ERROR CORRECTION CODING

Mathematical Methods and Algorithms

TODD K. MOON

WWW.
LINK AVAILABLE

Error Correction Coding

Mathematical Methods and Algorithms

This Page Intentionally Left Blank

Error Correction Coding

This Page Intentionally Left Blank

Error Correction Coding

Mathematical Methods and Algorithms

Todd K. Moon
Utah State University

A JOHN WILEY & SONS, INC., PUBLICATION

Copyright © 2005 by John Wiley & Sons, Inc. All rights reserved.

Published by John Wiley & Sons, Inc., Hoboken, New Jersey.
Published simultaneously in Canada.

No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning or otherwise, except as permitted under Section 107 or 108 of the 1976 United States Copyright Act, without either the prior written permission of the Publisher, or authorization through payment of the appropriate per-copy fee to the Copyright Clearance Center, Inc., 222 Rosewood Drive, Danvers, MA 01923, (978) 750-8400, fax (978) 646-8600, or on the web at www.copyright.com. Requests to the Publisher for permission should be addressed to the Permissions Department, John Wiley & Sons, Inc., 111 River Street, Hoboken, NJ 07030, (201) 748-6011, fax (201) 748-6008.

Limit of Liability/Disclaimer of Warranty: While the publisher and author have used their best efforts in preparing this book, they make no representation or warranties with respect to the accuracy or completeness of the contents of this book and specifically disclaim any implied warranties of merchantability or fitness for a particular purpose. No warranty may be created or extended by sales representatives or written sales materials. The advice and strategies contained herein may not be suitable for your situation. You should consult with a professional where appropriate. Neither the publisher nor author shall be liable for any loss of profit or any other commercial damages, including but not limited to special, incidental, consequential, or other damages.

For general information on our other products and services please contact our Customer Care Department within the U.S. at 877-762-2974, outside the U.S. at 317-572-3993 or fax 317-572-4002.

Wiley also publishes its books in a variety of electronic formats. Some content that appears in print, however, may not be available in electronic format.

Library of Congress Cataloging-in-Publication Data:

Moon, Todd K.
Error correction coding : mathematical methods and algorithms / Todd K. Moon.
p. cm.
Includes bibliographical references and index.
ISBN 0-471-64800-0 (cloth)
1. Engineering mathematics. 2. Error-correcting codes (Information theory) I. Title.

TA331.M66 2005
621.382'0285'572—dc22 2004031019

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

Error Correction Coding

This Page Intentionally Left Blank

Preface

The purpose of this book is to provide a comprehensive introduction to error correction coding, including both classical block- and trellis-based codes and the recent developments in iteratively decoded codes such as turbo codes and low-density parity-check codes. The presentation is intended to provide a background useful both to engineers, who need to understand algorithmic aspects for the deployment and implementation of error correction coding, and to researchers, who need sufficient background to prepare them to read, understand, and ultimately contribute to the research literature. The practical algorithmic aspects are built upon a firm foundation of mathematics, which are carefully motivated and developed.

Pedagogical Features

Since its inception, coding theory has drawn from a rich and interacting variety of mathematical areas, including detection theory, information theory, linear algebra, finite geometries, combinatorics, optimization, system theory, probability, algebraic geometry, graph theory, statistical designs, Boolean functions, number theory, and modern algebra. The level of sophistication has increased over time: algebra has progressed from vector spaces to modules; practice has moved from polynomial interpolation to rational interpolation; Viterbi makes way for BCJR. This richness can be bewildering to students, particularly engineering students who are unaccustomed to posing problems and thinking abstractly. It is important, therefore, to motivate the mathematics carefully.

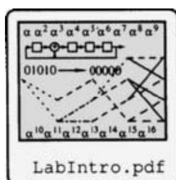
Some of the major pedagogical features of the book are as follows.

- While most engineering-oriented error-correction-coding textbooks clump the major mathematical concepts into a single chapter, in this book the concepts are developed over several chapters so they can be put to more immediate use. I have attempted to present the mathematics "just in time," when they are needed and well-motivated. Groups and linear algebra suffice to describe linear block codes. Cyclic codes motivate polynomial rings. The design of cyclic codes motivates finite fields and associated number-theoretical tools. By interspersing the mathematical concepts with applications, a deeper and broader understanding is possible.
- For most engineering students, finite fields, the Berlekamp-Massey algorithm, the Viterbi algorithm, BCJR, and other aspects of coding theory are initially abstract and subtle. Software implementations of the algorithms brings these abstractions closer to a meaningful reality, bringing deeper understanding than is possible by simply working homework problems and taking tests. Even when students grasp the concepts well enough to do homework on paper, these programs provide a further emphasis, as well as tools to *help* with the homework. The understanding becomes *experiential*, more than merely conceptual.

  Understanding of any subject typically improves when the student him- or herself has the chance to teach the material to someone (or something) else. A student must develop an especially clear understanding of a concept in order to "teach" it to something as dim-witted and literal-minded as a computer. In this process the

computer can provide feedback to the student through debugging and program testing that reinforces understanding.

In the coding courses I teach, students implement a variety of encoders and decoders, including Reed-Solomon encoders and decoders, convolutional encoders, turbo code decoders, and LDPC decoders. As a result of these programming activities, students move beyond an on-paper understanding, gaining a perspective of what coding theory can do and how to put it to work. A colleague of mine observed that many students emerge from a first course in coding theory more confused than informed. My experience with these programming exercises is that my students are, if anything, overconfident, and feel ready to take on a variety of challenges.



In this book, programming exercises are presented in a series of 13 Laboratory Exercises. These are supported with code providing most of the software "infrastructure," allowing students to focus on the particular algorithm they are implementing.

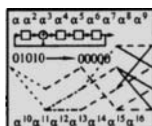
These labs also help with the coverage of the course material. In my course I am able to offload classroom instruction of some topics for students to read, with the assurance that the students will learn it solidly on their own as they implement it. (The Euclidean algorithm is one of these topics in my course.)

Research in error control coding can benefit from having a flexible library of tools for the computations, particularly since analytical results are frequently not available and simulations are required. The laboratory assignments presented here can form the foundation for a research library, with the added benefit that having written major components, the researcher can easily modify and extend them.

It is in light of these pedagogic features that this book bears the subtitle *Mathematical Methods and Algorithms.*

There is sufficient material in this book for a one- or two-semester course based on the book, even for instructors who prefer to focus less on implementational aspects and the laboratories.

Over 150 programs, functions and data files are associated with the text. The programs are written in Matlab,[1] C, or C++. Some of these include complete executables which provide "tables" of primitive polynomials (over any prime field), cyclotomic cosets and minimal polynomials, and BCH codes (not just narrow sense), avoiding the need to tabulate this material. Other functions include those used to make plots and compute results in the book. These provide example of how the theory is put into practice. Other functions include those used for the laboratory exercises. The files are highlighted in the book by the icon



as in the marginal note above. The files are available at the website

`http://ftp.wiley.com/public/sci_tech_med/error_control`

Other aspects of the book include the following:

[1] Matlab is a registered trademard of The Mathworks, Inc.

- Many recent advances in coding have resulted from returning to the perspective of coding as a detection problem. Accordingly, the book starts off with a digital communication framework with a discussion of **detection theory**.
- Recent codes are capable of nearly achieving capacity. It is important, therefore, to understand what capacity is and what it means to transmit at capacity. Chapter 1 also summarizes **information theory**, to put coding into its historical and modern context. This information theory also is used in the EXIT chart analysis of turbo and LDPC codes.
- Pedagogically, Hamming codes are used to set the stage for the book by using them to demonstrate block codes, cyclic codes, trellises and Tanner graphs.
- Homework exercises are drawn from a variety of sources and are at a variety of levels. Some are numerical, testing basic understanding of concepts. Others provide the opportunity to prove or extend results from the text. Others extend concepts or provide new results. Because of the programming laboratories, exercises requiring decoding by hand of given bit sequences are few, since I am of the opinion that is better to know how to tell the computer than to do it by hand. I have drawn these exercises from a variety of sources, including problems that I faced as a student and those which I have given to students on homework and exams over the years.
- Number theoretic concepts such as divisibility, congruence, and the Chinese remainder theorem are developed.
- At points throughout the book, connections between the coding theoretic concepts and related topics are pointed out, such as **public key cryptography** and **shift register sequences**. These add spice and motivate students with the understanding that the tools they are learning have broad applicability.
- There has been considerable recent progress made in decoding Reed-Solomon codes by re-examining their original definition. Accordingly, Reed-Solomon codes are defined both in this primordial way (as the image of a polynomial function) and also using a generator polynomial having roots that are consecutive powers of a primitive element. This sets the stage for several decoding algorithms for Reed-Solomon codes, including frequency-domain algorithms, **Welch-Berlekamp algorithm** and the **soft-input Guruswami-Sudan algorithm**.
- **Turbo codes**, including EXIT chart analysis, are presented, with both BCJR and SOVA decoding algorithms. Both probabilistic and likelihood decoding viewpoints are presented.
- **LDPC codes** are presented with an emphasis on the decoding algorithm. Density evolution analysis is also presented.
- **Decoding algorithms on graphs** which subsume both turbo code and LDPC code decoders, are presented.
- A summary of **log likelihood algebra**, used in soft-decision decoding, is presented.
- **Space-time codes**, used for multi-antenna systems in fading channels, are presented.

Courses of Study

A variety of courses of study are possible. In the one-semester course I teach, I move quickly through principal topics of block, trellis, and iteratively-decoded codes. Here is an outline

of one possible one-semester course:

Chapter 1: Major topics only.

Chapter 2: All.

Chapter 3: Major topics.

Chapter 4: Most. Leave CRC codes and LFSR to labs.

Chapter 5: Most. Leave Euclidean algorithm to lab; skip CRT; skip RSA.

Chapter 6: Basic topics.

Chapter 12: Most. Skip puncturing, stack-oriented algorithms and trellis descriptions of block codes

Chapter 13: Most. Skip the V.34 material.

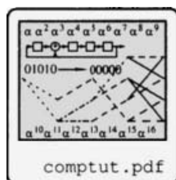
Chapter 14: Basic definition and the BCJR algorithm.

Chapter 15: Basic definition and the sum-product decoder.

A guide in selecting material for this course is: follow the labs. To get through all 13 labs, selectivity is necessary.

An alternative two-semester course could be a semester devoted to block codes followed by a semester on trellis and iteratively decoded codes. A two semester sequence could move straight through the book, with possible supplements from the research literature on topics of particular interest to the instructor.

The reader should be aware that theorems, lemmas, and corollaries are all numbered sequentially using the same counter in a chapter. Examples, definitions, figures, tables, and equations each have their own counters. Definitions, proofs and examples are all terminated by the symbol □.



Use of Computers

The computer-based labs provide a means of working out some of the computational details that otherwise might require drudgery. There are in addition many tools available, both for modest cost and for free. The brief tutorial `comptut.pdf` provides an introduction to `gap` and `magma`, both of which can be helpful to students doing homework or research in this area.

Acknowledgments

In my mind, the purposes of a textbook are these:

1. To provide a topographical map into the field of study, showing the peaks and the important roads to them. (However, in an area as rich as coding theory, it is unfortunately impossible to be exhaustive.)
2. To provide specific details about important techniques.
3. To present challenging exercises that will strengthen students' understanding of the material and present new perspectives.
4. To have some reference value, so that practitioners can continue to use it.

5. To provide references to literature that will lead readers to make their own discoveries. (With a rapidly-changing field, the references can only provide highlights; web-based searches have changed the nature of the game. Nevertheless, having a starting point in the literature is still important.)

A significant difficulty I have faced is selection. The terrain is so richly textured that it cannot be mapped out in a single book. Every conference and every issue of the *IEEE Transactions on Information Theory* yields new and significant results. Publishing restrictions and practicality limit this book from being encyclopedic. My role as author has been merely to select what parts of the map to include and to present them in a pedagogically useful way. In so doing, I have aimed to choose tools for the general practitioner and student. Other than that selective role, no claim of creation is intended; I hope I have given credit as appropriate where it is due.

This book is a result of teaching a course in error correction coding at Utah State University for over a decade. Over that time, I have taught out of the books [33], [373], and [203], and my debt to these books is clear. Parts of some chapters grew out of lecture notes based on these books and the connections will be obvious. I have felt compelled to include many of the exercises from the first coding course I took out of [203]. These books have defined for me the *sine qua non* of error-correction coding texts. I am also indebted to [220] for its rich theoretical treatment, [303] for presentation of trellis coding material, [350] for discussion of bounds, [141] for exhaustive treatment of turbo coding methods, and to the many great researchers and outstanding expositors whose works have contributed to my understanding.

I am grateful for the supportive environment at Utah State University that has made it possible to undertake and to complete this task. Students in coding classes over the years have contributed to this material, and the students in ECE 7670 class of Spring 2005 have combed carefully through the text. Stewart Weber and Ray Rallison have improved my C++ code. Thanks to Ojas Chauhan, who produced the performance curves for convolutional codes. I am especially grateful to John Crockett, who gave a particularly careful reading and contributed to the EXIT charts for LDPC codes. He also did the solutions for the first three chapters of the solutions manual. With all the help I have had in trying to produce clean copy, I alone am responsible for any remaining errors.

To my six wonderful children — Leslie, Kyra, Kaylie, Jennie, Kiana, and Spencer — and my wife Barbara, who have seen me slip away too often and too long to write, I express my deep gratitude for their trust and patience. In the end, all I do is for them.

T.K.M
Logan, UT, Mar. 2005

This Page Intentionally Left Blank

Contents

Preface vii

List of Program Files xxxi

List of Laboratory Exercises xxxii

List of Algorithms xxxiv

List of Figures xl

List of Tables xlii

List of Boxes xliii

Part I Introduction and Foundations 1

1 A Context for Error Correction Coding 2

1.1 Purpose of This Book . . . 2
1.2 Introduction: Where Are Codes? . . . 2
1.3 The Communications System . . . 4
1.4 Basic Digital Communications . . . 9
1.4.1 Binary Phase-Shift Keying . . . 10
1.4.2 More General Digital Modulation . . . 11
1.5 Signal Detection . . . 14
1.5.1 The Gaussian Channel . . . 14
1.5.2 MAP and ML Detection . . . 16
1.5.3 Special Case: Binary Detection . . . 18
1.5.4 Probability of Error for Binary Detection . . . 19
1.5.5 Bounds on Performance: The Union Bound . . . 22
1.5.6 The Binary Symmetric Channel . . . 23
1.5.7 The BSC and the Gaussian Channel Model . . . 25
1.6 Memoryless Channels . . . 25
1.7 Simulation and Energy Considerations for Coded Signals . . . 26
1.8 Some Important Definitions . . . 27
1.8.1 Detection of Repetition Codes Over a BSC . . . 28
1.8.2 Soft-Decision Decoding of Repetition Codes Over the AWGN . . . 32
1.8.3 Simulation of Results . . . 33
1.8.4 Summary . . . 33
1.9 Hamming Codes . . . 34
1.9.1 Hard-Input Decoding Hamming Codes . . . 35
1.9.2 Other Representations of the Hamming Code . . . 36
An Algebraic Representation . . . 37
A Polynomial Representation . . . 37

A Trellis Representation 38
The Tanner Graph Representation 38
1.10 The Basic Questions 39
1.11 Historical Milestones of Coding Theory 40
1.12 A Bit of Information Theory 40
1.12.1 Definitions for Discrete Random Variables 40
Entropy and Conditional Entropy 40
Relative Entropy, Mutual Information, and Channel Capacity 41
1.12.2 Definitions for Continuous Random Variables 43
1.12.3 The Channel Coding Theorem 45
1.12.4 "Proof" of the Channel Coding Theorem 45
1.12.5 Capacity for the Continuous-Time AWGN Channel 49
1.12.6 Transmission at Capacity with Errors 51
1.12.7 The Implication of the Channel Coding Theorem 52
Lab 1 Simulating a Communications Channel 53
Objective 53
Background 53
Use of Coding in Conjunction with the BSC 53
Assignment 54
Programming Part 54
Resources and Implementation Suggestions 54
1.13 Exercises 56
1.14 References 60

Part II Block Codes **61**

2 Groups and Vector Spaces **62**
2.1 Introduction 62
2.2 Groups 62
2.2.1 Subgroups 65
2.2.2 Cyclic Groups and the Order of an Element 66
2.2.3 Cosets 67
2.2.4 Lagrange's Theorem 68
2.2.5 Induced Operations; Isomorphism 69
2.2.6 Homomorphism 72
2.3 Fields: A Prelude 73
2.4 Review of Linear Algebra 75
2.5 Exercises 80
2.6 References 82

3 Linear Block Codes **83**
3.1 Basic Definitions 83
3.2 The Generator Matrix Description of Linear Block Codes 84
3.2.1 Rudimentary Implementation 86
3.3 The Parity Check Matrix and Dual Codes 86
3.3.1 Some Simple Bounds on Block Codes 88
3.4 Error Detection and Correction over Hard-Input Channels 90

3.4.1 Error Detection 90
3.4.2 Error Correction: The Standard Array 90
3.5 Weight Distributions of Codes and Their Duals 95
3.6 Hamming Codes and Their Duals 97
3.7 Performance of Linear Codes 98
3.7.1 Error detection performance 99
3.7.2 Error Correction Performance 100
3.7.3 Performance for Soft-Decision Decoding 103
3.8 Erasure Decoding 104
3.8.1 Binary Erasure Decoding 105
3.9 Modifications to Linear Codes 105
3.10 Best Known Linear Block Codes 107
3.11 Exercises 107
3.12 References 112

4 Cyclic Codes, Rings, and Polynomials **113**
4.1 Introduction 113
4.2 Basic Definitions 113
4.3 Rings 114
4.3.1 Rings of Polynomials 115
4.4 Quotient Rings 116
4.5 Ideals in Rings 118
4.6 Algebraic Description of Cyclic Codes 120
4.7 Nonsystematic Encoding and Parity Check 122
4.8 Systematic Encoding 124
4.9 Some Hardware Background 126
4.9.1 Computational Building Blocks 126
4.9.2 Sequences and Power series 127
4.9.3 Polynomial Multiplication 128
Last-Element-First Processing 128
First-Element-First Processing 128
4.9.4 Polynomial division 129
Last-Element-First Processing 129
4.9.5 Simultaneous Polynomial Division and Multiplication 132
First-Element-First Processing 132
4.10 Cyclic Encoding 133
4.11 Syndrome Decoding 137
4.12 Shortened Cyclic Codes 143
Method 1: Simulating the Extra Clock Shifts 144
Method 2: Changing the Error Pattern Detection Circuit 147
4.13 Binary CRC Codes 147
4.13.1 Byte-Oriented Encoding and Decoding Algorithms 150
4.13.2 CRC Protecting Data Files or Data Packets 153
Appendix 4.A Linear Feedback Shift Registers 154
Appendix 4.A.1 Basic Concepts 154
Appendix 4.A.2 Connection With Polynomial Division 157
Appendix 4.A.3 Some Algebraic Properties of Shift Sequences 160

Lab 2 Polynomial Division and Linear Feedback Shift Registers . . . 161
Objective . . . 161
Preliminary Exercises . . . 161
Programming Part: `BinLFSR` . . . 161
Resources and Implementation Suggestions . . . 161
Programming Part: `BinPolyDiv` . . . 162
Follow-On Ideas and Problems . . . 162
Lab 3 CRC Encoding and Decoding . . . 162
Objective . . . 163
Preliminary . . . 163
Programming Part . . . 163
Resources and Implementation Suggestions . . . 163
4.14 Exercises . . . 165
4.15 References . . . 170

5 Rudiments of Number Theory and Algebra **171**
5.1 Motivation . . . 171
5.2 Number Theoretic Preliminaries . . . 175
5.2.1 Divisibility . . . 175
5.2.2 The Euclidean Algorithm and Euclidean Domains . . . 177
5.2.3 The Sugiyama Algorithm . . . 182
5.2.4 Congruence . . . 184
5.2.5 The ϕ Function . . . 185
5.2.6 Some Cryptographic Payoff . . . 186
Fermat's Little Theorem . . . 186
RSA Encryption . . . 187
5.3 The Chinese Remainder Theorem . . . 188
5.3.1 The CRT and Interpolation . . . 190
The Evaluation Homomorphism . . . 190
The Interpolation Problem . . . 191
5.4 Fields . . . 193
5.4.1 An Examination of $\mathbb{R}$ and $\mathbb{C}$ . . . 194
5.4.2 Galois Field Construction: An Example . . . 196
5.4.3 Connection with Linear Feedback Shift Registers . . . 199
5.5 Galois Fields: Mathematical Facts . . . 200
5.6 Implementing Galois Field Arithmetic . . . 204
5.6.1 Zech Logarithms . . . 204
5.6.2 Hardware Implementations . . . 205
5.7 Subfields of Galois Fields . . . 206
5.8 Irreducible and Primitive polynomials . . . 207
5.9 Conjugate Elements and Minimal Polynomials . . . 209
5.9.1 Minimal Polynomials . . . 212
5.10 Factoring $x^n - 1$ . . . 215
5.11 Cyclotomic Cosets . . . 217
Appendix 5.A How Many Irreducible Polynomials Are There? . . . 218
Appendix 5.A.1 Solving for I_m Explicitly: The Moebius Function . . . 222
Lab 4 Programming the Euclidean Algorithm . . . 223

Objective . . . 223
Preliminary Exercises . . . 223
Background . . . 223
Programming Part . . . 223
Lab 5 Programming Galois Field Arithmetic . . . 224
Objective . . . 224
Preliminary Exercises . . . 224
Programming Part . . . 224
5.12 Exercises . . . 225
5.13 References . . . 234

6 BCH and Reed-Solomon Codes: Designer Cyclic Codes **235**
6.1 BCH Codes . . . 235
6.1.1 Designing BCH Codes . . . 235
6.1.2 The BCH Bound . . . 237
6.1.3 Weight Distributions for Some Binary BCH Codes . . . 239
6.1.4 Asymptotic Results for BCH Codes . . . 240
6.2 Reed-Solomon Codes . . . 242
6.2.1 Reed-Solomon Construction 1 . . . 242
6.2.2 Reed-Solomon Construction 2 . . . 243
6.2.3 Encoding Reed-Solomon Codes . . . 244
6.2.4 MDS Codes and Weight Distributions for RS Codes . . . 245
6.3 Decoding BCH and RS Codes: The General Outline . . . 247
6.3.1 Computation of the Syndrome . . . 247
6.3.2 The Error Locator Polynomial . . . 248
6.3.3 Chien Search . . . 248
6.4 Finding the Error Locator Polynomial . . . 250
6.4.1 Simplifications for Binary Codes and Peterson's Algorithm . . . 251
6.4.2 Berlekamp-Massey Algorithm . . . 253
6.4.3 Characterization of LFSR Length in Massey's Algorithm . . . 255
6.4.4 Simplifications for Binary Codes . . . 259
6.5 Non-Binary BCH and RS Decoding . . . 261
6.5.1 Forney's Algorithm . . . 262
6.6 Euclidean Algorithm for the Error Locator Polynomial . . . 266
6.7 Erasure Decoding for Nonbinary BCH or RS codes . . . 267
6.8 Galois Field Fourier Transform Methods . . . 269
6.8.1 Equivalence of the Two Reed-Solomon Code Constructions . . . 274
6.8.2 Frequency-Domain Decoding . . . 275
6.9 Variations and Extensions of Reed-Solomon Codes . . . 276
6.9.1 Simple Modifications . . . 276
6.9.2 Generalized Reed-Solomon Codes and Alternant Codes . . . 277
6.9.3 Goppa Codes . . . 278
6.9.4 Decoding Alternant Codes . . . 280
6.9.5 The McEliece Public Key Cryptosystem . . . 280
Lab 6 Programming the Berlekamp-Massey Algorithm . . . 281
Background . . . 281
Assignment . . . 281

Preliminary Exercises . . . 281
Programming Part . . . 281
Resources and Implementation Suggestions . . . 282
Lab 7 Programming the BCH Decoder . . . 283
Objective . . . 283
Preliminary Exercises . . . 283
Programming Part . . . 283
Resources and Implementation Suggestions . . . 283
Follow-On Ideas and Problems . . . 284
Lab 8 Reed-Solomon Encoding and Decoding . . . 284
Objective . . . 284
Background . . . 284
Programming Part . . . 284
Appendix 6.A Proof of Newton's Identities . . . 285
6.10 Exercises . . . 287
6.11 References . . . 291

7 Alternate Decoding Algorithms for Reed-Solomon Codes **293**
7.1 Introduction: Workload for Reed-Solomon Decoding . . . 293
7.2 Derivations of Welch-Berlekamp Key Equation . . . 293
7.2.1 The Welch-Berlekamp Derivation of the WB Key Equation . . . 294
7.2.2 Derivation From the Conventional Key Equation . . . 298
7.3 Finding the Error Values . . . 300
7.4 Methods of Solving the WB Key Equation . . . 302
7.4.1 Background: Modules . . . 302
7.4.2 The Welch-Berlekamp Algorithm . . . 303
7.4.3 Modular Solution of the WB Key Equation . . . 310
7.5 Erasure Decoding with the Welch-Berlekamp Key Equation . . . 321
7.6 The Guruswami-Sudan Decoding Algorithm and Soft RS Decoding . . . 322
7.6.1 Bounded Distance, ML, and List Decoding . . . 322
7.6.2 Error Correction by Interpolation . . . 323
7.6.3 Polynomials in Two Variables . . . 324
Degree and Monomial Order . . . 325
Zeros and Multiple Zeros . . . 328
7.6.4 The GS Decoder: The Main Theorems . . . 330
The Interpolation Theorem . . . 331
The Factorization Theorem . . . 331
The Correction Distance . . . 333
The Number of Polynomials in the Decoding List . . . 335
7.6.5 Algorithms for Computing the Interpolation Step . . . 337
Finding Linearly Dependent Columns: The Feng-Tzeng Algorithm 338
Finding the Intersection of Kernels: The Kötter Algorithm . . . 342
7.6.6 A Special Case: $m = 1$ and $L = 1$ . . . 348
7.6.7 The Roth-Ruckenstein Algorithm . . . 350
What to Do with Lists of Factors? . . . 354
7.6.8 Soft-Decision Decoding of Reed-Solomon Codes . . . 358
Notation . . . 358

A Factorization Theorem . . . 360
Mapping from Reliability to Multiplicity . . . 361
The Geometry of the Decoding Regions . . . 363
Computing the Reliability Matrix . . . 364
7.7 Exercises . . . 365
7.8 References . . . 368

8 Other Important Block Codes **369**
8.1 Introduction . . . 369
8.2 Hadamard Matrices, Codes, and Transforms . . . 369
8.2.1 Introduction to Hadamard Matrices . . . 369
8.2.2 The Paley Construction of Hadamard Matrices . . . 371
8.2.3 Hadamard Codes . . . 374
8.3 Reed-Muller Codes . . . 375
8.3.1 Boolean Functions . . . 375
8.3.2 Definition of the Reed-Muller Codes . . . 376
8.3.3 Encoding and Decoding Algorithms for First-Order RM Codes . . . 379
Encoding $RM(1, m)$ Codes . . . 379
Decoding $RM(1, m)$ Codes . . . 379
Expediting Decoding Using the Fast Hadamard Transform . . . 382
8.3.4 The Reed Decoding Algorithm for $RM(r, m)$ Codes, $r \geq 1$ . . . 384
Details for an $RM(2, 4)$ Code . . . 384
A Geometric Viewpoint . . . 387
8.3.5 Other Constructions of Reed-Muller Codes . . . 391
8.4 Building Long Codes from Short Codes: The Squaring Construction . . . 392
8.5 Quadratic Residue Codes . . . 396
8.6 Golay Codes . . . 398
8.6.1 Decoding the Golay Code . . . 400
Algebraic Decoding of the $\mathcal{G}_{23}$ Golay Code . . . 400
Arithmetic Decoding of the $\mathcal{G}_{24}$ Code . . . 401
8.7 Exercises . . . 403
8.8 References . . . 404

9 Bounds on Codes **406**
9.1 The Gilbert-Varshamov Bound . . . 409
9.2 The Plotkin Bound . . . 410
9.3 The Griesmer Bound . . . 411
9.4 The Linear Programming and Related Bounds . . . 413
9.4.1 Krawtchouk Polynomials . . . 415
9.4.2 Character . . . 415
9.4.3 Krawtchouk Polynomials and Characters . . . 416
9.5 The McEliece-Rodemich-Rumsey-Welch Bound . . . 418
9.6 Exercises . . . 420
9.7 References . . . 424

10 Bursty Channels, Interleavers, and Concatenation **425**
10.1 Introduction to Bursty Channels 425
10.2 Interleavers 425
10.3 An Application of Interleaved RS Codes: Compact Discs 427
10.4 Product Codes 430
10.5 Reed-Solomon Codes 431
10.6 Concatenated Codes 432
10.7 Fire Codes 433
10.7.1 Fire Code Definition 433
10.7.2 Decoding Fire Codes: Error Trapping Decoding 435
10.8 Exercises 437
10.9 References 438

11 Soft-Decision Decoding Algorithms **439**
11.1 Introduction and General Notation 439
11.2 Generalized Minimum Distance Decoding 441
11.2.1 Distance Measures and Properties 442
11.3 The Chase Decoding Algorithms 445
11.4 Halting the Search: An Optimality Condition 445
11.5 Ordered Statistic Decoding 447
11.6 Exercises 449
11.7 References 450

Part III Codes on Graphs 451

12 Convolutional Codes **452**
12.1 Introduction and Basic Notation 452
12.1.1 The State 456
12.2 Definition of Codes and Equivalent Codes 458
12.2.1 Catastrophic Encoders 461
12.2.2 Polynomial and Rational Encoders 464
12.2.3 Constraint Length and Minimal Encoders 465
12.2.4 Systematic Encoders 468
12.3 Decoding Convolutional Codes 469
12.3.1 Introduction and Notation 469
12.3.2 The Viterbi Algorithm 471
12.3.3 Some Implementation Issues 481
The Basic Operation: Add-Compare-Select 481
Decoding Streams of Data: Windows on the Trellis 481
Output Decisions 482
Hard and Soft Decoding; Quantization 484
Synchronization Issues 486
12.4 Some Performance Results 487
12.5 Error Analysis for Convolutional Codes 491
12.5.1 Enumerating Paths Through the Trellis 493
Enumerating on More Complicated Graphs: Mason's Rule 496

12.5.2 Characterizing the Node Error Probability P_e and the Bit Error Rate P_b . . . 498
12.5.3 A Bound on P_d for Discrete Channels . . . 501
Performance Bound on the BSC . . . 503
12.5.4 A Bound on P_d for BPSK Signaling Over the AWGN Channel . . . 503
12.5.5 Asymptotic Coding Gain . . . 504
12.6 Tables of Good Codes . . . 505
12.7 Puncturing . . . 507
12.7.1 Puncturing to Achieve Variable Rate . . . 509
12.8 Suboptimal Decoding Algorithms for Convolutional Codes . . . 510
12.8.1 Tree Representations . . . 511
12.8.2 The Fano Metric . . . 511
12.8.3 The Stack Algorithm . . . 515
12.8.4 The Fano Algorithm . . . 517
12.8.5 Other Issues for Sequential Decoding . . . 520
12.8.6 A Variation on the Viterbi Algorithm: The M Algorithm . . . 521
12.9 Convolutional Codes as Block Codes . . . 522
12.10 Trellis Representations of Block and Cyclic Codes . . . 523
12.10.1 Block Codes . . . 523
12.10.2 Cyclic Codes . . . 524
12.10.3 Trellis Decoding of Block Codes . . . 525
Lab 9 Programming Convolutional Encoders . . . 526
Objective . . . 526
Background . . . 526
Programming Part . . . 526
Lab 10 Convolutional Decoders: The Viterbi Algorithm . . . 528
Objective . . . 528
Background . . . 528
Programming Part . . . 528
12.11 Exercises . . . 529
12.12 References . . . 533

13 Trellis Coded Modulation **535**
13.1 Adding Redundancy by Adding Signals . . . 535
13.2 Background on Signal Constellations . . . 535
13.3 TCM Example . . . 537
13.3.1 The General Ungerboeck Coding Framework . . . 544
13.3.2 The Set Partitioning Idea . . . 545
13.4 Some Error Analysis for TCM Codes . . . 546
13.4.1 General Considerations . . . 546
13.4.2 A Description of the Error Events . . . 548
13.4.3 Known Good TCM Codes . . . 552
13.5 Decoding TCM Codes . . . 554
13.6 Rotational Invariance . . . 556
Differential Encoding . . . 558
Constellation Labels and Partitions . . . 559
13.7 Multidimensional TCM . . . 561

13.7.1 Some Advantages of Multidimensional TCM 562
13.7.2 Lattices and Sublattices 563
Basic Definitions 563
Common Lattices 565
Sublattices and Cosets 566
The Lattice Code Idea 567
Sources of Coding Gain in Lattice Codes 567
Some Good Lattice Codes 571
13.8 The V.34 Modem Standard 571
Lab 11 Trellis-Coded Modulation Encoding and Decoding 578
Objective 578
Background 578
Programming Part 578
13.9 Exercises 578
13.10 References 580

Part IV Iteratively Decoded Codes **581**

14 Turbo Codes **582**
14.1 Introduction 582
14.2 Encoding Parallel Concatenated Codes 584
14.3 Turbo Decoding Algorithms 586
14.3.1 The MAP Decoding Algorithm 588
14.3.2 Notation 588
14.3.3 Posterior Probability 590
14.3.4 Computing α_t and β_t 592
14.3.5 Computing γ_t 593
14.3.6 Normalization 594
14.3.7 Summary of the BCJR Algorithm 596
14.3.8 A Matrix/Vector Formulation 597
14.3.9 Comparison of the Viterbi and BCJR Algorithms 598
14.3.10 The BCJR Algorithm for Systematic Codes 598
14.3.11 Turbo Decoding Using the BCJR Algorithm 600
The Terminal State of the Encoders 602
14.3.12 Likelihood Ratio Decoding 602
Log Prior Ratio $\lambda_{p,t}$ 603
Log Posterior $\lambda_{s,t}^{(0)}$ 605
14.3.13 Statement of the Turbo Decoding Algorithm 605
14.3.14 Turbo Decoding Stopping Criteria 605
The Cross Entropy Stopping Criterion 606
The Sign Change Ratio (SCR) Criterion 607
The Hard Decision Aided (HDA) Criterion 608
14.3.15 Modifications of the MAP Algorithm 608
The Max-Log-MAP Algorithm 608
14.3.16 Corrections to the Max-Log-MAP Algorithm 609
14.3.17 The Soft Output Viterbi Algorithm 610
14.4 On the Error Floor and Weight Distributions 612

14.4.1 The Error Floor 612
14.4.2 Spectral Thinning and Random Interleavers 614
14.4.3 On Interleavers 618
14.5 EXIT Chart Analysis 619
14.5.1 The EXIT Chart 622
14.6 Block Turbo Coding 623
14.7 Turbo Equalization 626
14.7.1 Introduction to Turbo Equalization 626
14.7.2 The Framework for Turbo Equalization 627
Lab 12 Turbo Code Decoding 629
Objective 629
Background 629
Programming Part 629
14.8 Exercises 629
14.9 References 632

15 Low-Density Parity-Check Codes **634**
15.1 Introduction 634
15.2 LDPC Codes: Construction and Notation 635
15.3 Tanner Graphs 638
15.4 Transmission Through a Gaussian Channel 638
15.5 Decoding LDPC Codes 640
15.5.1 The Vertical Step: Updating $q_{mn}(x)$ 641
15.5.2 Horizontal Step: Updating $r_{mn}(x)$ 644
15.5.3 Terminating and Initializing the Decoding Algorithm 647
15.5.4 Summary of the Algorithm 648
15.5.5 Message Passing Viewpoint 649
15.5.6 Likelihood Ratio Decoder Formulation 649
15.6 Why Low-Density Parity-Check Codes? 653
15.7 The Iterative Decoder on General Block Codes 654
15.8 Density Evolution 655
15.9 EXIT Charts for LDPC Codes 659
15.10 Irregular LDPC Codes 660
15.10.1 Degree Distribution Pairs 662
15.10.2 Some Good Codes 664
15.10.3 Density Evolution for Irregular Codes 664
15.10.4 Computation and Optimization of Density Evolution 667
15.10.5 Using Irregular Codes 668
15.11 More on LDPC Code Construction 668
15.11.1 A Construction Based on Finite Geometries 668
15.11.2 Constructions Based on Other Combinatoric Objects 669
15.12 Encoding LDPC Codes 669
15.13 A Variation: Low-Density Generator Matrix Codes 671
15.14 Serial Concatenated Codes; Repeat-Accumulate Codes 671
15.14.1 Irregular RA Codes 673
Lab 13 Programming an LDPC Decoder 674
Objective 674

Background . . . 674
Assignment . . . 675
Numerical Considerations . . . 675
15.15 Exercises . . . 676
15.16 References . . . 679

16 Decoding Algorithms on Graphs 680
16.1 Introduction . . . 680
16.2 Operations in Semirings . . . 681
16.3 Functions on Local Domains . . . 681
16.4 Factor Graphs and Marginalization . . . 686
16.4.1 Marginalizing on a Single Variable . . . 687
16.4.2 Marginalizing on All Individual Variables . . . 691
16.5 Applications to Coding . . . 694
16.5.1 Block Codes . . . 694
16.5.2 Modifications to Message Passing for Binary Variables . . . 695
16.5.3 Trellis Processing and the Forward/Backward Algorithm . . . 696
16.5.4 Turbo Codes . . . 699
16.6 Summary of Decoding Algorithms on Graphs . . . 699
16.7 Transformations of Factor Graphs . . . 700
16.7.1 Clustering . . . 700
16.7.2 Stretching Variable Nodes . . . 701
16.7.3 Exact Computation of Graphs with Cycles . . . 702
16.8 Exercises . . . 706
16.9 References . . . 708

Part V Space-Time Coding 709

17 Fading Channels and Space-Time Codes 710
17.1 Introduction . . . 710
17.2 Fading Channels . . . 710
17.2.1 Rayleigh Fading . . . 712
17.3 Diversity Transmission and Reception: The MIMO Channel . . . 714
17.3.1 The Narrowband MIMO Channel . . . 716
17.3.2 Diversity Performance with Maximal-Ratio Combining . . . 717
17.4 Space-Time Block Codes . . . 719
17.4.1 The Alamouti Code . . . 719
17.4.2 A More General Formulation . . . 721
17.4.3 Performance Calculation . . . 721
Real Orthogonal Designs . . . 723
Encoding and Decoding Based on Orthogonal Designs . . . 724
Generalized Real Orthogonal Designs . . . 726
17.4.4 Complex Orthogonal Designs . . . 727
Future Work . . . 728
17.5 Space-Time Trellis Codes . . . 728
17.5.1 Concatenation . . . 729
17.6 How Many Antennas? . . . 732

17.7 Estimating Channel Information 733
17.8 Exercises 733
17.9 References 734

A Log Likelihood Algebra **735**
A.1 Exercises 737

References **739**

Index **750**

This Page Intentionally Left Blank

List of Program Files

`comptut.pdf` Introduction to `gap` and `magma` x
`qf.c` Q function 20
`qf.m` Q function 20
`bpskprobplot.m` Set up to plot prob. of error for BPSK 21
`bpskprob.m` Plot probability of error for BPSK 21
`repcodeprob.m` Compute error probability for $(n, 1)$ repetition codes 32
`testrepcode.cc` Test the repetition code performance 33
`repcodes.m` Plot results for repetition code 33
`mindist.m` Find minimum distance using exhaustive search 34
`hamcode74pe.m` Probability of error for (7, 4) Hamming code 36
`nchoosektest.m` Compute $\binom{n}{k}$ and test for $k < 0$ 36
`plotcapcmp.m` Capacity of the AWGN channel and BAWGNC channel . . 45
`cawgnc2.m` Compute capacity of AWGN channel 45
`cbawgnc2.m` Compute the capacity of the BAWGN channel 45
`h2.m` Compute the binary entropy function 45
`plotcbawn2.m` Plot capacity for AWGN and BAWGN channels 51
`cbawgnc.m` Compute capacity for BAWGN channel 51
`cawgnc.m` Compute capacity for AWGN channel 51
`philog.m` Compute the log ϕ function associated with the BAWGNC 51
`phifun.m` Compute the ϕ function associated with the BAWGNC . . 51
`gaussj2` Gaussian elimination over $GF(2)$ 86
`Hammsphere` Compute the number of points in a Hamming sphere 89
`genstdarray.c` Generate a standard array for a code 91
`progdetH15.m` Probability of error detection for (15, 11) Hamming code . 100
`progdet.m` Probability of error detection for (31, 21) Hamming code . 100
`polyadd.m` Add polynomials 116
`polysub.m` Subtract polynomials 116
`polymult.m` Multiply polynomials 116
`polydiv.m` Divide polynomials (compute quotient and remainder) . . . 116
`polyaddm.m` Add polynomials modulo a number 116
`polysubm.m` Subtract polynomials modulo a number 116
`polymultm.m` Multiply polynomials modulo a number 116
`primitive.txt` Table of primitive polynomials 155
`BinLFSR.h` (lab, complete) Binary LFSR class 162
`BinLFSR.cc` (lab, complete) Binary LFSR class 162
`testBinLFSR.cc` (lab, complete) Binary LFSR class tester 162
`MakeLFSR` (lab, complete) Makefile for tester 162
`BinPolyDiv.h` (lab, complete) Binary polynomial division 162
`BinPolyDiv.cc` (lab, incomplete) Binary polynomial division 162
`testBinPolyDiv.cc` (lab, complete) Binary polynomial division test 162
`gcd.c` A simple example of the Euclidean algorithm 181
`crtgamma.m` Compute the gammas for the CRT 189

fromcrt.m	Convert back from CRT representation to an integer	189
tocrt.m	Compute the CRT representation of an integer	189
testcrt.m	An example of CRT calculations	189
testcrp.m	Test a polynomial CRT calculation	189
tocrtpoly.m	Compute the CRT representation of a polynomial	189
fromcrtpolym.	Compute a polynomial from a CRT representation	189
crtgammapoly.m	Compute the gammas for the CRT representation	189
primfind	**Executable:** Find primitive polynomials in $GF(p)[x]$	209
cyclomin	**Executable:** Cyclotomic cosets and minimal polynomials	217
ModAr.h	(lab, complete) Modulo arithmetic class	223
ModAr.cc	(lab, complete) Modulo arithmetic class	223
ModArnew.cc	Templatized Modulo arithmetic class	223
testmodarnew.cc	Templatized Modulo arithmetic class tester	223
testmodar1.cc	(lab, complete) Test Modulo arithmetic class	223
polynomialT.h	(lab, complete) Templatized polynomial class	223
polynomialT.cc	(lab, complete) Templatized polynomial class	223
testpoly1.cc	(lab, complete) Demonstrate templatized polynomial class	223
testgcdpoly.cc	(lab, complete) Test the polynomial GCD function	224
gcdpoly.cc	(lab, **incomplete**) Polynomial GCD function	224
GF2.h	(lab, complete) GF(2) class	224
GFNUM2m.h	(lab, complete) Galois field $GF(2^m)$ class	224
GFNUM2m.cc	(lab, **incomplete**) Galois field $GF(2^m)$ class	224
testgfnum.cc	(lab, complete) Test Galois field class	224
bchweight.m	Weight distribution of BCH code from weight of dual	240
bchdesigner	**Executable:** Design a t-error correcting binary BCH code	241
reedsolwt.m	Compute weight distribution for an (n, k) RS code	246
masseymodM.m	Return the shortest LFSR for data sequence	258
erase.mag	Erasure decoding example in magma	268
testBM.cc	(lab, complete) Test the Berlekamp-Massey algorithm	282
Chiensearch.h	(lab, complete) Chien Search class	283
Chiensearch.cc	(lab, **incomplete**) Chien Search class	283
testChien.cc	(lab, complete) Test the Chien Search class	283
BCHdec.h	(lab, complete) BCHdec decoder class	283
BCHdec.cc	(lab, **incomplete**) BCHdec decoder class	283
testBCH.cc	(lab, complete) BCHdec decoder class tester	283
RSenc.h	(lab, complete) RS encoder class header	284
RSenc.cc	(lab, complete) RS encoder class	284
RSdec.h	(lab, complete) RS decoder class header	284
RSdec.cc	(lab, **incomplete**) RS decoder class	284
testRS.cc	(lab, complete) Test RS decoder	284
rsencode.cc	(lab, complete) Encode a file of data using RS encoder	285
rsdecode.cc	(lab, complete) Decode a file of data using RS decoder	285
bsc.c	**Executable:** Simulate a binary symmetric channel	285
testpxy.cc	Demonstrate concepts relating to 2-variable polynomials	325
computekm.m	Compute K_m for the Guruswami-Sudan decoder	333
computeLm.cc	Compute the maximum list length for $GS(m)$ decoding	337
computeLm.m	Compute the maximum list length for $GS(m)$ decoding	337

testft.m	Test the Feng-Tzeng algorithm	341
fengtzeng.m	Poly. such that the first $l+1$ columns are lin. dep.	341
invmodp.m	Compute the inverse of a number modulo p	341
testGS1.cc	Test the GS decoder (Kotter part)	346
kotter.cc	Kotter interpolation algorithm	346
testGS3.cc	Test the GS decoder	347
testGS5.cc	Test the GS decoder	350
kotter1.cc	Kotter algorithm with $m = 1$	350
testGS2.cc	Test the Roth-Ruckenstein algorithm	354
rothruck.cc	Roth-Ruckenstein algorithm (find y-roots)	354
rothruck.h	Roth-Ruckenstein algorithm (find y-roots)	354
Lbarex.m	Average performance of a $GS(m)$ decoder	357
computetm.m	Compute T_m, error correction capability for GS decoder	357
computeLbar.m	Avg. no. of codewords in sphere around random pt.	357
computeLm.m	Compute maximum length of list of $GF(m)$ decoder	357
pi2ml	Koetter-Vardy algorithm to map reliability to multiplicity	362
genrm.cc	Create a Reed-Muller generator matrix	376
rmdecex.m	$RM(1,3)$ decoding example	381
hadex.m	Computation of H_8	382
testfht.cc	Test the fast Hadamard transform	383
fht.cc	Fast Hadamard transform	383
fht.m	Fast Hadamard transform	383
rmdecex2.m	$RM(2,4)$ decoding example	387
testQR.cc	Example of arithmetic for QR code decoding	397
golaysimp.m	Derive equations for algebraic Golay decoder	401
testGolay.cc	Test the algebraic Golay decoder	401
golayrith.m	Arithmetic Golay decoder	402
plotbds.m	Plot bounds for binary codes	407
simplex1.m	Linear program solution to problems in standard form	413
pivottableau.m	Main function in simplex1.m	413
reducefree.m	Auxiliary linear programming function	413
restorefree.m	Auxiliary linear programming function	413
krawtchouk.m	Compute Krawtchouk polynomials recursively	415
lpboundex.m	Solve the linear programming for the LP bound	418
utiltkm.cc	Sort and random functions	440
utiltkm.h	Sort and random functions	440
concodequant.m	Compute the quantization of the Euclidean metric	486
chernoff1.m	Chernoff bounds for convolutional performance	502
plotconprob.m	Plot performance bounds for a convolutional code	504
finddfree	**Executable:** Find d_{free} for connection coefficients	506
teststack.m	Test the stack algorithm	515
stackalg.m	The stack algorithm for convolutional decoding	515
fanomet.m	Compute the Fano metric for convolutionally coded data	515
fanoalg.m	The Fano algorithm for convolutional decoding	517
BinConv.h	(lab, complete) Base class for binary convolutional encoder	526
BinConvFIR.h	(lab, complete) Binary feedforward convolutional encoder	526
BinConvFIR.cc	(lab, **incomplete**) Binary feedforward convolutional encoder	526

BinConvIIR.h	(lab, complete) Binary recursive convolutional encoder	526
BinConvIIR.cc	(lab, **incomplete**) Binary recursive convolutional encoder	526
testconvenc	(lab, complete) Test convolutional encoders	526
Convdec.h	(lab, complete) Convolutional decoder class	528
Convdec.cc	(lab, **incomplete**) Convolutional decoder class	528
BinConvdec01.h	(lab, complete) Binary conv. decoder class, BSC metric	528
BinConvdec01.h	(lab, complete) Binary conv. decoder class, BSC metric	528
BinConvdecBPSK.h	(lab, complete) Bin. conv. decoder, soft metric with BPSK	528
BinConvdecBPSK.cc	(lab, complete) Bin. conv. decoder, soft metric with BPSK	528
testconvdec.cc	(lab, complete) Test the convolutional decoder	529
makeB.m	Make the B matrix for an example code	549
tcmt1.cc	Test the constellation for a rotationally invariant code	549
tcmrot2.cc	Test the constellation for a rotationally invariant code	557
lattstuff.m	Generator matrices for A_2, D_4, E_6, E_8, Λ_{16}, Λ_{24}	563
voln.m	Compute volume of n-dimensional unit-radius sphere	563
latta2.m	Plot A_2 lattice	568
lattz2m	Plot Z_2 lattice	568
BCJR.h	(lab, complete) BCJR algorithm class header	629
BCJR.cc	(lab, **incomplete**) BCJR algorithm class	629
testbcjr.cc	(lab, complete) Test BCJR algorithm class	629
testturbodec2.cc	(lab, complete) Test the turbo decoder	629
makgenfromA.m	Find systematic generator matrix for a parity check matrix	635
gaussj2.m	Gauss-Jordan elimination over $GF(2)$ on a matrix	635
Agall.m	A parity check matrix for an LDPC code	637
Agall.txt	Sparse representation of the matrix	637
writesparse.m	Write a matrix into a file in sparse format	637
ldpc.m	Demonstrate LDPC decoding	648
galdecode.m	LDPC decoder (nonsparse representation)	648
ldpclogdec.m	Log-likelihood LDPC decoder	652
psifunc.m	Plot the Ψ function used in density evolution	656
densev1.m	An example of density evolution	658
densevtest.m	Plot density evolution results	658
Psi.m	Plot the Ψ function used in density evolution	658
Psiinv.m	Compute Ψ^{-1} used in density evolution	658
threshtab.m	Convert threshold table to E_b/N_0	658
ldpcsim.mat	LDPC decoder simulation results	660
exit1.m	Plot histograms of LDPC decoder outputs	660
loghist.m	Find histograms	660
exit3.m	Plot mutual information as a function of iteration	660
dotrajectory.m	Mutual information as a function of iteration	660
exit2.m	Plot EXIT chart	660
doexitchart.m	Take mutual information to EXIT chart	660
getinf.m	Convert data to mutual information	660
getinfs.m	Convert multiple data to mutual information	660
sparseHno4.m	Make a sparse check matrix with no cycles of girth 4	668
Asmall.txt	(lab, complete) A matrix in sparse representation	675
galdec.h	(lab, complete) LDPC decoder class header	675

galdec.cc	(lab, **incomplete**) LDPC decoder class	675
galtest.cc	(lab, complete) LDPC decoder class tester	675
ldpc.m	(lab, complete) Demonstrate LDPC decoding	675
galdecode.m	(lab, complete) LDPC decoder (not sparse representation)	675
galtest2.cc	(lab, complete) Prob. of error plots for LDPC code	675
A1-2.txt	(lab, complete) Rate 1/2 parity check matrix, sparse format	675
A1-4.txt	(lab, complete) Rate 1/4 parity check matrix, sparse format	675
testgdl2.m	Test the generalized distributive law	695
gdl.m	A generalized distributive law function	695
fyx0.m	Compute $f(y\|x)$ for the GDL framework	695
fadeplot.m	Plot realizations of the amplitude of a fading channel	712
jakes.m	Jakes method for computing amplitude of a fading channel	712
fadepbplot.m	Plot BPSK performance over Rayleigh fading channel	714

List of Laboratory Exercises

Lab 1 Simulating a Communications Channel 53
Lab 2 Polynomial Division and Linear Feedback Shift Registers 161
Lab 3 CRC Encoding and Decoding 162
Lab 4 Programming the Euclidean Algorithm 223
Lab 5 Programming Galois Field Arithmetic 224
Lab 6 Programming the Berlekamp-Massey Algorithm 281
Lab 7 Programming the BCH Decoder 283
Lab 8 Reed-Solomon Encoding and Decoding 284
Lab 9 Programming Convolutional Encoders 526
Lab 10 Convolutional Decoders: The Viterbi Algorithm 528
Lab 11 Trellis-Coded Modulation Encoding and Decoding 578
Lab 12 Turbo Code Decoding 629
Lab 13 Programming an LDPC Decoder 674

List of Algorithms

1.1 Hamming Code Decoding . . . 35
1.2 Outline for simulating a digital communications channel . . . 53
1.3 Outline for simulating (n, k)-coded digital communications . . . 53
1.4 Outline for simulating (n, k) Hamming-coded digital communications . . . 54
4.1 Fast CRC encoding for a stream of bytes . . . 152
4.2 Binary linear feedback shift register . . . 162
4.3 Binary polynomial division . . . 162
5.1 Extended Euclidean Algorithm . . . 181
5.2 Modulo Arithmetic . . . 223
5.3 Templatized Polynomials . . . 223
5.4 Polynomial GCD . . . 223
5.5 $GF(2^m)$ arithmetic . . . 224
6.1 Massey's Algorithm (pseudocode) . . . 258
6.2 Massey's Algorithm for Binary BCH Decoding . . . 259
6.3 Test Berlekamp-Massey algorithm . . . 282
6.4 Chien Search . . . 283
6.5 BCH Decoder . . . 283
6.6 Reed-Solomon Encoder Declaration . . . 284
6.7 Reed-Solomon Decoder Declaration . . . 284
6.8 Reed-Solomon Decoder Testing . . . 284
6.9 Reed-Solomon File Encoder and Decoder . . . 285
6.10 Binary Symmetric Channel Simulator . . . 285
7.1 Welch-Berlekamp Interpolation . . . 308
7.2 Welch-Berlekamp Interpolation, Modular Method . . . 316
7.3 Welch-Berlekamp Interpolation, Modular Method v. 2 . . . 319
7.4 Welch-Berlekamp Interpolation, Modular Method v. 3 . . . 321
7.5 The Feng-Tzeng Algorithm . . . 341
7.6 Kötters Interpolation for Guruswami-Sudan Decoder . . . 346
7.7 Guruswami-Sudan Interpolation Decoder with $m = 1$ and $L = 1$ . . . 349
7.8 Roth-Ruckenstein Algorithm for Finding y-roots of $Q(x, y)$ . . . 353
7.9 Koetter-Vardy Algorithm for Mapping from Π to M . . . 362
8.1 Decoding for $RM(1, m)$ Codes . . . 381
8.2 Arithmetic Decoding of the Golay $\mathcal{G}_{24}$ Code . . . 402
11.1 Generalized Minimum Distance (GMD) Decoding . . . 441
11.2 Chase-2 Decoder . . . 445
11.3 Chase-3 Decoder . . . 445
11.4 Ordered Statistic Decoding . . . 449
12.1 The Viterbi Algorithm . . . 475
12.2 The Stack Algorithm . . . 515
12.3 Base Class for Binary Convolutional Encoder . . . 526
12.4 Derived classes for FIR and IIR Encoders . . . 526
12.5 Test program for convolutional encoders . . . 526

12.6 Base Decoder Class Declarations 528
12.7 Convolutional decoder for binary (0,1) data 528
12.8 Convolutional decoder for BPSK data 528
12.9 Test the convolutional decoder 529
14.1 The BCJR (MAP) Decoding Algorithm, Probability Form 596
14.2 The Turbo Decoding Algorithm, Probability Form 601
14.3 BCJR algorithm 629
14.4 Test the turbo decoder algorithm 629
15.1 Iterative Decoding Algorithm for Binary LDPC Codes 648
15.2 Iterative Log Likelihood Decoding Algorithm for Binary LDPC Codes 652
15.3 LDPC decoder class declarations 675
15.4 Matlab code to test LDPC decoding 675
15.5 Make performance plots for LDPC codes 675

List of Figures

1.1 The binary entropy function $H_2(p)$. 4
1.2 A general framework for digital communications. 6
1.3 Signal constellation for BPSK. 10
1.4 Juxtaposition of signal waveforms. 11
1.5 Two-dimensional signal space. 12
1.6 8-PSK signal constellation. 13
1.7 Correlation processing (equivalent to matched filtering). 15
1.8 Conditional densities in BPSK modulation. 18
1.9 Distributions when two signals are sent in Gaussian noise. 20
1.10 Probability of error for BPSK signaling. 21
1.11 Probability of error bound for 8-PSK modulation. 22
1.12 A binary symmetric channel. 23
1.13 Communication system diagram and BSC equivalent. 25
1.14 Energy for a coded signal. 26
1.15 Probability of error for coded bits, before correction. 27
1.16 A (3, 1) binary repetition code. 29
1.17 A representation of decoding spheres. 30
1.18 Performance of the (3, 1) and (11, 1) repetition code over BSC. 32
1.19 Performance of the (7, 4) Hamming code in the AWGN channel. 37
1.20 The trellis of a (7, 4) Hamming code. 39
1.21 The Tanner graph for a (7, 4) Hamming code. 39
1.22 Capacities of AWGNC, BAWGNC, and BSC. 46
1.23 Relationship between input and output entropies for a channel. 48
1.24 Capacity lower bounds on P_b as a function of SNR. 52
1.25 Regions for bounding the Q function. 57

2.1 An illustration of cosets. 67
2.2 A lattice partitioned into cosets. 72

3.1 Error detection performance for a (15,11) Hamming code. 100
3.2 Demonstrating modifications on a Hamming code. 107

4.1 A circuit for multiplying two polynomials, last-element first. 128
4.2 High-speed circuit for multiplying two polynomials, last-element first. 129
4.3 A circuit for multiplying two polynomials, first-element first. 129
4.4 High-speed circuit for multiplying two polynomials, first-element first. 130
4.5 A circuit to perform polynomial division. 131
4.6 A circuit to divide by $g(x) = x^5 + x + 1$. 131
4.7 Realizing $h(x)/g(x)$ (first-element first), controller canonical form. 133
4.8 Realizing $h(x)/g(x)$ (first-element first), observability form. 134
4.9 Realization of $H(x) = (1 + x)/(1 + x^3 + x^4)$, controller form. 134
4.10 Realization of $H(x) = (1 + x)/(1 + x^3 + x^4)$, observability form. 134

4.11 Nonsystematic encoding of cyclic codes. 135
4.12 Circuit for systematic encoding using $g(x)$. 136
4.13 Systematic encoder for (7, 4) code with generator $g(x) = 1 + x + x^3$. 136
4.14 A systematic encoder using the parity check polynomial. 137
4.15 A systematic encoder for the Hamming code using $h(x)$. 137
4.16 A syndrome computation circuit for a cyclic code example. 139
4.17 Cyclic decoder with $r(x)$ shifted in the left end of syndrome register. 140
4.18 Decoder for a (7,4) Hamming code, input on the left. 141
4.19 Cyclic decoder when $r(x)$ is shifted in right end of syndrome register. 143
4.20 Hamming decoder with input fed into right end of the syndrome register. . . . 144
4.21 Meggitt decoders for the (31,26) Hamming code. 145
4.22 Multiply $r(x)$ by $\rho(x)$ and compute the remainder modulo $g(x)$. 146
4.23 Decoder for a shortened Hamming code. 147
4.24 Linear feedback shift register. 154
4.25 Linear feedback shift register with $g(x) = 1 + x + x^2 + x^4$. 155
4.26 Linear feedback shift register with $g(x) = 1 + x + x^4$. 156
4.27 Linear feedback shift register, reciprocal polynomial convention. 158
4.28 Another LFSR circuit. 169
4.29 An LFSR with state labels. 169

5.1 LFSR labeled with powers of α to illustrate Galois field elements. 200
5.2 Multiplication of β by α. 205
5.3 Multiplication of an arbitrary β by α^4. 205
5.4 Multiplication of β by an arbitrary field element. 206
5.5 Subfield structure of $GF(2^{24})$. 207

6.1 Chien search algorithm. 249

7.1 Remainder computation when errors are in message locations. 295
7.2 Comparing BD, ML, and list decoding. 323
7.3 K_m as a function of m for a (32, 8) Reed-Solomon code. 333
7.4 Fraction of errors corrected as a function of rate. 335
7.5 An example of the Roth-Ruckenstein Algorithm over $GF(5)$. 355
7.6 An example of the Roth-Ruckenstein Algorithm over $GF(5)$ (cont'd). 356
7.7 Convergence of $\overline{M}$ to Π. 363
7.8 Computing the reliability function. 364

8.1 An encoder circuit for a $RM(1, 3)$ code. 379
8.2 Signal flow diagram for the fast Hadamard transform. 384
8.3 Binary adjacency relationships in three and four dimensions. 388
8.4 Planes shaded to represent the equations orthogonal on bit m_{34}. 388
8.5 Parity check geometric descriptions for vectors of the $RM(2, 4)$ code. 390
8.6 Parity check geometric descriptions for vectors of the $RM(2, 4)$ code. 391

9.1 Comparison of lower bound and various upper bounds. 407
9.2 A linear programming problem. 413
9.3 Finding Stirling's formula. 422

10.1 A 3×4 interleaver and deinterleaver. 426
10.2 A cross interleaver and deinterleaver system. 428
10.3 The CD recording and data formatting process. 429
10.4 The error correction encoding in the compact disc standard. 429
10.5 The product code $\mathcal{C}_1 \times \mathcal{C}_2$. 431
10.6 A concatenated code. 432
10.7 Deep-space concatenated coding system. 433
10.8 Error trapping decoder for burst-error correcting codes. 436

11.1 Signal labels for soft-decision decoding. 440

12.1 A rate $R = 1/2$ convolutional encoder. 453
12.2 A systematic $R = 1/2$ encoder. 454
12.3 A systematic $R = 2/3$ encoder. 455
12.4 A systematic $R = 2/3$ encoder with more efficient hardware. 455
12.5 Encoder, state diagram, and trellis for $G(x) = [1 + x^2, 1 + x + x^2]$. 458
12.6 State diagram and trellis for a rate $R = 2/3$ systematic encoder. 459
12.7 A feedforward $R = 2/3$ encoder. 460
12.8 A less efficient feedforward $R = 2/3$ encoder. 461
12.9 Processing stages for a convolutional code. 469
12.10 Notation associated with a state transition. 472
12.11 The Viterbi step: Select the path with the best metric. 474
12.12 Path through trellis corresponding to true sequence. 476
12.13 Add-compare-select Operation. 481
12.14 A two-bit quantization of the soft-decision metric. 485
12.15 Quantization thresholds for 4- and 8-level quantization. 487
12.16 Bit error rate for different constraint lengths. 488
12.17 Bit error rate for various quantization and window lengths. 489
12.18 Viterbi algorithm performance as a function of quantizer threshold spacing. . . . 490
12.19 BER performance as a function of truncation block length. 490
12.20 Error events due to merging paths. 491
12.21 Two decoding examples. 492
12.22 The state diagram and graph for diverging/remerging paths. 494
12.23 Rules for simplification of flow graphs. 495
12.24 Steps simplifying the flow graph for a convolutional code. 495
12.25 State diagram labeled with input/output weight and branch length. 496
12.26 A state diagram to be enumerated. 497
12.27 Performance of a $(3, 1)$ convolutional code with $d_{\text{free}} = 5$. 505
12.28 Trellises for a punctured code. 509
12.29 A tree representation for a rate $R = 1/2$ code. 512
12.30 Stack contents for stack algorithm decoding example. 516
12.31 Flowchart for the Fano algorithm. 519
12.32 The trellis of a $(7, 4)$ Hamming code. 524
12.33 A systematic encoder for a $(7, 4, 3)$ Hamming code. 524
12.34 A trellis for a cyclically encoded (7,4,3) Hamming code. 525
12.35 State diagram and trellis 527

13.1 PSK signal constellations. 537

13.2 QAM Signal constellations (overlaid). 537
13.3 Three communication scenarios. 538
13.4 Set partitioning of an 8-PSK signal. 539
13.5 $R = 2/3$ trellis coded modulation example. 540
13.6 A TCM encoder employing subset selection and a four-state trellis. 542
13.7 An 8-state trellis for 8-PSK TCM. 543
13.8 Block diagram of a TCM encoder. 544
13.9 Set partitioning on a 16-QAM constellation. 545
13.10 Partition for 8-ASK signaling. 546
13.11 A correct path and an error path. 548
13.12 Example trellis for four-state code. 549
13.13 Trellis coder circuit. 552
13.14 TCM encoder for QAM constellations. 553
13.15 Mapping of edge (i, j) to edge $(f_\phi(i), f_\phi(j))$. 556
13.16 Encoder circuit for rotationally invariant TCM code. 557
13.17 Trellis for the rotationally invariant code of Figure 13.16 558
13.18 32-cross constellation for rotationally invariant TCM code. 558
13.19 A portion of the lattice $\mathbb{Z}^2$ and its cosets. 564
13.20 Hexagonal lattice. 565
13.21 $\mathbb{Z}^2$ and its partition chain and cosets. 568
13.22 Block diagram for a trellis lattice coder. 569
13.23 Lattice and circular boundaries for various constellations. 570
13.24 16-state trellis encoder for use with V.34 standard. 572
13.25 Trellis diagram of V.34 encoder. 573
13.26 The 192-point constellation employed in the V.34 standard. 575
13.27 Partition steps for the V.34 signal constellation. 576
13.28 Orbits of some of the points under rotation. 576

14.1 Decoding results for a (37,21,65536) code. 583
14.2 Block diagram of a turbo encoder. 585
14.3 Block diagram of a turbo encoder with puncturing. 586
14.4 Example turbo encoder with $G(x) = 1/1 + x^2$. 587
14.5 Block diagram of a turbo decoder. 587
14.6 Processing stages for BCJR algorithm. 589
14.7 One transition of the trellis for the encoder. 590
14.8 A log likelihood turbo decoder. 604
14.9 Trellis with metric differences and bits for SOVA. 612
14.10 State sequences for an encoding example. 616
14.11 Arrangements of $n_1 = 3$ detours in a sequence of length $N = 7$. 616
14.12 A 6×6 "square" sequence written into the 120×120 interleaver. 618
14.13 Variables used in iterative decoder for EXIT chart analysis. 620
14.14 Qualitative form of the transfer characteristic $I_E = T(I_A)$. 622
14.15 Trajectories of mutual information in iterated decoding. 623
14.16 Turbo BCH encoding. 624
14.17 Structure of an implementation of a parallel concatenated code. 624
14.18 A trellis for a cyclically encoded (7,4,3) Hamming code. 625
14.19 Framework for a turbo equalizer. 627

14.20 Trellis associated with a channel with $L = 2$. 628
14.21 Example for a (3, 2, 2) parity check code. 631

15.1 Bipartite graph associated with the parity check matrix A. 638
15.2 A parity check tree associated with the Tanner graph. 640
15.3 A subset of the Tanner graph. 642
15.4 The two-tier tree. 644
15.5 The trellis associated with finding $r_{mn}(x)$. 646
15.6 Processing information through the graph determined by A. 650
15.7 Conditional independence among the sets of bits. 651
15.8 Illustration of the decoding performance of LPDC codes. 654
15.9 Comparison of hard-decision Hamming and iterative decoding. 655
15.10 The function $\Psi(x)$ compared with $\tanh(x/2)$. 657
15.11 Behavior of density evolution for a $R = 1/3$ code. 658
15.12 Messages from bits to checks and from checks to bits. 659
15.13 Histograms of the bit-to-check information for various decoder iterations. . . . 661
15.14 Decoder information at various signal-to-noise ratios. 661
15.15 EXIT charts at various signal-to-noise ratios. 662
15.16 Result of permutation of rows and columns. 670
15.17 Serially concatenated codes and their iterative decoding. 672
15.18 A repeat-accumulate encoder. 672
15.19 The Tanner graph for a (3, 1) RA code with two input bits. 673
15.20 Tanner graph for an irregular repeat-accumulate code. 674

16.1 Factor graph for some examples. 687
16.2 Factor graph for a DFT. 687
16.3 Graphical representations of marginalization. 688
16.4 Conversion from factor graph to expression tree. 689
16.5 Message passing in the sum-product algorithm. 692
16.6 Steps of processing in the sum-product algorithm. 692
16.7 The factor (Tanner) graph for a Hamming code. 694
16.8 Graph portions to illustrate simplifications. 696
16.9 A trellis and a factor graph representation of it. 697
16.10 Message designation for forward/backward algorithm. 698
16.11 The factor graph for a turbo code. 700
16.12 Demonstration of clustering transformations. 701
16.13 Stretching transformations. 702
16.14 Eliminating an edge in a cycle. 703
16.15 Transformations on the DFT factor graph. 705

17.1 Multiple reflections from transmitter to receiver. 711
17.2 Simulation of a fading channel. 713
17.3 Diversity performance of quasi-static, flat-fading channel with BPSK. 714
17.4 Multiple transmit and receive antennas across a fading channel. 715
17.5 Two receive antennas and a maximal ratio combiner receiver. 717
17.6 A two-transmit antenna diversity scheme: the Alamouti code. 719
17.7 A delay diversity scheme. 728
17.8 8-PSK constellation and the trellis for a delay-diversity encoder. 729

17.9 Space-time codes with diversity 2 for 4-PSK having 8, 16, and 32 states. 730
17.10 Space-time codes with diversity 2 for 8-PSK having 16 and 32 states. 731
17.11 Performance of codes with 4-PSK that achieve diversity 2. 732
17.12 Performance of codes with 8-PSK that achieve diversity 2. 732

A.1 Log likelihood ratio. 735

List of Tables

1.1 Historical Milestones 41

3.1 The Standard Array for a Code 92

4.1 Codewords in the Code Generated by $g(x) = 1 + x^2 + x^3 + x^4$ 123
4.2 Computation Steps for Long Division Using a Shift Register Circuit 132
4.3 Computing the Syndrome and Its Cyclic Shifts 139
4.4 Operation of the Meggitt decoder, Input from the Left 142
4.5 Operation of the Meggitt Decoder, Input from the Right 146
4.6 CRC Generators 150
4.7 Lookup Table for CRC-ANSI. Values for t and $R(t)$ are expressed in hex. . . . 152
4.8 LFSR Example with $g(x) = 1 + x + x^2 + x^4$ and Initial State 1. 155
4.9 LFSR Example with $g(x) = 1 + x + x^2 + x^4$ 155
4.10 LFSR Example with $g(x) = 1 + x + x^2 + x^4$ 156
4.11 LFSR example with $g(x) = 1 + x + x^4$ 157
4.12 Barker Codes 170

5.1 Representations of $GF(2^4)$ Using $g(\alpha) = 1 + \alpha + \alpha^4$ 199
5.2 Conjugacy Classes over $GF(2^3)$ with Respect to $GF(2)$ 214
5.3 Conjugacy Classes over $GF(2^4)$ with Respect to $GF(2)$ 214
5.4 Conjugacy Classes over $GF(2^5)$ with Respect to $GF(2)$ 215
5.5 Conjugacy Classes over $GF(4^2)$ with Respect to $GF(4)$ 216
5.6 Cyclotomic Cosets modulo 15 with Respect to $GF(7)$ 218

6.1 Weight Distribution of the Dual of a Double-Error-Correcting Primitive Binary BCH Code of Length $n = 2^m - 1, m \geq 3, m$ Odd 240
6.2 Weight Distribution of the Dual of a Double-Error-Correcting Primitive Binary Narrow-Sense BCH Code, $n = 2^m - 1, m \geq 4, m$ Even 240
6.3 Weight Distribution of the Dual of a Triple-Error Correcting Primitive Binary Narrow-Sense BCH Code, $n = 2^m - 1, m \geq 5, m$ Odd 241
6.4 Weight Distribution of the Dual of a Triple-Error Correcting Primitive Binary Narrow-Sense BCH Code, $n = 2^m - 1, m \geq 6, m$ Even 241
6.5 Berlekamp-Massey algorithm for input sequence $\{1, 1, 1, 0, 1, 0, 0\}$ 259
6.6 Berlekamp-Massey Algorithm for a Double-Error Correcting Code 259
6.7 Simplified Berlekamp-Massey Algorithm for a double-error correcting code . . 260
6.8 Berlekamp-Massey Algorithm for a Triple-Error Correcting Code 265

7.1 Monomials Ordered Under $(1, 3)$-revlex Order 326
7.2 Monomials Ordered Under $(1, 3)$-lex Order 326

8.1 Extended Quadratic Residue Codes $\mathcal{Q}$ 398

8.2 Weight Distributions for the $\mathcal{G}_{23}$ and $\mathcal{G}_{24}$ Codes 400

10.1 Performance Specification of the Compact Disc Coding System 430

12.1 Quantized Branch Metrics Using Linear Quantization 487
12.2 Comparison of Free Distance for Systematic and Nonsystematic Code 508
12.3 Best Known Convolutional Codes Obtained by Puncturing a $R = 1/2$ Code . . 510
12.4 Performance of Fano Algorithm as a Function of Δ 520

13.1 Average Energy Requirements for Some QAM Constellations 536
13.2 Maximum Free-Distance Trellis codes for 8-PSK Constellation 554
13.3 Maximum Free-Distance Trellis Codes for 16-PSK Constellation 555
13.4 Maximum Free-Distance Trellis Codes for AM Constellations 555
13.5 Encoder Connections and Coding Gains for QAM Trellis Codes 555
13.6 Attributes of Some Lattices 565
13.7 Comparison of Average Signal Energy 569
13.8 Some Good Multidimensional TCM Codes [303] 571
13.9 Bit Converter: Sublattice Partition of 4D Rectangular Lattice 574
13.10 4D Block Encoder 574

14.1 α_t' and β_t' Example Computations 597
14.2 Posterior Input Bit Example Computations 597

15.1 Threshold Values for Various LDPC Codes 659
15.2 Degree Distribution Pairs for $R = 1/2$ Codes for Transmission on an AWGN . 664

16.1 Some Commutative Semirings 681
16.2 Some Special Cases of Message Passing 691

List of Boxes

Box 1.1 The Union Bound 22
Box 2.1 One-to-One and Onto Functions 70
Box 3.1 Error Correction and Least-Squares 90
Box 3.2 The UDP Protocol 105
Box 4.1 The Division Algorithm 114
Box 5.1 Èveriste Galois (1811–1832) 197
Box 9.1 O and o Notation 408
Box 9.2 The Cauchy-Schwartz Inequality 411
Box 12.1 Graphs: Basic Definitions 457

This Page Intentionally Left Blank

Part I

Introduction and Foundations

Chapter 1

A Context for Error Correction Coding

I will make weak things become strong unto them ... — Ether 12:27

... he denies that any error in the machine is responsible for the so-called errors in the answers. He claims that the Machines are self correcting and that it would violate the fundamental laws of nature for an error to exist in the circuits of relays.

— Isaac Asimov
I, Robot

1.1 Purpose of This Book

Error control coding in the context of digital communication has a history dating back to the middle of the twentieth century. In recent years, the field has been revolutionized by codes which are capable of approaching the theoretical limits of performance, the *channel capacity*. This has been impelled by a trend away from purely combinatoric and discrete approaches to coding theory toward codes which are more closely tied to a physical channel and soft decoding techniques.

The purpose of this book is to present error correction/detection coding in a modern setting, covering both traditional concepts thoroughly as well as modern developments in soft-decision and iteratively decoded codes and recent decoding algorithms for algebraic codes. An attempt has been made to maintain some degree of balance between the mathematics and their engineering implications by presenting both the mathematical methods used in understanding the codes as well as the algorithms which are used to efficiently encode and decode.

1.2 Introduction: Where Are Codes?

Error correction coding is the means whereby errors which may be introduced into digital data as a result of transmission through a communication channel can be corrected based upon received data. *Error detection coding* is the means whereby errors can be detected based upon received information. Collectively, error correction and error detection coding are *error control coding*. Error control coding can provide the difference between an operating communications system and a dysfunctional system. It has been a significant enabler in the telecommunications revolution, the internet, digital recording, and space exploration. Error control coding is nearly ubiquitous in modern, information-based society. Every compact disc, CD-ROM, or DVD employs codes to protect the data embedded in the plastic disk. Every hard disk drive employs correction coding. Every phone call made over a digital cellular phone employs it. Every packet transmitted over the internet has a protective coding "wrapper" used to determine if the packet has been received correctly. Even

everyday commerce takes advantage of error detection coding, as the following examples illustrate.

Example 1.1 The ISBN (international standard book number) is used to uniquely identify books. An ISBN such as 0-201-36186-8 can be parsed as

$$\underbrace{0}_{\text{country}} - \underbrace{20}_{\text{publisher}} - \underbrace{1 - 36186}_{\text{book no.}} - \underbrace{8}_{\text{check}}.$$

Hyphens do not matter. The first digit indicates a country/language, with 0 for the United States. The next two digits are a publisher code. The next six digits are a publisher-assigned book number. The last digit is a check digit, used to validate if the code is correct using what is known as a weighted code.

An ISBN is checked as follows: The cumulative sum of the digits is computed, then the cumulative sum of the cumulative sum is computed. For a valid ISBN, the sum-of-the-sum must be equal to 0, modulo 11. The character X is used for the check digit 10. For this ISBN, we have

digit	cumulative sum	cumulative sum
0	0	0
2	2	2
0	2	4
1	3	7
3	6	13
6	12	25
1	13	38
8	21	59
6	27	86
8	35	121

The final sum-of-the-sum is 121, which is equal to 0 modulo 11 (i.e., the remainder after dividing by 11 is 0). □

Example 1.2 The Universal Product Codes (UPC) employed on the bar codes of most merchandise employ a simple error detection system to help ensure reliability in scanning. In this case, the error detection system consists of a simple parity check. A UPC consists of a 12-digit sequence, which can be parsed as

$$\underbrace{0\ 16000}_{\substack{\text{manufacturer}\\ \text{identification}\\ \text{number}}} \quad \underbrace{66610}_{\substack{\text{item}\\ \text{number}}} \quad \underbrace{8}_{\substack{\text{parity}\\ \text{check}}}.$$

Denoting the digits as $u_1, u_2, \ldots, u_{12}$, the parity digit u_{12} is determined such that

$$3(u_1 + u_3 + u_5 + u_7 + u_9 + u_{11}) + (u_2 + u_4 + u_6 + u_8 + u_{10} + u_{12})$$

is a multiple of 10. In this case,

$$3(0 + 6 + 0 + 6 + 6 + 0) + (1 + 0 + 0 + 6 + 1 + 8) = 70.$$

If, when a product is scanned, the parity condition does not work, the operator is flagged so that the object may be re-scanned. □

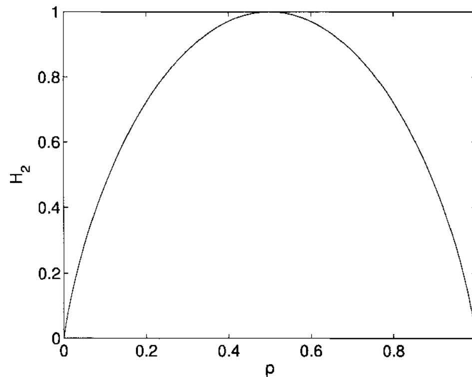


Figure 1.1: The binary entropy function $H_2(p)$.

1.3 The Communications System

Appreciation of the contributions of coding and an understanding of its limitations require some awareness of information theory and how its major theorems delimit the performance of a digital communication system. In fact, information theory is increasingly relevant to coding theory, because with recent advances in coding theory it is now possible to achieve the performance bounds of information theory, whereas in the past the bounds were more of a backdrop to the action on the stage of coding research and practice. Part of this success has come by placing the coding problem more fully in its communications context, marrying the coding problem more closely to the signal detection problem, instead of treating the coding problem mostly as one of discrete combinatorics.

Information theory treats *information* almost as a physical quantity which can be measured, transformed, stored, and moved from place to place. A fundamental concept of information theory is that information is conveyed by the resolution of uncertainty. Information can be measured by the amount of uncertainty resolved. For example, if a digital source always emits the same value, say 1, then no information is gained by observing that the source has produced, yet again, the output 1. Probabilities are used to mathematically describe the uncertainty. For a discrete random variable X (i.e., one which produces discrete outputs, such as $X = 0$ or $X = 1$), the *information* conveyed by observing an outcome x is $-\log_2 P(X = x)$ bits. (If the logarithm is base 2, the units of information are in **bits**. If the natural logarithm is employed, the units of information are in **nats**.) For example, if $P(X = 1) = 1$ (the outcome 1 is certain), then observing $X = 1$ yields $-\log_2(1) = 0$ bits of information. On the other hand, observing $X = 0$ in this case yields $-\log_2(0) = \infty$: total surprise at observing an impossible outcome.

The *entropy* is the *average information.* For a binary source X having two outcomes occurring with probabilities p and $1 - p$, the binary entropy function, denoted as either $H_2(X)$ (indicating that it is the entropy of the source) or $H_2(p)$ (indicating that it is a function of the outcome probabilities) is

$$H_2(X) = H_2(p) = E[-\log_2 P(X)] = -p\log_2(p) - (1 - p)\log_2(1 - p) \text{ bits.}$$

A plot of the binary entropy function as a function of p is shown in Figure 1.1. The peak information of 1 occurs when $p = \frac{1}{2}$.

Example 1.3 A fair coin is tossed once per second, with the outcomes being 'head' and 'tail' with equal probability. Each toss of the coin generates an event that can be described with $H_2(0.5) = 1$ bit of information. The sequence of tosses produces information at a rate of 1 bit per second.

An unfair coin, with $P(\text{head}) = 0.01$ is tossed. The average information generated by each throw in this case is $H_2(0.01) = 0.0808$ bits.

Another unfair coin, with $P(\text{head}) = 1$ is tossed. The information generated by each throw in this case is $H_2(1) = 0$ bits. □

For a source X having M outcomes $x_1, x_2, \ldots, x_M$, with probabilities $P(X = x_i) = p_i, i = 1, 2, \ldots, M$, the entropy is

$$H(X) = E[-\log_2 P(X)] = -\sum_{i=1}^{M} p_i \log_2 p_i \text{ bits}. \tag{1.1}$$

Note: The "bit" as a measure of entropy (or information content) is different from the "bit" as a measure of storage. For the unfair coin having $P(\text{head}) = 1$, the actual information content determined by a toss of the coin is 0: there is no information gained by observing that the outcome is again 1. For this process with this unfair coin, the entropy rate — that is, the amount of actual information it generates — is 0. However, if the coin outcomes were for some reason to be stored directly, without the benefit of some kind of coding, each outcome would require 1 bit of storage (even though they don't represent any new information).

With the prevalence of computers in our society, we are accustomed to thinking in terms of "bits" — e.g., a file is so many bits long, the register of a computer is so many bits wide. But these are "bits" as a measure of storage size, not "bits" as a measure of actual information content. Because of the confusion between "bit" as a unit of information content and "bit" as an amount of storage, the unit of information content is sometimes called a *Shannon*, in homage to the founder of information theory, Claude Shannon.[1]

A digital communication system embodies functionality to perform physical actions on information. Figure 1.2 illustrates a fairly general framework for a single digital communication link. In this link, digital data from a *source* are encoded and modulated (and possibly encrypted) for communication over a *channel*. At the other end of the channel, the data are demodulated, decoded (and possibly decrypted), and sent to a *sink*. The elements in this link all have mathematical descriptions and theorems from information theory which govern their performance. The diagram indicates the realm of applicability of three major theorems of information theory.

There are actually many kinds of codes employed in a communication system. In the description below we point out where some of these codes arise. Throughout the book we make some connections between these codes and our major focus of study, error correction codes.

The source is the data to be communicated, such as a computer file, a video sequence, or a telephone conversation. For our purposes, it is represented in digital form, perhaps as a result of an analog-to-digital conversion step. Information-theoretically, sources are viewed as streams of random numbers governed by some probability distribution.

[1]This mismatch of object and value is analogous to the physical horse, which may or may not be capable of producing one "horsepower" of power, 550 ft-lbs/sec. Thermodynamicists can dodge the issue by using the SI unit of Watts for power, information theorists might sidestep confusion by using the Shannon. Both of these units honor founders of their respective disciplines.

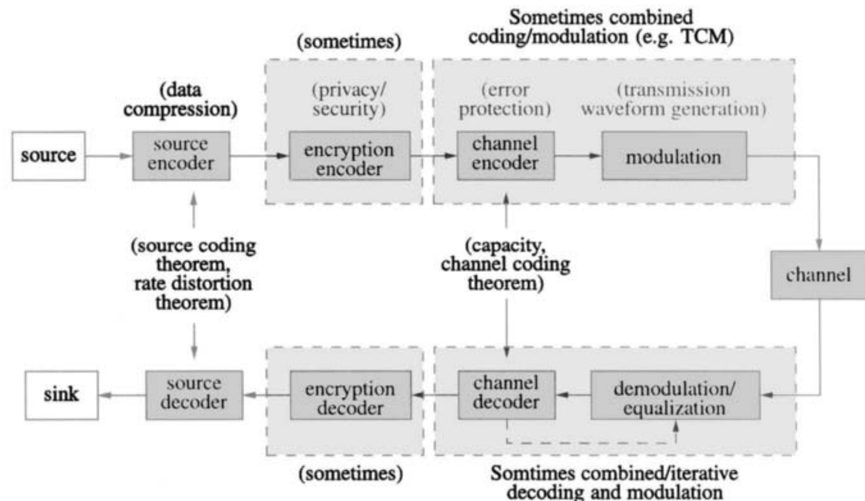


Figure 1.2: A general framework for digital communications.

Every source of data has a measure of the information that it represents, which (in principle) can be exactly quantified in terms of entropy.

The source encoder performs data compression by removing redundancy.

As illustrated in Example 1.3, the number of bits used to store the information from a source may exceed the number of bits of actual information content. That is, the number of bits to represent the data may exceed the number of mathematical bits — Shannons — of actual information content.

The amount a particular source of data can be compressed without any loss of information (*lossless* compression) is governed theoretically by the *source coding theorem* of information theory, which states that a source of information can be represented without any loss of information in such a way that the amount of storage required (in bits) is equal to the amount of information content — the entropy — in bits or Shannons. To achieve this lower bound, it may be necessary for long blocks of the data to be jointly encoded.

Example 1.4 For the unfair coin with $P(\text{head}) = 0.01$, the entropy is $H(0.01) = 0.0808$. Therefore, 10,000 such (independent) tosses convey 808 bits (Shannons) of information, so theoretically the information of 10,000 tosses of the coin can be represented *exactly* using only 808 (physical) bits of information. □

Thus a bit (in a computer register) in principle *can* represent an actual (mathematical) bit of information content, if the source of information is represented correctly.

In compressing a data stream, a source encoder removes redundancy present in the data. For compressed binary data, 0 and 1 occur with equal probability in the compressed data (otherwise, there would be some redundancy which could be exploited to further compress the data). Thus it is frequently assumed at the channel coder that 0 and 1 occur with equal probability.

The source encoder employs special types of codes to do the data compression, called collectively source codes or data compression codes. Such coding techniques include Huffman coding, run-length coding, arithmetic coding, Lempel-Ziv coding, and combinations of these, all of which fall beyond the scope of this book.

If the data need to be compressed below the entropy rate of the source, then some kind of distortion must occur. This is called *lossy* data compression. In this case, another theorem governs the representation of the data. It is possible to do lossy compression in a way that minimizes the amount of distortion for a given rate of transmission. The theoretical limits of lossy data compression are established by the *rate-distortion theorem* of information theory. One interesting result of rate-distortion theory says that for a binary source having equiprobable outcomes, the minimal rate to which the data can be compressed with the average distortion per bit equal to p is

$$r = 1 - H_2(p) \qquad p \leq \frac{1}{2}. \tag{1.2}$$

Lossy data compression uses its own kinds of codes as well.

The encrypter hides or scrambles information so that unintended listeners are unable to discern the information content. The codes used for encryption are generally different from the codes used for error correction.

Encryption is often what the layperson frequently thinks of when they think of "coding," but as we are seeing, there are many other different kinds of codes.

The channel coder is the first step in the error correction or error detection process.

The channel coder adds redundant information to the stream of input symbols in a way that allows errors which are introduced into the channel to be corrected. **This book is essentially dedicated to the study of the channel coder and its corresponding channel decoder.**

It may seem peculiar to remove redundancy with the source encoder, then turn right around and add redundancy back in with the channel encoder. However, the redundancy in the source typically depends on the source in an unstructured way and may not provide uniform protection to all the information in the stream, nor provide any indication of how errors occurred or how to correct them. The redundancy provided by the channel coder, on the other hand, is introduced in a structured way, precisely to provide error control capability.

Treating the problems of data compression and error correction separately, rather than seeking a jointly optimal source/channel coding solution, is asymptotically optimal (as the block sizes get large). This fact is called the *source-channel separation theorem* of information theory. (There has been recent work on combined source/channel coding for finite — practical — block lengths, in which the asymptotic theorems are not invoked. This work falls outside the scope of this book.)

Because of the redundancy introduced by the channel coder, there must be more symbols at the output of the coder than at the input. Frequently, a channel coder operates by accepting a block of k input symbols and producing at its output a block of n symbols, with $n > k$. The **rate** of such a channel coder is

$$R = k/n,$$

so that $R < 1$.

The input to the channel coder is referred to as the *message symbols* (or, in the case of binary codes, the *message bits*). The input may also be referred to as the *information symbols* (or bits).

The modulator converts symbol sequences from the channel encoders into signals appropriate for transmission over the channel. Many channels require that the signals be sent as a continuous-time voltage, or an electromagnetic waveform in a specified frequency band. The modulator provides the appropriate channel-conforming representation.

Included within the modulator block one may find codes as well. Some channels (such as magnetic recording channels) have constraints on the maximum permissible length of runs of 1s. Or they might have a restriction that the sequence must be DC-free. Enforcing such constraints employs special codes. Treatment of such runlength-limited codes appears in [206]; see also [157].

Some modulators employ mechanisms to ensure that the signal occupies a broad bandwidth. This *spread-spectrum* modulation can serve to provide multiple-user access, greater resilience to jamming, low-probability of detection, and other advantages. (See, e.g., [386].) Spread-spectrum systems frequently make use of pseudorandom sequences, some of which are produced using linear feedback shift registers as discussed in Section Appendix 4.A.

The channel is the medium over which the information is conveyed. Examples of channels are telephone lines, internet cables, fiber-optic lines, microwave radio channels, high frequency channels, cell phone channels, etc. These are channels in which information is conveyed between two distinct places. Information may also be conveyed between two distinct times, for example, by writing information onto a computer disk, then retrieving it at a later time. Hard drives, diskettes, CD-ROMs, DVDs, and solid state memory are other examples of channels.

As signals travel through a channel they are corrupted. For example, a signal may have noise added to it; it may experience time delay or timing jitter, or suffer from attenuation due to propagation distance and/or carrier offset; it may be multiply reflected by objects in its path, resulting in constructive and/or destructive interference patterns; it may experience inadvertent interference from other channels, or be deliberately jammed. It may be filtered by the channel response, resulting in interference among symbols. These sources of corruption in many cases can all occur simultaneously.

For purposes of analysis, channels are frequently characterized by mathematical models, which (it is hoped) are sufficiently accurate to be representative of the attributes of the actual channel, yet are also sufficiently abstracted to yield tractable mathematics. Most of our work in this book will assume one of two idealized channel models, the binary symmetric channel (BSC) and the additive white Gaussian noise channel (AWGN), which are described in Section 1.5. While these idealized models do not represent all of the possible problems a signal may experience, they form a starting point for many, if not most, of the more comprehensive channel models. The experience gained by studying these simpler channels models forms a foundation for more accurate and complicated channel models. (As exceptions to the AWGN or BSC rule, in Section 14.7, we comment briefly on convolutive channels and turbo equal-

ization, while in Chapter 17, coding for quasi-static Rayleigh flat fading channels are discussed.)

Channels have different information-carrying capabilities. For example, a dedicated fiber-optic line is capable of carrying more information than a plain-old-telephone-service (POTS) pair of copper wires. Associated with each channel is a quantity known as the **capacity**, C, which indicates how much information it can carry **reliably**.

The reliable information a channel can carry is intimately related to the use of error correction coding. The governing theorem from information theory is Shannon's **channel coding theorem**, which states essentially this: Provided that the rate R of transmission is less than the capacity C, *there exists* a code such that the probability of error can be made arbitrarily small.

As suggested by Figure 1.2, the channel encoding and modulation may be combined in what is known as *coded modulation*.

The demodulator/equalizer receives the signal from the channel and converts it into a sequence of symbols. This typically involves many functions, such as filtering, demodulation, carrier synchronization, symbol timing estimation, frame synchronization, and matched filtering, followed by a detection step in which decisions about the transmitted symbols are made. We will not concern ourselves in this book with these important details, but will focus on issues related to channel encoding and decoding.

The channel decoder exploits the redundancy introduced by the channel encoder to correct any errors that may have been introduced. As suggested by the figure, demodulation, equalization and decoding may be combined. Particularly in recent work, turbo equalizers are used in a powerful combination. This is introduced in Chapter 14.

The decrypter removes any encryption.

The source decoder provides an uncompressed representation of the data.

The sink is the ultimate destination of the data.

As this summary description has indicated, there are many different kinds of codes employed in communications. This book treats only error correction (or detection) codes. However, there is significant overlap in the mathematical tools employed for error correction codes and other kinds of codes. So, for example, while this is not a book on encryption, a couple of encryption codes are presented in this book, where they are right near our main topic. (In fact, one public key cryptosystem *is* an error correction coding scheme. See Section 6.9.5.) And there is a certain duality between some channel coding methods and some source coding methods. So studying error correction does provide a foundation for other aspects of the communication system.

1.4 Basic Digital Communications

The study of modulation/channel/demodulation/detection falls in the realm of "digital communications," and many of its issues (e.g., filtering, synchronization, carrier tracking) lie beyond the scope of this book. Nevertheless, some understanding of digital communications is necessary here, because modern coding theory has achieved some of its successes by careful application of detection theory, in particular in maximum *a posteriori* (MAP) and maximum likelihood (ML) receivers. Furthermore, performance of codes is often plotted

in terms of signal to noise ratio, which is understood in the context of the modulation of a physical waveform. Coded modulation relies on signal constellations beyond simple binary modulation, so an understanding of them is important.

The material in this section is standard for a digital communications course. However, it is germane to our treatment here, because these concepts are employed in the development of soft-decision decoding algorithms.

1.4.1 Binary Phase-Shift Keying

In digital communication, a stream of bits (i.e., a sequence of 1s and 0s) is mapped to a waveform for transmission over the channel. Binary phase-shift keying (BPSK) is a form of amplitude modulation in which a bit is represented by the sign of the transmitted waveform. (It is called "phase-shift" keying because the sign change represents a 180° phase shift.) Let $\{\ldots, b_{-2}, b_{-1}, b_0, b_1, b_2, \ldots\}$ represent a sequence of bits, $b_i \in \{0, 1\}$ which arrive at a rate of one bit every T seconds. The bits are assumed to be randomly generated with probabilities $P_1 = P(b_i = 1)$ and $P_0 = P(b_i = 0)$. While typically 0 and 1 are equally likely, we will initially retain a measure of generality and assume that $P_1 \neq P_0$ necessarily. It will frequently be of interest to map the set $\{0, 1\}$ to the set $\{-1, 1\}$. We will denote $\tilde{b}_i$ as the ± 1-valued bit corresponding to the $\{0, 1\}$-valued bit b_i. Either of the mappings

$$\tilde{b}_i = (2b_i - 1) \qquad \text{or} \qquad \tilde{b}_i = -(2b_i - 1)$$

may be used in practice, so some care is needed to make sure the proper mapping is understood.

Here, let $a_i = \sqrt{E_b}(2b_i - 1) = -\sqrt{E_b}(-1)^{b_i} = \sqrt{E_b}\tilde{b}_i$ be a mapping of bit b_i (or $\tilde{b}_i$) into a transmitted signal amplitude. This signal amplitude multiplies a waveform $\varphi_1(t)$, where $\phi_1(t)$ is a unit-energy signal,

$$\int_{-\infty}^{\infty} \varphi_1(t)^2\, dt = 1,$$

which has support[2] over $[0, T)$. Thus, a bit b_i arriving at time iT can be represented by the signal $a_i\varphi_1(t - iT)$. The energy required to transmit a single bit b_i is

$$\int_{-\infty}^{\infty} (a_i\varphi_1(t))^2\, dt = E_b.$$

Thus E_b is thus the energy expended per bit.

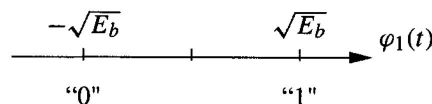


Figure 1.3: Signal constellation for BPSK.

It is helpful to think of the transmitted signals $\sqrt{E_b}\varphi_1(t)$ and $-\sqrt{E_b}\varphi_1(t)$ as points $\sqrt{E_b}$ and $-\sqrt{E_b}$ in a one-dimensional **signal space**, where the coordinate axis is the "φ_1" axis. The two points in the signal space are plotted with their corresponding bit assignment in Figure 1.3. The points in the signal space employed by the modulator are called the **signal constellation**, so Figure 1.3 is a signal constellation with two points (or signals).

A sequence of bits to be transmitted can be represented by a juxtaposition of $\varphi_1(t)$ waveforms, where the waveform representing bit b_i starts at time iT. Then the sequence of

[2]Strictly speaking, functions not having this limited support can be used, but assuming support over $[0, T)$ makes the discussion significantly simpler. Also, the signal $\varphi_i(t)$ can in general be complex, but we restrict attention here to real signals.

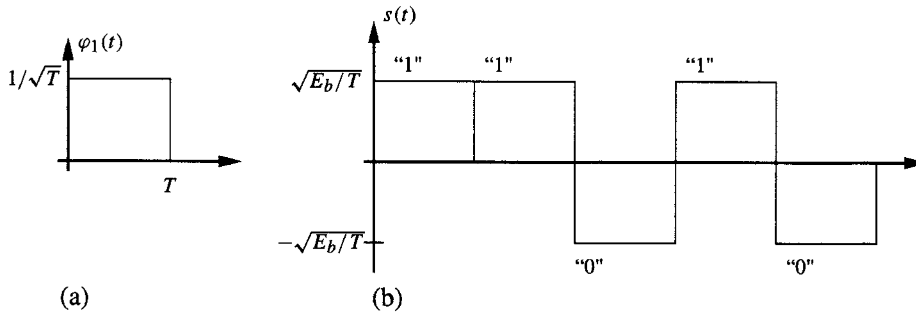


Figure 1.4: Juxtaposition of signal waveforms.

bits is represented by the signal

$$s(t) = \sum_i a_i \varphi_1(t - iT). \tag{1.3}$$

Example 1.5 With $\varphi_1(t)$ as shown in Figure 1.4(a) and the bit sequence $\{1, 1, 0, 1, 0\}$, the signal $s(t)$ is as shown in Figure 1.4(b). □

1.4.2 More General Digital Modulation

The concept of signal spaces generalizes immediately to higher dimensions and to larger signal constellations; we restrict our attention here to no more than two dimensions. Let $\varphi_2(t)$ be a unit-energy function which is orthogonal to $\varphi_1(t)$. That is,

$$\int_{-\infty}^{\infty} \varphi_2(t)^2\, dt = 1 \qquad \text{and} \qquad \int_{-\infty}^{\infty} \varphi_1(t)\varphi_2(t)\, dt = 0.$$

In this case, we are defining "orthogonality" with respect to the inner product

$$\langle \varphi_1(t), \varphi_2(t) \rangle = \int_{-\infty}^{\infty} \varphi_1(t)\varphi_2(t)\, dt.$$

We say that $\{\varphi_1(t), \varphi_2(t)\}$ form an orthonormal set if they both have unit energy and are orthogonal:

$$\langle \varphi_1(t), \varphi_1(t) \rangle = 1 \qquad \langle \varphi_2(t), \varphi_2(t) \rangle = 1 \qquad \langle \varphi_1(t), \varphi_2(t) \rangle = 0.$$

The orthonormal functions $\varphi_1(t)$ and $\varphi_2(t)$ define the coordinate axes of a two-dimensional signal space, as suggested by Figure 1.5. Corresponding to every point (x_1, y_1) of this two-dimensional signal space is a signal (i.e., a function of time) $s(t)$ obtained as a linear combination of the coordinate functions:

$$s(t) = x_1\varphi_1(t) + y_1\varphi_2(t).$$

That is, there is a one-to-one correspondence between "points" in space and their represented signals. We can represent this as

$$s(t) \leftrightarrow (x_1, y_1).$$

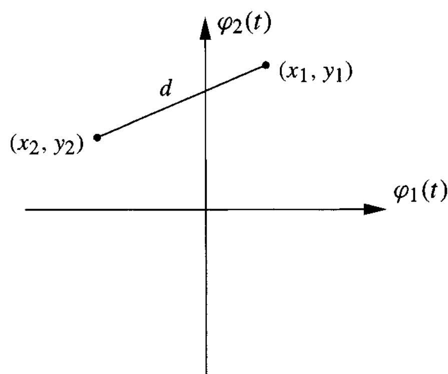


Figure 1.5: Two-dimensional signal space.

The geometric concepts of distance and angle can be expressed in terms of the signal space points. For example, let

$$\begin{aligned} s_1(t) &= x_1\varphi_1(t) + y_1\varphi_2(t) \qquad (\text{i.e., } s_1(t) \leftrightarrow (x_1, y_1)) \\ s_2(t) &= x_2\varphi_1(t) + y_2\varphi_2(t) \qquad (\text{i.e., } s_2(t) \leftrightarrow (x_2, y_2)) \end{aligned} \tag{1.4}$$

We define the squared distance between $s_1(t)$ and $s_2(t)$ as

$$d^2(s_1(t), s_2(t)) = \int_{-\infty}^{\infty} (s_1(t) - s_2(t))^2\, dt, \tag{1.5}$$

and the inner product between $s_1(t)$ and $s_2(t)$ as

$$\langle s_1(t), s_2(t)\rangle = \int_{-\infty}^{\infty} s_1(t)s_2(t)\, dt. \tag{1.6}$$

Rather than computing distance using the integral (1.5), we can equivalently and more easily compute using the coordinates in signal space (see Figure 1.5):

$$d^2(s_1(t), s_2(t)) = (x_1 - x_2)^2 + (y_1 - y_2)^2. \tag{1.7}$$

This is the familiar squared **Euclidean distance** between the points (x_1, y_1) and (x_2, y_2). Also, rather than computing the inner product using the integral (1.6), we equivalently compute using the coordinates in signal space:

$$\langle s_1(t), s_2(t)\rangle = x_1x_2 + y_1y_2. \tag{1.8}$$

This is the familiar inner product (or dot product) between the points (x_1, y_1) and (x_2, y_2):

$$\langle (x_1, y_1), (x_2, y_2)\rangle = x_1x_2 + y_1y_2.$$

The point here is that we can use the signal space geometry to gain insight into the nature of the signals, using familiar concepts of distance and angle.

We can use this two-dimensional signal space for digital information transmission as follows. Let $M = 2^m$, for some integer m, be the number of points in the signal constellation. M-ary transmission is obtained by placing M points (a_{1k}, a_{2k}), $k = 0, 1, \ldots, M-1$, in this

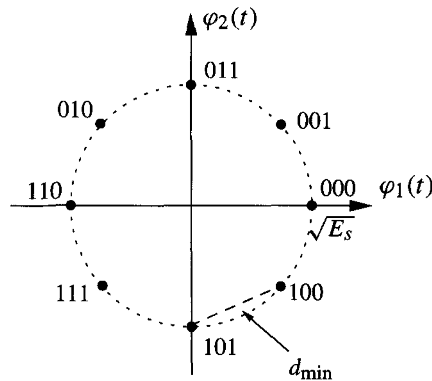


Figure 1.6: 8-PSK signal constellation.

signal space and assigning a unique pattern of m bits to each of these points. These points are the signal constellation. Let

$$\mathcal{S} = \{(a_{1k}, a_{2k}), k = 0, 1, \ldots, M-1\}$$

denote the set of points in the signal constellation.

Example 1.6 Figure 1.6 shows 8 points arranged in two-dimensional space in a constellation known as 8-PSK (phase-shift keying). Each point has a three-bit designation. The signal corresponding to the point (a_{1k}, a_{2k}) is selected by three input bits and transmitted. Thus the signal

$$s_k(t) = a_{1k}\varphi_1(t) + a_{2k}\varphi_2(t), \qquad (a_{1k}, a_{2k}) \in \mathcal{S}$$

carries three bits of information.

Note that the assignments of bits to constellation points in Figure 1.6 is such that adjacent points differ by only one bit. Such an assignment is called Gray code order. Since it is most probable that errors will move from a point to an adjacent point, this reduces the probability of *bit* error. □

Associated with each signal $s_k(t) = a_{1k}\varphi_1(t) + a_{2k}\varphi_2(t)$ and signal constellation point $(a_{1k}, a_{2k}) \in \mathcal{S}$ is a signal energy,

$$E_k = \int_{-\infty}^{\infty} (s_k(t))^2\, dt = a_{1k}^2 + a_{2k}^2.$$

The *average signal energy* E_s is obtained by averaging all the signal energies, usually by assuming that each signal point is used with equal probability:

$$E_s = \frac{1}{M}\sum_{k=0}^{M-1}\int_{-\infty}^{\infty} s_k(t)^2\, dt = \frac{1}{M}\sum_{k=0}^{M-1}(a_{1k}^2 + a_{2k}^2).$$

The average energy per signal E_s can be related to the average energy per bit E_b by

$$E_b = \frac{\text{energy per signal}}{\text{number of bits/signal}} = \frac{E_s}{m}.$$

To send a sequence of bits using M-ary modulation, the bits are partitioned into blocks of m successive bits, where the data rate is such that m bits arrive every T_s seconds. The ith

m-bit set is then used to index a point $(a_{1i}, a_{2i}) \in \mathcal{S}$. This point corresponds to the signal which is transmitted over the signal interval for the m bits. These signals are juxtaposed to form the transmitted signal:

$$s(t) = \sum_i a_{1i}\varphi_1(t - iT_s) + a_{2i}\varphi_2(t - iT_s), \qquad (a_{1i}, a_{2i}) \in \mathcal{S}. \tag{1.9}$$

The point (a_{1i}, a_{2i}) is thus the point set at time i. Equation (1.9) can be expressed in its signal space vector equivalent, by simply letting $\mathbf{s}_i = [a_{1i}, a_{2i}]^T$ denote the vector transmitted at time i.

In what follows, we will express the operations in terms of the two-dimensional signal space. Restricting to a one-dimensional signal space (as for BPSK transmission), or extending to higher-dimensional signal spaces is straightforward.

In most channels, the signal $s(t)$ is mixed with some carrier frequency before transmission. However, for simplicity we will restrict attention to the baseband transmission case.

1.5 Signal Detection

1.5.1 The Gaussian Channel

The signal $s(t)$ is transmitted over the channel. Of all the possible disturbances that might be introduced by the channel, we will deal only with additive noise, resulting in the received signal

$$r(t) = s(t) + n(t). \tag{1.10}$$

In an additive white Gaussian noise (AWGN) channel, the signal $n(t)$ is a white Gaussian noise process, having the properties that

$$E[n(t)] = 0 \qquad \forall t,$$

$$R_n(\tau) = E[n(t)n(t-\tau)] = \frac{N_0}{2}\delta(\tau),$$

and all sets of samples are jointly Gaussian distributed. The quantity $N_0/2$ is the (two-sided) noise power spectral density.

Due to the added noise, the signal $r(t)$ is typically not a point in the signal constellation, nor, in fact, is $r(t)$ probably even *in* the signal space — it cannot be expressed as a linear combination of the basis functions $\varphi_1(t)$ and $\varphi_2(t)$. The detection process to be described below corresponds to the geometric operations of (1) projecting $r(t)$ onto the signal space; and (2) finding the closest point in the signal space to this projected function.

At the receiver, optimal detection requires first passing the received signal through a filter "matched" to the transmitted waveform. This is the projection operation, projecting $r(t)$ onto the signal space. To detect the ith signal starting at iT_s, the received signal is correlated with the waveforms $\varphi_1(t - iT_s)$ and $\varphi_2(t - iT_s)$ to produce the point (R_{1i}, R_{2i})

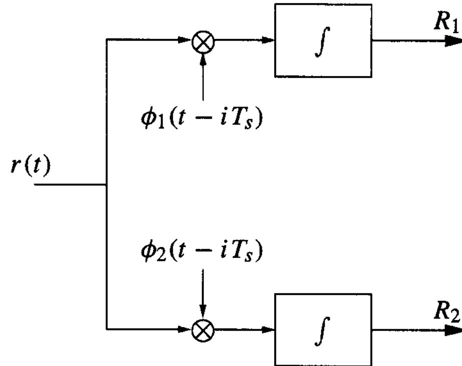


Figure 1.7: Correlation processing (equivalent to matched filtering).

in signal space[3]:

$$\begin{aligned} R_{1i} &= \langle r(t), \varphi_1(t - iT_s)\rangle = \int_{iT_s}^{(i+1)T_s} r(t)\varphi_1(t - iT_s)\, dt, \\ R_{2i} &= \langle r(t), \varphi_2(t - iT_s)\rangle = \int_{iT_s}^{(i+1)T_s} r(t)\varphi_2(t - iT_s)\, dt. \end{aligned} \tag{1.11}$$

The processing in (1.11) is illustrated in Figure 1.7. Using (1.9) and (1.10), it is straightforward to show that

$$R_{1i} = a_{1i} + N_{1i} \quad \text{and} \quad R_{2i} = a_{2i} + N_{2i}, \tag{1.12}$$

where (a_{1i}, a_{2i}) is the transmitted point in the signal constellation for the ith symbol. The point (a_{1i}, a_{2i}) is not known at the receiver — it is, in fact, what the receiver needs to decide — so at the receiver (a_{1i}, a_{2i}) is a random variable.

The noise random variables N_{1i} and N_{2i} defined by

$$N_{1i} = \int_{iT_s}^{(i+1)T_s} \varphi_1(t - iT_s)n(t)\, dt \quad \text{and} \quad N_{2i} = \int_{iT_s}^{(i+1)T_s} \varphi_2(t - iT_s)n(t)\, dt$$

have the following properties: N_{1i} and N_{2i} are Gaussian random variables, with

$$E[N_{1i}] = 0 \quad \text{and} \quad E[N_{2i}] = 0 \tag{1.13}$$

and[4]

$$\text{var}[N_{1i}] \triangleq \sigma^2 = \frac{N_0}{2} \quad \text{and} \quad \text{var}[N_{2i}] \triangleq \sigma^2 = \frac{N_0}{2}. \tag{1.14}$$

Also,

$$E[N_{1i}N_{2i}] = 0; \tag{1.15}$$

[3]The operation in (1.11) can equivalently be performed by passing $r(t)$ through filters with impulse response $\varphi_1(-t)$ and $\varphi_2(-t)$ and sampling the output at time $t = iT_s$. This is referred to as a *matched filter*. The matched filter implementation and the correlator implementation provide identical outputs.

[4]The symbol $\triangleq$ means "is defined to be equal to."

that is, N_{1i} and N_{2i} are uncorrelated and hence, being Gaussian, are independent. The probability density function (pdf) for N_{1i} or N_{2i} is

$$p_N(n_i) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2\sigma^2} n_i^2}$$

It will be convenient to express (1.12) in vector form. Let $\mathbf{R}_i = [R_{1i}, R_{2i}]^T$ (received vector), $\mathbf{S}_i = [a_{1i}, a_{2i}]^T$ (sent vector), and $\mathbf{N}_i = [N_{1i}, N_{2i}]^T$ (noise vector). Then

$$\mathbf{R}_i = \mathbf{S}_i + \mathbf{N}_i.$$

Then $\mathbf{N}_i$ is jointly Gaussian distributed, with 0 mean and covariance matrix

$$E[\mathbf{N}_i\mathbf{N}_i^T] = \sigma^2 \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \sigma^2 I = R_N.$$

Explicitly, the pdf of the vector $\mathbf{N}_i$ is

$$p_{\mathbf{N}}(\mathbf{n}) = \frac{1}{2\pi\sqrt{\det(R_N)}} \exp\left[-\frac{1}{2}\mathbf{n}^T R_N^{-1}\mathbf{n}\right] = \frac{1}{2\pi\sigma^2}\exp\left[-\frac{1}{2\sigma^2}(n_1^2 + n_2^2)\right].$$

Let $P(\mathbf{s})$ be used to denote $P(\mathbf{S} = \mathbf{s}) = P_{\mathbf{S}}(\mathbf{s})$ for vectors $\mathbf{s} = [a_{1k}, a_{2k}]^T \in \mathcal{S}$. Let $P(\mathbf{s}|\mathbf{r})$ be used to denote $P(\mathbf{S} = \mathbf{s}|\mathbf{R} = \mathbf{r}) = P_{\mathbf{S}|\mathbf{R}}(\mathbf{S} = \mathbf{s}|\mathbf{R} = \mathbf{r})$ for an observed value of the random variable $\mathbf{R} = \mathbf{r}$. Note that conditioned on knowing that the transmitted signal is $\mathbf{S} = \mathbf{s}$, $\mathbf{R}$ is a Gaussian random variable with conditional density

$$\begin{aligned} p_{R|S}(\mathbf{r}|\mathbf{s}) = p_N(\mathbf{r} - \mathbf{s}) &= \frac{1}{2\pi\sqrt{\det(R_N)}} \exp\left[-\frac{1}{2}(\mathbf{r} - \mathbf{s})^T R_N^{-1}(\mathbf{r} - \mathbf{s})\right] \\ &= C \exp\left[-\frac{1}{2\sigma^2}\|\mathbf{r} - \mathbf{s}\|^2\right], \end{aligned} \tag{1.16}$$

where $\|\mathbf{r}-\mathbf{s}\|^2$ is the squared Euclidean distance between $\mathbf{r}$ and $\mathbf{s}$ and C is a quantity that does not depend on either $\mathbf{R}$ or $\mathbf{S}$. The quantity $p_{R|S}(\mathbf{r}|\mathbf{s})$ is called the *likelihood function*. The likelihood function $p_{R|S}(\mathbf{r}|\mathbf{s})$ is typically viewed as a function of the *conditioning* argument, with the observed values $\mathbf{r}$ as fixed parameters.

The signal point $\mathbf{s} \in \mathcal{S}$ depends uniquely upon the transmitted bits mapped to the signal constellation point. Conditioning upon knowing the transmitted signal is thus equivalent to conditioning on knowing the transmitted bits. Thus the notation $p(\mathbf{r}|\mathbf{s})$ is used interchangeably with $p(\mathbf{r}|\mathbf{b})$, when $\mathbf{s}$ is the signal used to represent the bits $\mathbf{b}$. For example, for BPSK modulation, we could write either $p(r|s = \sqrt{E_b})$ or $p(r|b = 1)$ or even $p(r|\tilde{b} = 1)$, since by the modulation described above the amplitude $\sqrt{E_b}$ is transmitted when the input bit is $b = 1$ (or $\tilde{b} = 1$).

1.5.2 MAP and ML Detection

Let $\mathbf{S}$ denote the transmitted value, where $\mathbf{S} \in \mathcal{S}$ is chosen with prior probability $P(\mathbf{S} = \mathbf{s})$, or, more briefly, $P(\mathbf{s})$. The receiver uses the received point $\mathbf{R} = \mathbf{r}$ to make a decision about what the transmitted signal $\mathbf{S}$ is. Let us denote the estimated decision as $\hat{\mathbf{s}} = [\hat{a}_1, \hat{a}_2]^T \in \mathcal{S}$. We will use the notation $P(\mathbf{s}|\mathbf{r})$ as a shorthand for $P(\mathbf{S} = \mathbf{s}|\mathbf{r})$.

Theorem 1.1 *The decision rule which minimizes the probability of error is to choose* $\hat{\mathbf{s}}$ *to be that value of* $\mathbf{s}$ *which maximizes* $P(\mathbf{S} = \mathbf{s}|\mathbf{r})$, *where the possible values of* $\mathbf{s}$ *are those in the signal constellation* $\mathcal{S}$. *That is,*

$$\hat{\mathbf{s}} = \arg\max_{\mathbf{s}\in\mathcal{S}} P(\mathbf{s}|\mathbf{r}). \tag{1.17}$$

Proof Let us denote the constellation as $\mathcal{S} = \{\mathbf{s}_i, i = 1, 2, \ldots, M\}$. Let $p(\mathbf{r}|\mathbf{s}_i)$ denote the pdf of the received signal when $\mathbf{S} = \mathbf{s}_i$ is the transmitted signal. Let Ω denote the space of possible received values; in the current case $\Omega = \mathbb{R}^2$. Let us partition the space Ω into regions Ω_i, where the decision rule is expressed as: set $\hat{\mathbf{s}} = \mathbf{s}_i$ if $\mathbf{r} \in \Omega_i$. That is,

$$\Omega_i = \{\mathbf{r} \in \Omega : \text{ decide } \hat{\mathbf{s}} = \mathbf{s}_i\}.$$

The problem, then, is to determine the partitions Ω_i. By the definition of the partition, the conditional probability of a correct answer when $\mathbf{S} = \mathbf{s}_i$ is sent is

$$P(\hat{\mathbf{s}} = \mathbf{s}_i | \mathbf{S} = \mathbf{s}_i) = \int_{\Omega_i} p(\mathbf{r}|\mathbf{s}_i)\, d\mathbf{r}.$$

Denote the conditional probability of error when signal $\mathbf{S} = \mathbf{s}_i$ is sent as $P_i(\mathcal{E})$:

$$P_i(\mathcal{E}) = P(\hat{\mathbf{s}} \neq \mathbf{s}_i | \mathbf{S} = \mathbf{s}_i).$$

Then we have

$$P_i(\mathcal{E}) = 1 - P(\hat{\mathbf{s}} = \mathbf{s}_i | \mathbf{S} = \mathbf{s}_i) = 1 - \int_{\Omega_i} p(\mathbf{r}|\mathbf{s}_i)\, d\mathbf{r}.$$

The average probability of error is

$$\begin{aligned} P(\mathcal{E}) &= \sum_{i=1}^{M} P_i(\mathcal{E}) P(\mathbf{S} = \mathbf{s}_i) = \sum_{i=1}^{M} P(\mathbf{S} = \mathbf{s}_i) \left[1 - \int_{\Omega_i} p(\mathbf{r}|\mathbf{s}_i)\, d\mathbf{r} \right] d\mathbf{r} \\ &= 1 - \sum_{i=1}^{M} \int_{\Omega_i} p(\mathbf{r}|\mathbf{s}_i) P(\mathbf{S} = \mathbf{s}_i)\, d\mathbf{r}. \end{aligned}$$

The probability of a correct answer is

$$P(\mathcal{C}) = 1 - P(\mathcal{E}) = \sum_{i=1}^{M} \int_{\Omega_i} p(\mathbf{r}|\mathbf{s}_i) P(\mathbf{S} = \mathbf{s}_i)\, d\mathbf{r} = \sum_{i=1}^{M} \int_{\Omega_i} P(\mathbf{S} = \mathbf{s}_i | \mathbf{r}) p(\mathbf{r})\, d\mathbf{r}.$$

Since $p(\mathbf{r}) \geq 0$, to maximize $P(\mathcal{C})$, the region of integration Ω_i should be chosen precisely so that it covers the region where $P(\mathbf{S} = \mathbf{s}_i | \mathbf{r})$ is the largest possible. That is,

$$\Omega_i = \{\mathbf{r} : P(\mathbf{S} = \mathbf{s}_i | \mathbf{r}) > P(\mathbf{S} = \mathbf{s}_j | \mathbf{r}), i \neq j\}.$$

This is equivalent to (1.17). □

Using Bayes' rule we can write (1.17) as

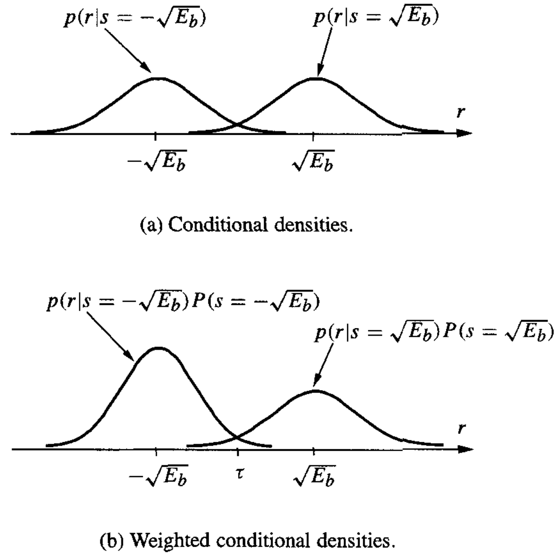
$$\hat{\mathbf{s}} = \arg\max_{\mathbf{s} \in \mathcal{S}} P(\mathbf{s}|\mathbf{r}) = \arg\max_{\mathbf{s} \in \mathcal{S}} \frac{p_{R|S}(\mathbf{r}|\mathbf{s}) P(\mathbf{s})}{p_R(\mathbf{r})}.$$

Since the denominator of the last expression does not depend on $\mathbf{s}$, we can further write

$$\boxed{\hat{\mathbf{s}} = \arg\max_{\mathbf{s} \in \mathcal{S}} p_{R|S}(\mathbf{r}|\mathbf{s}) P(\mathbf{s}).} \tag{1.18}$$

This is called the maximum *a posteriori* (MAP) decision rule. In the case that all the prior probabilities are equal (or are assumed to be equal), this rule can be simplified to

$$\boxed{\hat{\mathbf{s}} = \arg\max_{\mathbf{s} \in \mathcal{S}} p_{R|S}(\mathbf{r}|\mathbf{s}).}$$



(a) Conditional densities.

(b) Weighted conditional densities.

Figure 1.8: Conditional densities in BPSK modulation.

This is called the maximum likelihood (ML) decision rule.

Note: We will frequently suppress the subscript on a probability density function or distribution function, letting the arguments themselves indicate the intended random variables. We could thus write $p(\mathbf{r}|\mathbf{s})$ in place of $p_{R|S}(\mathbf{r}|\mathbf{s})$.

Once the decision $\hat{\mathbf{s}}$ is made, the corresponding bits are determined by the bit-to-constellation mapping. The output of the receiver is thus an estimate of the bits.

By the form of (1.16), we see that the ML decision rule for the Gaussian noise channel selects that point $\hat{\mathbf{s}} \in \mathcal{S}$ which is closest to $\mathbf{r}$ in squared **Euclidean distance,** $\|\mathbf{r}-\hat{\mathbf{s}}\|^2$.

1.5.3 Special Case: Binary Detection

For the case of binary transmission in a one-dimensional signal space, the signal constellation consists of the points $\mathcal{S} = \{\sqrt{E_b}, -\sqrt{E_b}\}$, corresponding, respectively, to the bit $b = 1$ or $b = 0$ (respectively, using the current mapping). The corresponding likelihood functions are

$$p(r|s=\sqrt{E_b}) = \frac{1}{2\pi}e^{-\frac{1}{2\sigma^2}(r-\sqrt{E_b})^2} \qquad p(r|s=-\sqrt{E_b}) = \frac{1}{2\pi}e^{-\frac{1}{2\sigma^2}(r+\sqrt{E_b})^2}.$$

These densities are plotted in Figure 1.8(a). We see $r|s = \sqrt{E_b}$ is a Gaussian with mean $\sqrt{E_b}$. The MAP decision rule compares the weighted densities $p(r|s=\sqrt{E_b})P(s=\sqrt{E_b})$ and $p(r|s=-\sqrt{E_b})P(s=-\sqrt{E_b})$. Figure 1.8(b) shows these densities in the case that $P(s=-\sqrt{E_b}) > P(s=\sqrt{E_b})$. Clearly, there is a threshold point τ at which

$$p(r|s=\sqrt{E_b})P(s=\sqrt{E_b}) = p(r|s=-\sqrt{E_b})P(s=-\sqrt{E_b}).$$

In this case, the decision rule (1.18) simplifies to

$$\hat{s} = \begin{cases} \sqrt{E_b} \text{ (i.e., } b_i = 1) & \text{if } r > \tau \\ -\sqrt{E_b} \text{ (i.e., } b_i = 0) & \text{if } r < \tau. \end{cases} \tag{1.19}$$

The threshold value can be computed explicitly as

$$\tau = \frac{\sigma^2}{2\sqrt{E_b}} \ln \frac{P(s = -\sqrt{E_b})}{P(s = \sqrt{E_b})}. \tag{1.20}$$

In the case that $P(s = \sqrt{E_b}) = P(s = -\sqrt{E_b})$, the decision threshold is at $\tau = 0$, as would be expected.

Binary detection problems are also frequently expressed in terms of *likelihood ratios*. For binary detection, the problem is one of determining, say, if $b = 1$ or if $b = 0$. The detection rule (1.18) becomes a test between

$$p(r|b = 1)P(b = 1) \text{ and } P(r|b = 0)P(b = 0).$$

This can be expressed as a *ratio*,

$$\frac{p(r|b = 1)P(b = 1)}{p(r|b = 0)P(b = 0)}.$$

In the case of equal priors, we obtain the likelihood ratio

$$L(r) = \frac{p(r|b = 1)}{p(r|b = 0)}.$$

For many channels, it is more convenient to use the *log likelihood ratio*

$$\Lambda(r) = \log \frac{p(r|b = 1)}{p(r|b = 0)},$$

where the natural logarithm is usually used. The decision is made that $\hat{b} = 1$ if $\Lambda(r) > 0$ and $\hat{b} = 0$ if $\Lambda(r) < 0$.

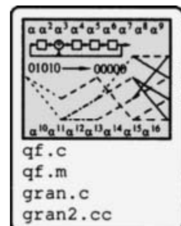
For the Gaussian channel with BPSK modulation, we have

$$\Lambda(r) = \log \frac{p(r|a = \sqrt{E_b})}{p(r|a = -\sqrt{E_b})} = \log \frac{\exp(-\frac{1}{2\sigma^2}(r - \sqrt{E_b})^2)}{\exp(-\frac{1}{2\sigma^2}(r + \sqrt{E_b})^2)} = \frac{2\sqrt{E_b}}{\sigma^2} r = L_c r, \tag{1.21}$$

where $L_c = \frac{2\sqrt{E_b}}{\sigma^2}$ is called the **channel reliability**[5].

The quantity $\Lambda(r) = L_c r$ can be used as *soft information* in a decoding system. The quantity $\text{sign}(\Lambda(r))$ is referred to as *hard information* in a decoding system. Most early error correction decoding algorithms employed hard information — actual estimated bit values — while there has been a trend toward increasing use of soft information decoders, which generally provide better performance.

1.5.4 Probability of Error for Binary Detection



Even with optimum decisions at the demodulator, errors can still be made with some probability (otherwise, error correction coding would not ever be needed). For binary detection problems in Gaussian noise, the probabilities can be expressed using the $Q(x)$ function, which is the probability that a unit Gaussian $N \sim \mathcal{N}(0, 1)$ exceeds x:

[5] In some sources (e.g. [134]) the channel reliability L_c is expressed alternatively as equivalent to $2E_b/\sigma^2$. This is in some ways preferable, since it is unitless.

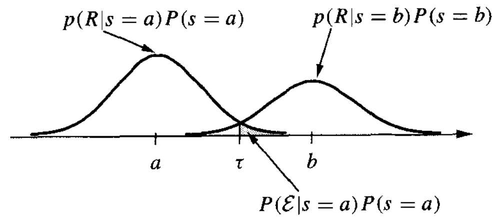
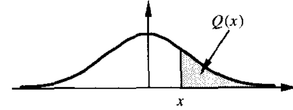


Figure 1.9: Distributions when two signals are sent in Gaussian noise.

$$Q(x) = P(N > x) = \frac{1}{\sqrt{2\pi}} \int_x^\infty e^{-n^2/2}\,dn.$$



The Q function has the properties that

$$Q(x) = 1 - Q(-x) \qquad Q(0) = \frac{1}{2} \qquad Q(-\infty) = 1 \qquad Q(\infty) = 0.$$

For a Gaussian random variable Z with mean μ and variance σ^2, $Z \sim \mathcal{N}(\mu, \sigma^2)$, it is straightforward to show that

$$P(Z > x) = \frac{1}{\sqrt{2\pi}\sigma} \int_x^\infty e^{-(z-\mu)^2/2\sigma^2}\,dz = Q\left(\frac{x-\mu}{\sigma}\right).$$

Suppose there are two points a and b along an axis, and that

$$R = s + N,$$

where s is one of the two points, and $N \sim \mathcal{N}(0, \sigma^2)$. The distributions $P(R|s = a)P(s = a)$ and $P(R|s = b)P(s = b)$ are plotted in Figure 1.9. A decision threshold τ is also shown. When a is sent, an error is made when $R > \tau$. Denoting $\mathcal{E}$ as the error event, this occurs with probability

$$P(\mathcal{E}|s = a) = P(R > \tau) = \frac{1}{\sqrt{2\pi}\sigma} \int_\tau^\infty e^{-\frac{1}{2\sigma^2}(r-a)^2}\,dr = Q\left(\frac{\tau - a}{\sigma}\right).$$

When b is sent, an error is made when $R < \tau$, which occurs with probability

$$P(\mathcal{E}|s = b) = P(R < \tau) = 1 - P(R > \tau) = 1 - Q\left(\frac{\tau - b}{\sigma}\right) = Q\left(\frac{b - \tau}{\sigma}\right).$$

The overall probability of error is

$$\begin{aligned} P(\mathcal{E}) &= P(\mathcal{E}|s = a)P(s = a) + P(\mathcal{E}|s = b)P(s = b) \\ &= Q\left(\frac{\tau - a}{\sigma}\right) P(s = a) + Q\left(\frac{b - \tau}{\sigma}\right) P(s = b). \end{aligned} \tag{1.22}$$

An important special case is when $P(s = a) = P(s = b) = \frac{1}{2}$. Then the decision threshold is at the midpoint $\tau = (a + b)/2$. Let $d = |b - a|$ be the distance between the two signals. Then (1.22) can be written

$$\boxed{P(\mathcal{E}) = Q\left(\frac{d}{2\sigma}\right).} \tag{1.23}$$

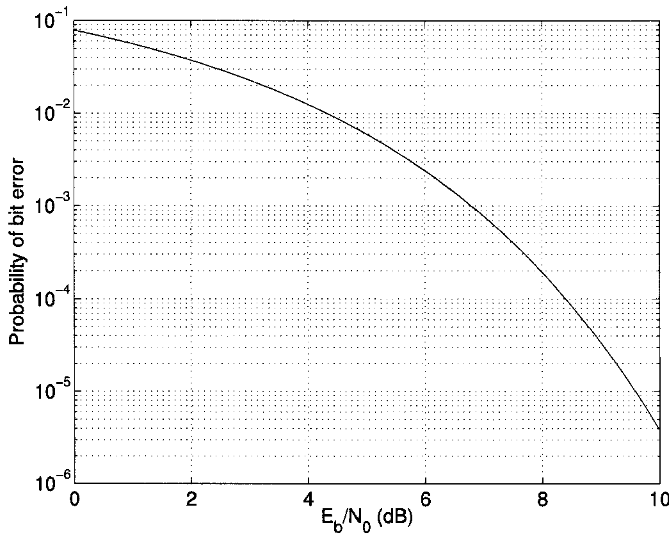


Figure 1.10: Probability of error for BPSK signaling.

Even in the case that the signals are transmitted in multidimensional space, provided that the covariance of the noise is of the form $\sigma^2 I$, the probability of error is still of the form (1.23). That is, if

$$\mathbf{R} = \mathbf{s} + \mathbf{N}$$

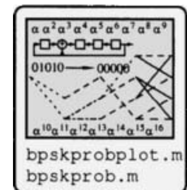
are n-dimensional vectors, with $\mathbf{N} \sim \mathcal{N}(\mathbf{0}, \sigma^2 I)$, and $\mathbf{S} \in \{\mathbf{a}, \mathbf{b}\}$ are two equiprobable transmitted vectors, then the probability of decision error is $P(\mathcal{E}) = Q\left(\frac{d}{2\sigma}\right)$, where $d = \|\mathbf{a} - \mathbf{b}\|$ is the Euclidean distance between vectors. This formula is frequently used in characterizing the performance of codes.

For the particular case of BPSK signaling, we have $a = -\sqrt{E_b}$, $b = \sqrt{E_b}$, and $d = 2\sqrt{E_b}$. The probability $P(\mathcal{E})$ is denoted as P_b, the "probability of a bit error." Thus,

$$P_b = Q\left(\frac{\tau + \sqrt{E_b}}{\sigma}\right) P(-\sqrt{E_b}) + Q\left(\frac{\sqrt{E_b} - \tau}{\sigma}\right) P(\sqrt{E_b}). \tag{1.24}$$

When $P(\sqrt{E_b}) = P(-\sqrt{E_b})$, then $\tau = 0$. Recalling that the variance for the channel is expressed as $\sigma^2 = \frac{N_0}{2}$, we have for BPSK transmission

$$\boxed{P_b = Q(\sqrt{E_b}/\sigma) = Q\left(\sqrt{\frac{2E_b}{N_0}}\right).} \tag{1.25}$$



The quantity E_b/N_0 is frequently called the (bit) signal-to-noise ratio (SNR).

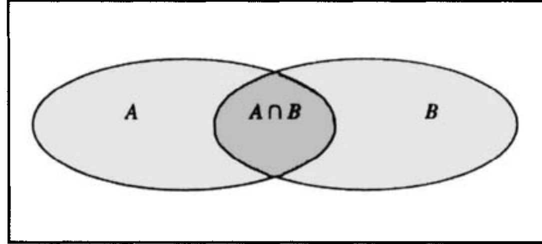
Figure 1.10 shows the probability of bit error for a BPSK as a function of the signal-to-noise ratio in dB (decibel), where

$$E_b/N_0 \text{ dB} = 10 \log_{10} E_b/N_0,$$

for the case $P(\sqrt{E_b}) = P(-\sqrt{E_b})$.

Box 1.1: The Union Bound

For sets A and B, we have $P(A \cup B) = P(A) + P(B) - P(A \cap B)$.



Then, since $P(A \cap B) \geq 0$, clearly $P(A \cup B) \leq P(A) + P(B)$.

1.5.5 Bounds on Performance: The Union Bound

For some signal constellations, exact expressions for the probability of error are difficult or inconvenient to obtain. In many cases it is more convenient to obtain a *bound* on the probability of error using the union bound. (See Box 1.1.) Consider, for example, the 8-PSK constellation in Figure 1.11. If the point labeled $\mathbf{s}_0$ is transmitted, then an error occurs if the received signal falls in either shaded area. Let A be the event that the received signal falls on the incorrect side of threshold line L_1 and let B be the event that the received signal falls on the incorrect side of the line L_2. Then

$$\Pr(\text{symbol decoding error}|\mathbf{s}_0 \text{ sent}) = P(A \cup B).$$

The events A and B are not disjoint, as is apparent from Figure 1.11. The exact probability

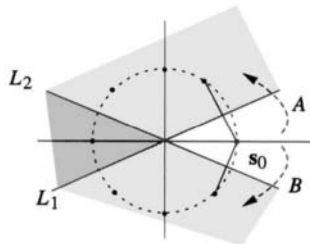


Figure 1.11: Probability of error bound for 8-PSK modulation.

computation is made more difficult by the overlapping region. Using the union bound, however, the probability of error can bounded as

$$\Pr(\text{symbol decoding error}|\mathbf{s}_0 \text{ sent}) \leq P(A) + P(B)$$

The event A occurs with the probability that the transmitted signal falls on the wrong side of the line L_1; similarly for B. Assuming that the noise is independent Gaussian with variance σ^2 in each coordinate direction, this probability is

$$P(A) = Q\left(\frac{d_{\min}}{2\sigma}\right),$$

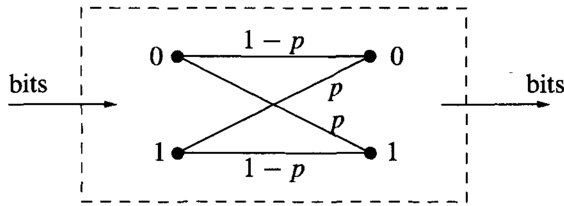


Figure 1.12: A binary symmetric channel.

where $d_{\min}$ is the minimum distance between signal points. Denote the probability of a symbol error by P_s. Assuming that all symbols are sent with equal probability, we have $P_s = \Pr(\text{symbol decoding error}|\mathbf{s}_0 \text{ sent})$, where the probability is bounded by

$$P_s \leq 2Q\left(\frac{d_{\min}}{2\sigma}\right). \tag{1.26}$$

The factor 2 multiplying the Q function is the number of nearest neighbors around each constellation point. The probability of error is dominated by the minimum distance between points: better performance is obtained with larger distance. As E_s/N_0 (the symbol SNR) increases, the probability of falling in the intersection region decreases and the bound (1.26) becomes increasingly tight.

For signal constellations larger than BPSK, it common to plot the probability of a *symbol* error *vs.* the SNR in E_s/N_0, where E_s is the average signal energy. However, when the bits are assigned in Gray code order, then a symbol error is likely to be an adjacent symbol, so that only a single bit error occurs

$$P_b \approx P_s \text{ for sufficiently large SNR.} \tag{1.27}$$

More generally, the probability of detection error for a symbol s which has K neighbors in signal space at a distance $d_{\min}$ from it can be bounded by

$$P_s \leq KQ\left(\frac{d_{\min}}{2\sigma}\right), \tag{1.28}$$

and the bound becomes increasingly tight as the SNR increases.

1.5.6 The Binary Symmetric Channel

The binary symmetric channel (BSC) is a simplified channel model which contemplates only the transmission of bits over the channel; it does not treat details such as signal spaces, modulation, or matched filtering. The BSC accepts 1 bit per unit of time and transmits that bit with a probability of error p. A representation of the BSC is shown in Figure 1.12. An incoming bit of 0 or 1 is transmitted through the channel unchanged with probability $1 - p$, or flipped with probability p. The sequence of output bits in a BSC can be modeled as

$$R_i = S_i + N_i, \tag{1.29}$$

where $R_i \in \{0, 1\}$ are the output bits, $S_i \in \{0, 1\}$ are the input bits, $N_i \in \{0, 1\}$ represents the possible bit errors, where N_i is 1 if an error occurs on bit i. The addition in (1.29) is **modulo 2 addition**, according to the addition table

$$0 + 0 = 0 \qquad 0 + 1 = 1 \qquad 1 + 1 = 0,$$

so that if $N_i = 1$, then R_i is the bit complement of S_i. The BSC is an instance of a *memoryless channel.* This means each of the errors N_i is statistically independent of all the other N_i. The probability that bit i has an error is $P(N_i = 1) = p$, where p is called the BSC crossover probability. The sequence $\{N_i, i \in \mathbb{Z}\}$ can be viewed as an independent and identically distributed (i.i.d.) Bernoulli(p) random process.

Suppose that S is sent over the channel and R is received. The likelihood function $P(R|S)$ is

$$P(R|S) = \begin{cases} 1-p & \text{if } R = S \\ p & \text{if } R \neq S. \end{cases} \tag{1.30}$$

Now suppose that the sequence $\mathbf{s} = [s_1, s_2, \ldots, s_n]$ is transmitted over a BSC and that the received sequence is $\mathbf{R} = [r_1, r_2, \ldots, r_n]$. Because of independent noise samples, the likelihood function factors,

$$P(\mathbf{R}|\mathbf{S}) = \prod_{i=1}^{n} P(R_i|S_i). \tag{1.31}$$

Each factor in the product is of the form (1.30). Thus there is a factor $(1-p)$ every time R_i agrees with S_i, and a factor p every time R_i differs from S_i. To represent this, we introduce the *Hamming distance.*

Definition 1.1 The **Hamming distance** between a sequence $\mathbf{x} = [x_1, x_2, \ldots, x_n]$ and a sequence $\mathbf{y} = [y_1, y_2, \ldots, y_n]$ is the number of positions that the corresponding elements differ:

$$\boxed{d_H(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{n} [x_i \neq y_i].} \tag{1.32}$$

Here we have used the notation (Iverson's convention [126])

$$[x_i \neq y_i] = \begin{cases} 1 & \text{if } x_i \neq y_i \\ 0 & \text{if } x_i = y_i. \end{cases}$$

□

Using the notation of Hamming distance, we can write the likelihood function (1.31) as

$$P(\mathbf{R}|\mathbf{S}) = \underbrace{(1-p)^{n-d_H(\mathbf{R},\mathbf{S})}}_{\substack{\text{number of places} \\ \text{they are the same}}} \quad \underbrace{p^{d_H(\mathbf{R},\mathbf{S})}}_{\substack{\text{number of places} \\ \text{they differ}}}.$$

The likelihood function can also be written as

$$P(\mathbf{R}|\mathbf{S}) = \left(\frac{p}{1-p}\right)^{d_H(\mathbf{R},\mathbf{S})} (1-p)^n.$$

Consider now the detection problem of deciding if the sequence $\mathbf{S}_1$ or the sequence $\mathbf{S}_2$ was sent, where each occur with equal probability. The maximum likelihood decision rule says to choose that value of $\mathbf{S}$ for which $\frac{p}{1-p}^{d_H(\mathbf{R},\mathbf{S})}(1-p)^n$ is the largest. Assuming that $p < \frac{1}{2}$, this corresponds to choosing that value of $\mathbf{S}$ for which $d_H(\mathbf{R}, \mathbf{S})$ is the *smallest*, that is, the vector $\mathbf{S}$ nearest to $\mathbf{R}$ in Hamming distance.

We see that for detection in a Gaussian channel, the *Euclidean* distance is the appropriate distance for detection. For the BSC, the *Hamming* distance is the appropriate distance for detection.

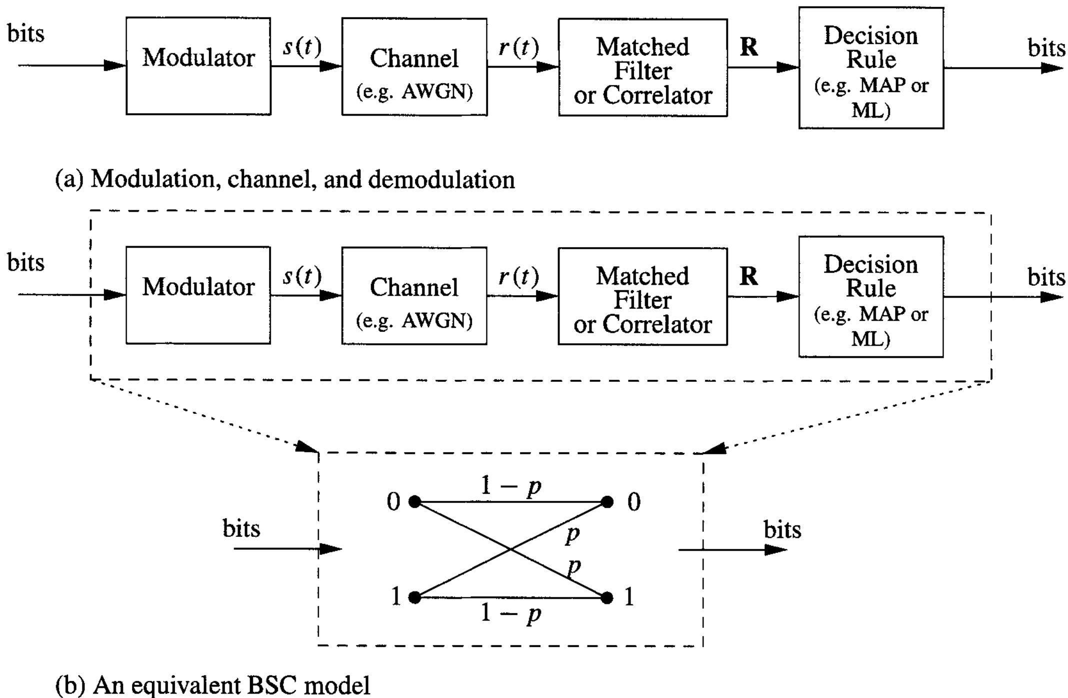


Figure 1.13: (a) System diagram showing modulation, channel, and demodulation; (b) BSC equivalent.

1.5.7 The BSC and the Gaussian Channel Model

At a sufficiently coarse level of detail, the modulator/demodulator system with the additive white Gaussian noise channel can be viewed as a BSC. The modulation, channel, and detector collectively constitute a "channel" which accepts bits at the input and emits bits at the output. The end-to-end system viewed at this level, as suggested by the dashed box in Figure 1.13(b), forms a BSC. The crossover probability p can be computed based on the system parameters,

$$p = P(\text{bit out} = 0|\text{bit in} = 1) = P(\text{bit out} = 1|\text{bit in} = 0) = P_b = Q(\sqrt{2E_b/N_0}).$$

In many cases the probability of error is computed using a BSC with an "internal" AWGN channel, so that the probability of error is produced as a function of E_b/N_0.

1.6 Memoryless Channels

A memoryless channel is one in which the output r_n at the nth symbol time depends only on the input at time n. Thus, given the input at time n, the output at time n is statistically independent of the outputs at other times. That is, for a sequence of received signals

$$\mathbf{R} = (R_1, R_2, \ldots, R_m)$$

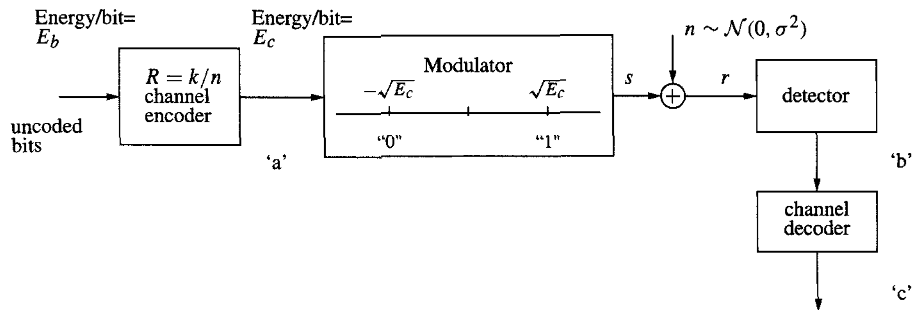


Figure 1.14: Energy for a coded signal.

depending on transmitted signals $S_1, S_2, \ldots, S_m$, the likelihood function

$$p(R_1, R_2, \ldots, R_m | S_1, S_2, \ldots, S_m)$$

can be factored as

$$p(R_1, R_2, \ldots, R_m | S_1, S_2, \ldots, S_m) = \prod_{i=1}^{m} p(R_i | S_i).$$

Both the additive Gaussian channel and the binary symmetric channel that have been introduced are memoryless channels. We will almost universally assume that the channels are memoryless channels. The bursty channels discussed in Chapter 10 and the convolutive channel introduced in Chapter 14 are exceptions to this.

1.7 Simulation and Energy Considerations for Coded Signals

In channel coding, k input bits yield n output bits, where $n > k$. Let $R = k/n$ be the code rate. A transmission budget which allocates E_b Joules/bit for the uncoded data must spread that energy over more coded bits. Let

$$E_c = RE_b$$

denote the "energy per coded bit." We thus have $E_c < E_b$. Consider the framework shown in Figure 1.14. From point 'a' to point 'b,' there is conventional (uncoded) BPSK modulation scheme, except that the energy per bit is E_c. Thus, at point 'b' the probability of error can be computed as

$$P_{b,coded} = Q\left(\sqrt{\frac{2E_c}{N_0}}\right).$$

Since $E_c < E_b$, this is *worse* performance than uncoded BPSK would have had. Figure 1.15 shows the probability of error of coded bits for $R = 1/2$ and $R = 1/3$ error correction codes at point 'b' in Figure 1.14. At the receiver, the detected coded bits are passed to the channel decoder, the error correction stage, which attempts to correct errors. Clearly, in order to be of any value the code must be strong enough so that the bits emerging at point 'c' of Figure 1.14 can compensate for the lower energy per bit in the channel, plus correct other errors. Fortunately, we will see that this is in fact the case.

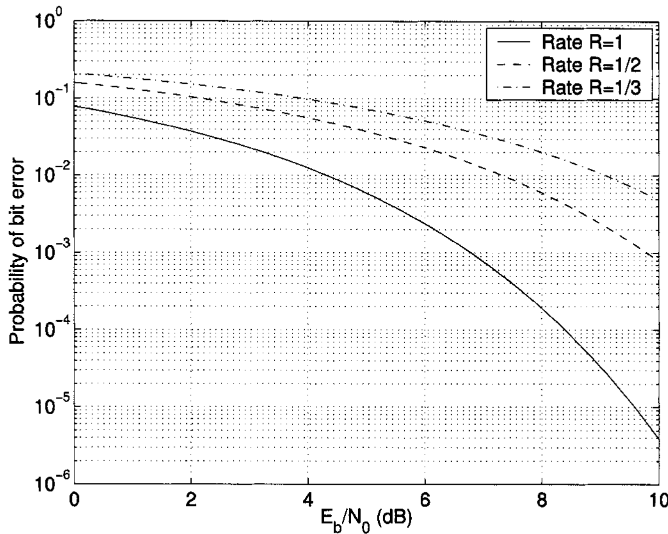


Figure 1.15: Probability of error for coded bits, before correction.

Now consider how this system might be simulated in software. It is common to simulate the modulator at point 'a' of Figure 1.14 as having fixed amplitudes and to adjust the variance σ^2 of the noise n in the channel. One of the primary considerations, therefore, is how to set σ^2.

Frequently it is desired to simulate performance at a particular SNR, E_b/N_0. Let $\gamma = E_b/N_0$ denote the desired signal to noise ratio at which to simulate. Frequently, this is expressed in dB, so we have

$$\gamma = 10^{(\text{SNR,dB})/10}.$$

Recalling that $\sigma^2 = N_0/2$, and knowing γ, we have

$$\gamma = \frac{E_b}{2\sigma^2},$$

so

$$\sigma^2 = \frac{E_b}{2\gamma}.$$

Since $E_b = E_c/R$, we have

$$\sigma^2 = \frac{E_c}{2R\gamma}.$$

It is also common in simulation to normalize, so that the simulated signal amplitude is $E_c = 1$.

1.8 Some Important Definitions and a Trivial Code: Repetition Coding

In this section we introduce the important coding concepts of code rate, Hamming distance, minimum distance, Hamming spheres, and the generator matrix. These concepts are intro-

duced by means of a simple, even trivial, example of an error correction code, the repetition code.

Let $\mathbb{F}_2$ denote the set (field) with two elements in it, 0 and 1. In this field, arithmetic operations are defined as:

$$0+0=0 \quad 0+1=1 \quad 1+0=1 \quad 1+1=0$$
$$0\cdot 0=0 \quad 0\cdot 1=0 \quad 1\cdot 0=0 \quad 1\cdot 1=1.$$

Let $\mathbb{F}_2^n$ denote the (vector) space of n-tuples of elements of $\mathbb{F}_2$.

An (n, k) binary code is a set of 2^k distinct points in $\mathbb{F}_2^n$. Another way of putting this: An (n, k) binary code is a code that accepts k bits as input and produces n bits as output.

Definition 1.2 The **rate** of an (n, k) code is

$$R = \frac{k}{n}.$$

□

The $(n, 1)$ **repetition code**, where n is odd, is the code obtained by repeating the 1-bit input n times in the output codeword. That is, the codeword representing the input 0 is a block of n 0s and the codeword representing the input 1 is a block of n 1s. The *code* $\mathcal{C}$ consists of the set of two codewords

$$\mathcal{C} = \{[0, 0, \ldots, 0], [1, 1, \ldots, 1]\} \subset \mathbb{F}_2^n.$$

Letting m denote the message, the corresponding codeword is

$$\mathbf{c} = [\underbrace{m, m, m, \ldots, m}_{n \text{ copies}}].$$

This is a rate $R = 1/n$ code.

Encoding can be represented as a matrix operation. Let G be the $1 \times n$ **generator matrix** given by

$$G = \begin{bmatrix} 1 & 1 & \cdots & 1 \end{bmatrix}.$$

Then the encoding operation is

$$\mathbf{c} = mG.$$

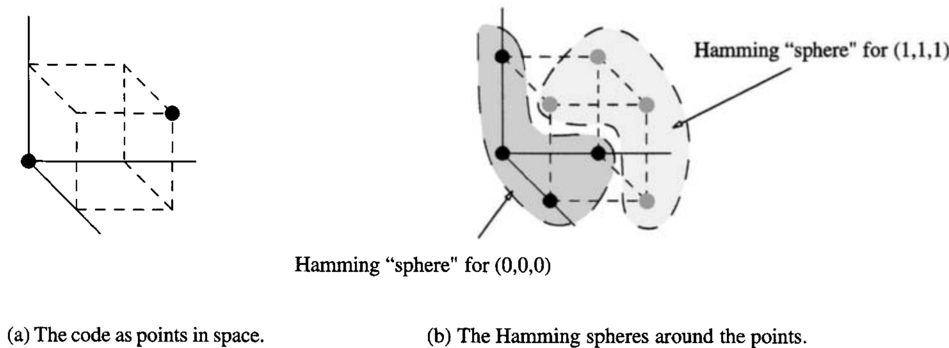
1.8.1 Detection of Repetition Codes Over a BSC

Let us first consider decoding of this code when transmitted through a BSC with crossover probability $p < 1/2$. Denote the output of the BSC by

$$\mathbf{r} = \mathbf{c} + \mathbf{n},$$

where the addition is modulo 2 and $\mathbf{n}$ is a binary vector of length n, with 1 in the positions where the channel errors occur. Assuming that the codewords are selected with equal probability, maximum likelihood decoding is appropriate. As observed in Section 1.5.6, the maximum likelihood decoding rule selects the codeword in $\mathcal{C}$ which is closest to the received vector $\mathbf{r}$ in Hamming distance. For the repetition code, this decoding rule can be expressed as a majority decoding rule: If the majority of received bits are 0, decode a 0; otherwise, decode a 1. For example, take the $(7, 1)$ repetition code and let $m = 1$. Then the codeword is $\mathbf{c} = [1, 1, 1, 1, 1, 1, 1]$. Suppose that the received vector is

$$\mathbf{r} = [1, 0, 1, 1, 0, 1, 1].$$



(a) The code as points in space. (b) The Hamming spheres around the points.

Figure 1.16: A (3, 1) binary repetition code.

Since 5 out of the 7 bits are 1, the decoded value is

$$\hat{m} = 1.$$

An error *detector* can also be established. If the received vector $\mathbf{r}$ is not one of the codewords, we *detect* that the channel has introduced one or more errors into the transmitted codeword.

The codewords in a code C can be viewed as points in n-dimensional space. For example, Figure 1.16(a) illustrates the codewords as points $(0, 0, 0)$ and $(1, 1, 1)$ in 3-dimensional space. (Beyond three dimensions, of course, the geometric viewpoint cannot be plotted, but it is still valuable conceptually.) In this geometric setting, we use the **Hamming distance** to measure distances between points.

Definition 1.3 The **minimum distance** $d_{\min}$ of a code C is the smallest Hamming distance between any two codewords in the code:

$$\boxed{d_{\min} = \min_{\mathbf{c}_i, \mathbf{c}_j \in C, \mathbf{c}_i \neq \mathbf{c}_j} d_H(\mathbf{c}_i, \mathbf{c}_j).}$$

□

The two codewords in the $(n, 1)$ repetition code are clearly a (Hamming) distance n apart.

In this geometric setting the ML decoding algorithm may be expressed as: Choose the codeword $\hat{\mathbf{c}}$ which is closest to the received vector $\mathbf{r}$. That is,

$$\hat{\mathbf{c}} = \arg\min_{\mathbf{c} \in C} d_H(\mathbf{r}, \mathbf{c}).$$

A different decoder is based on constructing a sphere around each codeword.

Definition 1.4 The **Hamming sphere** of radius t around a codeword $\mathbf{c}$ consists of all vectors which are at a Hamming distance $\leq t$ from $\mathbf{c}$. □

For example, for the (3, 1) repetition code, the codewords and the points in their Hamming spheres are

Codeword	Points in its sphere
(0,0,0)	(0,0,0),(0,0,1),(0,1,0),(1,0,0)
(1,1,1)	(1,1,1),(1,1,0),(1,0,1),(0,1,1),

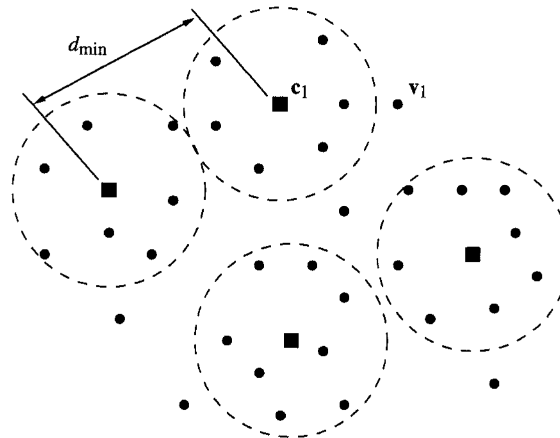


Figure 1.17: A representation of decoding spheres.

as illustrated in Figure 1.16(b).

When the Hamming spheres around each codeword are all taken to have the same radius, the largest such radius producing nonoverlapping spheres is determined by the separation between the *nearest* two codewords in the code, d_{min}. The radius of the spheres in this case is $t = \lfloor (d_{\text{min}} - 1)/2 \rfloor$, where the notation $\lfloor x \rfloor$ means to take the greatest integer $\leq x$. Figure 1.17 shows the idea of these Hamming spheres. The black squares represent codewords in n-dimensional space and black dots represent other vectors in n-dimensional space. The dashed lines indicate the boundaries of the Hamming spheres around the codewords. If a vector $\mathbf{r}$ falls inside the sphere around a codeword, then it is closer to that codeword than to any other codeword. By the ML criterion, $\mathbf{r}$ should decode to that codeword inside the sphere. When all the spheres have radius $t = \lfloor (d_{\text{min}} - 1)/2 \rfloor$, this decoding rule referred to as *bounded distance decoding*.

The decoder will make a decoding error if the channel noise moves the received vector $\mathbf{r}$ into a sphere other than the sphere the true codeword is in. Since the centers of the spheres lie a distance at least d_{min} apart, the decoder is guaranteed to *decode* correctly provided that no more than t errors occur in the received vector $\mathbf{r}$. The number t is called the **random error correction capability** of the code. If d_{min} is even and two codewords lie exactly d_{min} apart and the channel introduces $d_{\text{min}}/2$ errors, then the received vector lies right on the boundary of two spheres. In this case, given no other information, the decoder must choose one of the two codewords arbitrarily; half the time it will make an error.

Note from Figure 1.17 that in a bounded distance decoder there may be vectors that fall outside the Hamming spheres around the codewords, such as the vector labeled $\mathbf{v}_1$. If the received vector $\mathbf{r} = \mathbf{v}_1$, then the nearest codeword is $\mathbf{c}_1$. A bounded distance decoder, however, would not be able to decode if $\mathbf{r} = \mathbf{v}_1$, since it can only decode those vectors that fall in spheres of radius t. The decoder might have to declare a decoding failure in this case.

A true maximum likelihood (ML) decoder, which chooses the nearest codeword to the received vector, would be able to decode. Unfortunately, ML decoding is computationally very difficult for large codes. Most of the algebraic decoding algorithms in this book are only bounded distance decoders. An interesting exception are the decoders presented in Chapters 7 and 11, which actually produce *lists* of codeword candidates. These decoders

are called *list decoders*.

If the channel introduces fewer than $d_{\min}$ errors, then these can be *detected*, since $\mathbf{r}$ cannot be another codeword in this case. In summary, for a code with minimum distance $d_{\min}$:

Guaranteed error correction capability:	$t = \lfloor (d_{\min} - 1)/2 \rfloor$
Guaranteed error detection capability:	$d_{\min} - 1$

Having defined the repetition code, let us now characterize its probability of error performance as a function of the BSC crossover probability p. For the $(n, 1)$ repetition code, $d_{\min} = n$, and $t = (n-1)/2$ (remember n is odd). Suppose in particular that $n = 3$, so that $t = 1$. Then the decoder will make an error if the channel causes either 2 or 3 bits to be in error. Using P_e^n to denote the probability of decoding error for a code of length n, we have

$$\begin{aligned} P_e^3 &= \text{Prob(2 channel errors)} + \text{Prob(3 channel errors)} \\ &= 3p^2(1-p) + p^3 = 3p^2 - 2p^3. \end{aligned}$$

If $p < \frac{1}{2}$, then $P_e^3 < p$, that is, the decoder will have fewer errors than using the channel without coding.

Let us now examine the probability of decoding error for a code of length n. Note that it doesn't matter what the transmitted codeword was; the probability of error depends only on the error introduced by the channel. Clearly, the decoder will make an error if more than half of the received bits are in error. More precisely, if more than t bits are in error, the decoder will make an error. The probability of error can be expressed as

$$P_e^n = \sum_{i=t+1}^{n} \text{Prob}(i \text{ channel errors occur out of } n \text{ transmitted bits}).$$

The probability of exactly i bits in error out of n bits, where each bit is drawn at random with probability p is[6]

$$\binom{n}{i} p^i (1-p)^{n-i},$$

so that

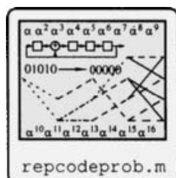
$$\begin{aligned} P_e^n &= \sum_{i=t+1}^{n} \binom{n}{i} p^i (1-p)^{n-i} \\ &= \binom{n}{t+1} (1-p)^n \left(\frac{p}{1-p}\right)^{t+1} + \text{terms of higher degree in } p. \end{aligned}$$

It would appear that as the code length increases, and thus t increases, the probability of decoder error decreases. (This is substantiated in Exercise 1.16b.) Thus, it is possible to obtain *arbitrarily small* probability of error, but at the cost of a very low rate: $R = 1/n \to 0$ as $P_e^N \to 0$.

Let us now consider using this repetition code for communication over the AWGN channel. Let us suppose that the transmitter has $P = 1$ Watt (W) of power available and that we want to send information at 1 bit/second. There is thus $E_b = 1$ Joule (J) of energy available for each bit of information. Now the information is coded using an $(n, 1)$ repetition

[6] The binomial coefficient is $\binom{n}{i} = \frac{n!}{i!(n-i)!}$

code. To maintain the information rate of 1 bit/second, we must send n coded bits/second. With n times as many bits to send, there is still only 1 W of power available, which must be shared among all the coded bits. The energy available for each coded bit, which we denote as E_c, is $E_c = E_b/n$. Thus, because of coding, there is less energy available for each bit to convey information! The probability of error for the AWGN channel (i.e., the binary crossover probability for the effective BSC) is



$$p = Q(\sqrt{2E_c/N_0}) = Q(\sqrt{2E_b/nN_0}).$$

The crossover probability p is higher as a result of using a code! However, the hope is that the error decoding capability of the *overall* system is better. Nevertheless, for the repetition code, this hope is in vain.

Figure 1.18 shows the probability of error for repetition codes (here, consider only the hard-decision decoding). The coded performance is worse than the uncoded performance, and the performance gets worse with increasing n.

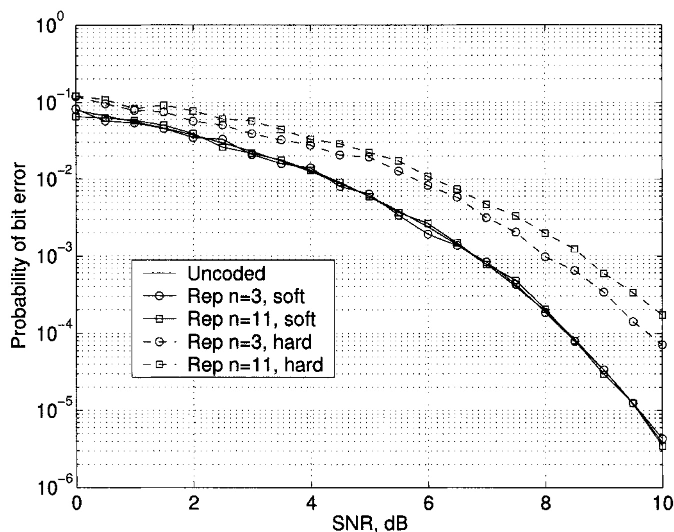


Figure 1.18: Performance of the (3, 1) and (11, 1) repetition code over BSC using both hard- and soft-decision decoding.

1.8.2 Soft-Decision Decoding of Repetition Codes Over the AWGN

Let us now consider decoding over the AWGN using a *soft-decision* decoder. Since the repetition code has a particularly simple codeword structure, it is straightforward to describe the soft-decision decoder and characterize its probability of error.

The likelihood function is

$$p(\mathbf{r}|\mathbf{c}) = \prod_{i=1}^{n} p(r_i|c_i),$$

so that the log likelihood ratio

$$\Lambda(\mathbf{r}) = \log \frac{p(\mathbf{r}|m = 1)}{p(\mathbf{r}|m = 0)}$$

can be computed using (1.21) as

$$\Lambda(\mathbf{r}) = L_c \sum_{i=1}^{n} r_i.$$

Then the decoder decides $\hat{m} = 1$ if $\Lambda(\mathbf{r}) > 0$, or $\hat{m} = 0$ if $\Lambda(\mathbf{r}) < 0$. Since the threshold is 0 and L_c is a positive constant, the decoder decides

$$\hat{m} = \begin{cases} 1 & \text{if } \sum_{i=1}^{n} r_i > 0 \\ 0 & \text{if } \sum_{i=1}^{n} r_i < 0. \end{cases}$$

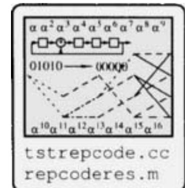
The soft-decision decoder performs superior to the hard-decision decoder. Suppose the vector $(-\sqrt{E_c}, -\sqrt{E_c}, \ldots, -\sqrt{E_c})$ is sent (corresponding to the all-zero codeword). If one of the r_i happens to be greater than 0, but other of the r_i are correspondingly less than 0, the erroneous positive quantities might be canceled out by the other symbols. In fact, it is straightforward to show (see Exercise 1.18) that the probability of error for the $(n, 1)$ repetition code with soft-decision decoding is

$$P_b = Q(\sqrt{2E_b/N_0}). \tag{1.33}$$

That is, it is the same as for uncoded transmission — still not effective as a code, but better than hard-decision decoding.

1.8.3 Simulation of Results

While it is possible for these simple codes to compute explicit performance curves, it is worthwhile to consider how the performance might also be simulated, since other codes that we will examine may be more difficult to analyze. The program here illustrates a framework for simulating the performance of codes. The probability of error is estimated by running codewords through a simulated Gaussian channel until a specified number of errors has occurred. Then the estimated probability of error is the number of errors counted divided by the number of bits generated.



tstrepcode.cc
repcoderes.m

Figure 1.18 shows the probability of error for uncoded transmission and both hard- and soft-decision decoding of (3, 1) and (11, 1) codes.

1.8.4 Summary

This lengthy example on a nearly useless code has introduced several concepts that will be useful for other codes:

- The concept of minimum distance of a code.
- The probability of decoder error.
- The idea of a generator matrix.
- The fact that not every code is good![7]
- Recognition that soft-decision decoding is superior to hard-input decoding in terms of probability of error.

[7]Despite the fact that these are very low-rate codes and historically of little interest, repetition codes are an essential component of a very powerful, recently introduced code, the repeat accumulate code introduced in Section 15.14.

Prior to the proof of Shannon's channel coding theorem and the research it engendered, communication engineers were in a quandary. It was believed that to obtain totally reliable communication, it would be necessary to transmit very slow rates, essentially employing repetition codes to catch any errors and using slow symbol rate to increase the energy per bit. However, Shannon's theorem dramatically changed this perspective, indicating that it is not necessary to slow the rate of communication to zero. It is only necessary to use better codes.

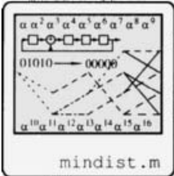
1.9 Hamming Codes

As a second example we now introduce Hamming codes. These are codes which are much better than repetition codes and were the first important codes discovered. Hamming codes lie at the intersection of many different kinds of codes, so we will use them also to introduce several important themes which will be developed throughout the course of this book.

A (7, 4) Hamming code produces 7 bits of output for every 4 bits of input. Hamming codes are *linear block codes*, which means that the encoding operation can be described in terms of a 4×7 generator matrix, such as

$$G = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix}. \tag{1.34}$$

The codewords are obtained as linear combination of the *rows* of G, where all the operations are computed modulo 2 in each vector element. That is, the code is the row space of G. For a message vector $\mathbf{m} = [m_1, m_2, m_3, m_4]$ the codeword is



$$\mathbf{c} = \mathbf{m}G.$$

For example, if $\mathbf{m} = [1, 1, 0, 0]$ then

$$\mathbf{c} = [1, 1, 0, 1, 0, 0, 0] + [0, 1, 1, 0, 1, 0, 0] = [1, 0, 1, 0, 1, 0, 0].$$

It can be verified that the minimum distance of the Hamming code is $d_{\min} = 3$, so the code is capable of correcting 1 error in every block of n bits.

The codewords for this code are

$$\begin{aligned} &[0,0,0,0,0,0,0], [1,1,0,1,0,0,0], [0,1,1,0,1,0,0], [1,0,1,1,1,0,0] \\ &[0,0,1,1,0,1,0], [1,1,1,0,0,1,0], [0,1,0,1,1,1,0], [1,0,0,0,1,1,0] \\ &[0,0,0,1,1,0,1], [1,1,0,0,1,0,1], [0,1,1,1,0,0,1], [1,0,1,0,0,0,1] \\ &[0,0,1,0,1,1,1], [1,1,1,1,1,1,1], [0,1,0,0,0,1,1], [1,0,0,1,0,1,1]. \end{aligned} \tag{1.35}$$

The Hamming decoding algorithm presented here is slightly more complicated than for the repetition code. (There are other decoding algorithms.)

Every (n, k) linear block code has associated with it a $(n-k) \times n$ matrix H called the *parity check matrix*, which has the property that

$$\mathbf{v}H^T = \mathbf{0} \text{ if and only if the vector } \mathbf{v} \text{ is a codeword.} \tag{1.36}$$

The parity check matrix is not unique. For the generator G of (1.34), the parity check matrix can be written as

$$H = \begin{bmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix}. \tag{1.37}$$

It can be verified that $GH^T = \mathbf{0}$.

The matrix H can be expressed in terms of its columns as

$$H = \begin{bmatrix}\mathbf{h}_1 & \mathbf{h}_2 & \mathbf{h}_3 & \mathbf{h}_4 & \mathbf{h}_5 & \mathbf{h}_6 & \mathbf{h}_7\end{bmatrix}.$$

It may be observed that the columns of H consist of the binary representations of the numbers 1 through $7 = n$, though not in numerical order. On the basis of this observation, we can generalize to other Hamming codes. Hamming codes of length $n = 2^m - 1$ and dimension $k = 2^m - m - 1$ exist for every $m \geq 2$, having parity check matrices whose columns are binary representations of the numbers from 1 through n.

1.9.1 Hard-Input Decoding Hamming Codes

Suppose that a codeword $\mathbf{c}$ is sent and the received vector is

$$\mathbf{r} = \mathbf{c} + \mathbf{n} \text{ (addition modulo 2).}$$

The first decoding step is to compute the *syndrome*

$$\mathbf{s} = \mathbf{r}H^T = (\mathbf{c} + \mathbf{n})H^T = \mathbf{n}H^T.$$

Because of property (1.36), the syndrome depends only on the error $\mathbf{n}$ and not on the transmitted codeword. The codeword information is "projected away."

Since a Hamming code is capable of correcting only a single error, suppose that $\mathbf{n}$ is all zeros except at a single position,

$$\mathbf{n} = [n_1, n_2, n_3, \ldots, n_7] = [0, \ldots, 0, 1, 0, \ldots, 0]$$

where the 1 is equal to n_i. (That is, the error is in the ith position.)

Let us write H^T in terms of its rows:

$$H^T = \begin{bmatrix}\mathbf{h}_1^T \\ \mathbf{h}_2^T \\ \vdots \\ \mathbf{h}_n^T\end{bmatrix}.$$

Then the syndrome is

$$\mathbf{s} = \mathbf{r}H^T = \mathbf{n}H^T = \begin{bmatrix}n_1 & n_2 & \ldots & n_n\end{bmatrix}\begin{bmatrix}\mathbf{h}_1^T \\ \mathbf{h}_2^T \\ \vdots \\ \mathbf{h}_n^T\end{bmatrix} = \mathbf{h}_i^T.$$

The error position i is the column i of H that is equal to the (transpose of) the syndrome $\mathbf{s}^T$.

Algorithm 1.1 Hamming Code Decoding

1. For the received binary vector $\mathbf{r}$, compute the syndrome $\mathbf{s} = \mathbf{r}H^T$. If $\mathbf{s} = \mathbf{0}$, then the decoded codeword is $\hat{\mathbf{c}} = \mathbf{r}$.
2. If $\mathbf{s} \neq \mathbf{0}$, then let i denote the column of H which is equal to $\mathbf{s}^T$. There is an error in position i of $\mathbf{r}$. The decoded codeword is $\hat{\mathbf{c}} = \mathbf{r} + \mathbf{n}_i$, where $\mathbf{n}_i$ is a vector which is all zeros except for a 1 in the ith position.

This decoding procedure fails if more than one error occurs.

Example 1.7 Suppose that the message

$$\mathbf{m} = [m_1, m_2, m_3, m_4] = [0, 1, 1, 0]$$

is encoded, resulting in the codeword

$$\mathbf{c} = [0, 1, 1, 0, 1, 0, 0] + [0, 0, 1, 1, 0, 1, 0] = [0, 1, 0, 1, 1, 1, 0].$$

When $\mathbf{c}$ is transmitted over a BSC, the vector

$$\mathbf{r} = [0, 1, 1, 1, 1, 1, 0]$$

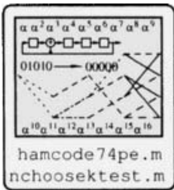
is received. The decoding algorithm proceeds as follows:

1. The syndrome $\mathbf{s} = [0, 1, 1, 1, 1, 1, 0]H^T = [1, 0, 1]$ is computed.
2. This syndrome corresponds to column 3 of H. The decoded value is therefore

$$\hat{\mathbf{c}} = \mathbf{r} + [0, 0, 1, 0, 0, 0, 0] = [0, 1, 0, 1, 1, 1, 0],$$

which is the transmitted codeword.

□



The expression for the probability of bit error is significantly more complicated for Hamming codes than for repetition codes. We defer on the details of these computations to the appropriate location (Section 3.7) and simply plot the results here. The available energy per encoded bit is

$$E_c = E_b(k/n) = 4/7E_b,$$

so, as for the repetition code, there is less energy available per bit. This represents a loss of $10\log_{10}(4/7) = -2.4$ dB of energy per transmitted bit compared to the uncoded system. Note, however, that the decrease in energy per bit is not as great as for the repetition code, since the rate is higher. Figure 1.19 shows the probability of bit error for uncoded channels (the solid line), and for the coded bits — that is, the bits coded with energy E_c per bit — (the dashed line). The figure also shows the probability of bit error for the bits *after* they have been through the decoder (the dash-dot line). In this case, the decoded bits *do* have a lower probability of error than the uncoded bits. For the uncoded system, to achieve a probability of error of $P_b = 10^{-6}$ requires an SNR of 10.5 dB, while for the coded system, the same probability of error is achieved with 10.05 dB. The code was able to overcome the 2.4 dB of loss due to rate, and add another 0.45 dB of improvement. We say that the *coding gain* of the system (operated near 10 dB) is 0.45 dB: we can achieve the same performance as a system expending 10.5 dB SNR per bit, but with only 10.05 dB of expended transmitter energy per bit.

Also shown in Figure 1.19 is the asymptotic (most accurate for large SNR) performance of soft-decision decoding. This is somewhat optimistic, being better performance than might be achieved in practice. But it does show the potential that soft-decision decoding has: it is significantly better than the hard-input decoding.

1.9.2 Other Representations of the Hamming Code

In the brief introduction to the Hamming code, we showed that the encoding and decoding operations have matrix representations. This is because Hamming codes are **linear block codes**, which will be explored in Chapter 3. There are other representations for Hamming and other codes. We briefly introduce these here as bait and lead-in to further chapters. As these representations show, descriptions of codes involve algebra, polynomials, graph theory, and algorithms on graphs, in addition to the linear algebra we have already seen.

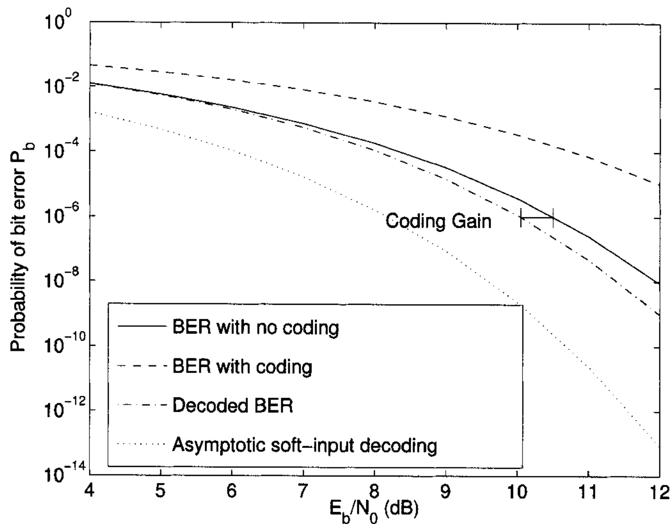


Figure 1.19: Performance of the (7, 4) Hamming code in the AWGN channel.

An Algebraic Representation

The columns of the parity check matrix H can be represented using special symbols. That is, we could write

$$H = \begin{bmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix}$$

as

$$H = \begin{bmatrix} \boldsymbol{\beta}_1 & \boldsymbol{\beta}_2 & \boldsymbol{\beta}_3 & \boldsymbol{\beta}_4 & \boldsymbol{\beta}_5 & \boldsymbol{\beta}_6 & \boldsymbol{\beta}_7 \end{bmatrix},$$

where each β_i represents a 3-tuple. Then the syndrome $\mathbf{s} = \mathbf{r}H^T$ can be represented as

$$\mathbf{s} = \sum_{i=1}^{7} r_i \boldsymbol{\beta}_i.$$

Then $\mathbf{s} = \boldsymbol{\beta}_j$ for some j, which indicates the column where the error occurred. This turns the decoding problem into a straightforward algebra problem.

A question we shall take up later is how to generalize this operation. That is, can codes be defined which are capable of correcting more than a single error, for which finding the errors can be computed using algebra? In order to explore this question, we will need to carefully define how to perform algebra on discrete objects (such as the columns of H) so that addition, subtraction, multiplication, and division are defined in a meaningful way. Such algebraic operations are defined in Chapters 2 and 5.

A Polynomial Representation

Examination of the codewords in (1.35) reveals an interesting fact: if $\mathbf{c}$ is a codeword, then so is every cyclic shift of $\mathbf{c}$. For example, the codeword [1, 1, 0, 1, 0, 0, 0] has the cyclic

shifts

$$[0,1,1,0,1,0,0], [0,0,1,1,0,1,0], [0,0,0,1,1,0,1], [1,0,0,0,1,1,0],$$
$$[0,1,0,0,0,1,1], [1,0,1,0,0,0,1],$$

which are also codewords. Codes for which all cyclic shifts of every codeword are also codewords are called **cyclic** codes. As we will find in Chapter 4, Hamming codes, like most block codes of modern interest, are cyclic codes. In addition to the representation using a generator matrix, cyclic codes can also be represented using polynomials. For the $(7,4)$ Hamming code, there is a *generator polynomial* $g(x) = x^3 + x + 1$ and a corresponding *parity-check polynomial* $h(x) = x^4+x^2+x+1$, which is a polynomial such that $h(x)g(x) = x^7 + 1$. The encoding operation can be represented using polynomial multiplication (with coefficient operations modulo 2). For this reason, the study of polynomial operations and the study of algebraic objects built out of polynomials is of great interest. The parity check polynomial can be used to check if a polynomial is a code polynomial: A polynomial $r(x)$ is a code polynomial if and only if $r(x)h(x)$ is a multiple of $x^7 + 1$. This provides a parity check condition: compute $r(x)h(x)$ modulo $x^7 + 1$. If this is not equal to 0, then $r(x)$ is not a code polynomial.

Example 1.8 The message $\mathbf{m} = [m_0, m_1, m_2, m_3] = [0, 1, 1, 0]$ can be represented as a polynomial as

$$m(x) = m_0 + m_1 x + m_2 x^2 + m_3 x^3 = 0 \cdot 1 + 1 \cdot x + 1 \cdot x^2 + 0 \cdot x^3 = x + x^2.$$

The code polynomial is obtained by $c(x) = m(x)g(x)$, or

$$\begin{aligned} c(x) &= (x + x^2)(1 + x + x^3) = (x + x^2 + x^4) + (x^2 + x^4 + x^5) \\ &= x + 2x^2 + x^3 + x^4 + x^5 = x + x^3 + x^4 + x^5, \end{aligned}$$

(where $2x^2 = 0$ modulo 2), which corresponds to the code vector $\mathbf{c} = [0, 1, 0, 1, 1, 1, 0]$. □

A Trellis Representation

As we will see in Chapter 12, there is a graph associated with a block code. This graph is called the *Wolf trellis* for the code. We shall see that paths through the graph correspond to vectors $\mathbf{v}$ that satisfy the parity check condition $\mathbf{v}H^T = \mathbf{0}$. For example, Figure 1.20 shows the trellis corresponding to the parity check matrix (1.37). The trellis states at the kth stage are obtained by taking all possible binary linear combinations of the first k columns of H. In Chapter 12, we will develop decoding a algorithm which essentially finds the best path through the graph. One such decoding algorithm is called the *Viterbi algorithm.* Such decoding algorithms will allow us to create soft-decision decoding algorithms for block codes.

The Viterbi algorithm is also used for decoding codes which are defined using graphs similar to that of Figure 1.20. Such codes are called *convolutional codes.*

The Tanner Graph Representation

Every linear block code also has another graph which represents it called the *Tanner graph.* For a parity check matrix, the Tanner graph has one node to represent each column of H (the "bit nodes") and one node to represent each row of H (the "check nodes"). Edges occur only between bit nodes and check nodes. There is an edge between a bit node and a check node if there is a 1 in the parity check matrix at the corresponding location. For example, for the parity check matrix of (1.37), the Tanner graph representation is shown in Figure 1.21.

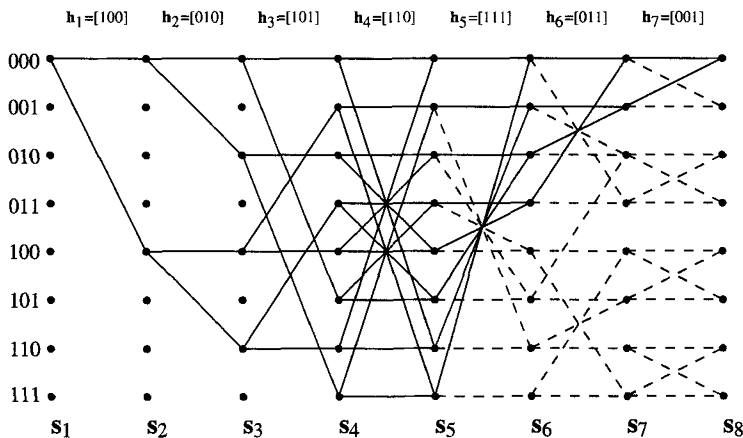


Figure 1.20: The trellis of a (7, 4) Hamming code.

Algorithms to be presented in Chapter 15 describe how to propagate information through the graph in order to perform decoding. These algorithms are usually associated with codes which are iteratively decoded, such as turbo codes and low-density parity-check codes. These modern families of codes have very good behavior, sometimes nearly approaching capacity.

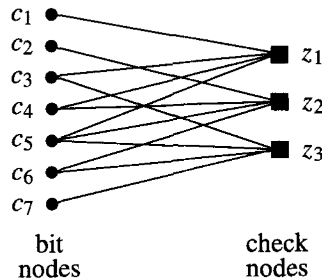


Figure 1.21: The Tanner graph for a (7, 4) Hamming code.

1.10 The Basic Questions

The two simple codes we have examined so far bring out issues relevant for the codes we will investigate:

1. How is the code described and represented?
2. How is encoding accomplished?
3. How is decoding accomplished? (This frequently takes some cleverness!)
4. How are codewords to be represented, encoded, and decoded, in a computationally tractable way?

5. What is the performance of the code? What are the properties of the code? (e.g., How many codewords? What are the weights of the codewords?)
6. Are there other families of codes which can provide better coding gains?
7. How can these codes be found and described?
8. Are there constraints on allowable values of n, k, and $d_{\min}$?
9. Is there some limit to the amount of coding gain possible?
10. For a given available SNR, is there a lower limit on the probability of error that can be achieved?

Questions of this nature shall be addressed throughout the remainder of this book, presenting the best answers available at this time.

1.11 Historical Milestones of Coding Theory

We present in Table 1.1 a brief summary of major accomplishments in coding theory and some of the significant contributors to that theory, or expositors who contributed by bringing together the significant contributions to date. Some dates and contributions may not be exactly as portrayed here; it is difficult to sift through the sands of recent history. Also, significant contributions to coding are made *every month*, so this cannot be a complete list.

1.12 A Bit of Information Theory

The channel coding theorem governs the ultimate limits of error correction codes. To understand what it implies, we need to introduce a little bit of information theory and state some results. However, it lies beyond the scope of the book to provide a full in-depth coverage.

1.12.1 Information Theoretic Definitions for Discrete Random Variables

Entropy and Conditional Entropy

We first present information-theoretic concepts for discrete random variables. Let X be a discrete random variable taking values in a set $\mathcal{A}_x = \{x_1, x_2, \ldots, x_m\}$ with probability $P(X = x_i) = p_i$. We have seen that the entropy is

$$H(X) = E[-\log_2 P(X)] = -\sum_{x \in \mathcal{A}_x} P(X = x) \log_2 P(X = x) \text{ (bits)}.$$

The entropy represents the uncertainty there is about X prior to its measurement; equivalently, it is the amount of information gained when X is measured.

Now suppose that $Y = f(X)$ for some probabilistic function $f(X)$. For example, Y might be the output of a noisy channel that has X as the input. Let $\mathcal{A}_y$ denote the set of possible Y outcomes. We define $H(X|y)$ as the uncertainty remaining about X when Y is measured as $Y = y$:

$$H(X|y) = E[-\log_2 P_{X|Y}(X|y)] = -\sum_{x \in \mathcal{A}_x} P_{X|Y}(x|y) \log_2 P_{X|Y}(x|y) \text{ (bits)}.$$

Table 1.1: Historical Milestones

Year	Milestone
1948	Shannon publishes "A Mathematical Theory of Communication" [309]
1950	Hamming describes Hamming codes [137]
1954	Reed [284] and Muller [248] both present Reed-Muller codes and their decoders
1955	Elias introduces convolutional codes [76]
1957	Prange introduces cyclic codes [271]
1959	A. Hocquenghem [151] and ...
1960	Bose and Ray-Chaudhuri [36] describe BCH codes
	Reed&Solomon produce eponymous codes [286]
	Peterson provides a solution to BCH decoding [261]
1961	Peterson produces his book [260], later extended and revised by Peterson and Weldon [262]
1962	Gallager introduces LDPC codes [112]
	2400 BPS modem commercially available (4-PSK) (see [100])
1963	The Fano algorithm for decoding convolutional codes introduced [80]
	Massey unifies the study of majority logic decoding [224]
1966	Forney produces an in-depth study of concatenated codes [87] and introduces generalized minimum distance decoding [88]
1967	Berlekamp introduces a fast algorithm for BCH/Reed-Solomon decoding [22]
	Rudolph initiates the study of finite geometries for coding [299]
	4800 BPS modem commercially available (8-PSK) (see [100])
1968	Berlekamp produces *Algebraic Coding Theory* [25]
	Gallager produces *Information theory and reliable communication* [111]
1969	Jelinek describes the stack algorithm for decoding convolutional codes [165]
	Massey introduces his algorithm for BCH decoding [222]
	Reed-Muller code flies on *Mariner* deep space probes using Green machine decoder
1971	Viterbi introduces the algorithm for ML decoding of convolutional codes [359]
	9600 BPS modem commercially available (16-QAM) (see [100])
1972	The BCJR algorithm is described in the open literature [10]
1973	Forney elucidates the Viterbi algorithm [89]
1975	Sugiyama et al. propose the use of the Euclidean algorithm for decoding [324]
1977	MacWilliams and Sloane produce the encyclopedic *The Theory of Error Correcting Codes* [220]
	Voyager deep space mission uses a concatenated RS/convolutional code (see [231])
1978	Wolf introduces a trellis description of block codes [377]
1980	14,400 BPS modem commercially available (64-QAM) (see [100])
	Sony and Phillips standardize the compact disc, including a shortened Reed-Solomon code
1981	Goppa introduces algebraic-geometry codes [123, 124]
1982	Ungerboeck describes trellis-coded modulation [345]
1983	Lin & Costello produce their engineering textbook [203]
	Blahut publishes his textbook [33]
1984	14,400 BPS TCM modem commercially available (128-TCM) (see [100])
1985	19,200 BPS TCM modem commercially available (160-TCM) (see [100])
1993	Berrou, Glavieux, and Thitimajshima announce turbo codes [28]
1994	The $\mathbb{Z}_4$ linearity of families of nonlinear codes is announced [138]
1995	MacKay resuscitates LDPC codes [218]
	Wicker publishes his textbook [373]
1996	33,600 BPS modem (V.34) modem is commercially available (see [98])
1998	Alamouti describes a space-time code [3]
1999	Guruswami and Sudan present a list decoder for RS and AG codes [128]
2000	Aji and McEliece [2] (and others [195]) synthesize several decoding algorithms using message passing ideas
2002	Hanzo, Liew, and Yeap characterize turbo algorithms in [141]
2003	Koetter and Vardy extend the GS algorithm for soft-decision decoding of RS codes [191]
2004	Lin&Costello second edition [204]
2005	Moon produces what is hoped to be a valuable book!

Then the average uncertainty in X, averaged over the outcomes Y, is called the *conditional entropy*, $H(X|Y)$, computed as

$$H(X|Y) = \sum_{y \in \mathcal{A}_y} H(X|y) P_Y(y) = -\sum_{y \in \mathcal{A}_y} \sum_{x \in \mathcal{A}_x} P_{X|Y}(x|y) P_Y(y) \log_2 P_{X|Y}(x|y)$$

$$= -\sum_{y \in \mathcal{A}_y} \sum_{x \in \mathcal{A}_x} P_{X,Y}(x, y) \log_2 P_{X|Y}(x|y) \text{ (bits).}$$

Relative Entropy, Mutual Information, and Channel Capacity

Definition 1.5 An important information-theoretic quantity is the **Kullback-Leibler distance** $D(P||Q)$ between two probability mass functions, also known as the relative entropy or the cross entropy. Let $P(X)$ and $Q(X)$ be two probability mass functions on the

outcomes in $\mathcal{A}_x$. We define

$$D(P||Q) = E_P\left[\log\frac{P(X)}{Q(X)}\right] = \sum_{x\in\mathcal{A}_x} P(X)\log\frac{P(x)}{Q(x)}.$$

□

Lemma 1.2 $D(P||Q) \geq 0$, *with equality if and only if* $P = Q$; *that is, if the two distributions are the same.*

Proof We use the inequality $\log x \leq x - 1$, with equality only at $x = 1$. This inequality appears so frequently in information theory it has been termed the *information theory inequality*. Then

$$\begin{aligned} D(P||Q) &= \sum_{x\in\mathcal{A}_x} P(x)\log\frac{P(x)}{Q(x)} = -\sum_{x\in\mathcal{A}_x} P(x)\log\frac{Q(x)}{P(x)} \\ &\geq \sum_{x\in\mathcal{A}_x} P(x)\left[1-\frac{Q(x)}{P(x)}\right] \qquad \text{(information theory inequality)} \\ &= \sum_{x\in\mathcal{A}_x} P(x) - Q(x) = 0. \end{aligned}$$

□

Definition 1.6 The **mutual information** between a random variable X and Y is the Kullback-Leibler distance between the joint distribution $P(X, Y)$ and the product of the marginals $P(X)P(Y)$:

$$I(X;Y) = D(P(X,Y)||P(X)P(Y)). \tag{1.38}$$

□

If X and Y are independent, so that $P(X, Y) = P(X)P(Y)$, then $I(X; Y) = 0$. That is, Y tells no information at all about X.

Using the definitions, it is straightforward to show that the mutual information can also be written as

$$I(X;Y) = H(X) - H(X|Y).$$

The mutual information is the difference between the average uncertainty in X and the uncertainty in X there still is after measuring Y. Thus, it quantifies how much information Y tells about X. Since the definition (1.38) is symmetric, we also have

$$I(X;Y) = H(Y) - H(Y|X).$$

In light of Lemma 1.2, we see that mutual information $I(X; Y)$ can never be negative.

With the definition of the mutual information, we can now define the channel capacity.

Definition 1.7 The **channel capacity** C of a channel with input X and output Y is defined as the maximum mutual information between X and Y, where the maximum is taken over all possible input distributions.

$$C = \max_{p_X(x)} I(X;Y).$$

For the BSC with crossover probability p, it is straightforward to show (see Exercise 1.31) that the capacity is

$$\boxed{C = 1 - H_2(p).}$$

1.12.2 Information Theoretic Definitions for Continuous Random Variables

Let Y be a continuous random variable taking on values in an (uncountable) set $\mathcal{A}_y$, with pdf $p_Y(y)$. The differential entropy is defined as

$$H(Y) = -E[\log_2 p_Y(y)] = -\int_{\mathcal{A}_y} p_Y(y) \log_2 p_Y(y)\, dy.$$

Whereas entropy for discrete random variables is always nonnegative, differential entropy (for a continuous random variable) can be positive or negative.

Example 1.9 Let $Y \sim \mathcal{N}(0, \sigma^2)$. Then

$$\begin{aligned}
H(Y) &= -E\left[\log_2 \frac{1}{\sqrt{2\pi}\sigma} e^{-Y^2/2\sigma^2}\right] = -E\left[\log_2 \frac{1}{\sqrt{2\pi}\sigma} + \log_2(e)(-\frac{1}{2\sigma^2})(Y^2)\right] \\
&= \log_2(e)\frac{1}{2\sigma^2}E[Y^2] + \frac{1}{2}\log_2 2\pi\sigma^2 \\
&= \frac{1}{2}\log_2(e) + \frac{1}{2}\log_2 2\pi\sigma^2 = \frac{1}{2}\log_2 2\pi e\sigma^2 \text{ (bits).}
\end{aligned}$$

□

It can be shown that, for a continuous random variable with mean 0 and variance σ^2, the Gaussian $\mathcal{N}(0, \sigma^2)$ has the largest differential entropy.

Let X be a discrete-valued random variable taking on values in the alphabet $\mathcal{A}_x$ with probability $\Pr(X = x) = P_X(x), x \in \mathcal{A}_x$ and let X be passed through a channel which produces a continuous-valued output Y for $Y \in \mathcal{A}_y$. A typical example of this is the additive white Gaussian noise channel, where

$$Y = X + N,$$

and $N \sim \mathcal{N}(0, \sigma^2)$. Let

$$p_{XY}(x, y) = p_{Y|X}(y|x)P_X(x)$$

denote the joint distribution of X and Y and let

$$p_Y(y) = \sum_{x\in\mathcal{A}_x} p_{XY}(x, y) = \sum_{x\in\mathcal{A}_x} p_{XY}(y|x)P_X(x)$$

denote the pdf of Y. Then the mutual information $I(X; Y)$ is computed as

$$\begin{aligned}
I(X; Y) = D(p_{XY}(X, Y)||P_X(X)p_Y(Y)) &= \int_{\mathcal{A}_y} \sum_{\mathcal{A}_x} p_{XY}(x, y) \log_2 \frac{p_{XY}(x, y)}{p_Y(y)P_X(x)}\, dy \\
&= \sum_{x\in\mathcal{A}_x} \int_{y\in\mathcal{A}_y} p_{Y|X}(y|x)P_X(x) \log_2 \frac{p_{Y|X}(y|x)}{\sum_{x'\in\mathcal{A}_x} p(y|x')P_X(x')}\, dy.
\end{aligned}$$

Example 1.10 Suppose $\mathcal{A}_x = \{a, -a\}$ (e.g., BPSK modulation with amplitude a) with probabilities $P(X = a) = P(X = -a) = \frac{1}{2}$. Let $N \sim \mathcal{N}(0, \sigma^2)$ and let

$$Y = X + N.$$

Because the channel has only binary inputs, this is referred to as the *binary* additive white Gaussian noise channel (BAWGNC). Then

$$\begin{aligned} I(X;Y) &= \frac{1}{2}\left[\int_{-\infty}^{\infty} p(y|a)\log_2 \frac{p(y|a)}{\frac{1}{2}(p(y|a)+p(y|-a))} + p(y|-a)\log_2 \frac{p(y|-a)}{\frac{1}{2}(p(y|a)+p(y|-a))}\,dy\right] \\ &= \frac{1}{2}\int_{-\infty}^{\infty} p(y|a)\log_2 p(y|a) + p(y|-a)\log_2 p(y|-a) \qquad (1.39) \\ &\qquad - (p(y|a)+p(y|-a))\log_2\left[\frac{1}{2}(p(y|a)+p(y|-a))\right] dy \\ &= \frac{1}{2}\left[-H(Y) - H(Y) - \int_{-\infty}^{\infty} (p(y|a)+p(y|-a))\log_2\left[\frac{1}{2}(p(y|a)+p(y|-a))\right] dy\right] \\ &= \boxed{-\int_{-\infty}^{\infty} \phi(y, E_b, \sigma^2)\log_2 \phi(y, E_b, \sigma^2)\,dy - \frac{1}{2}\log_2 2\pi e\sigma^2 \text{ (bits)},} \qquad (1.40) \end{aligned}$$

where we define the function

$$\phi(y, a, \sigma^2) = \frac{1}{\sqrt{8\pi\sigma^2}}\left[e^{-(y-a)^2/2\sigma^2} + e^{-(y+a)^2/2\sigma^2}\right].$$

□

When both the channel input X and the output Y are continuous random variables, then the mutual information is

$$I(X;Y) = D(p_{XY}(X,Y)\|p_X(x)p_Y(Y)) = \int_{\mathcal{A}_x}\int_{\mathcal{A}_y} p_{XY}(x,y)\log_2 \frac{p_{XY}(x,y)}{p_Y(y)p_X(x)}\,dydx.$$

Example 1.11 Let $X \sim \mathcal{N}(0, \sigma_x^2)$ and $N \sim \mathcal{N}(0, \sigma_n^2)$, independent of X. Let $Y = X + N$. Then $Y \sim \mathcal{N}(0, \sigma_x^2 + \sigma_n^2)$.

$$\begin{aligned} I(X;Y) &= H(Y) - H(Y|X) = H(Y) - H(X+N|X) = H(Y) - H(N|X) = H(Y) - H(N) \\ &= \frac{1}{2}\log_2 2\pi e\sigma_y^2 - \frac{1}{2}\log_2 2\pi e\sigma_n^2 \\ &= \boxed{\frac{1}{2}\log_2\left(1 + \frac{\sigma_x^2}{\sigma_n^2}\right) \text{ (bits).}} \qquad (1.41) \end{aligned}$$

The quantity σ_x^2 represents the average power in the transmitted signal X and σ_n^2 represents the average power in the noise signal N. This channel is called the additive white Gaussian noise channel (AWGNC). □

As for the discrete channel, the channel capacity C of a channel with input X and output Y is the maximum mutual information between X and Y, where the maximum is over all input distributions. In Example 1.10, the maximizing distribution is, in fact, the uniform distribution, $P(X = a) = P(X = -a) = \frac{1}{2}$, so (1.40) is the capacity for the BAWGNC. In Example 1.11, the maximizing distribution is, in fact, the Gaussian distribution (since this maximizes the entropy of the output), so (1.41) is the capacity for the AWGNC.

1.12.3 The Channel Coding Theorem

The channel capacity has been *defined* as the maximum mutual information between the input and the output. But Shannon's the channel coding theorem, tells us what the capacity *means*. Recall that an error correction code has a rate $R = k/n$, where k is the number of input symbols and n is the number of output symbols, the length of the code. The channel coding theorem says this:

> Provided that the coded rate of transmission R is less than the channel capacity, for any given probability of error ϵ specified, there is an error correction code of length n_0 such that there exist codes of length n exceeding n_0 for which the decoded probability of error is less than ϵ.

That is, provided that we transmit at a rate less than capacity, arbitrarily low probabilities of error can be obtained, if a sufficiently long error correction code is employed. The capacity is thus the amount of information that can be transmitted *reliably* through the channel *per channel use*.

A converse to the channel coding theorem states that for a channel with capacity C, if $R > C$, then the probability of error is bounded away from zero: reliable transmission is not possible.

The channel coding theorem is an *existence* theorem; it tells us that codes exist that can be used for reliable transmission, but not how to find practical codes. Shannon's remarkable proof used random codes. But as the code gets long, the decoding complexity of a truly random (unstructured) code goes up exponentially with the length of the code. Since Shannon's proof, engineers and mathematicians have been looking for ways of constructing codes that are both good (meaning they can correct a lot of errors) and practical, meaning that they have some kind of structure that makes decoding of sufficiently low complexity that decoders can be practically constructed.

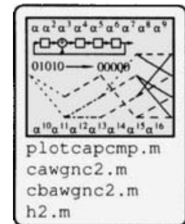


Figure 1.22 shows a comparison of the capacity of the AWGNC and the BAWGNC channels as a function of E_c/σ^2 (an SNR measure). In this figure, we observe that the capacity of the AWGNC increases with SNR beyond one bit per channel use, while the BAWGNC asymptotes to a maximum of one bit per channel use — if only binary data is put into the channel, only one bit of useful information can be obtained. It is always the case that

$$C_{\text{AWGNC}} > C_{\text{BAWGNC}}.$$

Over all possible input distributions, the Gaussian distribution is information maximizing, so C_{AWGNC} is an upper bound on capacity for any modulation or coding that might be employed. However, at very low SNRs, C_{AWGNC} and C_{BAWGNC} are very nearly equal.

Figure 1.22 also shows the capacity of the equivalent BSC, with crossover probability $p = Q(\sqrt{E_c/\sigma^2})$ and capacity $C_{\text{BSC}} = 1 - H_2(p)$. This corresponds to hard-input decoding. Clearly, there is some loss of potential rate due to hard-input decoding, although the loss diminishes as the SNR increases.

1.12.4 "Proof" of the Channel Coding Theorem

In this section we present a "proof" of the channel coding theorem. While mathematically accurate, it is not complete. The arguments can be considerably tightened, but are sufficient to show the main ideas of coding. Also, the proof is only presented for the discrete

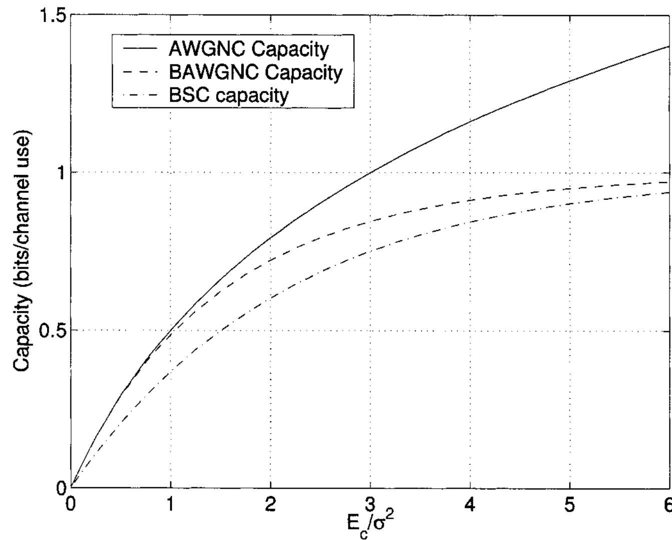


Figure 1.22: Capacities of AWGNC, BAWGNC, and BSC.

input/discrete channel case. The intuition, however, generally carries over to the Gaussian channel.

An important preliminary concept is the "asymptotic equipartition property" (AEP). Let X be a random variable taking values in a set $\mathcal{A}_x$. Let $\mathbf{X} = (X_1, X_2, \ldots, X_n)$ be an i.i.d. (independent, identically distributed) random vector and let $\mathbf{x}$ denote an outcome of $\mathbf{X}$.

Theorem 1.3 *(AEP) As $n \to \infty$, there is a set of "typical" outcomes $\mathcal{T}$ for which*

$$P(\mathbf{X} = \mathbf{x}) \approx 2^{-nH(X)}, \quad \mathbf{x} \in \mathcal{T}, \tag{1.42}$$

By the AEP, most of the probability is "concentrated" in the typical set. That is, a "typical" outcome is likely to occur, while an outcome which is not "typical" is not likely to occur. Since the "typical" outcomes all have approximately the same probability (1.42), there must be approximately $2^{nH(X)}$ outcomes in the typical set $\mathcal{T}$.[8]

Proof We sketch the main idea of the proof. Let the outcome space for a random variable Y be $\mathcal{A}_y = \{b_1, b_2, \ldots, b_K\}$, occurring with probabilities $P_i = P(Y = b_i)$. Out of n samples of the i.i.d. variable Y, let n_i be the number of outcomes that are equal to b_i. By the law of large numbers,[9] when n is large,

$$\frac{n_i}{n} \approx P_i. \tag{1.43}$$

[8]This observation is the basis for lossless data compression occurring in a source coder.

[9]Thorough proof of the AEP merely requires putting all of the discussion in the formal language of the weak law of large numbers.

The product of n observations can be written as

$$\begin{aligned} y_1 y_2 \cdots y_n &= b_1^{n_1} b_2^{n_2} \cdots b_K^{n_K} = \left[b_1^{(n_1/n)} b_2^{(n_2/n)} \cdots b_K^{(n_K/n)}\right]^n \\ &= \left[2^{\frac{n_1}{n}\log_2 b_1} 2^{\frac{n_2}{n}\log_2 b_2} \cdots 2^{\frac{n_K}{n}\log_2 b_K}\right]^n = \left[2^{\sum_{i=1}^K \frac{n_i}{n}\log_2 b_i}\right]^n \\ &\approx \left[2^{\sum_{i=1}^K P_i \log_2 b_i}\right]^n \qquad \text{(by (1.43))} \\ &= \left[2^{E[\log_2 Y]}\right]^n. \end{aligned} \tag{1.44}$$

Now suppose that Y is, in fact, a function of a random variable X, $Y = f(X)$. In particular, suppose $f(x) = p_X(x) = P(X = x)$. Then by (1.44),

$$y_1 y_2 \cdots y_n = f(x_1) f(x_2) \cdots f(x_n) = \prod_{i=1}^n p_X(x_i) \approx \left[2^{E[\log_2 p_X(X)]}\right]^n = 2^{-nH(X)}.$$

This establishes (1.42). □

Let X be a binary source with entropy $H(X)$ and let each X be transmitted through a memoryless channel to produce the output Y. Consider transmitting the sequence of i.i.d. outcomes $x_1, x_2, \ldots, x_n$. While the number of possible sequences is $M = 2^n$, the typical set has only about $2^{nH(X)}$ sequences in it. Let the total possible number of output sequences $\mathbf{y} = y_1, y_2, \ldots, y_n$ be N. There are about $2^{nH(Y)} \leq N$ typical output sequences. For each typical output sequence $\mathbf{y}$ there are approximately $2^{nH(X|Y)}$ input sequences that could have caused it. Furthermore, each input sequence $\mathbf{x}$ typically could produce $2^{nH(Y|X)}$ output sequences. This is summarized in Figure 1.23(a).

Now let X be coded by a rate-R code to produce a coded sequence which selects, out of the 2^n possible input sequences, only 2^{nR} of these. In Figure 1.23(b), these coded sequences are denoted with filled squares, ■. The mapping which selects the 2^{nR} points is the *code*. Rather than select any particular code, we contemplate using all possible codes *at random* (using, however, only the typical sequences). Under the random code, a sequence selected at random is a codeword with probability

$$\frac{2^{nR}}{2^{nH(X)}} = 2^{n(R-H(X))}.$$

Now consider the problem of correct decoding. A sequence $\mathbf{y}$ is observed. It can be decoded correctly if there is only one code vector $\mathbf{x}$ that could have caused it. From Figure 1.23(b), the probability that none of the points in the "fan" leading to $\mathbf{y}$ other than the original code point is a message is

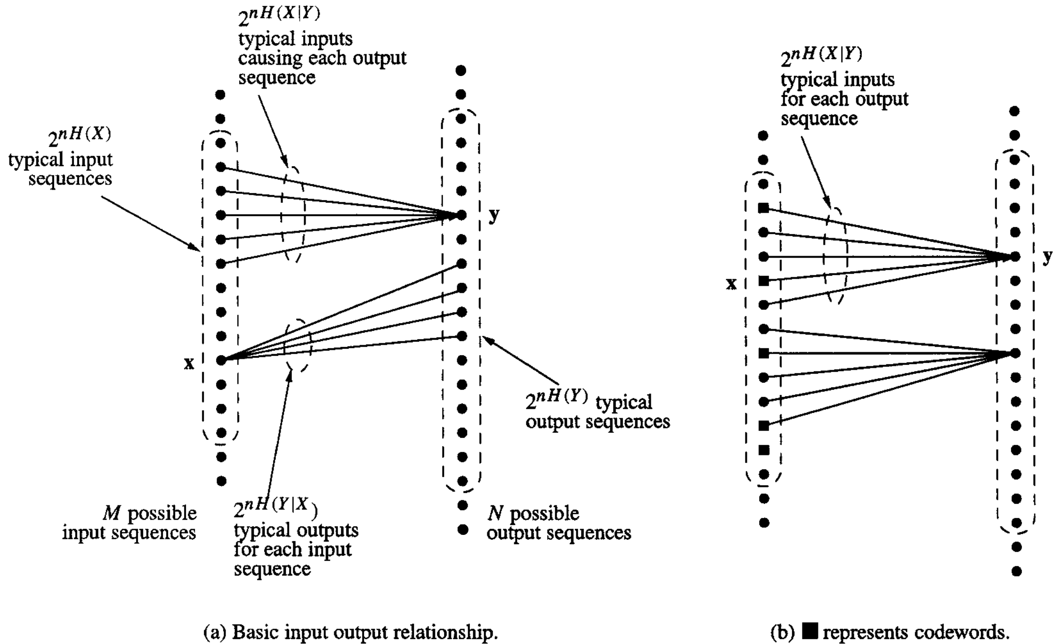
$$\begin{aligned} P &= (\text{probability a point } \mathbf{x} \text{ is not a codeword})^{(\text{typical number of inputs for this } \mathbf{y})} \\ &= (1 - 2^{n(R-H(X))})^{2^{nH(X|Y)}}. \end{aligned}$$

If we now choose $R < \max_{P_X(x)} H(X) - H(X|Y)$, that is, choose $R <$ the capacity C, then

$$R - H(X) + H(X|Y) < 0$$

for any input distribution $P_X(x)$. In this case,

$$R - H(X) = -H(X|Y) - \eta$$



(a) Basic input output relationship. (b) ■ represents codewords.

Figure 1.23: Relationship between input and output entropies for a channel. Each • or ■ represents a sequence.

for some $\eta > 0$. Then

$$P = (1 - 2^{n(-H(X|Y)-\eta)})^{2^{nH(X|Y)}}.$$

Expanding out using the binomial expansion,

$$P = 1 - 2^{nH(X|Y)} 2^{n(-H(X|Y)-\eta)} + \text{higher order terms},$$

so as $n \to \infty$,

$$P \to 1 - 2^{-n\eta} \to 1.$$

Thus the probability that none of the points except the original code point leading to **y** is a codeword approaches 1, so that the probability of decoding error — due to multiple codewords mapping to a single received vector — approaches 0.

We remark that if the *average* of an ensemble approaches zero, then there are elements in the ensemble that must approach 0. Thus there are codes (not randomly selected) for which the probability of error approaches zero as $n \to \infty$.

There are two other ways of viewing the coding rate requirement. The $2^{nH(Y|X)}$ typical sequences resulting from transmitting a vector **x** must partition the $2^{nH(Y)}$ typical output sequences, so that each observed output sequence can be attributed to a unique input sequence. The number of subsets in this partition is

$$\frac{2^{nH(Y)}}{2^{nH(Y|X)}} = 2^{n(H(Y)-H(Y|X))},$$

so the condition $R < H(Y) - H(Y|X)$ must be enforced. Alternatively, the $2^{nH(X)}$ typical input sequences must be partitioned so that the $2^{nH(X|Y)}$ typical input sequences associated with an observation $\mathbf{y}$ are disjoint. There must be $2^{n(H(X)-H(X|Y))}$ distinct subsets, so again the condition $R < H(X) - H(X|Y)$ must be enforced.

Let us summarize what we learn from the proof of the channel coding theorem:

- As long as $R < C$, arbitrarily reliable transmission is possible.
- The code lengths, however, may have to be long to achieve the desired reliability. The closer R is to C, the larger we would expect n to need to be in order to obtain some specified level of performance.
- Since the theorem was based on ensembles of random codes, it does not specify what the best code should be. We don't know how to "design" the best codes, we only know that they exist.
- However, random codes have a high probability of being good. So we are likely to get a good code simply by picking one at random!

So what, then, is the issue? Why the need for decades of research in coding theory, if a code can simply be selected at random? The answer has to do with the complexity of representing and *decoding* the code. To represent a random code of length n, there must be memory to store all the codewords, which requires $n2^{Rn}$ bits. Furthermore, to decode a received word $\mathbf{y}$, ML decoding for a random code requires that a received vector $\mathbf{y}$ must be compared with all 2^{Rn} possible codewords. For a $R = 1/2$ code with $n = 1000$ (a relatively modest code length and a low-rate code), 2^{500} comparisons must be made for each received vector. This is prohibitively expensive, beyond practical feasibility for even massively parallel computing systems, let alone a portable communication device.

Ideally, we would like to explore the space of codes parameterized by rate, probability of decoding error, block length (which governs latency), and encoding and decoding complexity, identifying thereby all achievable tuples of $(R, P, n, \chi_E, \chi_D)$, where P is the probability of error and χ_E and χ_D are the encoding and decoding complexities. This is an overwhelmingly complex task. The essence of coding research has taken the pragmatic stance of identifying families of codes which have some kind of algebraic or graphical *structure* that will enable representation and decoding with manageable complexity. In some cases what is sought are codes in which the encoding and decoding can be accomplished readily using algebraic methods — essentially so that decoding can be accomplished by solving sets of equations. In other cases, codes employ constraints on certain graphs to reduce the encoding and decoding complexity. Most recently, families of codes have been found for which very long block lengths can be effectively obtained with low complexity using very sparse representations, which keep the decoding complexity in check. Describing these codes and their decoding algorithms is the purpose of this book.

The end result of the decades of research in coding is that the designer has a rich palette of code options, with varying degrees of rate and encode and decode complexity. This book presents many of the major themes that have emerged from this research.

1.12.5 Capacity for the Continuous-Time AWGN Channel

Let X_i be a zero-mean random variable with $E[X_i^2] = \sigma_x^2$ which is input to a discrete AWGN channel, so that

$$R_i = X_i + N_i,$$

where the N_i are i.i.d. $N_i \sim \mathcal{N}(0, \sigma_n^2)$. The capacity of this channel is

$$C = \frac{1}{2}\log_2(1 + \frac{\sigma_x^2}{\sigma_n^2}) \text{ bits/channel use.}$$

Now consider sending a continuous-time signal $x(t)$ according to

$$x(t) = \sum_{i=1}^{n} X_i \varphi_i(t),$$

where the $\varphi_i(t)$ functions are orthonormal over $[0, T]$. Let us suppose that the transmitter power available is P watts, so that the energy dissipated in T seconds is $E = PT$. This energy is also expressed as

$$E = \int_0^T x^2(t)\,dt = \sum_{i=1}^{n} X_i^2.$$

We must therefore have

$$\sum_{i=1}^{n} X_i^2 = PT$$

or $nE[X_i^2] = PT$, so that $\sigma_x^2 = PT/n$.

Now consider transmitting a signal $x(t)$ through a continuous-time channel with bandwidth W. By the sampling theorem (frequently attributed to Nyquist, but in this context it is frequently called *Shannon's* sampling theorem), a signal of bandwidth W can be exactly characterized by $2W$ samples/second — any more samples than this cannot convey any more information about this bandlimited signal. So we can get $2W$ independent channel uses per second over this bandlimited channel. There are $n = 2WT$ symbols transmitted over T seconds.

If the received signal is

$$R(t) = x(t) + N(t)$$

where $N(t)$ is a white Gaussian noise random process with two-sided power spectral density $N_0/2$, then in the discrete-time sample

$$R_i = x_i + N_i$$

where $R_i = \int_0^T R(t)\varphi_i(t)\,dt$, the variance of N_i is $\sigma^2 = N_0/2$. The capacity for this bandlimited channel is

$$\begin{aligned} C &= \left(\frac{1}{2}\log_2\left(1 + \frac{PT/n}{N_0/2}\right) \text{ bits/channel use}\right)(2W \text{ channel uses/second}) \\ &= W\log_2\left(1 + \frac{2PT}{nN_0}\right) \text{ bits/second.} \end{aligned}$$

Now using $n = 2WT$ we obtain

$$C = W\log_2(1 + P/N_0W) \text{ bits/second.} \tag{1.45}$$

Since P is the average transmitted power, in terms of its units we have

$$P = \frac{\text{energy}}{\text{second}} = (\text{energy/bit})(\text{ bits/second}).$$

Since E_b is the energy/bit and the capacity is the rate of transmission in bits per second, we have $P = E_b C$, giving

$$C = W \log_2 \left(1 + \frac{C}{W}\frac{E_b}{N_0}\right). \tag{1.46}$$

Let $\eta = C/W$ be the spectral efficiency in bits/second/Hz; this is the data rate available for each Hertz of channel bandwidth. From (1.46),

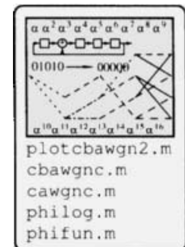
$$\eta = \log_2(1 + \eta \frac{E_b}{N_0})$$

or

$$E_b/N_0 = \frac{2^\eta - 1}{\eta}. \tag{1.47}$$

For BPSK the spectral efficiency is $\eta = 1$ bit/second/Hz, so (1.47) indicates that it is theoretically possible to transmit arbitrarily reliably at $E_b/N_0 = 1$, which is 0 dB. In principle, then, it should be possible to devise a coding scheme which could transmit BPSK-modulated signals arbitrarily reliably at an SNR of 0 dB. By contrast, for uncoded transmission when $E_b/N_0 = 9.6$ dB the BPSK performance shown in Figure 1.10 has $P_b = 10^{-5}$. There is at least 9.6 dB of coding gain possible. The approximately 0.44 dB of gain provided by the (7,4) Hamming code of Section 1.9 falls over 9 dB short of what is theoretically possible!

1.12.6 Transmission at Capacity with Errors



By the channel coding theorem, zero probability of error is attainable provided that the transmission rate is less than the capacity. What if we allow a non-vanishing probability of error. What is the maximum rate of transmission? Or, equivalently, for a given rate, which is the minimum SNR that will allow transmission at that rate, with a specified probability of error?

The theoretical tools we need to address these questions are the separation theorem and rate-distortion theory. The separation theorem says that we can consider separately and optimally (at least, asymptotically) data compression and error correction. Suppose that the source has a rate of r bits/second. First compress the information so that the bits of the compressed signal match the bits of the source signal with probability p. From rate distortion theory, this produces a source at rate $1 - H_2(p)$ per source bit (see (1.2)). These compressed bits, at a rate $r(1 - H_2(p))$ are then transmitted over the channel with vanishingly small probability of error. We must therefore have $r(1 - H_2(p)) < C$. The maximum rate achievable with average distortion (i.e., probability of bit error) p, which we denote as $C^{(p)}$ is therefore

$$C^{(p)} = \frac{C}{1 - H_2(p)}.$$

Figure 1.24 shows the required SNR E_b/N_0 for transmission at various rates for both the BAWGNC and the AWGNC. For any given line in the plot, the region to the right of the plot is achievable — it should theoretically be possible to transmit at that probability of error at that SNR. Curves such as these therefore represent a goal to be achieved by a particular code: we say that we are transmitting at capacity if the performance falls on the curve.

We note the following from the plot:

- At very low SNR, the binary channel and the AWGN channel have very similar performance. This was also observed in conjunction with Figure 1.22.

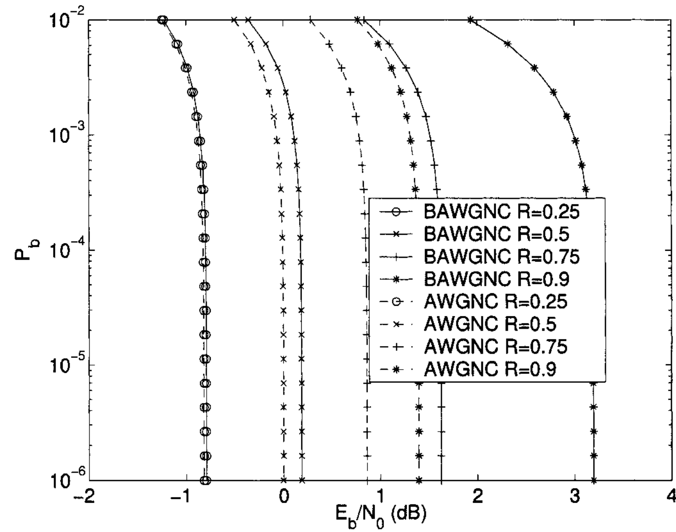


Figure 1.24: Capacity lower bounds on P_b as a function of SNR.

- The higher the rate, the higher the required SNR.
- The vertical asymptote (as $P_b \to 0$) is the capacity C for that channel.

1.12.7 The Implication of the Channel Coding Theorem

The implication of the channel coding theorem, fundamentally, is that for a block code of length n and rate $R = k/n$, the probability of a block decoding error can be bounded as

$$P(E) \leq 2^{-nE_b(R)}, \tag{1.48}$$

where $E_b(R)$ is a positive function of R for $R < C$. Work on a class of codes known as convolutional codes — to be introduced in Chapter 12 has shown (see, e.g., [357]) that

$$P(E) \leq 2^{(m+1)nE_c(R)}, \tag{1.49}$$

where m is the memory of the code and $E_c(R)$ is positive for $R < C$. The problem, as we shall see (and what makes coding such a fascinating topic) is that, in the absence of some kind of structure, as either n or m grow, the complexity can grow exponentially.

Programming Laboratory 1: Simulating a Communications Channel

Objective

In this lab, you will simulate a BPSK communication system and a coded system with a Hamming code employing hard-input decoding rules.

Background

Reading: Sections 1.5, 1.7, 1.9.

In the case of BPSK, an exact expression for the probability of error is available, (1.25). However, in many more interesting communication systems, a closed form expression for the probability of error is not available or is difficult to compute. Results must be therefore obtained by simulation of the system.

One of the great strengths of the signal-space viewpoint is that probability of error simulations can be made based only on points in the signal space. In other words, it suffices to simulate random variables as in the matched filter output (1.12), rather than creating the continuous-time functions as in (1.10). (However, for other kinds of questions, a simulation of the continuous-time function might be necessary. For example, if you are simulating the effect of synchronization, timing jitter, delay, or fading, simulating the time signal is probably necessary.)

A framework for simulating a communication system from the signal space point of view for the purpose of computing the probability of error is as follows:

Algorithm 1.2 Outline for simulating digital communications

1 Initialization: Store the points in the signal constellation.
  Fix E_b (typically $E_b = 1$).
2 FOR each signal-to-noise ratio $\gamma = E_b/N_0$:
3   Compute $N_0 = E_b/\gamma$ and $\sigma^2 = N_0/2$.
4   DO:
5   Generate some random bit(s) (the "transmitted" bits)
      according to the bit probabilities
6   Map the bit(s) into the signal constellation
      (e.g.,BPSK or 8-PSK) to create signal **s**
7   Generate a Gaussian random vector **n** (noise) with
      variance $\sigma^2 = N_0/2$ in each signal direction.
8   Add the noise to the signal to create the matched filter output
      signal $\mathbf{r} = \mathbf{s} + \mathbf{n}$.
9   Perform a detection on the symbol
      (e.g., find closest point in signal constellation to **r**)
10   From the detected symbol, determine the detected bits
11   Compare detected bits with the transmitted bits
12   Accumulate the number of bits in error
13 UNTIL at least N bit errors have been counted.
14 The estimated probability of error at this SNR is
    $P_e \approx \frac{\text{number of errors counted}}{\text{number of bits generated}}$
15 End FOR

As a general rule, the more errors N you count, the smaller will be the variance of your estimate of the probability of error. However, the bigger N is, the longer the simulation will take to run. For example, if the probability of error is near 10^{-6} at some particular value of SNR, around one million bits must be generated before you can expect an error. If you choose $N = 100$, then 100 million bits must be generated to estimate the probability of error, for just that *one* point on the plot!

Use of Coding in Conjunction with the BSC

For an (n, k) code having rate $R = k/n$ transmitted with energy per bit equal to E_b, the energy per coded bit is $E_c = E_b R$. It is convenient to fix the coded energy per bit in the simulation. To simulate the BSC channel with coding, the following outline can be used.

Algorithm 1.3 Outline for simulating (n, k)-coded digital communications

1 Initialization: Store the points in the signal constellation.
  Fix E_c (typically $E_c = 1$). Compute R.
2 FOR each signal-to-noise ratio $\gamma = E_b/N_0$:
3   Compute $N_0 = E_c/(R\gamma)$ and $\sigma^2 = N_0/2$.
4   Compute the BSC crossover probability $p = Q(\sqrt{2E_c/N_0})$.
5   DO:
6   Generate a block of k "transmitted" input bits
      and accumulate the number of bits generated
7   Encode the input bits to n codeword bits
8   Pass the n bits through the BSC
      (flip each bit with probability p)
9   Run the n bits through the decoder to produce k output bits
10   Compare the decoded output bits with the input bits
11   Accumulate the number of bits in error
12 UNTIL at least N bit errors have been counted.
13 The estimated probability of error is
    $P_e \approx \frac{\text{number of errors counted}}{\text{number of bits generated}}$
14 End FOR

The encoding and decoding operations depend on the kind of code used. In this lab, you will use codes which are among the simplest possible, the Hamming codes.

Since for linear codes the codeword is irrelevant, the simulation can be somewhat simplified by assuming that the input bits are all zero, so that the codeword is also all zero. For the Hamming code, the simulation can be arranged as follows:

Algorithm 1.4 Outline for simulating (n, k) Hamming-coded digital communications

1 Fix E_c (typically $E_c = 1$). Compute R.
2 FOR each signal-to-noise ratio $\gamma = E_b/N_0$:
3 Compute $N_0 = E_c/(R\gamma)$ and $\sigma^2 = N_0/2$.
4 Compute the BSC crossover probability $p = Q(\sqrt{2E_c/N_0})$.
5 DO:
6 Generate $\mathbf{r}$ as a vector of n random bits which are 1 with probability p
7 Increment the number of bits generated by k.
8 Compute the syndrome $\mathbf{s} = \mathbf{r}H^T$.
9 If $\mathbf{s} \neq 0$, determine the error location based on the column of H which is equal to $\mathbf{s}$ and complement that bit of $\mathbf{r}$
10 Count the number of decoded bits (out of k) in $\mathbf{r}$ which match the all-zero message bits
11 Accumulate the number of bits in error.
12 UNTIL at least N bit errors have been counted.
13 Compute the probability of error.
14 End FOR

The **coding gain** for a coded system is the difference in the SNR required between uncoded and coded systems achieving the same probability of error. Usually the coding gain is expressed in dB.

Assignment

Preliminary Exercises Show that if X is a random variable with mean 0 and variance 1 then

$$Y = aX + b$$

is a random variable with mean b and variance a^2.

Programming Part

BPSK Simulation

1) Write a program that will simulate a BPSK communication system with unequal prior bit probabilities. Using your program, create data from which to plot the probability of bit error obtained from your simulation for SNRs in the range from 0 to 10 dB, for the three cases that $P_0 = 0.5$ (in which case your plot should look much like Figure 1.10), $P_0 = 0.25$, and $P_0 = 0.1$. Decide on an appropriate value of N.

2) Prepare data from which to plot the theoretical probability of error (1.24) for the same three values of P_0. (You may want to combine these first two programs into a single program.)

3) Plot the simulated probability of error on the same axes as the theoretical probability of error. The plots should have E_b/N_0 in dB as the horizontal axis and the probability as the vertical axis, plotted on a logarithmic scale (e.g., `semilogy` in Matlab).

4) Compare the theoretical and simulated results. Comment on the accuracy of the simulation and the amount of time it took to run the simulation. Comment on the importance of theoretical models (where it is possible to obtain them).

5) Plot the probability of error for $P_0 = 0.1$, $P_0 = 0.25$ and $P_0 = 0.5$ on the same axes. Compare them and comment.

8-PSK Simulation

1) Write a program that will simulate an 8-PSK communication system with equal prior bit probabilities. Use a signal constellation in which the points are numbered in Gray code order. Make your program so that you can estimate both the symbol error probability and the bit error probability. Decide on an appropriate value of N.

2) Prepare data from which to plot the bound on the probability of symbol error P_s using (1.26) and probability of bit error P_b using (1.27).

3) Plot the simulated probability of symbol error and bit error on the same axes as the bounds on the probabilities of error.

4) Compare the theoretical and simulated results. Comment on the accuracy of the bound compared to the simulation and the amount of time it took to run the simulation.

Coded BPSK Simulation

1) Write a program that will simulate performance of the $(7, 4)$ Hamming code over a BSC channel with channel crossover probability $p = Q(\sqrt{2E_b/N_0})$ and plot the probability of error as a function of E_b/N_0 in dB. On the same plot, plot the theoretical probability of error for uncoded BPSK transmission. Identify what the coding gain is for a probability of error $P_b = 10^{-5}$.

2) Repeat this for a $(15, 11)$ Hamming code. (See page 97 and equations (3.6) and (3.4).)

Resources and Implementation Suggestions

- A unit Gaussian random variable has mean zero and variance 1. Given a unit Gaussian random variable, using the preliminary exercise, it is straightforward to generate a Gaussian random variable with any desired variance.

  The function `gran` provides a unit Gaussian random variable, generated using the Box-Muller transformation of two uniform random variables. The function `gran2` returns two unit Gaussian random variables. This is useful for simulations in two-dimensional signal constellations.

- There is nothing in this lab that makes the use of C++ imperative, as opposed to C. However, you may find it useful to use C++ in the following ways:

- Create an AWGN class to represent a 1-D or 2-D channel.
- Create a BSC class.

- Create a Hamming code class to take care of encoding and decoding (as you learn more about coding algorithms, you may want to change how this is done).
- In the literature, points in two-dimensional signal constellations are frequently represented as points in the complex plane. You may find it convenient to do similarly, using the complex number capabilities that are present in C++.
- Since the horizontal axis of the probability of error plot is expressed as a ratio E_b/N_0, there is some flexibility in how to proceed. Given a value of E_b/N_0, you can either fix N_0 and determine E_b, or you can fix E_b and determine N_0. An example of how this can be done is in `testrepcode.cc`.
- The function `uran` generates a uniform random number between 0 and 1. This can be used to generate a bit which is 1 with probability p.
- The Q function, used to compute the theoretical probability of error, is implemented in the function `qf`.
- There are two basic approaches to generating the sequence of bits in the simulation. One way is to generate and store a large array of bits (or their resulting signals) then processing them all together. This is effective in a language such as Matlab, where vectorized operations are faster than using `for` loops. The other way, and the way recommended here, is to generate each signal separately and to process it separately. This is recommended because it is not necessarily known in advance how many bits should be generated. The number of bits to be generated could be extremely large — in the millions or even billions when the probability of error is small enough.
- For the Hamming encoding and decoding operation, vector/matrix multiply operations over GF(2) are required, such as $\mathbf{c} = \mathbf{m}G$. ($GF(2)$ is addition/subtraction/multiplication/division modulo 2.) These could be done in the conventional way using nested `for` loops. However, for short binary codes, a computational simplification is possible. Write G in terms of its columns as

$$G = \begin{bmatrix}\mathbf{g}_1 & \mathbf{g}_2 & \cdots & \mathbf{g}_n\end{bmatrix}$$

Then the encoding process can be written as a series of vector/vector products (inner products)

$$\begin{aligned}\mathbf{c} &= [c_1, c_2, \ldots, c_n] \\ &= \begin{bmatrix}\mathbf{m}\mathbf{g}_1 & \mathbf{m}\mathbf{g}_2 & \cdots & \mathbf{m}\mathbf{g}_n\end{bmatrix}.\end{aligned}$$

Let us consider the inner product operation: it consists of element-by-element multiplication, followed by a sum.

Let m be an integer variable, whose bits represent the elements of the message vector $\mathbf{m}$. Also, let $g[i]$ be an integer variable in C whose bits represent the elements of the column $\mathbf{g}_k$. Then the element-by-element multiplication involved in the product $\mathbf{m}\mathbf{g}_k$ can be written simply using the bitwise-and operator `&` in C. How, then, to sum up the elements of the resulting vector? One way, of course, is to use a `for` loop, such as:

```
// Compute c=m*G, where m is a bit-vector,
// and G is represented by g[i]
c = 0;    // set vector of bits to 0
for(i = 0; i < n; i++) {
   mg = m & g[i];
   // mod-2 multiplication
   // of all elements
   bitsum=0;
   for(j = 0, mask=1; j < n; j++) {
   // mask selects a single bit
      if(mg & mask) {
         bitsum++;
         // accumulate if the bit != 0
      }
      mask <<= 1;
      // shift mask over by 1 bit
   }
   bitsum = bitsum % 2; // mod-2 sum
   c = c | bitsum*(1<<i);
   // assign to vector of bits ...
}
```

However, for sufficiently small codes (such as in this assignment) the inner `for` loop can be eliminated by *precomputing* the sums. Consider table below. For a given number m, the last column provides the sum of all the bits in m, modulo 2.

m	m (binary)	$\sum m$	$s[m] = \sum m \pmod 2$
0	0000	0	0
1	0001	1	1
2	0010	1	1
3	0011	2	0
4	0100	1	1
5	0101	2	0
6	0110	2	0
7	0111	3	1
8	1000	1	1
9	1001	2	0
10	1010	2	0
11	1011	3	1
12	1100	2	0
13	1101	3	1
14	1110	3	1
15	1111	4	0

To use this in a program, precompute the table of bit sums, then use this to look up the result. An outline follows:

```
// Compute the table s, having all
// the bit sums modulo 2
// ...

// Compute c=m*G, where
// m is a bit-vector, and
// G is represented by g[i]
c = 0;
for(i = 0; i < n; i++) {
   c = c | s[m & g[i]]*(1<<i);
   // assign to vector of bits
}
```

1.13 Exercises

1.1 Weighted codes. Let $s_1, s_2, \ldots, s_n$ be a sequence of digits, each in the range $0 \leq s_i < p$, where p is a prime number. The weighted sum is

$$W = ns_1 + (n-1)s_2 + (n-2)s_3 + \cdots + 2s_{n-1} + s_n.$$

The final digit s_n is selected so that W modulo p is equal to 0. That is, $W \equiv 0 \pmod{p}$. W is called the checksum.

(a) Show that the weighted sum W can be computed by computing the cumulative sum sequence $t_1, t_2, \ldots, t_n$ by

$$t_1 = s_1, t_2 = s_1 + s_2, \ldots, t_n = s_1 + s_2 + \cdots s_n$$

then computing the cumulative sum sequence

$$w_1 = t_1, w_2 = t_1 + t_2, \ldots, w_n = t_1 + t_2 + \cdots + t_n,$$

with $W = w_n$.

(b) Suppose that the digits s_k and s_{k+1} are interchanged, with $s_k \neq s_{k+1}$, and then a new checksum W' is computed. Show that if the original sequence satisfies $W \equiv 0 \pmod{p}$, then the modified sequence cannot satisfy $W' \equiv 0 \pmod{p}$. Thus, interchanged digits can be detected.

(c) For a sequence of digits of length $< p$, suppose that digit s_k is altered to some $s'_k \neq s_k$, and a new checksum W' is computed. Show that if the original sequence satisfies $W \equiv 0 \pmod{p}$, then the modified sequence cannot satisfy $W' \equiv 0 \pmod{p}$. Thus, a single modified digit can be detected. Why do we need the added restriction on the length of the sequence?

(d) See if the ISBN 0-13-139072-4 is valid.

(e) See if the ISBN 0-13-193072-4 is valid.

1.2 See if the UPCs 0 59280 00020 0 and 0 41700 00037 9 are valid.

1.3 A coin having $P(\text{head}) = 0.001$ is tossed 10,000 times, each toss independent. What is the lower limit on the number of bits it would take to accurately describe the outcomes? Suppose it were possible to send only 100 bits of information to describe all 10,000 outcomes. What is the minimum average distortion per bit that must be accrued sending the information in this case?

1.4 Show that the entropy of a source X with M outcomes described by (1.1) is maximized when all the outcomes are equally probable: $p_1 = p_2 = \cdots = p_M$.

1.5 Show that (1.7) follows from (1.5) using (1.4).

1.6 Show that (1.12) is true and that the mean and variance of N_{1i} and N_{2i} are as in (1.13) and (1.14).

1.7 Show that the decision rule and threshold in (1.19) and (1.20) are correct.

1.8 Show that (1.24) is correct.

1.9 Show that if X is a random variable with mean 0 and variance 1 that $Y = aX + b$ is a random variable with mean b and variance a^2.

1.10 Show that the detection rule for 8-PSK

$$\hat{\mathbf{s}} = \arg\max_{\mathbf{s} \in \mathcal{S}} \mathbf{r}^T \mathbf{s}$$

follows from (1.18) when all points are equally likely.

1.11 Consider a series of M BSCs, each with transition probability p, where the outputs of each BSC is connected to the inputs of the next in the series. Show that the resulting overall channel is a BSC and determine the crossover probability as a function of M. What happens as $M \to \infty$? *Hint*: To simplify, consider the difference of $(x+y)^n$ and $(x-y)^n$.

1.12 [246] **Bounds and approximations to the Q function.** For many analyses it is useful to have analytical bounds and approximations to the Q function. This exercise introduces some of the most important of these.

(a) Show that

$$\sqrt{2\pi}\, Q(x) = \frac{1}{x} e^{-x^2/2} - \int_x^{\infty} \frac{1}{y^2} e^{-y^2/2} dy \qquad x > 0$$

Hint: integrate by parts.

(b) Show that

$$0 < \int_x^{\infty} \frac{1}{y^2} e^{-y^2/2} dy < \frac{1}{x^3} e^{-x^2/2}.$$

(c) Hence conclude that

$$\frac{1}{\sqrt{2\pi} x} e^{-x^2/2}(1 - 1/x^2) < Q(x) < \frac{1}{\sqrt{2\pi} x} e^{-x^2/2} \qquad x > 0.$$

(d) Plot these lower and upper bounds on a plot with $Q(x)$ (use a log scale).

(e) Another useful bound is $Q(x) \leq \frac{1}{2} e^{-x^2/2}$. Derive this bound. *Hint*: Identify $[Q(\alpha)]^2$ as the probability that the zero-mean unit-Gaussian random variables lie in the shaded region shown on the left in Figure 1.25, (the region $[\alpha, \infty) \times [\alpha, \infty)$). This probability is exceeded by the probability that (x, y) lies in the shaded region shown on the right (extended out to ∞). Evaluate this probability.

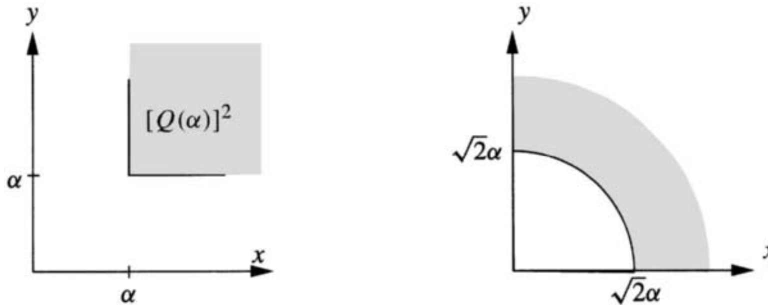


Figure 1.25: Regions for bounding the Q function.

1.13 Let $V_2(n, t)$ be the number of points in a Hamming sphere of "radius" t around a binary codeword of length n. That is, it is the number of points within a Hamming distance t of a binary vector. Determine a formula for $V_2(n, t)$.

1.14 Show that the Hamming distance satisfies the triangle inequality. That is, for three binary vectors $\mathbf{x}$, $\mathbf{y}$, and $\mathbf{z}$ of length n, show that

$$d_H(\mathbf{x}, \mathbf{z}) \leq d_H(\mathbf{x}, \mathbf{y}) + d_H(\mathbf{y}, \mathbf{z}).$$

1.15 Show that for BPSK modulation with amplitudes $\pm\sqrt{E_c}$, the Hamming distance d_H and the Euclidean distance d_E between a pair of codewords are related by $d_E = 2\sqrt{E_c d_H}$.

1.16 In this problem, we will demonstrate that the probability of error for a repetition code decreases exponentially with the code length. Several other useful facts will also be introduced by this problem.

(a) Show that

$$2^{-nH_2(p)} = (1-p)^n \left(\frac{p}{1-p}\right)^{np}.$$

(b) Justify the steps of the proof of the following fact:

$$\text{If } 0 \le p \le \frac{1}{2} \text{ then } \sum_{0 \le i \le pn} \binom{n}{i} \le 2^{nH_2(p)}.$$

$$\begin{aligned} 1 = (p + (1-p))^n &\ge \sum_{0 \le i \le pn} \binom{n}{i} p^i (1-p)^{n-i} \\ &\ge \sum_{0 \le i \le pn} \binom{n}{i} (1-p)^n \left(\frac{p}{1-p}\right)^{pn} \\ &= 2^{-nH_2(p)} \sum_{0 \le i \le pn} \binom{n}{i}. \end{aligned}$$

(c) Show that the probability of error for a repetition code can be written as

$$P_e^n = \sum_{j=0}^{n-(t+1)} \binom{n}{j} (1-p)^j p^{(n-j)},$$

where $t = \lfloor (n-1)/2 \rfloor$.

(d) Show that

$$P_e^n \le \left[2\sqrt{p(1-p)}\right]^n$$

1.17 [220, p. 14] Identities on $\binom{n}{k}$. We can define

$$\binom{x}{m} = \begin{cases} \frac{x(x-1)(x-2)\cdots(x-m+1)}{m!} & \text{if } m \text{ is a positive integer} \\ 1 & \text{if } m = 0 \\ 0 & \text{otherwise.} \end{cases}$$

Show that

(a) $\binom{n}{k} = \frac{n!}{k!(n-k)!}$ if k is a nonnegative integer.
(b) $\binom{n}{k} = 0$ if n is an integer and $k > n$ is a nonzero integer.
(c) $\binom{n}{k} + \binom{n}{k-1} = \binom{n+1}{k}$.
(d) $(-1)^k \binom{-n}{k} = \binom{n+k-1}{k}$.
(e) $\sum_{k=0}^{n} \binom{n}{k} = 2^n$.
(f) $\sum_{k \text{ even}} \binom{n}{k} = \sum_{k \text{ odd}} \binom{n}{k} = 2^{n-1}$ if $n \ge 1$.
(g) $\sum_{k=0}^{n} (-1)^k \binom{n}{k} = 0$ if $n \ge 1$.

1.18 Show that for soft-decision decoding on the $(n, 1)$ repetition code, (1.33) is correct.

1.19 For the $(n, 1)$ code used over a BSC with crossover probability p, what is the probability that an error event occurs which is not detected?

1.20 Hamming code decoding.

(a) For G in (1.34) and H in (1.37), verify that $GH^T = \mathbf{0}$. (Recall that operations are computed modulo 2.)

(b) Let $\mathbf{m} = [1, 1, 0, 0]$. Determine the transmitted Hamming codeword when the generator of (1.34) is used.

(c) Let $\mathbf{r} = [1, 1, 1, 1, 1, 0, 0]$. Using Algorithm 1.1, determine the transmitted codeword $\mathbf{c}$. Also determine the transmitted message $\mathbf{m}$.

(d) The message $\mathbf{m} = [1, 0, 0, 1]$ is encoded to form the codeword $\mathbf{c} = [1, 1, 0, 0, 1, 0, 1]$. The vector $\mathbf{r} = [1, 0, 1, 0, 1, 0, 0]$ is received. Decode $\mathbf{r}$ to obtain $\hat{\mathbf{c}}$. Is the codeword $\hat{\mathbf{c}}$ found the same as the original $\mathbf{c}$? Why or why not?

1.21 For the (7, 4) Hamming code generator polynomial $g(x) = 1 + x + x^3$, generate all possible code polynomials $c(x)$. Verify that they correspond to the codewords in (1.35). Take a nonzero codeword $c(x)$ and compute $c(x)h(x)$ modulo $x^7 + 1$. Do this also for two other nonzero codewords. What is the check condition for this code?

1.22 Is it possible that the polynomial $g(x) = x^4 + x^3 + x^2 + 1$ is a generator polynomial for a cyclic code?

1.23 For the parity check matrix

$$H = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \end{bmatrix}$$

draw the Wolf trellis and the Tanner graph.

1.24 Let X be a random variable taking on the values $\mathcal{A}_x = \{a, b, c, d\}$ with probabilities

$$P(X = a) = \frac{1}{2} \quad P(X = b) = \frac{1}{4} \quad P(X = c) = \frac{1}{8} \quad P(X = d) = \frac{1}{8}.$$

Determine $H(X)$. Suppose that 100 measurements of independent draws of X are made per second. Determine what the entropy rate of this source is. Determine how to encode the X data to achieve this rate.

1.25 Show that the information inequality $\log x \leq x - 1$ is true.

1.26 Show that for a discrete random variable X, $H(X) \geq 0$.

1.27 Show that $I(X; Y) \geq 0$ and that $I(X; Y) = 0$ only if X and Y are independent. *Hint*: Use the information inequality.

1.28 Show that the formulas $I(X; Y) = H(X) - H(X|Y)$ and $I(X; Y) = H(Y) - H(Y|X)$ follow from the definition (1.38).

1.29 Show that $H(X) \geq H(X|Y)$. *Hint*: Use the previous two problems.

1.30 Show that the mutual information $I(X; Y)$ can be written as

$$I(X; Y) = \sum_{x \in \mathcal{A}_x} P_X(x) \sum_{y \in \mathcal{A}_y} P_{Y|X}(y|x) \log_2 \frac{P_{Y|X}(y|x)}{\sum_{x' \in \mathcal{A}_x} P_X(x') P_{Y|X}(y|x')}$$

1.31 For a BSC with crossover probability p having input X and output Y, let the probability of the inputs be $P(X = 0) = q$ and $P(X = 1) = 1 - q$.

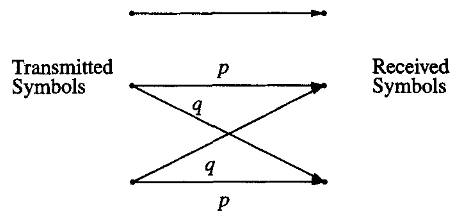
(a) Show that the mutual information is

$$I(X; Y) = H(Y) + p \log_2 p + (1 - p) \log_2 (1 - p)$$

(b) By maximizing over q show that the channel capacity per channel use is

$$C = 1 - H_2(p) \text{ (bits)}.$$

1.32 Consider the channel model shown here, which accepts three different symbols.



The first symbol is not affected by noise, while the second and third symbols have a probability p of not being corrupted, and a probability q of being changed into the other of the pair. Let $\alpha = -p \log p - q \log q$, and let P be the probability that the first symbol is chosen and let Q be the probability that either of the other two is chosen, so that $P + 2Q = 1$.

(a) Show that $H(X) = -P \log P - 2Q \log Q$.

(b) Show that $H(X|Y) = 2Q\alpha$.

(c) Choose the input distribution (i.e., choose P and Q) in such a way to maximize $I(X; Y) = H(X) - H(X|Y))$ subject to $P + 2Q = 1$. What is the capacity for this channel?

1.33 Let $X \sim \mathcal{U}(-a, a)$ (that is, X is uniformly distributed on $[-a, a]$). Compute $H(X)$. Compare $H(X)$ with the entropy of a Gaussian distribution having the same variance.

1.34 Let $g(x)$ denote the pdf of a random variable X with variance σ^2. Show that

$$H(X) \leq \frac{1}{2} \log_2 2\pi e \sigma^2.$$

with equality if and only if X is Gaussian. *Hint*: Let $p(x)$ denote the pdf of a Gaussian r.v. with variance σ^2 and consider $D(g||p)$. Also, note that $\log p(x)$ is quadratic in x.

1.35 Show that $H(X + N|X) = H(N)$.

1.14 References

The information age was heralded with Shannon's work [309]. Thorough coverage of information theory appears in [59], [111] and [382]. The books [228] and [357] place coding theory in its information theoretic context. Our discussion of the AEP follows [15], while our "proof" of the channel coding theorem closely follows Shannon's original [309]. More analytical proofs appear in the textbooks cited above. See also [350]. Discussion about tradeoffs with complexity are in [288], as is the discussion in Section 1.12.6.

The detection theory and signal space background is available in most books on digital communication. See, for example, [276, 15, 246, 267].

Hamming codes were presented in [137]. The trellis representation was presented first in [377]; a thorough treatment of the concept appears in [205]. The Tanner graph representation appears in [330]; see also [112]. Exercise 1.16b comes from [350, p. 21].

The discussion relating to simulating communication systems points out that such simulations can be very slow. Faster results can in some cases be obtained using *importance sampling*. Some references on importance sampling are [84, 211, 308, 316].

Part II

Block Codes

Chapter 2

Groups and Vector Spaces

2.1 Introduction

Linear block codes form a group and a vector space. Hence, the study of the properties of this class of codes benefits from a formal introduction to these concepts. The codes, in turn, reinforce the concepts of groups and subgroups that are valuable in the remainder of our study.

Our study of groups leads us to cyclic groups, subgroups, cosets, and factor groups. These concepts, important in their own right, also build insight in understanding the construction of extension fields which are essential for some coding algorithms to be developed.

2.2 Groups

A **group** formalizes some of the basic rules of arithmetic necessary for cancellation and solution of some simple algebraic equations.

Definition 2.1 A **binary operation** $*$ on a set is a rule that assigns to each ordered pair of elements of the set (a, b) some element of the set. (Since the operation returns an element in the set, this is actually defined as *closed* binary operation. We assume that all binary operations are closed.) □

Example 2.1 On the set of positive integers, we can define a binary operation $*$ by $a * b = \min(a, b)$. □

Example 2.2 On the set of real numbers, we can define a binary operation $*$ by $a * b = a$ (i.e., the first argument). □

Example 2.3 On the set of real numbers, we can define a binary operation $*$ by $a * b = a + b$. That is, the binary operation is regular addition. □

Definition 2.2 A **group** $\langle G, * \rangle$ is a set G together with a binary operation $*$ on G such that:

G1 The operator is **associative**: for any $a, b, c \in G$, $(a * b) * c = a * (b * c)$.

G2 There is an element $e \in G$ called the **identity element** such that $a * e = e * a = a$ for all $a \in G$.

G3 For every $a \in G$, there is an element $b \in G$ known as the **inverse** of a such that $a * b = e$. The inverse of a is sometimes denoted as a^{-1} (when the operator $*$ is multiplication-like) or as $-a$ (when the operator $*$ is addition-like).

Where the operation is clear from context, the group $\langle G, *\rangle$ may be denoted simply as G.

It should be noted that the notation $*$ and a^{-1} are generic labels to indicate the concept. The particular notation used is modified to fit the concept. Where the group operation is addition, the operator $+$ is used and the inverse of an element a is more commonly represented as $-a$. When the group operation is multiplication, either $\cdot$ or juxtaposition is used to indicate the operation and the inverse is denoted as a^{-1}.

Definition 2.3 If G has a finite number of elements, it is said to be a finite group. The **order** of a finite group G, denoted $|G|$, is the number of elements in G. □

This definition of order (of a group) is to be distinguished from the order of an element, given below. □

Definition 2.4 A group $\langle G, *\rangle$ is **commutative** if $a * b = b * a$ for every $a, b \in G$. □

Example 2.4 The set $\langle \mathbb{Z}, +\rangle$, which is the set of integers under addition, forms a group. The identity element is 0, since $0 + a = a + 0 = a$ for any $a \in \mathbb{Z}$. The inverse of any $a \in \mathbb{Z}$ is $-a$.

This is a commutative group. □

As a matter of convention, a group that is commutative with an additive-like operator is said to be an **Abelian** group (after the mathematician N.H. Abel).

Example 2.5 The set $\langle \mathbb{Z}, \cdot\rangle$, the set of integers under multiplication, does *not* form a group. There is a multiplicative identity, 1, but there is not a multiplicative inverse for every element in $\mathbb{Z}$. □

Example 2.6 The set $\langle \mathbb{Q} \setminus \{0\}, \cdot\rangle$, the set of rational numbers excluding 0, is a group with identity element 1. The inverse of an element a is $a^{-1} = 1/a$. □

The requirements on a group are strong enough to introduce the idea of cancellation. In a group G, if $a * b = a * c$, then $b = c$ (this is left cancellation). To see this, let a^{-1} be the inverse of a in G. Then

$$a^{-1} * (a * b) = a^{-1} * (a * c) = (a^{-1} * a) * c = e * c = c$$

and $a^{-1} * (a * b) = (a^{-1} * a) * b = e * b = b$, by the properties of associativity and identity.

Under group requirements, we can also verify that solutions to linear equations of the form $a * x = b$ are unique. Using the group properties we get immediately that $x = a^{-1}b$. If x_1 and x_2 are two solutions, such that $a * x_1 = b = a * x_2$, then by cancellation we get immediately that $x_1 = x_2$.

Example 2.7 Let $\langle \mathbb{Z}_5, +\rangle$ denote addition on the numbers $\{0, 1, 2, 3, 4\}$ modulo 5. The operation is demonstrated in tabular form in the table below:

+	0	1	2	3	4
0	0	1	2	3	4
1	1	2	3	4	0
2	2	3	4	0	1
3	3	4	0	1	2
4	4	0	1	2	3

Clearly 0 is the identity element. Since 0 appears in each row and column, every element has an inverse. By the uniqueness of solution, we must have every element appearing in every row and column, as it does. By the symmetry of the table it is clear that the operation is Abelian (commutative). Thus we verify that $\langle \mathbb{Z}_5, +\rangle$ is an Abelian group.

(Typically, when using a table to represent a group operation $a * b$, the first operand a is the row and the second operand b is the column in the table.) □

In general, we denote the set of numbers $0, 1, \ldots, n-1$ with addition modulo n by $\langle \mathbb{Z}_n, + \rangle$ or, more briefly, $\mathbb{Z}_n$.

Example 2.8 Consider the set of numbers $\{1, 2, 3, 4, 5\}$ using the operation of multiplication modulo 6. The operation is shown in the following table:

$\cdot$	1	2	3	4	5
1	1	2	3	4	5
2	2	4	0	2	4
3	3	0	3	0	3
4	4	2	0	4	2
5	5	4	3	2	1

The number 1 acts as an identity, but this does not form a group, since not every element has a multiplicative inverse. In fact, the only elements that have a multiplicative inverse are those that are relatively prime to 6, that is, those numbers that don't share a divisor with 6 other than one. We will see this example later in the context of rings. □

One way to construct groups is to take the Cartesian, or direct, product of groups. Given groups $\langle G_1, * \rangle, \langle G_2, * \rangle, \ldots, \langle G_r, * \rangle$, the direct product group $G_1 \times G_2 \times \cdots \times G_r$ has elements $(a_1, a_2, \ldots, a_r)$, where each $a_i \in G_i$. The operation in G is defined element-by-element. That is, if

$$(a_1, a_2, \ldots, a_r) \in G \text{ and } (b_1, b_2, \ldots, b_r) \in G,$$

then

$$(a_1, a_2, \ldots, a_r) * (b_1, b_2, \ldots, b_r) = (a_1 * b_1, a_2 * b_2, \ldots, a_r * b_r).$$

Example 2.9 The group $\langle \mathbb{Z}_2 \times \mathbb{Z}_2, + \rangle$ consists of two-tuples with addition defined element-by-element modulo two. An addition for the group table is shown here:

+	(0,0)	(0,1)	(1,0)	(1,1)
(0,0)	(0,0)	(0,1)	(1,0)	(1,1)
(0,1)	(0,1)	(0,0)	(1,1)	(1,0)
(1,0)	(1,0)	(1,1)	(0,0)	(0,1)
(1,1)	(1,1)	(1,0)	(0,1)	(0,0)

This group is called the Klein 4-group. □

Example 2.10 This example introduces the idea of *permutations* as elements in a group. It is interesting because it introduces a group operation that is function composition, as opposed to the mostly arithmetic group operations presented to this point.

A permutation of a set A is a one-to-one, onto function (a bijection) of a set A onto itself. It is convenient for purposes of illustration to let A be a set of n integers. For example,

$$A = \{1, 2, 3, 4\}.$$

A permutation p can be written in the notation

$$p = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 3 & 4 & 1 & 2 \end{pmatrix},$$

which means that

$$1 \rightarrow 3 \quad 2 \rightarrow 4 \quad 3 \rightarrow 1 \quad 4 \rightarrow 2.$$

There are $n!$ different permutations on n distinct elements.

We can think of p as an operator expressed in prefix notation. For example,

$$p \circ 1 = 3 \quad \text{or} \quad p \circ 4 = 2.$$

Let $p_1 = p$ and

$$p_2 = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 4 & 3 & 1 & 2 \end{pmatrix}$$

The *composition* permutation $p_2 \circ p_1$ first applies p_1, then p_2, so that

$$p_2 \circ p_1 = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 4 & 3 & 1 & 2 \end{pmatrix} \circ \begin{pmatrix} 1 & 2 & 3 & 4 \\ 3 & 4 & 1 & 2 \end{pmatrix} = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 4 & 3 \end{pmatrix}$$

This is again another permutation, so the operation of composition of permutations is closed under the set of permutations. The identity permutation is

$$e = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \end{pmatrix}.$$

There is an inverse permutation under composition. For example,

$$p_1^{-1} = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 3 & 4 & 1 & 2 \end{pmatrix}.$$

It can be shown that composition of permutations is associative: for three permutations p_1, p_2 and p_3, then $(p_1 \circ p_2) \circ p_3 = p_1 \circ (p_2 \circ p_3)$.

Thus the set of all $n!$ permutations on n elements forms a group, where the group operation is function composition. This group is referred to as the **symmetric group** on n letters. The group is commonly denoted by S_n.

It is also interesting to note that the composition is *not* commutative. This is clear from this example since

$$p_2 \circ p_1 \neq p_1 \circ p_2.$$

So S_4 is an example of a non-commutative group. □

2.2.1 Subgroups

Definition 2.5 A subgroup $\langle H, * \rangle$ of a group $\langle G, * \rangle$ is a group formed from a subset of elements in a group G with the same operation $*$. Notationally, we may write $H < G$ to indicate that H is a subgroup of G. (There should be no confusion using $<$ with comparisons between numbers because the operands are different in each case.) □

If the elements of H are a strict subset of the elements of G (i.e., $H \subset G$ but not $H = G$), then the subgroup is said to be a **proper** subgroup. If $H = G$, then H is an improper subgroup of G. The subgroups $H = \{e\} \subset G$ (e is the identity) and $H = G$ are said to be trivial subgroups.

Example 2.11 Let $G = \langle \mathbb{Z}_6, + \rangle$, the set of numbers $\{0, 1, 2, 3, 4, 5\}$ using addition modulo 6. Let $H = \langle \{0, 2, 4\}, + \rangle$, with addition taken modulo 6. As a set, $H \subset G$. It can be shown that H forms a group.

Let $K = \langle \{0, 3\}, + \rangle$, with addition taken modulo 6. Then K is a subgroup of G. □

Example 2.12 A variety of familiar groups can be arranged as subgroups. For example,

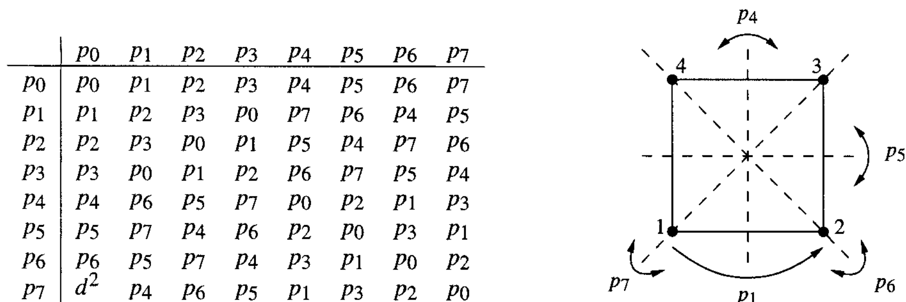
$$\langle \mathbb{Z}, + \rangle < \langle \mathbb{Q}, + \rangle < \langle \mathbb{R}, + \rangle < \langle \mathbb{C}, + \rangle.$$

□

Example 2.13 The group of permutations on 4 letters, S_4, has a subgroup formed by the permutations

$$p_0 = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \end{pmatrix} \qquad p_1 = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 3 & 4 & 1 \end{pmatrix}$$
$$p_2 = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 3 & 4 & 1 & 2 \end{pmatrix} \qquad p_3 = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 4 & 1 & 2 & 3 \end{pmatrix}$$
$$p_4 = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 1 & 4 & 3 \end{pmatrix} \qquad p_5 = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 4 & 3 & 2 & 1 \end{pmatrix}$$
$$p_6 = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 3 & 2 & 1 & 4 \end{pmatrix} \qquad p_7 = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 1 & 4 & 3 & 2 \end{pmatrix} \tag{2.1}$$

Compositions of these permutations is closed. These permutations correspond to the ways that the corners of a square can be moved to other corners by rotation about the center and reflection across edges or across diagonals (without bending the square). The geometric depiction of these permutations and the group operation table are shown here:



	p_0	p_1	p_2	p_3	p_4	p_5	p_6	p_7
p_0	p_0	p_1	p_2	p_3	p_4	p_5	p_6	p_7
p_1	p_1	p_2	p_3	p_0	p_7	p_6	p_4	p_5
p_2	p_2	p_3	p_0	p_1	p_5	p_4	p_7	p_6
p_3	p_3	p_0	p_1	p_2	p_6	p_7	p_5	p_4
p_4	p_4	p_6	p_5	p_7	p_0	p_2	p_1	p_3
p_5	p_5	p_7	p_4	p_6	p_2	p_0	p_3	p_1
p_6	p_6	p_5	p_7	p_4	p_3	p_1	p_0	p_2
p_7	d^2	p_4	p_6	p_5	p_1	p_3	p_2	p_0

This group is known as D_4. D_4 has a variety of subgroups of its own. (Can you find them?) □

2.2.2 Cyclic Groups and the Order of an Element

In a group G with operation $*$ or multiplication operation we use the notation a^n to indicate $a * a * a * \cdots * a$, with the operand a appearing n times. Thus $a^1 = a$, $a^2 = a * a$, etc. We take a^0 to be the identity element in the group G. We use a^{-2} to indicate $(a^{-1})(a^{-1})$, and a^{-n} to indicate $(a^{-1})^n$.

For a group with an additive operator $+$, the notation na is often used, which means $a + a + a + \cdots + a$, with the operand appearing n times. Throughout this section we use the a^n notation; making the switch to the additive operator notation is straightforward.

Let G be a group and let $a \in G$. Any subgroup containing a must also contain a^2, a^3, and so forth. The subgroup must contain $e = aa^{-1}$, and hence a^{-2}, a^{-3}, and so forth, are also in the subgroup.

Definition 2.6 For any $a \in G$, the set $\{a^n | n \in \mathbb{Z}\}$ generates a subgroup of G called the **cyclic subgroup**. The element a is said to be the **generator** of the subgroup. The cyclic subgroup generated by a is denoted as $\langle a \rangle$. □

Definition 2.7 If every element of a group can be generated by a single element, the group is said to be **cyclic**. □

Example 2.14 The group $\langle \mathbb{Z}_5, + \rangle$ is cyclic, since every element in the set can be generated by $a = 2$ (under the appropriate addition law):

$$\mathbf{2}, \quad 2+2=\mathbf{4}, \quad 2+2+2=\mathbf{1}, \quad 2+2+2+2=\mathbf{3}, \quad 2+2+2+2+2=\mathbf{0}.$$

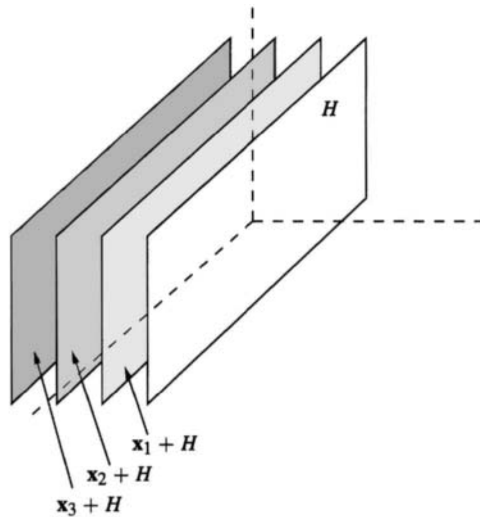


Figure 2.1: An illustration of cosets.

In this case we could write $\mathbb{Z}_5 = \langle 2 \rangle$. Observe that there are several generators for $\mathbb{Z}_5$. □

The permutation group S_3 is not cyclic: there is no element which generates the whole group.

Definition 2.8 In a group G, with $a \in G$, the smallest n such that a^n is equal to the identity in G is said to be the **order** of a. If no such n exists, a is of **infinite order**. □

The order of an element should not be confused with the order of a group, which is the number of elements in the group.

In $\mathbb{Z}_5$, the computations above show that the element 2 is of order 5. In fact, the order of every nonzero element in $\mathbb{Z}_5$ is 5.

Example 2.15 Let $G = \langle \mathbb{Z}_6, + \rangle$. Then

$$\langle 2 \rangle = \{0, 2, 4\} \qquad \langle 3 \rangle = \{0, 3\} \qquad \langle 5 \rangle = \{0, 1, 2, 3, 4, 5\} = \mathbb{Z}_6.$$

It is easy to verify that an element $a \in \mathbb{Z}_6$ is a generator for the whole group if and only if a and 6 are relatively prime. □

2.2.3 Cosets

Definition 2.9 Let H be a subgroup of $\langle G, * \rangle$ (where G is not necessarily commutative) and let $a \in G$. The **left coset** of H, $a * H$, is the set $\{a * h | h \in H\}$. The **right coset** of H is similarly defined, $H * a = \{h * a | h \in H\}$. □

Of course, in a commutative group, the left and right cosets are the same.

Figure 2.1 illustrates the idea of cosets. If G is the group $\langle \mathbb{R}^3, + \rangle$ and H is the white plane shown, then the cosets of H in G are the *translations* of H.

Let G be a group and let H be a subgroup of G. Let $a * H$ be a (left) coset of H in G. Then clearly $b \in a * H$ if and only if $b = a * h$ for some $h \in H$. This means (by cancellation) that we must have

$$a^{-1} * b \in H.$$

Thus to determine if a and b are in the same (left) coset of H, we determine if $a^{-1} * b \in H$.

Example 2.16 Let $G = \langle \mathbb{Z}, + \rangle$ and let

$$S_0 = 3\mathbb{Z} = \{\ldots, -6, -3, 0, 3, 6, \ldots\}.$$

Then S_0 is a subgroup of G. Now let us form the cosets

$$S_1 = S_0 + 1 = \{\ldots, -5, -2, 1, 4, 7, \ldots\}.$$

and

$$S_2 = S_0 + 2 = \{\ldots, -4, -1, 2, 5, 8, \ldots\}.$$

Note that neither S_1 nor S_2 are groups (they do not contain the identity). The sets S_0, S_1, and S_2 collectively cover the original group,

$$G = S_0 \cup S_1 \cup S_2.$$

Let us check whether $a = 4$ and $b = 6$ are in the same coset of S_0 by checking whether $(-a) + b \in S_0$. Since $-a + b = 2 \notin S_0$, a and b are not in the same coset. □

2.2.4 Lagrange's Theorem

Lagrange's theorem prescribes the size of a subgroup compared to the size of its group. This little result is used in a variety of ways in the developments to follow.

Lemma 2.1 *Every coset of H in a group G has the same number of elements.*

Proof We will show that every coset has the same number of elements as H. Let $a * h_1 \in a * H$ and let $a * h_2 \in a * H$ be two elements in the coset $a * H$. If $a * h_1 = a * h_2$ then by cancellation we must have $h_1 = h_2$. Thus the elements of a coset are uniquely identified by the elements in H. □

We summarize some important properties about cosets:

Reflexive An element a is in the same coset as itself.

Symmetric If a and b are in the same coset, then b and a are in the same coset.

Transitive If a and b are in the same coset, and b and c are in the same coset, then a and c are in the same coset.

Reflexivity, symmetricity, and transitivity are properties of the relation "in the same coset."

Definition 2.10 A relation which has the properties of being **reflexive, symmetric**, and **transitive** is said to be an **equivalence relation**. □

An important fact about equivalence relations is that every equivalence relation *partitions* its elements into *disjoint* sets. Let us consider here the particular case of cosets.

Lemma 2.2 *The distinct cosets of H in a group G are disjoint.*

Proof Suppose A and B are distinct cosets of H; that is, $A \neq B$. Assume that A and B are not disjoint, then there is some element c which is common to both. We will show that this implies that $A \subset B$. Let $b \in B$. For any $a \in A$, a and c are in the same coset (since c is in A). And c and b are in the same coset (since c is in B). By transitivity, a and b must be in the same coset. Thus every element of A is in B, so $A \subset B$. Turning the argument around, we find that $B \subset A$. Thus $A = B$.

This contradiction shows that distinct A and B must also be disjoint. □

Theorem 2.3 Lagrange's theorem *Let G be a group of finite order and let H be a subgroup of G. Then the order of H divides*[1] *the order of G. That is, $|H|$ divides $|G|$.*

Proof The set of cosets partition G into disjoint sets, each of which has the same number of elements, $|H|$. These disjoint sets completely cover G, since every element $g \in G$ is in some coset, $g * H$. So the number of elements of G must be equal a multiple of $|H|$. □

Lagrange's theorem can be stated more succinctly using a notation which we now introduce:

Definition 2.11 The vertical bar $\mid$ means **divides**. We write $a \mid b$ if a divides b (without remainder). □

Then Lagrange's theorem can be written: If $|G| < \infty$ and $H < G$, then $|H| \mid |G|$.

One implication of Lagrange's theorem is the following.

Lemma 2.4 *Every group of prime order is cyclic.*

Proof Let G be of prime order, let $a \in G$, and denote the identity in G by e. Let $H = \langle a \rangle$, the cyclic subgroup generated by a. Then $a \in H$ and $e \in H$. But by Theorem 2.3, the order of H must divide the order of G. Since G is of prime order, then we must have $|H| = |G|$; hence a generates G, so G is cyclic. □

2.2.5 Induced Operations; Isomorphism

Example 2.17 Let us return to the three cosets S_0, S_1, and S_2 defined in Example 2.16. We thus have a set of three objects, $S = \{S_0, S_1, S_2\}$. Let us define an addition operation on S as follows: for A, B and $C \in S$,

$$A + B = C \quad \text{if and only if } a + b = c \text{ for any } a \in A, b \in B \text{ and some } c \in C.$$

That is, addition of the sets is defined by **representatives** in the sets. The operation is said to be the induced operation on the cosets. For example,

$$S_1 + S_2 = S_0,$$

taking as representatives, for example, $1 \in S_1, 2 \in S_2$ and noting that $1 + 2 = 3 \in S_0$. Similarly,

$$S_1 + S_1 = S_2,$$

taking as representatives $1 \in S_1$ and noting that $1 + 1 = 2 \in S_2$. Based on this induced operation, an addition table can be built for the set S:

$+$	S_0	S_1	S_2
S_0	S_0	S_1	S_2
S_1	S_1	S_2	S_0
S_2	S_2	S_0	S_1

It is clear that this addition table defines a group, which we can call $\langle S, + \rangle$. Now compare this addition table with the addition table for $\mathbb{Z}_3$:

$+$	0	1	2
0	0	1	2
1	1	2	0
2	2	0	1

[1] That is, divides without remainder

Box 2.1: One-to-One and Onto Functions

Definition 2.13 A function $\phi : G \to \mathcal{G}$ is said to be **one-to-one** if $\phi(a) = \phi(b)$ implies $a = b$ for every a and b in G. That is, two distinct values $a, b \in G$ with $a \neq b$ do not map to the same value of ϕ. A one-to-one function is also called a **surjective** function. □

A contrasting example is $\phi(x) = x^2$, where $\phi : \mathbb{R} \to \mathbb{R}$, which is *not* one-to-one since $4 = \phi(2)$ and $4 = \phi(-2)$.

Definition 2.14 A function $\phi : G \to \mathcal{G}$ is said to be **onto** if for every $g \in \mathcal{G}$, there is an element $a \in G$ such that $\phi(a) = g$. An onto function is also called an **injective** function. □

That is, the function goes *onto* everything in $\mathcal{G}$. A contrasting example is $\phi(x) = x^2$, where $\phi : \mathbb{R} \to \mathbb{R}$, since the point $g = -3$ is not mapped onto by ϕ from any point in $\mathbb{R}$.

Definition 2.15 A function which is one-to-one and onto (i.e., surjective and injective) is called **bijective**. □

Bijective functions are always invertible. If $\phi : G \to \mathcal{G}$ is bijective, then $|G| = |\mathcal{G}|$ (the two sets have the same cardinality).

Structurally, the two addition tables are identical: entries in the second table are obtained merely by replacing S_k with k, for $k = 0, 1, 2$. We say that the group $\langle S, + \rangle$ and the group $\langle \mathbb{Z}_3, + \rangle$ are **isomorphic**. □

Definition 2.12 Two groups $\langle G, * \rangle$ and $\langle \mathcal{G}, \diamond \rangle$ are said to be (group) **isomorphic** if there exists a one-to-one, onto function $\phi : G \to \mathcal{G}$ called the **isomorphism** such that for every $a, b \in G$,

$$\phi(\underbrace{a * b}_{\substack{\text{operation}\\ \text{in } G}}) = \underbrace{\phi(a) \diamond \phi(b)}_{\substack{\text{operation}\\ \text{in } \mathcal{G}}}. \tag{2.2}$$

The fact that groups G and $\mathcal{G}$ are isomorphic are denoted by $G \cong \mathcal{G}$. □

We can thus write $S \cong \mathbb{Z}_3$ (where the operations are unstated but understood from context).

Whenever two groups are isomorphic they are, for all practical purposes, the same thing. Different objects in the groups may have different names, but they represent the same sorts of relationships among themselves.

Definition 2.16 Let $\langle G, * \rangle$ be a group, H a subgroup and let $S = \{H_0 = H, H_1, H_2, \ldots, H_M\}$ be the set of cosets of H in G. Then the **induced operation** between cosets A and B in S is defined by

$$A * B = C \text{ if and only if } a * b = c$$

for any $a \in A$, $b \in B$ and some $c \in C$, provided that this operation is well defined. The operation is **well defined** if for every $a \in A$ and $b \in B$, $a * b \in C$; there is thus no ambiguity in the induced operation. □

For commutative groups, the induced operation is always well defined. However, the reader should be cautioned that for noncommutative groups, the operation is well defined

only for normal subgroups.[2]

Example 2.18 Consider the group $G = \langle \mathbb{Z}_6, + \rangle$ and let $H = \{0, 3\}$. The cosets of H are

$$H_0 = \{0, 3\} \qquad H_1 = 1 + H = \{1, 4\} \qquad H_2 = 2 + H = \{2, 5\}.$$

Then, under the induced operation, for example, $H_2 + H_2 = H_1$ since $2 + 2 = 4$ and $4 \in H_1$. We could also choose different representatives from the cosets. We get

$$5 + 5 = 4$$

in G. Since $5 \in H_2$ and $4 \in H_1$, we again have $H_2 + H_2 = H_1$. If by choosing different elements from the addend cosets we were to end up with a different sum coset, the operation would not be well defined. Let us write the addition table for $\mathbb{Z}_6$ reordered and separated out by the cosets. The induced operation is clear. We observe that H_0, H_1 and H_2 themselves constitute a group, with addition table also shown.

		H_0		H_1		H_2	
	+	0	3	1	4	2	5
H_0	0	0	3	1	4	2	5
	3	3	0	4	1	5	2
H_1	1	1	4	2	5	3	0
	4	4	1	5	2	0	3
H_2	2	2	5	3	0	4	1
	5	5	2	0	3	1	4

+	H_0	H_1	H_2
H_0	H_0	H_1	H_2
H_1	H_1	H_2	H_0
H_2	H_2	H_0	H_1

The group of cosets is clearly isomorphic to $\langle \mathbb{Z}_3, + \rangle$: $\{H_0, H_1, H_2\} \cong \mathbb{Z}_3$. □

From this example, we see that the cosets themselves form a group.

Theorem 2.5 *If H is a subgroup of a commutative group $\langle G, * \rangle$, the induced operation $*$ on the set of cosets of H satisfies*

$$(a * b) * H = (a * H) * (b * H).$$

The proof is explored in Exercise 2.13. This defines an operation. Clearly, H itself acts as an identity for the operation defined on the set of cosets. Also, by Theorem 2.5, $(a * H) * (a^{-1} * H) = (a * a^{-1}) * H = H$, so every coset has an inverse coset. Thus the set of cosets of H form a group.

Definition 2.17 The group formed by the cosets of H in a commutative[3] group G with the induced operation is said to be the **factor group** of G modulo H, denoted by G/H. The cosets are said to be the **residue classes** of G modulo H. □

In the last example, we could write $\mathbb{Z}_3 \cong \mathbb{Z}_6/H_0$. From Example 2.17, the group of cosets was also isomorphic to $\mathbb{Z}_3$, so we can write

$$\mathbb{Z}/3\mathbb{Z} \cong \mathbb{Z}_3.$$

In general, it can be shown that

$$\mathbb{Z}/n\mathbb{Z} \cong \mathbb{Z}_n.$$

[2] A subgroup H of a group G is **normal** if $g^{-1}Hg = H$ for all $g \in G$. Clearly all Abelian groups are normal.

[3] Or of a normal subgroup in a noncommutative group.

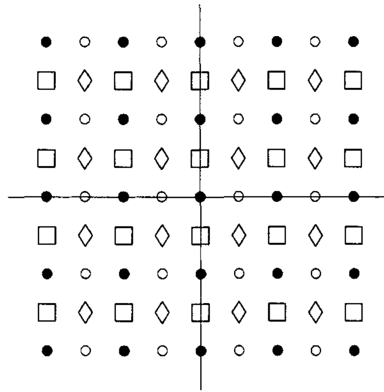


Figure 2.2: A lattice partitioned into cosets.

Example 2.19 A **lattice** is formed by taking all possible integer linear combinations of a set of basis vectors. That is, let $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n$ be a set of linearly independent vectors, let $V = [\mathbf{v}_1, \mathbf{v}_2, \cdots, \mathbf{v}_n]$. Then a lattice is formed from these basis vectors by

$$\Lambda = \{V\mathbf{z} : \mathbf{z} \in \mathbb{Z}^n\}.$$

For example, the lattice formed by $V = \left[\begin{smallmatrix} 1 & 0 \\ 0 & 1 \end{smallmatrix}\right]$ is the set of points with integer coordinates in the plane, denoted as $\mathbb{Z}^2$.

For the lattice $\Lambda = \mathbb{Z}^2$, let $\Lambda' = 2\mathbb{Z}^2$ be a subgroup. Then the cosets

$S_0 = \Lambda'$ (denoted by $\bullet$) $\qquad S_1 = (1, 0) + \Lambda'$ (denoted by $\circ$)

$S_2 = (0, 1) + \Lambda'$ (denoted by $\square$) $\qquad S_3 = (1, 1) + \Lambda'$ (denoted by $\lozenge$)

are indicated in Figure 2.2. It is straightforward to verify that

$$\Lambda/\Lambda' \cong \mathbb{Z}_2 \times \mathbb{Z}_2.$$

Such decompositions of lattices into subsets find application in trellis coded modulation, as we shall see in Chapter 13. $\square$

2.2.6 Homomorphism

For *isomorphism*, two sets G and $\mathcal{G}$ are structurally the same, as defined by (2.2), and they have the same number of elements (since there is a bijective function $\phi : G \to \mathcal{G}$). From an algebraic point of view, G and $\mathcal{G}$ are identical, even though they may have different names for their elements.

Homomorphism is a somewhat weaker condition: the sets must have the same algebraic structure, but they might have different numbers of elements.

Definition 2.18 The groups $\langle G, * \rangle$ and $\langle \mathcal{G}, \diamond \rangle$ are said to be (group) **homomorphic** if there exists a function (that is not necessarily one-to-one) $\phi : G \to \mathcal{G}$ called the **homomorphism** such that

$$\phi(\underbrace{a * b}_{\substack{\text{operation} \\ \text{in } G}}) = \underbrace{\phi(a) \diamond \phi(b)}_{\substack{\text{operation} \\ \text{in } \mathcal{G}}}. \tag{2.3}$$

□

Example 2.20 Let $G = \langle \mathbb{Z}, + \rangle$ and let $\mathcal{G} = \langle \mathbb{Z}_n, + \rangle$. Let $\phi : G \to \mathcal{G}$ be defined by $\phi(a) = a$ mod n, the remainder when a is divided by n. Let $a, b \in \mathbb{Z}$. We have (see Exercise 2.32)

$$\phi(a + b) = \phi(a) + \phi(b).$$

Thus $\langle \mathbb{Z}, + \rangle$ and $\langle \mathbb{Z}_n, + \rangle$ are homomorphic, although they clearly do not have the same number of elements. □

Theorem 2.6 *Let $\langle G, * \rangle$ be a commutative group and let H be a subgroup, so that G/H is the factor group. Let $\phi : G \to G/H$ be defined by $\phi(a) = a * H$. Then ϕ is a homomorphism. The homomorphism ϕ is said to be the* natural *or* canonical *homomorphism.*

Proof Let $a, b \in G$. Then by Theorem 2.5

$$\underbrace{\phi(a * b)}_{\substack{\text{operation} \\ \text{in } G}} = \underbrace{\phi(a) * \phi(b)}_{\substack{\text{operation} \\ \text{in } G/H}}.$$

□

Definition 2.19 The **kernel of a homomorphism** ϕ of a group G into a group $\mathcal{G}$ is the set of all elements of G which are mapped onto the identity element of $\mathcal{G}$ by ϕ. □

Example 2.21 For the canonical map $\mathbb{Z} \to \mathbb{Z}_n$ of Example 2.20, the kernel is $n\mathbb{Z}$, the set of multiples of n. □

2.3 Fields: A Prelude

We shall have considerably more to say about fields in Chapter 5, but we introduce the concept here since fields are used in defining vector spaces and simple linear block codes.

Definition 2.20 A **field** $\langle \mathbb{F}, +, \cdot \rangle$ is a set of objects $\mathbb{F}$ on which the operations of addition and multiplication, subtraction (or additive inverse), and division (or multiplicative inverse) apply in a manner analogous to the way these operations work for real numbers.

In particular, the addition operation $+$ and the multiplication operation $\cdot$ (or juxtaposition) satisfy the following :

F1 Closure under addition: For every a and b in $\mathbb{F}$, $a + b$ is also in $\mathbb{F}$.

F2 Additive identity: There is an element in $\mathbb{F}$, which we denote as 0, such that $a + 0 = 0 + a = a$ for every $a \in \mathbb{F}$.

F3 Additive inverse (subtraction): For every $a \in \mathbb{F}$, there exists an element b in $\mathbb{F}$ such that $a + b = b + a = 0$. The element b is frequently called the additive inverse of a and is denoted as $-a$.

F4 Associativity: $(a + b) + c = a + (b + c)$ for every $a, b, c \in \mathbb{F}$.

F5 Commutativity: $a + b = b + a$ for every $a, b \in \mathbb{F}$.

The first four requirements mean that the elements of $\mathbb{F}$ form a **group** under addition; with the fifth requirement, a **commutative group** is obtained.

F6 Closure under multiplication: For every a and b in $\mathbb{F}$, $a \cdot b$ is also in $\mathbb{F}$.

F7 Multiplicative identity: There is an element in $\mathbb{F}$, which we denote as 1, such that $a \cdot 1 = 1 \cdot a = a$ for every $a \in \mathbb{F}$ with $a \neq 0$.

F8 Multiplicative inverse: For every $a \in \mathbb{F}$ with $a \neq 0$, there is an element $b \in \mathbb{F}$ such that $a \cdot b = b \cdot a = 1$. The element b is called the multiplicative inverse, or reciprocal, of a and is denoted as a^{-1}.

F9 Associativity: $(a \cdot b) \cdot c = a \cdot (b \cdot c)$ for every $a, b, c \in \mathbb{F}$.

F10 Commutativity: $a \cdot b = b \cdot a$ for every $a, b \in \mathbb{F}$.

Thus the non-zero elements of $\mathbb{F}$ form a commutative group under multiplication.

F11 Multiplication distributes over addition: $a \cdot (b + c) = a \cdot b + a \cdot c$

The field $\langle \mathbb{F}, +, \cdot \rangle$ is frequently referred to simply as $\mathbb{F}$. A field with q elements in it may be denoted as $\mathbb{F}_q$. □

Example 2.22 The field with two elements in it, $\mathbb{F}_2 = \mathbb{Z}_2 = GF(2)$ has the following addition and multiplication tables

+	0	1
0	0	1
1	1	0

"exclusive or"

·	0	1
0	0	0
1	0	1

"and"

The field $GF(2)$ is very important to our work, since it is the field where the operations involved in *binary* codes work. However, we shall have occasion to use many other fields as well. □

Example 2.23 The field $\mathbb{F}_5 = \mathbb{Z}_5 = GF(5)$ has the following addition and multiplication tables:

+	0	1	2	3	4
0	0	1	2	3	4
1	1	2	3	4	0
2	2	3	4	0	1
3	3	4	0	1	2
4	4	0	1	2	3

·	0	1	2	3	4
0	0	0	0	0	0
1	0	1	2	3	4
2	0	2	4	1	3
3	0	3	1	4	2
4	0	4	3	2	1

□

There are similarly constructed fields for every prime p, denoted by either $GF(p)$ or $\mathbb{F}_p$.

Example 2.24 A field with four elements can be constructed with the following operation tables:

+	0	1	2	3
0	0	1	2	3
1	1	0	3	2
2	2	3	0	1
3	3	2	1	0

·	0	1	2	3
0	0	0	0	0
1	0	1	2	3
2	0	2	3	1
3	0	3	1	2

(2.4)

This field is called $GF(4)$. Note that it is definitely *not* the same as $\langle \mathbb{Z}_4, +, \cdot \rangle$! (Why not?) We learn in Chapter 5 how to construct such a field.

□

Just as for groups, we can define the concepts of isomorphism and homomorphism. Two fields $\langle F, +, \cdot \rangle$ and $\langle \mathcal{F}, +, \cdot \rangle$ are (field) isomorphic if there exists a bijective function $\phi : F \to \mathcal{F}$ such that for every $a, b \in F$,

$$\phi(\underbrace{a+b}_{\substack{\text{operation}\\ \text{in } F}}) = \underbrace{\phi(a)+\phi(b)}_{\substack{\text{operation}\\ \text{in } \mathcal{F}}} \qquad \phi(\underbrace{ab}_{\substack{\text{operation}\\ \text{in } F}}) = \underbrace{\phi(a)\phi(b)}_{\substack{\text{operation}\\ \text{in } \mathcal{F}}}.$$

For example, the field $\mathcal{F}$ defined on the elements $\{-1, 1\}$ with operation tables

+	−1	1
−1	−1	1
1	1	−1

·	−1	1
−1	−1	−1
1	−1	1

is isomorphic to the field $GF(2)$ defined above, with ϕ mapping $0 \to -1$ and $1 \to 1$. Fields F and $\mathcal{F}$ are homomorphic if such a structure-preserving map ϕ exists which is not necessarily bijective.

2.4 Review of Linear Algebra

Linear block codes are based on concepts from linear algebra. In this section we review concepts from linear algebra which are immediately pertinent to our study of linear block codes.

Up to this point, our examples have dealt primarily with binary alphabets having the symbols $\{0, 1\}$. As your algebraic and coding-theoretic skills are deepened you will learn that larger alphabets are feasible and often desirable for good codes. However, rather than present the algebra first and the codes second, it seems pedagogically worthwhile to present the basic block coding concepts first using binary alphabets and introduce the algebra for larger alphabets later. For the sake of generality, we present definitions in terms of larger alphabets, but for the sake of concrete exposition we present examples in this chapter using binary alphabets. For now, understand that we will eventually need to deal with alphabets with more than two symbols. We denote the number of symbols in the alphabet by q, where $q = 2$ usually in this chapter. Furthermore, the alphabets we use usually form a finite **field**, denoted here as $\mathbb{F}_q$, which is briefly introduced in Box 12.1 and thoroughly developed in Chapter 5.

Definition 2.21 Let V be a set of elements called **vectors** and let $\mathbb{F}$ be a field of elements called **scalars**. An addition operation $+$ is defined between vectors. A scalar multiplication operation $\cdot$ (or juxtaposition) is defined such that for a scalar $a \in \mathbb{F}$ and a vector $\mathbf{v} \in V$, $a \cdot \mathbf{v} \in V$. Then V is a **vector space** over F if $+$ and $\cdot$ satisfy the following:

V1 V forms a commutative group under $+$.

V2 For any element $a \in \mathbb{F}$ and $\mathbf{v} \in V$, $a \cdot \mathbf{v} \in V$.

Combining V1 and V2, we must have $a \cdot \mathbf{v} + b \cdot \mathbf{w} \in V$ for every $\mathbf{v}, \mathbf{w} \in V$ and $a, b \in \mathbb{F}$.

V3 The operations $+$ and $\cdot$ distribute:

$$(a+b)\cdot\mathbf{v} = a\cdot\mathbf{v} + b\cdot\mathbf{v} \quad \text{and} \quad a\cdot(\mathbf{u}+\mathbf{v}) = a\cdot\mathbf{u} + a\cdot\mathbf{v}$$

for all scalars $a, b \in \mathbb{F}$ and vectors $\mathbf{v}, \mathbf{u} \in V$.

V4 The operation $\cdot$ is associative: $(a\cdot b)\cdot\mathbf{v} = a\cdot(b\cdot\mathbf{v})$ for all $a, b \in \mathbb{F}$ and $\mathbf{v} \in V$.

$\mathbb{F}$ is called the scalar field of the vector space V. □

Example 2.25

1. The set of n-tuples $(v_0, v_1, \ldots, v_{n-1})$, with elements $v_i \in \mathbb{R}$ forms a vector space which we denote as $\mathbb{R}^n$, with addition defined element-by-element,

$$(v_0, v_1, \ldots, v_{n-1}) + (u_0, u_1, \ldots, u_{n-1}) = (v_0 + u_0, v_1 + u_1, \ldots, v_{n-1} + u_{n-1}),$$

   and scalar multiplication defined by

$$a\cdot(v_0, v_1, \ldots, v_{n-1}) = (av_0, av_1, \ldots, av_{n-1}). \tag{2.5}$$

2. The set of n-tuples of $(v_0, v_1, \ldots, v_{n-1})$ with elements $v_i \in \mathbb{F}_2$ forms a vector space which we denote as $\mathbb{F}_2^n$. There are 2^n elements in the vector space $\mathbb{F}_2^n$. For $n = 3$, the elements of the vector space are

$$\begin{array}{cccc} (0,0,0) & (0,0,1) & (0,1,0) & (0,1,1) \\ (1,0,0) & (1,0,1) & (1,1,0) & (1,1,1) \end{array}$$

3. In general, the set $V = \mathbb{F}_q^n$ of n-tuples of elements of the field $\mathbb{F}_q$ with element-by-element addition and scalar multiplication as in (2.5) constitutes a vector space. We call an n-tuple of elements of $\mathbb{F}_q$ simply an n-vector.

□

Definition 2.22 Let $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_k$ be vectors in a vector space V and let $a_1, a_2, \ldots, a_k$ be scalars in $\mathbb{F}$. The operation

$$a_1\mathbf{v}_1 + a_2\mathbf{v}_2 + \cdots a_k\mathbf{v}_k$$

is said to be a **linear combination** of the vectors. □

Notationally, observe that the linear combination

$$a_1\mathbf{v}_1 + a_2\mathbf{v}_2 + \cdots a_k\mathbf{v}_k$$

can be obtained by forming a matrix G by stacking the vectors as columns,

$$G = \begin{bmatrix} \mathbf{v}_1 & \mathbf{v}_2 & \cdots & \mathbf{v}_k \end{bmatrix}$$

then forming the product with the column vector of coefficients:

$$a_1\mathbf{v}_1 + a_2\mathbf{v}_2 + \cdots a_k\mathbf{v}_k = \begin{bmatrix} \mathbf{v}_1 & \mathbf{v}_2 & \cdots & \mathbf{v}_k \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_k \end{bmatrix}. \tag{2.6}$$

Alternatively, the vectors $\mathbf{v}_i$ can be envisioned as *row* vectors and stacked as rows. The linear combination can be obtained by the product with a row vector of coefficients:

$$a_1\mathbf{v}_1 + a_2\mathbf{v}_2 + \cdots a_k\mathbf{v}_k = \begin{bmatrix} a_1 & a_2 & \cdots & a_k \end{bmatrix} \begin{bmatrix} \mathbf{v}_1 \\ \mathbf{v}_2 \\ \vdots \\ \mathbf{v}_k \end{bmatrix}.$$

Definition 2.23 Let V be a vector space. A set of vectors $G = \{\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_k\}$, each in V, is said to be a **spanning set** for V if every vector $\mathbf{v} \in V$ can be written as a linear combination of the vectors in G. That is, for every $\mathbf{v} \in V$, there exists a set of scalars $a_1, a_2, \ldots, a_k$ such that $\mathbf{v} = a_1\mathbf{v}_1 + a_2\mathbf{v}_2 + \cdots + a_k\mathbf{v}_k$.

For a set of vectors G, the set of vectors obtained from every possible linear combination of vectors in G is called the **span** of G, $\text{span}(G)$. □

It may be verified that the span of a set of vectors is itself a vector space. In light of the notation in (2.6), it is helpful to think of G as a *matrix* whose columns are the vectors $\mathbf{v}_i$, and not simply as a set of vectors. If G is interpreted as a matrix, we take $\text{span}(G)$ as the set of linear combinations of the columns of G. The space obtained by the linear combination of the columns of a matrix G is called the **column space** of G. The space obtained by the linear combination of the rows of a matrix G is called the **row space** of G.

It may be that there is redundancy in the vectors of a spanning set, in the sense that not all of them are needed to span the space because some of them can be expressed in terms of other vectors in the spanning set. In such a case, the vectors in the spanning set are not linearly independent:

Definition 2.24 A set of vectors $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_k$ is said to be **linearly dependent** if a set of scalars $\{a_1, a_2, \ldots, a_k\}$ exists, with not all $a_i = 0$ such that

$$a_1\mathbf{v}_1 + a_2\mathbf{v}_2 + \cdots + a_k\mathbf{v}_k = 0.$$

A set of vectors which is not linearly dependent is **linearly independent**. □

From the definition, if a set of vectors $\{\mathbf{v}_1, \ldots, \mathbf{v}_k\}$ is linearly independent and there exists a set of coefficients $\{a_1, \ldots, a_k\}$ such that

$$a_1\mathbf{v}_1 + a_2\mathbf{v}_2 + \cdots + a_k\mathbf{v}_k = \mathbf{0},$$

then it must be the case that $a_1 = a_2 = \cdots = a_k = 0$.

Definition 2.25 A spanning set for a vector space V that has the smallest possible number of vectors in it is called a **basis** for V.

The number of vectors in a basis for V is the **dimension** of V. □

Clearly the vectors in a basis must be linearly independent (or it would be possible to form a smaller set of vectors).

Example 2.26 Let $V = \mathbb{F}_2^4$, the set of binary 4-tuples and let

$$G = \left[\begin{bmatrix}1\\0\\1\\0\end{bmatrix}, \begin{bmatrix}0\\1\\1\\0\end{bmatrix}, \begin{bmatrix}1\\1\\0\\0\end{bmatrix}\right]$$

be a set of vectors.

Let $W = \text{span}(G)$;

$$W = \left\{\begin{bmatrix}0\\0\\0\\0\end{bmatrix}, \begin{bmatrix}1\\0\\1\\0\end{bmatrix}, \begin{bmatrix}0\\1\\1\\0\end{bmatrix}, \begin{bmatrix}1\\1\\0\\0\end{bmatrix}\right\}.$$

It can be verified that W is a vector space.

The set G is a spanning set for W, but it is not a spanning set for V. However, G is not a basis for W; the set G has some redundancy in it, since the third vector is a linear combination of the first two:

$$\begin{bmatrix}1\\1\\0\\0\end{bmatrix} = \begin{bmatrix}1\\0\\1\\0\end{bmatrix} + \begin{bmatrix}0\\1\\1\\0\end{bmatrix}.$$

The vectors in G are not linearly independent. The third vector in G can be removed, resulting in the set

$$G' = \left[\begin{bmatrix}1\\0\\1\\0\end{bmatrix}, \begin{bmatrix}0\\1\\1\\0\end{bmatrix}\right],$$

which has $\mathrm{span}(G') = W$.

No spanning set for W has fewer vectors in it than does G', so $\dim(W) = 2$. □

Theorem 2.7 *Let V be a k-dimensional vector space defined over a scalar field with a finite number of elements q in it. Then the number of elements in V is $|V| = q^k$.*

Proof Every vector $\mathbf{v}$ in V can be written as

$$\mathbf{v} = a_1\mathbf{v}_1 + a_2\mathbf{v}_2 + \cdots + a_k\mathbf{v}_k.$$

Thus the number of elements in V is the number of distinct k-tuples $(a_1, a_2, \ldots, a_k)$ that can be formed, which is q^k. □

Definition 2.26 Let V be a vector space over a scalar field $\mathbb{F}$ and let $W \subset V$ be a vector space. That is, for any $\mathbf{w}_1$ and $\mathbf{w}_2 \in W$, $a\mathbf{w}_1 + b\mathbf{w}_2 \in W$ for any $a, b \in \mathbb{F}$. Then W is called a **vector subspace** (or simply a subspace) of F. □

Example 2.27 The set W in Example 2.26 is a vector space, and is a subset of V. So W is a vector subspace of V.

Note, as specified by Theorem 2.7, that W has $4 = 2^2$ elements in it. □

We now augment the vector space with a new operator called the **inner product**, creating an inner product space.

Definition 2.27 Let $\mathbf{u} = (u_0, u_1, \ldots, u_{n-1})$ and $\mathbf{v} = (v_0, v_1, \ldots, v_{n-1})$ be vectors in a vector space V, where $u_i, v_i \in \mathbb{F}$. The **inner product** is a function the accepts two vectors and returns a scalar. It may be written as $\langle \mathbf{u}, \mathbf{v} \rangle$ or as $\mathbf{u} \cdot \mathbf{v}$. It is defined as

$$\langle \mathbf{u}, \mathbf{v} \rangle = \mathbf{u} \cdot \mathbf{v} = \sum_{i=0}^{n-1} u_i \cdot v_i.$$

□

It is straightforward to verify the following properties:

1. Commutativity: $\mathbf{u} \cdot \mathbf{v} = \mathbf{v} \cdot \mathbf{u}$
2. Associativity: $a \cdot (\mathbf{u} \cdot \mathbf{v}) = (a \cdot \mathbf{u}) \cdot \mathbf{v}$
3. Distributivity: $\mathbf{u} \cdot (\mathbf{v} + \mathbf{w}) = \mathbf{u} \cdot \mathbf{v} + \mathbf{u} \cdot \mathbf{w}$.

In physics and elementary calculus, the inner product is often called the dot product and is used to describe the physical concept of orthogonality. We similarly define orthogonality for the vector spaces of interest to us, even though there may not be a physical interpretation.

Definition 2.28 Two vectors $\mathbf{u}$ and $\mathbf{v}$ are said to be **orthogonal** if $\mathbf{u} \cdot \mathbf{v} = 0$. When $\mathbf{u}$ and $\mathbf{v}$ are orthogonal, this is sometimes denoted as $\mathbf{u} \perp \mathbf{v}$. □

Combining the idea of vector subspaces with orthogonality, we get the concept of a dual space:

Definition 2.29 Let W be a k-dimensional subspace of a vector space V. The set of all vectors $\mathbf{u} \in V$ which are orthogonal to all the vectors of W is called the **dual space** of W (sometimes called the **orthogonal complement** of W or **nullspace**), denoted $W^{\perp}$. (The symbol $W^{\perp}$ is sometimes pronounced "W perp," for "perpendicular.") That is,

$$W^{\perp} = \{\mathbf{u} \in V : \mathbf{u} \cdot \mathbf{w} = 0 \text{ for all } \mathbf{w} \in W\}.$$

□

Geometric intuition regarding dual spaces frequently may be gained by thinking in three-dimensional space $\mathbb{R}^3$ and letting W be a plane through the origin and $W^{\perp}$ a line through the origin orthogonal to the plane.

Example 2.28 Let $V = \mathbb{F}_2^4$ and let W be as in Example 2.26. Then it can be verified that

$$W^{\perp} = \left\{ \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ 1 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \right\}.$$

Note that

$$W^{\perp} = \operatorname{span}\left(\left\{ \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ 1 \\ 1 \\ 0 \end{bmatrix} \right\} \right)$$

and that $\dim(W^{\perp}) = 2$. □

This example demonstrates the important principle stated in the following theorem.

Theorem 2.8 *Let V be a finite-dimensional vector space of n-tuples, $\mathbb{F}^n$, with a subspace W of dimension k. Let $U = W^{\perp}$ be the dual space of W. Then*

$$\dim(W^{\perp}) = \dim(V) - \dim(W) = n - k.$$

Proof Let $\mathbf{g}_1, \mathbf{g}_2, \ldots, \mathbf{g}_k$ be a basis for W and let

$$G = \begin{bmatrix} \mathbf{g}_1 & \mathbf{g}_2 & \cdots & \mathbf{g}_k \end{bmatrix}.$$

This is a rank k matrix, meaning that the dimension of its column space is k and the dimension of its row space is k. Any vector $\mathbf{w} \in W$ is of the form $\mathbf{w} = G\mathbf{x}$ for some vector $\mathbf{x} \in \mathbb{F}^k$. Any vector $\mathbf{u} \in U$ must satisfy $\mathbf{u}^T G\mathbf{x} = 0$ for all $\mathbf{x} \in \mathbb{F}^k$. This implies that $\mathbf{u}^T G = 0$. (That is, $\mathbf{u}$ is orthogonal to every basis vector for W.)

Let $\{\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_r\}$ be a basis for $W^{\perp}$, then extend this to a basis for the whole n-dimensional space, $\{\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_r, \mathbf{f}_1, \mathbf{f}_2, \ldots, \mathbf{f}_{n-r}\}$. Every vector $\mathbf{v}$ in the row space of G

is expressible (not necessarily uniquely) as $\mathbf{v} = \mathbf{b}^T G$ for some vector $\mathbf{b} \in V$. But since $\{\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_r, \mathbf{f}_1, \mathbf{f}_2, \ldots, \mathbf{f}_{n-r}\}$ spans V, $\mathbf{b}$ must be a linear combination of these vectors:

$$\mathbf{b} = a_1\mathbf{h}_1 + a_2\mathbf{h}_2 + \cdots a_r\mathbf{h}_r + a_{r+1}\mathbf{f}_1 + \cdots + a_n\mathbf{f}_{n-r}.$$

So a vector $\mathbf{v}$ in the row space of G can be written as

$$\mathbf{v} = a_1\mathbf{h}_1^T G + a_2\mathbf{h}_2^T G + \cdots + a_n\mathbf{f}_{n-r}^T G,$$

from which we observe that the row space of G is spanned by the vectors

$$\{\mathbf{h}_1^T G, \mathbf{h}_2^T G, \ldots, \mathbf{h}_r^T G, \mathbf{f}_1^T G, \ldots, \mathbf{f}_{n-r}^T G\}.$$

The vectors $\{\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_r\}$ are in $W^\perp$, so that $\mathbf{h}_i^T G = \mathbf{0}$ for $i = 1, 2, \ldots, r$. The remaining vectors $\{\mathbf{f}_1^T G, \ldots, \mathbf{f}_{n-r}^T G\}$ remain to span the k-dimensional row space of G. Hence, we must have $n - r \geq k$. Furthermore, these vectors are linearly independent, because if there is a set of coefficients $\{a_i\}$ such that

$$a_1(\mathbf{f}_1^T G) + \cdots a_{n-r}(\mathbf{f}_{n-r}^T G) = \mathbf{0},$$

then

$$(a_1\mathbf{f}_1^T + \cdots + a_{n-r}\mathbf{f}_{n-r}^T)G = \mathbf{0}.$$

But the vectors $\mathbf{f}_i$ are not in $W^\perp$, so we must have

$$a_1\mathbf{f}_1^T + \cdots + a_{n-r}\mathbf{f}_{n-r}^T = 0.$$

Since the vectors $\{\mathbf{f}_i\}$ are linearly independent, we must have $a_1 = a_2 = \cdots = a_{n-r} = 0$. Therefore, we must have $\dim \operatorname{span}(\{\mathbf{f}_1^T G, \ldots, \mathbf{f}_{n-r}^T G\}) = k$, so $n - r = k$. □

2.5 Exercises

2.1 A group can be constructed by using the rotations and reflections of a regular pentagon into itself. The group operator is "followed by" (e.g., a reflection ρ "followed by" a rotation r). This is a permutation group, as in Example 2.10.

(a) How many elements are in this group?
(b) Construct the group (i.e., show the "multiplication table" for the group).
(c) Is it an Abelian group?
(d) Find a subgroup with five elements and a subgroup with two elements.
(e) Are there any subgroups with four elements? Why?

2.2 Show that only one group exists with three elements "up to isomorphism." That is, there is only one way of filling out a binary operation table that satisfies all the requirements of a group.

2.3 Show that there are two groups with four elements, up to isomorphism. One of these groups is isomorphic to $\mathbb{Z}_4$. The other is called the Klein 4-group.

2.4 Prove that in a group G, the identity element is unique.

2.5 Prove that in a group G, the inverse a^{-1} of an element a is unique.

2.6 Let $G = \langle \mathbb{Z}_{16}, + \rangle$, the group of integers modulo 16. Let $H = \langle 4 \rangle$, the cyclic group generated by the element $4 \in G$.

(a) List the elements of H.
(b) Determine the cosets of G/H.
(c) Draw the "addition" table for G/H.

(d) To what group is G/H isomorphic?

2.7 Show that if G is an Abelian group and $\mathcal{G}$ is isomorphic to G, then $\mathcal{G}$ is also Abelian.

2.8 Let G be a cyclic group and let $\mathcal{G}$ be isomorphic to G. Show that $\mathcal{G}$ is also a cyclic group.

2.9 Let G be a cyclic group with generator a and let $\mathcal{G}$ be a group isomorphic to G. If $\phi : G \to \mathcal{G}$ is an isomorphism, show that for every $x \in G$, $\phi(x)$ is completely determined by $\phi(a)$.

2.10 An automorphism of a group G is an isomorphism of the group with itself, $\phi : G \to G$. Using Exercise 2.9, how many automorphisms are there of $\mathbb{Z}_2$? of $\mathbb{Z}_6$? of $\mathbb{Z}_8$? of $\mathbb{Z}_{17}$?

2.11 [106] Let G be a finite Abelian group of order n, and let r be a positive integer relatively prime to n (i.e., they have no factors in common except 1). Show that the map $\phi_r : G \to G$ defined by $\phi_r(a) = a^r$ is an isomorphism of G onto itself (an automorphism). Deduce that the equation $x^r = a$ always has a unique solution in a finite Abelian group G if r is relatively prime to the order of G.

2.12 Show that the induced operation defined in Definition 2.16 is well defined if G is commutative.

2.13 Prove Theorem 2.5.

2.14 Show for the lattice with coset decomposition in Figure 2.2 that $\Lambda/\Lambda' \cong \mathbb{Z}_2 \times \mathbb{Z}_2$.

2.15 Let G be a cyclic group with generator a and let $\phi : G \to G'$ be a homomorphism onto a group G'. Show that the value of ϕ on every element of G is determined by the value of the homomorphism $\phi(a)$.

2.16 Let G be a group and let $a \in G$. Let $\phi : \mathbb{Z} \to G$ be defined by $\phi(n) = a^n$. Show that ϕ is a homomorphism. Describe the image of ϕ in G.

2.17 Show that if G, G' and G'' are groups and $\phi : G \to G'$ and $\psi : G' \to G''$ are homomorphisms, then the composite function $\psi \circ \phi : G \to G''$ is a homomorphism.

2.18 Consider the set $S = \{0, 1, 2, 3\}$ with the operations

+	0	1	2	3
0	0	1	2	3
1	1	2	3	0
2	2	3	0	1
3	3	0	1	2

·	0	1	2	3
0	0	0	0	0
1	0	1	2	3
2	0	2	3	1
3	0	3	1	2

Is this a field? If not, why not?

2.19 Construct the addition and multiplication tables for $\langle \mathbb{Z}_4, +, \cdot \rangle$ and compare to the tables in (2.4). Does $\langle \mathbb{Z}_4, +, \cdot \rangle$ form a field?

2.20 Use the representation of $GF(4)$ in (2.4) to solve the following pair of equations:

$$2x + y = 3$$

$$x + 2y = 3.$$

2.21 Show that the vectors in a basis must be linearly independent.

2.22 Let $G = \{\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_k\}$ be a basis for a vector space V. Show that for every vector $\mathbf{v} \in V$, there is a *unique* representation for $\mathbf{v}$ as a linear combination of the vectors in G.

2.23 Show that if $\mathbf{u}$ is orthogonal to every basis vector for W, then $\mathbf{u} \perp W$.

2.24 The dual space $W^\perp$ of W is the set of vectors which are orthogonal to every vector in W. Show that the dual space $W^\perp$ of a vector space $W \subset V$ is a vector subspace of V.

2.25 Show that the set of binary polynomials (i.e., polynomials with binary coefficients, with operations in $GF(2)$) with degree less than r forms a vector space over $GF(2)$ with dimension r.

2.26 What is the dimension of the vector space spanned by the vectors

$$\{(1, 1, 0, 1, 0, 1), (0, 1, 0, 1, 1, 1), (1, 1, 0, 0, 1, 1), (0, 1, 1, 1, 0, 1), (1, 0, 0, 0, 0, 0)\}$$

over $GF(2)$?

2.27 Find a basis for the dual space to the vector space spanned by

$$\{(1,1,1,0,0),(0,1,1,1,0),(0,0,1,1,1)\}.$$

2.28 Let $S = \{\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_n\}$ be an arbitrary basis for the vector space V. Let $\mathbf{v}$ be an arbitrary vector in $\mathbf{v}$; it may be expressed as the linear combination

$$\mathbf{v} = a_1\mathbf{v}_1 + a_2\mathbf{v}_2 + \cdots + a_n\mathbf{v}_n.$$

Develop an expression for computing the coefficients $\{a_i\}$ in this representation.

2.29 Is it true that if $\mathbf{x}$, $\mathbf{y}$ and $\mathbf{z}$ are linearly independent vectors over $GF(q)$ then so also are $\mathbf{x}+\mathbf{y}$, $\mathbf{y}+\mathbf{z}$ and $\mathbf{z}+\mathbf{x}$?

2.30 Let V be a vector space and let $\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_k \in V$. Show that $\text{span}(\{\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_k\})$ is a vector space.

2.31 Let U and V be linear subspaces of a vector space S. Show that the intersection $U \cap V$ is also a subspace of S.

2.32 Let $G = \langle \mathbb{Z}, + \rangle$ and let $\mathcal{G} = \langle \mathbb{Z}_n, + \rangle$. Let $\phi : G \to \mathcal{G}$ be defined by $\phi(a) = a \mod n$. Show that $\phi(a+b) = \phi(a) + \phi(b)$.

2.33 In this exercise, let $\mathbf{x} \cdot \mathbf{y}$ denote the inner product *over the real numbers*. Let $\mathbf{x}$ and $\mathbf{y}$ be vectors of length n with elements from the set $\{-1, 1\}$. Show that $d_H(\mathbf{x}, \mathbf{y}) = \frac{n - \mathbf{x}\cdot\mathbf{y}}{2}$.

2.6 References

Group theory is presented in a variety of books; see, for example, [31] or [106]. Our summary of linear algebra was drawn from [373, 33] and [246]. Some of the exercises were drawn from [373] and [106].

Chapter 3

Linear Block Codes

3.1 Basic Definitions

Consider a source that produces symbols from an alphabet $\mathcal{A}$ having q symbols, where $\mathcal{A}$ forms a field. We refer to a tuple $(c_0, c_1, \ldots, c_{n-1}) \in \mathcal{A}^n$ with n elements as an n-vector or an n-tuple.

Definition 3.1 An (n, k) **block code** $\mathcal{C}$ over an alphabet of q symbols is a set of q^k n-vectors called **codewords** or **code vectors**. Associated with the code is an **encoder** which maps a **message**, a k-tuple $\mathbf{m} \in \mathcal{A}^k$, to its associated codeword. □

For a block code to be useful for error correction purposes, there should be a one-to-one correspondence between a message $\mathbf{m}$ and its codeword $\mathbf{c}$. However, for a given code $\mathcal{C}$, there may be more than one possible way of mapping messages to codewords.

A block code can be represented as an exhaustive list, but for large k this would be prohibitively complex to store and decode. The complexity can be reduced by imposing some sort of mathematical structure on the code. The most common requirement is *linearity*.

Definition 3.2 A block code $\mathcal{C}$ over a field $\mathbb{F}_q$ of q symbols of length n and q^k codewords is a q-ary **linear** (n, k) code if and only if its q^k codewords form a k-dimensional vector subspace of the vector space of all the n-tuples $\mathbb{F}_q^n$. The number n is said to be the **length** of the code and the number k is the **dimension** of the code. The **rate** of the code is $R = k/n$. □

In some literature, an (n, k) linear code is denoted using square brackets, $[n, k]$.

For a linear code, the sum of any two codewords is also a codeword. More generally, any linear combination of codewords is a codeword.

Definition 3.3 The **Hamming weight** $\text{wt}(\mathbf{c})$ of a codeword $\mathbf{c}$ is the number of nonzero components of the codeword. The minimum weight $w_{\min}$ of a code $\mathcal{C}$ is the smallest Hamming weight of any nonzero codeword: $w_{\min} = \min_{\mathbf{c} \in \mathcal{C}, \mathbf{c} \neq 0} \text{wt}(\mathbf{c})$. □

Recall from Definition 1.3 that the minimum distance is the smallest Hamming distance between any two codewords of the code.

Theorem 3.1 *For a linear code $\mathcal{C}$, the minimum distance $d_{\min}$ satisfies $d_{\min} = w_{\min}$. That is, the minimum distance of a linear block code is equal to the minimum weight of its nonzero codewords.*

Proof The result relies on the fact that linear combinations of codewords are codewords. If $\mathbf{c}_i$ and $\mathbf{c}_j$ are codewords, then so is $\mathbf{c}_i - \mathbf{c}_j$. The distance calculation can then be "translated to the origin":

$$d_{\min} = \min_{\mathbf{c}_i, \mathbf{c}_j \in \mathcal{C}, \mathbf{c}_i \neq \mathbf{c}_j} d_H(\mathbf{c}_i, \mathbf{c}_j) = \min_{\mathbf{c}_i, \mathbf{c}_j \in \mathcal{C}, \mathbf{c}_i \neq \mathbf{c}_j} d_H(\mathbf{c}_i - \mathbf{c}_j, \mathbf{c}_j - \mathbf{c}_j) = \min_{\mathbf{c} \in \mathcal{C}, \mathbf{c} \neq 0} w(\mathbf{c}).$$

□

An (n, k) code with minimum distance $d_{\min}$ is sometimes denoted as an $(n, k, d_{\min})$ code.

As described in Section 1.8.1, the random error correcting capability of a code with minimum distance $d_{\min}$ is $t = \lfloor (d_{\min} - 1)/2 \rfloor$.

3.2 The Generator Matrix Description of Linear Block Codes

Since a linear block code $\mathcal{C}$ is a k-dimensional vector space, there exist k linearly independent vectors which we designate as $\mathbf{g}_0, \mathbf{g}_1, \ldots, \mathbf{g}_{k-1}$ such that every codeword $\mathbf{c}$ in $\mathcal{C}$ can be represented as a linear combination of these vectors,

$$\mathbf{c} = m_0\mathbf{g}_0 + m_1\mathbf{g}_1 + \cdots + m_{k-1}\mathbf{g}_{k-1}, \tag{3.1}$$

where $m_i \in \mathbb{F}_q$. (For binary codes, all arithmetic in (3.1) is done modulo 2; for codes of $\mathbb{F}_q$, the arithmetic is done in $\mathbb{F}_q$.) Thinking of the $\mathbf{g}_i$ as *row* vectors[1] and stacking up, we form the $k \times n$ matrix G,

$$G = \begin{bmatrix} \mathbf{g}_0 \\ \mathbf{g}_1 \\ \vdots \\ \mathbf{g}_{k-1} \end{bmatrix}.$$

Let

$$\mathbf{m} = \begin{bmatrix} m_0 & m_1 & \cdots & m_{k-1} \end{bmatrix}.$$

Then (3.1) can be written as

$$\mathbf{c} = \mathbf{m}G, \tag{3.2}$$

and every codeword $\mathbf{c} \in \mathcal{C}$ has such a representation for some vector $\mathbf{m}$. Since the rows of G generate (or span) the (n, k) linear code $\mathcal{C}$, G is called a **generator matrix** for $\mathcal{C}$. Equation (3.2) can be thought of as an encoding operation for the code $\mathcal{C}$. Representing the code thus requires storing only k vectors of length n (rather than the q^k vectors that would be required to store all codewords of a nonlinear code).

Note that the representation of the code provided by G is not unique. From a given generator G, another generator G' can be obtained by performing row operations (nonzero linear combinations of the rows). Then an encoding operation defined by $\mathbf{c} = \mathbf{m}G'$ maps the message $\mathbf{m}$ to a codeword in $\mathcal{C}$, but it is not necessarily the same codeword that would be obtained using the generator G.

Example 3.1 The (7,4) Hamming code of Section 1.9 has the generator matrix

$$G = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix}. \tag{3.3}$$

To encode the message $\mathbf{m} = \begin{bmatrix} 1 & 0 & 0 & 1 \end{bmatrix}$, add the first and fourth rows of G (modulo 2) to obtain

$$\mathbf{c} = \begin{bmatrix} 1 & 1 & 0 & 0 & 1 & 0 & 1 \end{bmatrix}.$$

[1] Most signal processing and communication work employs column vectors by convention. However, a venerable tradition in coding theory has employed row vectors and we adhere to that through most of the book.

Another generator is obtained by replacing the first row of G with the sum of the first two rows of G:

$$G' = \begin{bmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix}.$$

For $\mathbf{m}$ the corresponding codeword is

$$\mathbf{c}' = \mathbf{m}G' = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

This is a different codeword than $\mathbf{c}$, but is still a codeword in C. □

Definition 3.4 Let C be an (n, k) block code (not necessarily linear). An encoder is **systematic** if the message symbols $m_0, m_1, \ldots, m_{k-1}$ may be found explicitly and unchanged in the codeword. That is, there are coordinates $i_0, i_1, \ldots, i_{k-1}$ (which are most frequently sequential, $i_0, i_0 + 1, \ldots, i_0 + k - 1$) such that $c_{i_0} = m_0, c_{i_1} = m_1, \ldots, c_{i_{k-1}} = m_{k-1}$.

For a linear code, the generator for a systematic encoder is called a *systematic generator*. □

It should be emphasized that being systematic is a property of the encoder and not a property of the code. For a linear block code, the encoding operation represented by G is systematic if an identity matrix can be identified among the rows of G. Neither the generator G nor G' of Example 3.1 are systematic.

Frequently, a systematic generator is written in the form

$$G = \begin{bmatrix} P & I_k \end{bmatrix} = \begin{bmatrix} p_{0,0} & p_{0,1} & \cdots & p_{0,n-k-1} & 1 & 0 & 0 & \cdots & 0 \\ p_{1,0} & p_{1,1} & \cdots & p_{1,n-k-1} & 0 & 1 & 0 & \cdots & 0 \\ p_{2,0} & p_{2,1} & \cdots & p_{2,n-k-1} & 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & & \vdots & & & & \vdots & \\ p_{k-1,0} & p_{k-1,1} & \cdots & p_{k-1,n-k-1} & 0 & 0 & 0 & \cdots & 1 \end{bmatrix}, \quad (3.4)$$

where I_k is the $k \times k$ identity matrix and P is a $k \times (n - k)$ matrix which generates *parity* symbols. The encoding operation is

$$\mathbf{c} = \mathbf{m}\begin{bmatrix} P & I_k \end{bmatrix} = \begin{bmatrix} \mathbf{m}P & \mathbf{m} \end{bmatrix}.$$

The codeword is divided into two parts: the part $\mathbf{m}$ consists of the message symbols, and the part $\mathbf{m}P$ consists of the **parity check symbols**.

Performing elementary row operations (replacing a row with linear combinations of some rows) does not change the row span, so that the same code is produced. If two columns of a generator are interchanged, then the corresponding positions of the code are changed, but the distance structure of the code is preserved.

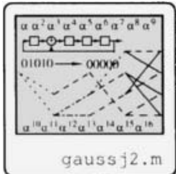
Definition 3.5 Two linear codes which are the same except for a permutation of the components of the code are said to be **equivalent** codes. □

Let G and G' be generator matrices of equivalent codes. Then G and G' are related by the following operations:

1. Column permutations,
2. Elementary row operations.

Given an arbitrary generator G, it is possible to put it into the form (3.4) by performing Gaussian elimination with pivoting.

Example 3.2 For G of (3.3), an equivalent generator in systematic form is



$$G'' = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}. \tag{3.5}$$

For the Hamming code with this generator, let the message be $\mathbf{m} = [m_0, m_1, m_2, m_3]$ and let the corresponding codeword be $\mathbf{c} = [c_0, c_1, \ldots, c_6]$. Then the parity bits are obtained by

$$\begin{aligned} c_0 &= m_0 + m_2 + m_3 \\ c_1 &= m_0 + m_1 + m_2 \\ c_2 &= m_1 + m_2 + m_3 \end{aligned}$$

and the systematically encoded bits are $c_3 = m_0$, $c_4 = m_1$, $c_5 = m_2$ and $c_6 = m_3$. □

3.2.1 Rudimentary Implementation

Implementing encoding operations for binary codes is straightforward, since the multiplication operation corresponds to the `and` operation and the addition operation corresponds to the `exclusive-or` operation. For software implementations, encoding is accomplished by straightforward matrix/vector multiplication. This can be greatly accelerated for binary codes by packing several bits into a single word (e.g., 32 bits in an `unsigned int` of four bytes). The multiplication is then accomplished using the bit `exclusive-or` operation of the language (e.g., the ^ operator of C). Addition must be accomplished by looping through the bits, or by precomputing bit sums and storing them in a table, where they can be immediately looked up.

3.3 The Parity Check Matrix and Dual Codes

Since a linear code $\mathcal{C}$ is a k-dimensional vector subspace of $\mathbb{F}_q^n$, by Theorem 2.8 there must be a dual space to $\mathcal{C}$ of dimension $n - k$.

Definition 3.6 The dual space to an (n, k) code $\mathcal{C}$ of dimension k is the $(n, n - k)$ **dual code** of $\mathcal{C}$, denoted by $\mathcal{C}^\perp$. A code $\mathcal{C}$ such that $\mathcal{C} = \mathcal{C}^\perp$ is called a **self-dual code**. □

As a vector space, $\mathcal{C}^\perp$ has a basis which we denote by $\{\mathbf{h}_0, \mathbf{h}_1, \ldots, \mathbf{h}_{n-k-1}\}$. We form a matrix H using these basis vectors as rows:

$$H = \begin{bmatrix} \mathbf{h}_0 \\ \mathbf{h}_1 \\ \vdots \\ \mathbf{h}_{n-k-1} \end{bmatrix}.$$

This matrix is known as the **parity check matrix** for the code $\mathcal{C}$. The generator matrix and the parity check matrix for a code satisfy

$$\boxed{GH^T = \mathbf{0}}.$$

The parity check matrix has the following important property:

Theorem 3.2 *Let C be an (n, k) linear code over $\mathbb{F}_q$ and let H be a parity check matrix for C. A vector $\mathbf{v} \in \mathbb{F}_q^n$ is a codeword if and only if*

$$\mathbf{v}H^T = \mathbf{0}.$$

That is, the codewords in C lie in the (left) nullspace of H.

(Sometimes additional linearly dependent rows are included in H, but the same result still holds.)

Proof Let $\mathbf{c} \in C$. By the definition of the dual code, $\mathbf{h} \cdot \mathbf{c} = \mathbf{0}$ for all $\mathbf{h} \in C^\perp$. Any row vector $\mathbf{h} \in C^\perp$ can be written as $\mathbf{h} = \mathbf{x}H$ for some vector $\mathbf{x}$. Since $\mathbf{x}$ is arbitrary, and in fact can select individual rows of H, we must have $\mathbf{c}\mathbf{h}_i^T = 0$ for $i = 0, 1, \ldots, n-k-1$. Hence $\mathbf{c}H^T = 0$.

Conversely, suppose that $\mathbf{v}H^T = 0$. Then $\mathbf{v}\mathbf{h}_i^T = 0$ for $i = 0, 1, \ldots, n-k-1$, so that $\mathbf{v}$ is orthogonal to the basis of the dual code, and hence orthogonal to the dual code itself. Hence, $\mathbf{v}$ must be in the code C. □

When G is in systematic form (3.4), a parity check matrix is readily determined:

$$H = \begin{bmatrix} I_{n-k} & -P^T \end{bmatrix}. \tag{3.6}$$

(For the field $\mathbb{F}_2$, $-1 = 1$, since 1 is its own additive inverse.) Frequently, a parity check matrix for a code is obtained by finding a generator matrix in systematic form and employing (3.6).

Example 3.3 For the systematic generator G'' of (3.5), a parity check matrix is

$$H = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix}. \tag{3.7}$$

It can be verified that $G''H^T = \mathbf{0}$. Furthermore, even though G is not in systematic form, it still generates the same code so that $GH^T = \mathbf{0}$. H is a generator for a $(7, 3)$ code, the dual code to the $(7, 4)$ Hamming code. □

The condition $\mathbf{c}H^T = \mathbf{0}$ imposes linear constraints among the bits of $\mathbf{c}$ called the **parity check equations**.

Example 3.4 The parity check matrix of (3.7) gives rise to the equations

$$\begin{aligned} c_0 + c_3 + c_5 + c_6 &= 0 \\ c_1 + c_3 + c_4 + c_5 &= 0 \\ c_2 + c_4 + c_5 + c_6 &= 0 \end{aligned}$$

or, equivalently, some equations for the parity symbols are

$$\begin{aligned} c_0 &= c_3 + c_5 + c_6 \\ c_1 &= c_3 + c_4 + c_5 \\ c_2 &= c_4 + c_5 + c_6. \end{aligned}$$

□

A parity check matrix for a code (whether systematic or not) provides information about the minimum distance of the code.

Theorem 3.3 *Let a linear block code $\mathcal{C}$ have a parity check matrix H. The minimum distance d_{min} of $\mathcal{C}$ is equal to the smallest positive number of columns of H which are linearly dependent. That is, all combinations of $d_{min} - 1$ columns are linearly independent, so there is some set of d_{min} columns which are linearly dependent.*

Proof Let the columns of H be designated as $\mathbf{h}_0, \mathbf{h}_1, \ldots, \mathbf{h}_{n-1}$. Then since $\mathbf{c}H^T = \mathbf{0}$ for any codeword $\mathbf{c}$, we have

$$\mathbf{0} = c_0\mathbf{h}_0 + c_1\mathbf{h}_1 + \cdots + c_{n-1}\mathbf{h}_{n-1}.$$

Let $\mathbf{c}$ be the codeword of smallest weight, $w = \text{wt}(\mathbf{c}) = d_{\min}$, with nonzero positions only at indices $i_1, i_2, \ldots, i_w$. Then

$$c_{i_1}\mathbf{h}_{i_1} + c_{i_2}\mathbf{h}_{i_2} + \cdots c_{i_w}\mathbf{h}_{i_w} = 0.$$

Clearly, the columns of H corresponding to the elements of $\mathbf{c}$ are linearly dependent.

On the other hand, if there were a linearly dependent set of $u < w$ columns of H, then there would be a codeword of weight u. □

Example 3.5 For the parity check matrix H of (3.7), the parity check condition is

$$\mathbf{c}H^T = [c_0, c_1, c_2, c_3, c_4, c_5, c_6] \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix}$$

$$= c_0[1,0,0] + c_1[0,1,0] + c_2[0,0,1] + c_3[1,1,0] + c_4[0,1,1] + c_5[1,1,1] + c_6[1,0,1]$$

The first, second, and fourth rows of H^T are linearly dependent, and no fewer rows of H^T are linearly dependent. □

3.3.1 Some Simple Bounds on Block Codes

Theorem 3.3 leads to a relationship between $d_{\min}$, n, and k:

Theorem 3.4 *The* **Singleton bound**. *The minimum distance for an (n, k) linear code is bounded by*

$$d_{\min} \leq n - k + 1. \tag{3.8}$$

Note: Although this bound is proved here for linear codes, it is also true for nonlinear codes. (See [220].)

Proof An (n, k) linear code has a parity check matrix with $n - k$ linearly independent rows. Since the row rank of a matrix is equal to its column rank, $\text{rank}(H) = n - k$. Any collection of $n - k + 1$ columns must therefore be linearly dependent. Thus by Theorem 3.3, the minimum distance cannot be larger than $n - k + 1$. □

A code for which $d_{\min} = n - k + 1$ is called a **maximum distance separable** (MDS) code.

Thinking geometrically, around each code point is a cloud of points corresponding to non-codewords. (See Figure 1.17.) For a q-ary code, there are $(q - 1)n$ vectors at a

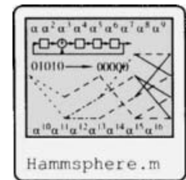
Hamming distance 1 away from a codeword, $(q-1)^2\binom{n}{2}$ vectors at a Hamming distance 2 away from a codeword and, in general, $(q-1)^l\binom{n}{l}$ vectors at a Hamming distance l from a codeword.

Example 3.6 Let C be a code of length $n = 4$ over $GF(3)$, so $q = 3$. Then the vectors at a Hamming distance of 1 from the [0, 0, 0, 0] codeword are

$$[1,0,0,0],[0,1,0,0],[0,0,1,0],[0,0,0,1]$$
$$[2,0,0,0],[0,2,0,0],[0,0,2,0],[0,0,0,2].$$

□

The vectors at Hamming distances $\leq t$ away from a codeword form a "sphere" called the **Hamming sphere** of radius t. The number of codewords in a Hamming sphere up to radius t for a code of length n over an alphabet of q symbols is denoted $V_q(n, t)$, where



$$V_q(n,t) = \sum_{j=0}^{t} \binom{n}{j}(q-1)^j. \tag{3.9}$$

The bounded distance decoding sphere of a codeword is the Hamming sphere of radius $t = \lfloor (d_{\min} - 1)/2 \rfloor$ around the codeword. Equivalently, a code whose random error correction capability is t must have a minimum distance between codewords satisfying $d_{\min} \geq 2t + 1$.

The **redundancy** of a code is essentially the number of parity symbols in a codeword. More precisely we have

$$r = n - \log_q M,$$

where M is the number of codewords. For a linear code we have $M = q^k$, so $r = n - k$.

Theorem 3.5 (**The Hamming Bound**) *A t-random error correcting q-ary code C must have redundancy r satisfying*

$$r \geq \log_q V_q(n, t).$$

Proof Each of M spheres in C has radius t. The spheres do not overlap, or else it would not be possible to decode t errors. The total number of points enclosed by the spheres must be $\leq q^n$. We must have

$$MV_q(n,t) \leq q^n$$

so

$$q^n/M \geq V_q(n,t),$$

from which the result follows by taking $\log_q$ of both sides. □

A code satisfying the Hamming bound with equality is said to be a **perfect code.** Actually, being perfect codes does not mean the codes are the best possible codes; it is simply a designation regarding how points fall in the Hamming spheres. The set of perfect codes is actually quite limited. It has been proved (see [220]) that the entire set of perfect codes is:

1. The set of all n-tuples, with minimum distance = 1 and $t = 0$.
2. Odd-length binary repetition codes.
3. Binary Hamming codes (linear) or other nonlinear codes with equivalent parameters.
4. The binary (23, 12, 7) Golay code G_{23}.
5. The ternary (i.e., over $GF(3)$) (11, 6, 5) code G_{11} and the (23,11,5) code G_{23}. These codes are discussed in Chapter 8.

Box 3.1: Error Correction and Least-Squares

The hard-input decoding problem is: Given $\mathbf{r} = \mathbf{m}G + \mathbf{e}$, compute $\mathbf{m}$. Readers familiar with least-squares problems (see, e.g., [246]) will immediately recognize the structural similarity of the decoding problem to least-squares. If a least-squares solution were possible, the decoded value could be written as

$$\hat{\mathbf{m}} = \mathbf{r}G^T(GG^T)^{-1},$$

reducing the decoding problem to numerical linear algebra. Why cannot least-squares techniques be employed here? In the first place, it must be recalled that in least squares, the distance function d is induced from an inner product, $d(x, y) = \langle x - y, x - y\rangle^{1/2}$, while in our case the distance function is the Hamming distance — which measures the likelihood — which is not induced from an inner product. The Hamming distance is a function $\mathbb{F}_q^n \times \mathbb{F}_q^n \to \mathbb{N}$, while the inner product is a function $\mathbb{F}_q^n \times \mathbb{F}_q^n \to \mathbb{F}_q^n$: the codomains of the Hamming distance and the inner product are different.

3.4 Error Detection and Correction over Hard-Input Channels

Definition 3.7 Let $\mathbf{r}$ be an n-vector over $\mathbb{F}_q$ and let H be a parity check matrix for a code $\mathcal{C}$. The vector

$$\boxed{\mathbf{s} = \mathbf{r}H^T} \tag{3.10}$$

is called the **syndrome** of $\mathbf{r}$. □

By Theorem 3.2, $\mathbf{s} = \mathbf{0}$ if and only if $\mathbf{r}$ is a codeword of $\mathcal{C}$. In medical terminology, a syndrome is a pattern of symptoms that aids in diagnosis; here $\mathbf{s}$ aids in diagnosing if $\mathbf{r}$ is a codeword or has been corrupted by noise. As we will see, it also aids in determining what the error is.

3.4.1 Error Detection

The syndrome can be used as an *error detection* scheme. Suppose that a codeword $\mathbf{c}$ in a binary linear block code $\mathcal{C}$ over $\mathbb{F}_q$ is transmitted through a hard channel (e.g., a binary code over a BSC) and that the n-vector $\mathbf{r}$ is received. We can write

$$\mathbf{r} = \mathbf{c} + \mathbf{e},$$

where the arithmetic is done in $\mathbb{F}_q$, and where $\mathbf{e}$ is the *error vector*, being 0 in precisely the locations where the channel does not introduce errors. The received vector $\mathbf{r}$ could be any of the vectors in $\mathbb{F}_q^n$, since any error pattern is possible. Let H be a parity check matrix for $\mathcal{C}$. Then the syndrome

$$\mathbf{s} = \mathbf{r}H^T = (\mathbf{c} + \mathbf{e})H^T = \mathbf{e}H^T.$$

From Theorem 3.2, $\mathbf{s} = \mathbf{0}$ if $\mathbf{r}$ is a codeword. However, if $\mathbf{s} \neq \mathbf{0}$, then an error condition has been *detected*: we do not know what the error is, but we do know that an error has occurred.

3.4.2 Error Correction: The Standard Array

Let us now consider one method of decoding linear block codes transmitted through a hard channel using maximum likelihood (ML) decoding. As discussed in Section 1.8.1, ML

decoding of a vector $\mathbf{r}$ consists of choosing the codeword $\mathbf{c} \in \mathcal{C}$ that is closest to $\mathbf{r}$ in Hamming distance. That is,

$$\hat{\mathbf{c}} = \arg\min_{\mathbf{c}\in\mathcal{C}} d_H(\mathbf{c}, \mathbf{r}).$$

Let the set of codewords in the code be $\{\mathbf{c}_0, \mathbf{c}_1, \ldots, \mathbf{c}_{M-1}\}$, where $M = q^k$. Let us take $\mathbf{c}_0 = \mathbf{0}$, the all-zero codeword. Let V_i denote the set of n-vectors which are closer to the codeword $\mathbf{c}_i$ than to any other codeword. (Vectors which are equidistant to more than one codeword are assigned into a set V_i at random.) The sets $\{V_i, i = 0, 1, \ldots, M-1\}$ partition the space of n-vectors into M disjoint subsets. If $\mathbf{r}$ falls in the set V_i, then, being closer to $\mathbf{c}_i$ than to any other codeword, $\mathbf{r}$ is decoded as $\mathbf{c}_i$. So, decoding can be accomplished if the V_i sets can be found.

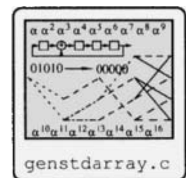
The *standard array* is a representation of the partition $\{V_i\}$. It is a two-dimensional array in which the columns of the array represent the V_i. The standard array is built as follows. First, every codeword $\mathbf{c}_i$ belongs in its own set V_i. Writing down the set of codewords thus gives the first row of the array. Now, from the remaining vectors in $\mathbb{F}_q^2$, find the vector $\mathbf{e}_1$ of smallest weight. This must lie in the set V_0, since it is closest to the codeword $\mathbf{c}_0 = \mathbf{0}$. But

$$d_H(\mathbf{e}_1 + \mathbf{c}_i, \mathbf{c}_i) = d_H(\mathbf{e}_1, \mathbf{0}),$$

for each i, so the vector $\mathbf{e}_1 + \mathbf{c}_i$ must also lie in V_i for each i. So $\mathbf{e}_1 + \mathbf{c}_i$ is placed into each V_i. The vectors $\mathbf{e}_1 + \mathbf{c}_i$ are included in their respective columns of the standard array to form the second row of the standard array. The procedure continues, selecting an unused vector of minimal weight and adding it to each codeword to form the next row of the standard array, until all q^n possible vectors have been used in the standard array. In summary:

1. Write down all the codewords of the code $\mathcal{C}$.
2. Select from the remaining unused vectors of $\mathbb{F}_q^n$ one of minimal weight, $\mathbf{e}$. Write $\mathbf{e}$ in the column under the all-zero codeword, then add $\mathbf{e}$ to each codeword in turn, writing the sum in the column under the corresponding codeword.
3. Repeat step 2 until all q^n vectors in $\mathbb{F}_q^n$ have been placed in the standard array.

Example 3.7 For a (7, 3) code, a generator matrix is



$$G = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}.$$

The codewords for this code are

row 1	0000000	0111100	1011010	1100110	1101001	1010101	0110011	0001111

From the remaining 7-tuples, one of minimal weight is selected; take (1000000). The second row is obtained by adding this to each codeword:

row 1	0000000	0111100	1011010	1100110	1101001	1010101	0110011	0001111
row 2	1000000	1111100	0011010	0100110	0101001	0010101	1110011	1001111

Now proceed until all 2^n vectors are used, selecting an unused vector of minimum weight and adding it to all the codewords. The result is shown in Table 3.1.

(The horizontal lines in the standard array separate the error patterns of different weights.) □

We make the following observations:

Table 3.1: The Standard Array for a Code

row 1	0000000	0111100	1011010	1100110	1101001	1010101	**0110011**	0001111
row 2	1000000	1111100	0011010	0100110	0101001	0010101	1110011	1001111
row 3	0100000	0011100	1111010	1000110	1001001	1110101	0010011	0101111
row 4	0010000	0101100	1001010	1110110	1111001	1000101	0100011	0011111
row 5	0001000	0110100	1010010	1101110	1100001	1011101	0111011	0000111
row 6	0000100	0111000	1011110	1100010	1101101	1010001	0110111	0001011
row 7	0000010	0111110	1011000	1100100	1101011	1010111	0110001	0001101
row 8	0000001	0111101	1011011	1100111	1101000	1010100	0110010	0001110
row 9	1100000	1011100	0111010	0000110	0001001	0110101	1010011	1101111
row 10	1010000	1101100	0001010	0110110	0111001	0000101	1100011	1011111
row 11	0110000	0001100	1101010	1010110	1011001	1100101	0000011	0111111
row 12	1001000	1110100	0010010	0101110	0100001	0011101	1111011	1000111
row 13	**0101000**	0010100	1110010	1001110	1000001	1111101	**0011011**	0100111
row 14	0011000	0100100	1000010	1111110	1110001	1001101	0101011	0010111
row 15	1000100	1111000	0011110	0100010	0101101	0010001	1110111	1001011
row 16	1110000	1001100	0101010	0010110	0011001	0100101	1000011	1111111

1. There are q^k codewords (columns) and q^n possible vectors, so there are q^{n-k} rows in the standard array. We observe, therefore, that: an (n, k) code is capable of correcting q^{n-k} different error patterns.
2. The difference (or sum, over $GF(2)$) of any two vectors in the same row of the standard array is a code vector. In a row, the vectors are $\mathbf{c}_i + \mathbf{e}$ and $\mathbf{c}_j + \mathbf{e}$. Then

$$(\mathbf{c}_i + \mathbf{e}) - (\mathbf{c}_j + \mathbf{e}) = \mathbf{c}_i - \mathbf{c}_j,$$

   which is a codeword, since linear codes form a vector subspace.
3. No two vectors in the same row of a standard array are identical. Because otherwise we have

$$\mathbf{e} + \mathbf{c}_i = \mathbf{e} + \mathbf{c}_j, \text{ with } i \neq j,$$

   which means $\mathbf{c}_i = \mathbf{c}_j$, which is impossible.
4. Every vector appears exactly once in the standard array. We know every vector must appear at least once, by the construction. If a vector appears in both the lth row and the mth row we must have

$$\mathbf{e}_l + \mathbf{c}_i = \mathbf{e}_m + \mathbf{c}_j$$

   for some i and j. Let us take $l < m$. We have

$$\mathbf{e}_m = \mathbf{e}_l + \mathbf{c}_i - \mathbf{c}_j = \mathbf{e}_l + \mathbf{c}_k$$

   for some k. This means that $\mathbf{e}_m$ is on the lth row of the array, which is a contradiction.

The rows of the standard array are called **cosets**. Each row is of the form

$$\mathbf{e} + \mathcal{C} = \{\mathbf{e} + \mathbf{c} : \mathbf{c} \in \mathcal{C}\}.$$

That is, the rows of the standard array are translations of $\mathcal{C}$. These are the same cosets we met in Section 2.2.3 in conjunction with groups.

The vectors in the first column of the standard array are called the **coset leaders**. They represent the error patterns that can be corrected by the code under this decoding strategy. The decoder of Example 3.7 is capable of correcting all errors of weight 1, 7 different error patterns of weight 2, and 1 error pattern of weight 3.

To decode with the standard array, we first locate the received vector $\mathbf{r}$ in the standard array. Then identify

$$\mathbf{r} = \mathbf{e} + \mathbf{c}$$

for a vector $\mathbf{e}$ which is a coset leader (in the left column) and a codeword $\mathbf{c}$ (on the top row). Since we designed the standard array with the smallest error patterns as coset leaders, the error codeword so identified in the standard array is the ML decision. The coset leaders are called the *correctable error patterns*.

Example 3.8 For the code of Example 3.7, let

$$\mathbf{r} = [0, 0, 1, 1, 0, 1, 1]$$

(shown in bold in the standard array) then its coset leader is $\mathbf{e} = [0, 1, 0, 1, 0, 0, 0]$ and the codeword is $\mathbf{c} = [0, 1, 1, 0, 0, 1, 1]$, which corresponds to the message $\mathbf{m} = [0, 1, 1]$, since the generator is systematic. □

Example 3.9 It is interesting to note that for the standard array of Example 3.7, not all $\binom{7}{2} = 21$ patterns of 2 errors are correctable. Only 7 patterns of two errors are correctable. However, there is one pattern of three errors which is correctable.

The minimum distance for this code is clearly 4, since the minimum weight of the nonzero codewords is 4. Thus, the code is *guaranteed* to correct only $\lfloor (4-1)/2 \rfloor = 1$ error and, in fact it does correct all patterns of single errors. □

As this decoding example shows, the standard array decoder may have coset leaders with weight higher than the random-error-correcting capability of the code $t = \lfloor (d_{\min} - 1)/2 \rfloor$.

This observation motivates the following definition.

Definition 3.8 A **complete error correcting decoder** is a decoder that given the received word $\mathbf{r}$, selects the codeword $\mathbf{c}$ which minimizes $d_H(\mathbf{r}, \mathbf{c})$. That is, it is the ML decoder for the BSC channel. □

If a standard array is used as the decoding mechanism, then complete decoding is achieved. On the other hand, if the rows of the standard array are filled out so that *all* instances of up to t errors appear in the table, and all other rows are left out, then a bounded distance decoder is obtained.

Definition 3.9 A t-error correcting **bounded distance decoder** selects the codeword $\mathbf{c}$ given the received vector $\mathbf{r}$ if $d_H(\mathbf{r}, \mathbf{c}) \leq t$. If no such $\mathbf{c}$ exists, then a **decoder failure** is declared. □

Example 3.10 In Table 3.1, only up to row 8 of the table would be used in a bounded distance decoder, which is capable of correcting up to $t = \lfloor (d_{\min} - 1)/2 \rfloor = \lfloor (4-1)/2 \rfloor = 1$ error. A received vector $\mathbf{r}$ appearing in rows 9 through 16 of the standard array would result in a decoding failure. □

A perfect code can be understood in terms of the standard array: it is one for which there are no "leftover" rows: all $\binom{n}{t}$ error patterns of weight t and all lighter error patterns appear as coset leaders in the table, with no "leftovers." What makes it "perfect" then, is that the bounded distance decoder is also the ML decoder.

The standard array can, in principle, be used to decode any linear block code, but suffers from a major problem: the memory required to represent the standard array quickly become excessive, and decoding requires searching the entire table to find a match for a received vector $\mathbf{r}$. For example, a (256, 200) binary code — not a particularly long code by modern

standards — would require $2^{256} \approx 1.2 \times 10^{77}$ vectors of length 256 bits to be stored in it and every decoding operation would require on average searching through half of the table.

A first step in reducing the storage and search complexity (which doesn't go far enough) is to use **syndrome decoding**. Let $\mathbf{e} + \mathbf{c}$ be a vector in the standard array. The syndrome for this vector is $\mathbf{s} = (\mathbf{e} + \mathbf{c})H^T = \mathbf{e}H^T$. Furthermore, every vector in the coset has the same syndrome: $(\mathbf{e} + \mathbf{c})H^T = \mathbf{e}H^T$. We therefore only need to store syndromes and their associated error patterns. This table is called the syndrome decoding table. It has q^{n-k} rows but only two columns, so it is smaller than the entire standard array. But is still impractically large in many cases.

With the syndrome decoding table, decoding is done as follows:

1. Compute the syndrome, $\mathbf{s} = \mathbf{r}H^T$
2. In the syndrome decoding table look up the error pattern $\mathbf{e}$ corresponding to $\mathbf{s}$.
3. Then $\mathbf{c} = \mathbf{r} + \mathbf{e}$.

Example 3.11 For the code of Example 3.7 a parity check matrix is

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}.$$

The syndrome decoding table is

Error	Syndrome
0000000	0000
1000000	1000
0100000	0100
0010000	0010
0001000	0001
0000100	0111
0000010	1011
0000001	1101
1100000	1100
1010000	1010
0110000	0110
1001000	1001
0101000	**0101**
0011000	0011
1000100	1111
1110000	1110

Suppose that $\mathbf{r} = [0, 0, 1, 1, 0, 1, 1]$, as before. The syndrome is

$$\mathbf{s} = \mathbf{r}H^T = \begin{bmatrix} 0 & 1 & 0 & 1 \end{bmatrix},$$

(in bold in the table) which corresponds to the coset leader $\mathbf{e} = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$. The decoded codeword is then

$$\hat{\mathbf{c}} = [0, 0, 1, 1, 0, 1, 1] + [0, 1, 0, 1, 0, 0, 0] = [0, 1, 1, 0, 0, 1, 1],$$

as before. □

Despite the significant reduction compared to the standard array, the memory requirements for the syndrome decoding table are still very high. It is still infeasible to use this technique for very long codes. Additional algebraic structure must be imposed on the code to enable decoding long codes.

3.5 Weight Distributions of Codes and Their Duals

The weight distribution of a code plays a significant role in calculating probabilities of error.

Definition 3.10 Let C be an (n, k) code. Let A_i denote the number of codewords of weight i in C. Then the set of coefficients $\{A_0, A_1, \ldots, A_n\}$ is called the **weight distribution** for the code.

It is convenient to represent the weight distribution as a polynomial,

$$A(z) = A_0 + A_1 z + A_2 z^2 + \cdots + A_n z^n. \tag{3.11}$$

This polynomial is called the **weight enumerator**. □

The weight enumerator is (essentially) the z-transform of the weight distribution sequence.

Example 3.12 For the code of Example 3.7, there is one codeword of weight 0. The rest of the codewords all have weight 4. So $A_0 = 1$, $A_4 = 7$. Thus

$$A(z) = 1 + 7z^4.$$

□

There is a relationship, known as the *MacWilliams identity*, between the weight enumerator of a linear code and its dual. This relationship is of interest because for many codes it is possible to directly characterize the weight distribution of the dual code, from which the weight distribution of the code of interest is obtained by the MacWilliams identity.

Theorem 3.6 *(The* **MacWilliams Identity***). Let C be an (n, k) linear block code over $\mathbb{F}_q$ with weight enumerator $A(z)$ and let $B(z)$ be the weight enumerator of $C^\perp$. Then*

$$B(z) = q^{-k}(1 + (q-1)z)^n A\left(\frac{1-z}{1+(q-1)z}\right), \tag{3.12}$$

or, turning this around algebraically,

$$A(z) = q^{-(n-k)}(1 + (q-1)z)^n B\left(\frac{1-z}{1+(q-1)z}\right). \tag{3.13}$$

The proof of this theorem reveals some techniques that are very useful in coding. We give the proof for codes over $\mathbb{F}_2$, but it is straightforward to extend to larger fields (once you are familiar with them). The proof relies on the **Hadamard transform**. For a function f defined on $\mathbb{F}_2^n$, the Hadamard transform $\hat{f}$ of f is

$$\hat{f}(\mathbf{u}) = \sum_{\mathbf{v} \in \mathbb{F}_2^n} (-1)^{\langle \mathbf{u}, \mathbf{v} \rangle} f(\mathbf{v}) = \sum_{\mathbf{v} \in \mathbb{F}_2^n} (-1)^{\sum_{i=0}^{n-1} u_i v_i} f(\mathbf{v}), \qquad \mathbf{u} \in \mathbb{F}_2^n,$$

where the sum is taken over all 2^n n-tuples $\mathbf{v} = (v_0, v_1, \ldots, v_{n-1})$, where each $v_i \in \mathbb{F}_2^n$.

Lemma 3.7 *If C is an (n, k) binary linear code and f is a function defined on $\mathbb{F}_2^n$,*

$$\sum_{\mathbf{u} \in C^\perp} f(\mathbf{u}) = \frac{1}{|C|} \sum_{\mathbf{u} \in C} \hat{f}(\mathbf{u}).$$

Here $|C|$ denotes the number of elements in the code C.

Proof of lemma.

$$\sum_{\mathbf{u}\in\mathcal{C}} \hat{f}(\mathbf{u}) = \sum_{\mathbf{u}\in\mathcal{C}}\sum_{\mathbf{v}\in\mathbb{F}_2^n}(-1)^{\langle\mathbf{u},\mathbf{v}\rangle}f(\mathbf{v}) = \sum_{\mathbf{v}\in\mathbb{F}_2^n} f(\mathbf{v})\sum_{\mathbf{u}\in\mathcal{C}}(-1)^{\langle\mathbf{u},\mathbf{v}\rangle}$$

$$= \sum_{\mathbf{v}\in\mathcal{C}^\perp} f(\mathbf{v})\sum_{\mathbf{u}\in\mathcal{C}}(-1)^{\langle\mathbf{u},\mathbf{v}\rangle} + \sum_{\mathbf{v}\in\mathcal{C}\setminus\{0\}} f(\mathbf{v})\sum_{\mathbf{u}\in\mathcal{C}}(-1)^{\langle\mathbf{u},\mathbf{v}\rangle},$$

where $\mathbb{F}_2^n$ has been partitioned into two disjoint sets, the dual code $\mathcal{C}^\perp$ and the nonzero elements of the code, $\mathcal{C}\setminus\{\mathbf{0}\}$. In the first sum, $\langle\mathbf{u},\mathbf{v}\rangle = 0$, so the inner sum is $|\mathcal{C}|$. In the second sum, for every $\mathbf{v}$ in the outer sum, $\langle\mathbf{u},\mathbf{v}\rangle$ takes on the values 0 and 1 equally often as $\mathbf{u}$ varies over $\mathcal{C}$ in the inner sum, so the inner sum is 0. Therefore

$$\sum_{\mathbf{u}\in\mathcal{C}} \hat{f}(\mathbf{u}) = \sum_{\mathbf{v}\in\mathcal{C}^\perp} f(\mathbf{v})\sum_{\mathbf{u}\in\mathcal{C}}(1) = |\mathcal{C}|\sum_{\mathbf{v}\in\mathcal{C}^\perp} f(\mathbf{v}).$$

□

Proof of Theorem 3.6. Note that the weight enumerator can be written as

$$A(z) = \sum_{\mathbf{c}\in\mathcal{C}} z^{\mathrm{wt}(\mathbf{c})}.$$

Let $f(\mathbf{u}) = z^{\mathrm{wt}(\mathbf{u})}$. Taking the Hadamard transform we have

$$\hat{f}(\mathbf{u}) = \sum_{\mathbf{v}\in\mathbb{F}_2^n}(-1)^{\langle\mathbf{u},\mathbf{v}\rangle}z^{\mathrm{wt}(\mathbf{v})}.$$

Writing out the inner product and the weight function explicitly on the vectors $\mathbf{u} = (u_0, u_1, \ldots, u_{n-1})$ and $\mathbf{v} = (v_0, v_1, \ldots, v_{n-1})$ we have

$$\hat{f}(\mathbf{u}) = \sum_{\mathbf{v}\in\mathbb{F}_2^n}(-1)^{\sum_{i=0}^{n-1}u_iv_i}\prod_{i=0}^{n-1}z^{v_i} = \sum_{\mathbf{v}\in\mathbb{F}_2^n}\prod_{i=0}^{n-1}(-1)^{u_iv_i}z^{v_i}.$$

The sum over the 2^n values of $\mathbf{v}\in\mathbb{F}_2^n$ can be expressed as n nested summations over the binary values of the elements of $\mathbf{v}$:

$$\hat{f}(\mathbf{u}) = \sum_{v_0=0}^{1}\sum_{v_1=0}^{1}\cdots\sum_{v_{n-1}=0}^{1}\prod_{i=0}^{n-1}(-1)^{u_iv_i}z^{v_i}.$$

Now the distributive law can be used to pull the product out of the summations[2],

$$\hat{f}(\mathbf{u}) = \prod_{i=0}^{n-1}\sum_{v_i=0}^{1}(-1)^{u_iv_i}z^{v_i}.$$

If $u_i = 0$ then the inner sum is $1+z$. If $u_i = 1$ then the inner sum is $1-z$. Thus

$$\hat{f}(\mathbf{u}) = (1+z)^{n-\mathrm{wt}(\mathbf{u})}(1-z)^{\mathrm{wt}(\mathbf{u})} = \left(\frac{1-z}{1+z}\right)^{\mathrm{wt}(\mathbf{u})}(1+z)^n$$

[2]The distributive law reappears in a generalized way in Chapter 16. We make the interesting observation here that the use of the distributive law gives rise to a "fast" Hadamard transform, analogous to and similarly derived as the fast Fourier transform.

Now applying Lemma 3.7 we obtain

$$B(z) = \frac{1}{|C|}(1+z)^n \sum_{\mathbf{u}\in C} \left(\frac{1-z}{1+z}\right)^{\mathrm{wt}(\mathbf{u})} = \frac{1}{|C|}(1+z)^n A\left(\frac{1-z}{1+z}\right).$$

□

3.6 Hamming Codes and Their Duals

We now formally introduce a family of binary linear block codes, the Hamming codes, and their duals.

Definition 3.11 For any integer $m \geq 2$, a $(2^m - 1, 2^m - m - 1, 3)$ binary code may be defined by its $m \times n$ parity check matrix H, which is obtained by writing all possible binary m-tuples, except the all-zero tuple, as the columns of H. For example, simply write out the m-bit binary representation for the numbers from 1 to n in order. Codes equivalent to this construction are called **Hamming codes**. □

For example, when $m = 4$ we get

$$H = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$
$$\begin{matrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \end{matrix}$$

as the parity check matrix for a $(15, 11)$ Hamming code. However, it is usually convenient to reorder the columns — resulting in an equivalent code — so that the identity matrix which is interspersed throughout the columns of H appears in the first m columns. We therefore write

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}.$$
$$\begin{matrix} 1 & 2 & 4 & 8 & 3 & 5 & 6 & 7 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \end{matrix}$$

It is clear from the form of the parity check matrix that for any m there exist three columns which add to zero; for example,

$$\begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}, \text{ and } \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix},$$

so by Theorem 3.3 the minimum distance is 3; Hamming codes are capable of correcting 1 bit error in the block, or detecting up to 2 bit errors.

An algebraic decoding procedure for Hamming codes was described in Section 1.9.1.

The dual to a $(2^m - 1, 2^m - m - 1)$ Hamming code is a $(2^m - 1, m)$ code called a **simplex code** or a **maximal-length feedback shift register code**.

Example 3.13 The parity check matrix of the (7, 4) Hamming code of (3.7) can be used as a generator of the (7, 3) simplex code with codewords

0000000	1001011
0101110	0010111
1100101	1011100
0111001	1110010

Observe that except for the zero codeword, all codewords have weight 4. □

In general, all of the codewords of the $(2^m - 1, m)$ simplex code have weight 2^{m-1} (see Exercise 3.12) and every pair of codewords is at a distance 2^{m-1} apart (which is why it is called a simplex). For example, for the $m = 2$ case, the codewords $\{(000), (101), (011), (110)\}$ form a tetrahedron. Thus the weight enumerator of the dual code is

$$B(z) = 1 + (2^m - 1)z^{2^{m-1}}. \tag{3.14}$$

From the weight enumerator of the dual, we find using (3.13) that the weight distribution of the Hamming code is

$$A(z) = \frac{1}{n+1}[(1+z)^n + n(1-z)(1-z^2)^{(n-1)/2}]. \tag{3.15}$$

Example 3.14 For the (7,4) Hamming code the weight enumerator is

$$A(z) = \frac{1}{8}\left[(1+z)^7 + 7(1-z)(1-z^2)^3\right] = 1 + 7z^3 + 7z^4 + z^7. \tag{3.16}$$

□

Example 3.15 For the (15,11) Hamming code the weight enumerator is

$$\begin{aligned} A(z) &= \frac{1}{16}((1+z)^{15} + 15(1-z)(1-z^2)^7) \\ &= 1 + 35z^3 + 105z^4 + 168z^5 + 280z^6 + 435z^7 + 435z^8 + 280z^9 + 168z^{10} \\ &\quad + 105z^{11} + 35z^{12} + z^{15}. \end{aligned} \tag{3.17}$$

□

3.7 Performance of Linear Codes

There are several different ways that we can characterize the error detecting and correcting capabilities of codes at the output of the channel decoder [373].

$P(E)$ is the **probability of decoder error**, also known as the **word error rate**. This is the probability that the *codeword* at the output of the decoder is not the same as the codeword at the input of the encoder.

$P_b(E)$ **or** P_b is the **probability of bit error**, also known as the **bit error rate.** This is the probability that the decoded message bits (extracted from a decoded codeword of a binary code) are not the same as the encoded message bits. Note that when a decoder error occurs, there may be anywhere from 1 to k message bits in error, depending on what codeword is sent, what codeword was decoded, and the mapping from message bits to codewords.

$P_u(E)$ is the **probability of undetected codeword error**, the probability that errors occurring in a codeword are not detected.

$P_d(E)$ is the **probability of detected codeword error**, the probability that one or more errors occurring in a codeword are detected.

P_{u_b} is the **undetected bit error rate**, the probability that a decoded message bit is in error and is contained within a codeword corrupted by an undetected error.

P_{d_b} is the **detected bit error rate**, the probability that a received message bit is in error and is contained within a codeword corrupted by a detected error.

$P(F)$ is the **probability of decoder failure**, which is the probability that the decoder is unable to decode the received vector (and is able to determine that it cannot decode).

In what follows, bounds and exact expressions for these probabilities will be developed.

3.7.1 Error detection performance

All errors with weight up to $d_{\min} - 1$ can be detected, so in computing the probability of detection only error patterns with weight $d_{\min}$ or higher need be considered. If a codeword $\mathbf{c}$ of a linear code is transmitted and the error pattern $\mathbf{e}$ happens to be a codeword, $\mathbf{e} = \mathbf{c}'$, then the received vector

$$\mathbf{r} = \mathbf{c} + \mathbf{c}'$$

is also a codeword. Hence, the error pattern would be undetectable. Thus, the probability that an error pattern is undetectable is precisely the probability that it is a codeword.

We consider only errors in transmission of binary codes over the BSC with crossover probability p. (Extension to codes with larger alphabets is discussed in [373].) The probability of any particular pattern of j errors in a codeword is $p^j(1-p)^{n-j}$. Recalling that A_j is the number of codewords in C of weight j, the probability that j errors form a codeword is $A_j p^j(1-p)^{n-j}$. The probability of undetectable error in a codeword is then

$$P_u(E) = \sum_{j=d_{\min}}^{n} A_j p^j (1-p)^{n-j}. \tag{3.18}$$

The probability of a detected codeword error is the probability that one or more errors occur minus the probability that the error is undetected:

$$P_d(E) = \sum_{j=1}^{n} \binom{n}{j} p^j (1-p)^{n-j} - P_u(E) = 1 - (1-p)^n - P_u(E).$$

Computing these probabilities requires knowing the weight distribution of the code, which is not always available. It is common, therefore, to provide *bounds* on the performance. A bound on $P_u(E)$ can be obtained by observing that the probability of undetected error is bounded above by the probability of occurrence of *any* error patterns of weight greater than or equal to $d_{\min}$. Since there are $\binom{n}{j}$ different ways that j positions out of n can be changed,

$$P_u(E) \le \sum_{j=d_{\min}}^{n} \binom{n}{j} p^j (1-p)^{n-j}. \tag{3.19}$$

A bound on $P_d(E)$ is simply

$$P_d(E) \le 1 - (1-p)^n.$$

Example 3.16 For the Hamming (7, 4) code with $A(z) = 1 + 7z^3 + 7z^4 + z^7$,

$$P_u(E) = 7p^3(1-p)^4 + 7p^4(1-p)^3 + p^7.$$

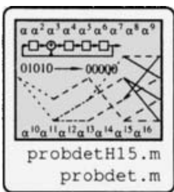
If $p = .01$ then $P_u(E) \approx 6.79 \times 10^{-6}$. The bound (3.19) gives $P_u(E) < 3.39 \times 10^{-5}$. □

The corresponding bit error rates can be bounded as follows. The undetected bit error rate P_{u_b} can be lower-bounded by assuming the undetected codeword error corresponds to only a single message bit error. P_{u_b} can be upper-bounded by assuming that the undetected codeword error corresponds to all k message bits being in error. Thus

$$\frac{1}{k}P_u(E) \leq P_{u_b} \leq P_u(E).$$

Similarly for P_{d_b}:

$$\frac{1}{k}P_d(E) \leq P_{d_b} \leq P_d(E).$$



Example 3.17 Figure 3.1 illustrates the detection probabilities for a BSC derived from a BPSK system, with $p = Q(\sqrt{2RE_b/N_0})$, for a Hamming (15,11) code. The weight enumerator is in (3.17). For comparison, the probability of an undetected error for the uncoded system is shown, in which *any* error is undetected, so $P_{u,\text{uncoded}} = 1 - (1 - p_{\text{uncoded}})^k$, where $p_{\text{uncoded}} = Q(\sqrt{2E_b/N_0})$. Note that the upper bound on P_u is not very tight, but the upper bound on P_d is tight — they are indistinguishable on the plot, since they differ by $P_u(E)$, which is orders of magnitude smaller than $P_d(E)$. The uncoded probability of undetected error is much greater than the coded P_u. □

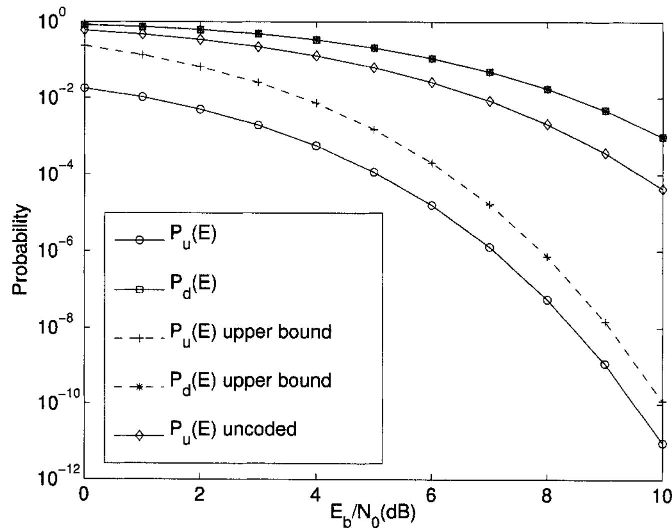


Figure 3.1: Error detection performance for a (15,11) Hamming code.

3.7.2 Error Correction Performance

An error pattern is correctable if and only if it is a coset leader in the standard array for the code, so the probability of correcting an error is the probability that the error is a coset leader. Let α_i denote the number of coset leaders of weight i. The numbers $\alpha_0, \alpha_1, \ldots, \alpha_n$

are called the **coset leader weight distribution**. Over a BSC with crossover probability p, the probability of j errors forming one of the coset leaders is $\alpha_j p^j(1-p)^{n-j}$. The probability of a decoding error is thus the probability that the error is *not* one of the coset leaders

$$P(E) = 1 - \sum_{j=0}^{n} \alpha_j p^j (1-p)^{n-j}.$$

This result applies to any linear code with a *complete decoder.*

Example 3.18 For the standard array in Table 3.1, the coset leader weight distribution is

$$\alpha_0 = 1 \quad \alpha_1 = 7 \quad \alpha_2 = 7 \quad \alpha_3 = 1.$$

If $p = 0.01$, then $P(E) = 0.0014$. □

Most hard-decision decoders are *bounded-distance* decoders, selecting the codeword $\mathbf{c}$ which lies within a Hamming distance of $\lfloor (d_{\min} - 1)/2 \rfloor$ of the received vector $\mathbf{r}$. An exact expression for the probability of error for a bounded-distance decoder can be developed as follows. Let P_l^j be the probability that a received word $\mathbf{r}$ is exactly Hamming distance l from a codeword of weight j.

Lemma 3.8 *[373, p. 249]*

$$P_l^j = \sum_{r=0}^{l} \binom{j}{l-r} \binom{n-j}{r} p^{j-l+2r} (1-p)^{n-j+l-2r}.$$

Proof Assume (without loss of generality) that the all-zero codeword was sent. Let $\mathbf{c}$ be a codeword of weight j, where $j \neq 0$. Let the coordinates of $\mathbf{c}$ which are 1 be called the 1-coordinates and let the coordinates of $\mathbf{c}$ which are 0 be called the 0-coordinates. There are thus j 1-coordinates and $n-j$ 0-coordinates of $\mathbf{c}$. Consider now the ways in which the received vector $\mathbf{r}$ can be a Hamming distance l away from $\mathbf{c}$. To differ in l bits, it must differ in an integer r number of 0-coordinates and $l-r$ 1-coordinates, where $0 \leq r \leq l$. The number of ways that $\mathbf{r}$ can differ from $\mathbf{c}$ in r of the 0-coordinates is $\binom{n-j}{r}$. The total probability of $\mathbf{r}$ differing from $\mathbf{c}$ in exactly r 0-coordinates is

$$\binom{n-j}{r} p^r (1-p)^{n-j-r}.$$

The number of ways that $\mathbf{r}$ can differ from $\mathbf{c}$ in $l-r$ of the 1-coordinates is $\binom{j}{j-(l-r)} = \binom{j}{l-r}$. Since the all-zero codeword was transmitted, the $l-r$ coordinates of $\mathbf{r}$ must be 0 (there was no crossover in the channel) and the remaining $j-(l-r)$ bits must be 1. The total probability of $\mathbf{r}$ differing from $\mathbf{c}$ in exactly $l-r$ 1-coordinates is

$$\binom{j}{l-r} p^{j-l+r} (1-p)^{l-r}.$$

The probability P_l^j is obtained by multiplying the probabilities on the 0-coordinates and the 1-coordinates (they are independent events since the channel is memoryless) and summing

over r:

$$P_l^j = \sum_{r=0}^{l} \binom{n-j}{r} p^r (1-p)^{n-j-r} \binom{j}{l-r} p^{j-l+r} (1-p)^{l-r}$$
$$= \sum_{r=0}^{l} \binom{j}{l-r} \binom{n-j}{r} p^{j-l+2r} (1-p)^{n+l-j-2r}.$$

□

The probability of error is now obtained as follows.

Theorem 3.9 *For a binary (n,k) code with weight distribution $\{A_i\}$, the probability of decoding error for a bounded distance decoder is*

$$P(E) = \sum_{j=d_{\min}}^{n} A_j \sum_{l=0}^{\lfloor (d_{\min}-1)/2 \rfloor} P_l^j. \tag{3.20}$$

Proof Assume that the all-zero codeword was sent. For a particular codeword of weight $j \neq 0$, the probability that the received vector $\mathbf{r}$ falls in the decoding sphere of that codeword is

$$\sum_{l=0}^{\lfloor (d_{\min}-1)/2 \rfloor} P_l^j.$$

Then the result follows by adding up over all possible weights, scaled by the number of codewords of weight j, A_j. □

The probability of decoder failure for the bounded distance decoder is the probability that the received codeword does not fall into any of the decoding spheres,

$$P(F) = 1 - \underbrace{\sum_{j=0}^{\lfloor (d_{\min}-1)/2 \rfloor} \binom{n}{j} p^j (1-p)^{n-j}}_{\substack{\text{probability of} \\ \text{falling in correct} \\ \text{decoding sphere}}} - \underbrace{P(E).}_{\substack{\text{probability of} \\ \text{falling in the} \\ \text{incorrect} \\ \text{decoding sphere}}}$$

Exact expressions to compute $P_b(E)$ require information relating the weight of the message bits and the weight of the corresponding codewords. This information is summarized in the number β_j, which is the total weight of the message blocks associated with codewords of weight j.

Example 3.19 For the $(7,4)$ Hamming code, $\beta_3 = 12$, $\beta_4 = 16$, and $\beta_7 = 4$. That is, the total weight of the messages producing codewords of weight 3 is 12; the total weight of messages producing codewords of weight 4 is 16. □

Modifying (3.20) we obtain

$$P_b(E) = \frac{1}{k} \sum_{j=d_{\min}}^{n} \beta_j \sum_{l=0}^{\lfloor (d_{\min}-1)/2 \rfloor} P_l^j.$$

(See `hamcode74pe.m`.) Unfortunately, while obtaining values for β_j for small codes is straightforward computationally, appreciably large codes require theoretical expressions which are usually unavailable.

The probability of decoder error can be easily bounded by the probability of *any* error patterns of weight greater than $\lfloor(d_{\min} - 1)/2\rfloor$:

$$P(E) \leq \sum_{j=\lfloor(d_{\min}+1)/2\rfloor}^{n} \binom{n}{j} p^j (1-p)^{n-j}.$$

An easy bound on probability of failure is the same as the bound on this probability of error.

Bounds on the probability of bit error can be obtained as follows. A lower bound is obtained by assuming that a decoder error causes a single bit error out of the k message bits. An upper bound is obtained by assuming that all k message bits are incorrect when the block is incorrectly decoded. This leads to the bounds

$$\frac{1}{k} P(E) \leq P_b(E) \leq P(E).$$

3.7.3 Performance for Soft-Decision Decoding

While all of the decoding in this chapter has been for hard-input decoders, it is interesting to examine the potential performance for soft-decision decoding. Suppose the codewords of an $(n, k, d_{\min})$ code C are modulated to a vector $\mathbf{s}$ using BPSK having energy $E_c = RE_b$ per coded bit and transmitted through an AWGN with variance $\sigma^2 = N_0/2$. The transmitted vector $\mathbf{s}$ is a point in n-dimensional space. In Exercise 1.15, it is shown that the Euclidean distance between two BPSK modulated codewords is related to the Hamming distance between the codewords by

$$d_E = 2\sqrt{E_c d_H}.$$

Suppose that there are K codewords (on average) at a distance $d_{\min}$ from a codeword. By the union bound (1.28), the probability of a block decoding error is given by

$$P(E) \approx K Q\left(\frac{d_{E,\min}}{2\sigma}\right) = K Q\left(\sqrt{\frac{2Rd_{min}E_b}{N_0}}\right).$$

Neglecting the multiplicity constant K, we see that we achieve essentially comparable performance compared to uncoded transmission when

$$\frac{E_b}{N_0} \text{ for uncoded} = \frac{Rd_{\min}E_b}{N_0} \text{ for coded.}$$

The asymptotic coding gain is the factor by which the coded E_b/N_0 can be decreased to obtain equivalent performance. (It is called asymptotic because it applies only as the SNR becomes large enough that the union bound can be regarded as a reasonable approximation.) In this case the asymptotic coding gain is

$$Rd_{\min}.$$

Recall that Figure 1.19 illustrated the advantage of soft-input decoding compared with hard-input decoding.

3.8 Erasure Decoding

An erasure is an error in which the error location is *known*, but the value of the error is not. Erasures can arise in several ways. In some receivers the received signal can be examined to see if it falls outside acceptable bounds. If it falls outside the bounds, it is declared as an erasure. (For example, for BPSK signaling, if the received signal is too close to the origin, an erasure might be declared.)

Example 3.20 Another way that an erasure can occur in packet-based transmission is as follows. Suppose that a sequence of codewords $\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_N$ are written into the *rows* of a matrix

c_{10}	c_{11}	c_{12}	$\cdots$	c_{1n-1}
c_{20}	c_{21}	c_{22}	$\cdots$	c_{2n-1}
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
c_{N0}	c_{N1}	c_{N2}	$\cdots$	c_{Nn-1}

then the *columns* are read out, giving the data sequence

$$[c_{10}, c_{20}, \ldots, c_{N0}], [c_{11}, c_{21}, \ldots, c_{N1}], [c_{12}, c_{22}, \ldots, c_{N2}], \ldots, [c_{1n-1}, c_{2n-1}, \ldots, c_{Nn-1}].$$

Suppose that these are now sent as a sequence of n data packets, each of length N, over a channel which is susceptible to packet loss, but where the loss of a packet is known at the receiver (such as the internet using a protocol that does not guarantee delivery, such as UDP). At the receiver, the packets are written into a matrix in column order — leaving an empty column corresponding to lost packets — then read out in row order. Suppose in this scheme that one of the packets, say the third, is lost in transmission. Then the data in the receiver interleaver matrix would look like

c_{10}	c_{11}	c_{12}	$\cdots$	c_{1n-1}
c_{20}	c_{21}	c_{22}	$\cdots$	c_{2n-1}
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
c_{N0}	c_{N1}	c_{N2}	$\cdots$	c_{Nn-1}

where the gray boxes indicate lost data. While a lost packet results in an entire column of lost data, it represents only *one* erased symbol from the de-interleaved codewords, a symbol whose location is known. □

Erasures can also sometimes be declared using concatenated coding techniques, where an outer code declares erasures at some symbol positions, which an inner code can then correct.

Consider the erasure capability for a code of distance $d_{\min}$. A single erased symbol removed from a code (with no additional errors) leaves a code with a minimum distance at least $d_{\min} - 1$. Thus f erased symbols can be "filled" provided that $f < d_{\min}$. For example, a Hamming code with $d_{\min} = 3$ can correct up to 2 erasures.

Now suppose that there are both errors and erasures. For a code with $d_{\min}$ experiencing a single erasure, there are still $n - 1$ unerased coordinates and the codewords are separated by a distance of at least $d_{\min} - 1$. More generally, if there are f erased symbols, then the distance among the remaining digits is at least $d_{\min} - f$. Letting t_f denote the random error decoding distance in the presence of f erasures, we can correct up to

$$t_f = \lfloor (d_{\min} - f - 1)/2 \rfloor$$

Box 3.2: The UDP Protocol

UDP — user datagram protocol — is one of the protocols in the TCP/IP protocol suite. The most common protocol, TCP, ensures packet delivery by acknowledging each packet successfully received, retransmitting packets which are garbled or lost in transmission. UDP, on the other hand, is an open-ended protocol which does not guarantee packet delivery. For a variety of reasons, it incurs lower delivery latency and as a result, it is of interest in near real-time communication applications. The application designer must deal with dropped packets using, for example, error correction techniques.

errors. If there are f erasures and e errors, they can be corrected provided that

$$2e + f < d_{\min}. \tag{3.21}$$

Since correcting an error requires determination of both the error position and the error value, while filling an erasure requires determination only of the error value, essentially twice the number of erasures can be filled as errors corrected.

3.8.1 Binary Erasure Decoding

We consider now how to simultaneously fill f erasures and correct e errors in a binary code with a given decoding algorithm [373, p. 229]. In this case, all that is necessary is to determine for each erasure whether the missing value should be a one or a zero. An erasure decoding algorithm for this case can be described as follows:

1. Place zeros in all erased coordinates and decode using the usual decoder for the code. Call the resulting codeword $\mathbf{c}_0$.
2. Place ones in all erased coordinates and decode using the usual decoder for the code. Call the resulting codeword $\mathbf{c}_1$.
3. Find which of $\mathbf{c}_0$ and $\mathbf{c}_1$ is closest to $\mathbf{r}$. This is the output code.

Let us examine why this decoder works. Suppose we have $(2e + f) < d_{\min}$ (so that correct decoding is possible). In assigning 0 to the f erased coordinates we thereby generated e_0 errors, $e_0 \le f$, so that the total number of errors to be corrected is $(e_0 + e)$. In assigning 1 to the f erased coordinates, we make e_1 errors, $e_1 \le f$, so that the total number of errors to be corrected is $(e_1 + e)$. Note that $e_0 + e_1 = f$, so that either e_0 or e_1 is less than or equal to $f/2$. Thus either

$$2(e + e_0) \le 2(e + f/2) \qquad \text{or} \qquad 2(e + e_1) \le 2(e + f/2),$$

and $2(e + f/2) < d_{\min}$, so that one of the two decodings must be correct.

Erasure decoding for nonbinary codes depends on the particular code structure. For example, decoding of Reed-Solomon codes is discussed in Section 6.7.

3.9 Modifications to Linear Codes

We introduce some minor modifications to linear codes. These are illustrated for some particular examples in Figure 3.2.

Definition 3.12 An (n, k, d) code is **extended** by adding an additional redundant coordinate, producing an $(n+1, k, d+1)$ code. □

Example 3.21 We demonstrate the operations by modifying a (7, 4, 3) Hamming code. The parity check matrix for an extended Hamming code, with an extra check bit that checks the parity of all the bits, can be written

$$H = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}.$$

The last row is the overall check bit row. By linear operations, this can be put in equivalent systematic form

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \end{bmatrix}$$

with the corresponding generator

$$G = \begin{bmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

See Figure 3.2. □

Definition 3.13 A code is **punctured** by deleting one of its parity symbols. An (n, k) code becomes an $(n-1, k)$ code. □

Puncturing an extended code can return it to the original code (if the extended symbols are the ones punctured.) Puncturing can reduce the weight of each codeword by its weight in the punctured positions. The minimum distance of a code is reduced by puncturing if the minimum weight codeword is punctured in a nonzero position. Puncturing an (n, k, d) code p times can result in a code with minimum distance as small as $d - p$.

Definition 3.14 A code is **expurgated** by deleting some of its codewords. It is possible to expurgate a linear code in such a way that it remains a linear code. The minimum distance of the code may increase. □

Example 3.22 If all the odd-weight codewords are deleted from the (7, 4) Hamming code, an even-weight subcode is obtained. □

Definition 3.15 A code is **augmented** by adding new codewords. It may be that the new code is not linear. The minimum distance of the code may decrease. □

Definition 3.16 A code is **shortened** by deleting a message symbol. This means that a row is removed from the generator matrix (corresponding to that message symbol) and a column is removed from the generator matrix (corresponding to the encoded message symbol). An (n, k) code becomes an $(n-1, k-1)$ code.

Shortened cyclic codes are discussed in more detail in section 4.12. □

Definition 3.17 A code is **lengthened** by adding a message symbol. This means that a row is added to the generator matrix (for the message symbol) and a column is added to represent the coded message symbol). An (n, k) code becomes an $(n+1, k+1)$ code. □

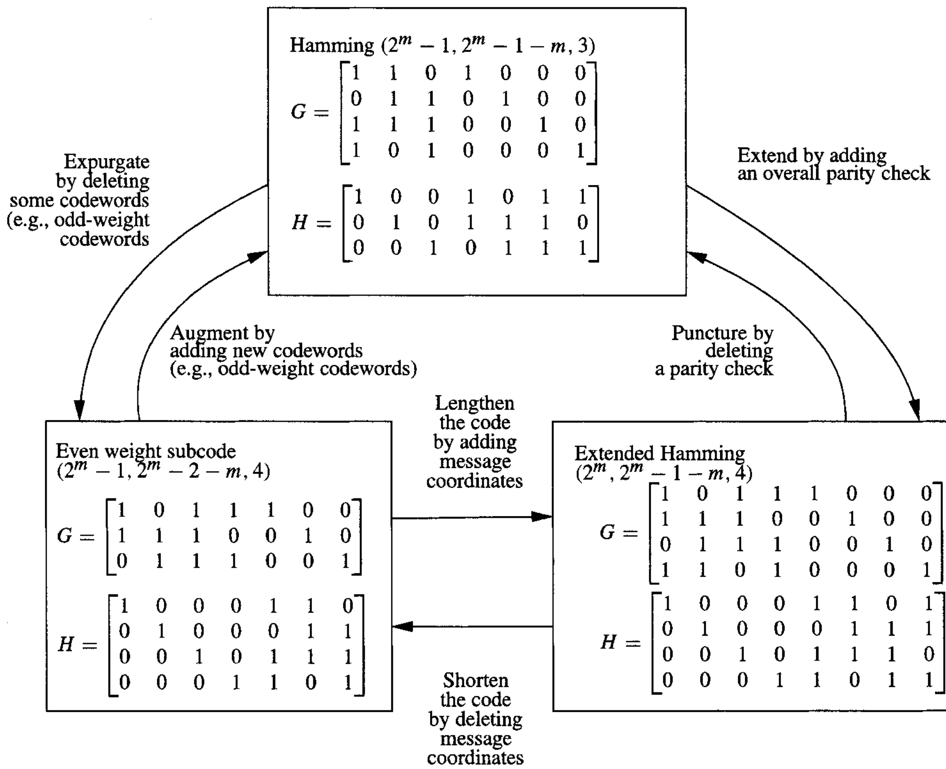


Figure 3.2: Demonstrating modifications on a Hamming code.

3.10 Best Known Linear Block Codes

Tables of the best known linear block codes are available. An early version appears in [220]. More recent tables can be found at [37].

3.11 Exercises

3.1 Find, by trial and error, a set of four binary codewords of length three such that each word is at least a distance of 2 from every other word.

3.2 Find a set of 16 binary words of length 7 such that each word is at least a distance of 3 from every other word. *Hint*: Hamming code.

3.3 Perhaps the simplest of all codes is the binary parity check code, a $(n, n-1)$ code, where $k = n-1$. Given a message vector $\mathbf{m} = (m_0, m_1, \ldots, m_{k-1})$, the codeword is $\mathbf{c} = (m_0, m_1, \ldots, m_{k-1}, b)$, where $b = \sum_{j=0}^{k-1} m_j$ (arithmetic in $GF(2)$) is the *parity bit*. Such a code is called an even parity code, since all codewords have even parity — an even number of 1 bits.

(a) Determine the minimum distance for this code.

(b) How many errors can this code correct? How many errors can this code detect?

(c) Determine a generator matrix for this code.

(d) Determine a parity check matrix for this code.

(e) Suppose that bit errors occur independently with probability p_c. The probability that a parity check is satisfied is the probability that an even number of bit errors occur in the received codeword. Verify the following expression for this probability:

$$\sum_{i=0, i \text{ even}}^{n} \binom{n}{i} p_c^i (1-p_c)^{n-i} = \frac{1+(1-2p_c)^n}{2}.$$

3.4 For the $(n, 1)$ repetition code, determine a parity check matrix.

3.5 [373] Let $p = 0.1$ be the probability that any bit in a received vector is incorrect. Compute the probability that the received vector contains undetected errors given the following encoding schemes:

(a) No code, word length $n = 8$.
(b) Even parity (see Exercise 3), word length $n = 4$.
(c) Odd parity, word length $n = 9$. (Is this a linear code?)
(d) Even parity, word length $= n$.

3.6 [204] Let $\mathcal{C}_1$ be an (n_1, k, d_1) binary linear systematic code with generator $G_1 = \begin{bmatrix} P_1 & I_k \end{bmatrix}$. Let $\mathcal{C}_2$ be an (n_2, k, d_2) binary linear systematic code with generator $G_2 = \begin{bmatrix} P_2 & I_k \end{bmatrix}$. Form the parity check matrix for an $(n_1 + n_2, k)$ code as

$$H = \begin{bmatrix} I_{n_1+n_2-k} & \begin{matrix} P_1^T \\ I_k \\ P_2^T \end{matrix} \end{bmatrix}.$$

Show that this code has minimum distance at least $d_1 + d_2$.

3.7 The generator matrix for a code over $GF(2)$ is given by

$$G = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}.$$

Find a generator matrix and parity-check matrix for an equivalent systematic code.

3.8 The generator and parity check matrix for a binary code are given by

$$G = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 0 \end{bmatrix} \qquad H = \begin{bmatrix} 1 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 \end{bmatrix}. \tag{3.22}$$

This code is small enough that it can be used to demonstrate several concepts from throughout the chapter.

(a) Verify that H is a parity check matrix for this generator.
(b) Draw a logic diagram schematic for an implementation of an encoder for the nonsystematic generator G using 'and' and 'xor' gates.
(c) Draw a logic diagram schematic for an implementation of a circuit that computes the syndrome.
(d) List the vectors in the orthogonal complement of the code.
(e) Form the standard array for this code.
(f) Form the syndrome decoding table for this code.
(g) How many codewords are there of weight $0, 1, \ldots, 6$? Determine the weight enumerator $A(z)$.

(h) Using the generator matrix in (3.22), find the codeword with $\mathbf{m} = [1, 1, 0]$ as message bits.
(i) Decode the received word $\mathbf{r} = [1, 1, 1, 0, 0, 1]$ using the generator of (3.22).
(j) Determine the weight enumerator for the dual code.
(k) Write down an explicit expression for $P_u(E)$ for this code. Evaluate this when $p = 0.01$.
(l) Write down an explicit expression for $P_d(E)$ for this code. Evaluate this when $p = 0.01$.
(m) Write down an explicit expression for $P(E)$ for this code. Evaluate this when $p = 0.01$.
(n) Write down an explicit expression for $P(E)$ for this code, assuming a bounded distance decoder is used. Evaluate this when $p = 0.01$.
(o) Write down an explicit expression for $P(F)$ for this code. Evaluate this when $p = 0.01$.
(p) Determine the generator G for an extended code, in systematic form.
(q) Determine the generator for a code which has expurgated all codewords of odd weight. Then express it in systematic form.

3.9 [203] Let a systematic $(8, 4)$ code have parity check equations

$$\begin{aligned} c_0 &= m_1 + m_2 + m_3 \\ c_1 &= m_0 + m_1 + m_2 \\ c_2 &= m_0 + m_1 + m_3 \\ c_3 &= m_0 + m_2 + m_3. \end{aligned}$$

(a) Determine the generator matrix G in for this code in systematic form. Also determine the parity check matrix H.
(b) Using Theorem 3.3, show that the minimum distance of this code is 4.
(c) Determine $A(z)$ for this code. Determine $B(z)$.
(d) Show that this is a self-dual code.

3.10 Show that a self-dual code has a generator matrix G which satisfies $GG^T = 0$.

3.11 Given a code with a parity-check matrix H, show that the coset with syndrome $\mathbf{s}$ contains a vector of weight w if and only if some linear combination of w columns of H equals $\mathbf{s}$.

3.12 Show that all of the nonzero codewords of the $(2^m - 1, m)$ simplex code have weight 2^{m-1}. *Hint*: Start with $m = 2$ and work by induction.

3.13 Show that (3.13) follows from (3.12).

3.14 Show that (3.15) follows from (3.14) using the MacWilliams identity.

3.15 Let $f(u_1, u_2) = u_1 u_2$, for $u_i \in \mathbb{F}_2$. Determine the Hadamard transform $\hat{f}$ of f.

3.16 The weight enumerator $A(z)$ of (3.11) for a code C is sometimes written as $W_A(x, y) = \sum_{i=0}^{n} A_i x^{n-i} y^i$.

(a) Show that $A(z) = W_A(x, y)|_{x=1,y=z}$.
(b) Let $W_B(x, y) = \sum_{i=0}^{n} B_i x^{n-i} y^i$ be the weight enumerator for the code dual to C. Show that the MacWilliams identity can be written as

$$W_B(x, y) = \frac{1}{q^k} W_A(x + y, x - y)$$

or

$$W_A(x, y) = \frac{1}{q^{n-k}} W_B(x + y, x - y). \tag{3.23}$$

(c) In the following subproblems, assume a binary code. Let $x = 1$ in (3.23). We can write

$$\sum_{i=0}^{n} A_i y^i = \frac{1}{2^{n-k}} \sum_{i=0}^{n} B_i (1 + y)^{n-i} (1 - y)^i. \tag{3.24}$$

Set $y = 1$ in this and show that $\sum_{i=0}^{n} \frac{A_i}{2^k} = 1$. Justify this result.

(d) Now differentiate (3.24) with respect to y and set $y = 1$ to show that

$$\sum_{i=1}^{n} \frac{iA_i}{2^k} = \frac{1}{2}(n - B_1).$$

If $B_1 = 0$, this gives the average weight.

(e) Differentiate (3.24) ν times with respect to y and set $y = 1$ to show that

$$\sum_{i=\nu}^{n} \binom{i}{\nu} \frac{A_i}{2^k} = \frac{1}{2^\nu} \sum_{i=0}^{\nu} (-1)^i \binom{n-i}{\nu-i} B_i.$$

Hint: Define $(x)_+^n = \begin{cases} 0 & n < 0 \\ x^n & n \geq 0 \end{cases}$. We have the following generalization of the product rule for differentiation:

$$\frac{d^\nu}{dy^\nu}(y+a)^p (y+b)^q = \sum_{j=0}^{\nu} \binom{\nu}{j} \frac{p!}{(p-j)!}(y+a)_+^{p-j} \frac{q!}{(q-(\nu-j))!}(y+b)_+^{q-(\nu-j)}.$$

(f) Now set $y = 1$ in (3.23) and write

$$\sum_{i=0}^{n} A_i x^{n-i} = \frac{1}{2^{n-k}} \sum_{i=0}^{n} B_i (x+1)^{n-i}(x-1)^i.$$

Differentiate ν times with respect to x and set $x = 1$ to show that

$$\sum_{i=0}^{n-\nu} \binom{n-i}{\nu} A_i = 2^{k-\nu} \sum_{i=0}^{\nu} \binom{n-i}{n-\nu} B_i, \qquad \nu = 0, 1, \ldots, n. \tag{3.25}$$

3.17 Let C be a binary (n, k) code with weight enumerator $A(z)$ and let $\overline{C}$ be the extended code of length $n + 1$,

$$\overline{C} = \left\{ (c_0, c_1, \ldots, c_n) : (c_0, \ldots, c_{n-1}) \in C, \sum_{i=0}^{n} c_i = 0 \right\}.$$

Determine the weight enumerator for $\overline{C}$.

3.18 [204] Let C be a linear code with both even- and odd-weight codewords. Show that the number of even-weight codewords is equal to the number of odd-weight codewords.

3.19 Show that for a binary code, $P_u(E)$ can be written as:

(a) $P_u(E) = (1-p)^n \left[A\left(\frac{p}{1-p}\right) - 1 \right]$

(b) and $P_u(E) = 2^{k-n} B(1-2p) - (1-p)^n$.

3.20 [373] Find the lower bound on required redundancy for the following codes.

(a) A single-error correcting binary code of length 7.
(b) A single-error correcting binary code of length 15.
(c) A triple-error correcting binary code of length 23.
(d) A triple-error correcting 4-ary code (i.e., $q = 4$) of length 23.

3.21 Show that all odd-length binary repetition codes are perfect.

3.22 Show that Hamming codes achieve the Hamming bound.

3.23 Determine the weight distribution for a binary Hamming code of length 31. Determine the weight distribution of its dual code.

3.24 The parity check matrix for a nonbinary Hamming code of length $n = (q^m - 1)/(q - 1)$ and dimension $k = (q^m - 1)/(q - 1) - m$ with minimum distance 3 can be constructed as follows. For each q-ary m-tuple of the base-q representation of the numbers from 1 to $q^m - 1$, select those for which the first nonzero element is equal to 1. The list of all such m-tuples as columns gives the generator H.

(a) Explain why this gives the specified length n.

(b) Write down a parity check matrix in systematic form for the (5, 3) Hamming code over the field of four elements.

(c) Write down the corresponding generator matrix. *Note*: in this field, every element is its own additive inverse: $1 + 1 = 0, 2 + 2 = 0, 3 + 3 = 0$.

3.25 [204] Let G be the generator matrix of an (n, k) binary code C and let no column of G be all zeros. Arrange all the codewords of C as rows of a $2^k \times n$ array.

(a) Show that no column of the array contains only zeros.

(b) Show that each column of the array consists of 2^{k-1} zeros and 2^{k-1} ones.

(c) Show that the set of all codewords with zeros in particular component positions forms a subspace of C. What is the dimension of this subspace?

(d) Show that the minimum distance d_{min} of this code must satisfy the following inequality, known as the **Plotkin bound**:

$$d_{min} \le \frac{n2^{k-1}}{2^k - 1}.$$

3.26 [204] Let Γ be the ensemble of *all* the binary systematic linear (n, k) codes.

(a) Prove that a nonzero binary vector $\mathbf{v}$ is contained in exactly $2^{(k-1)(n-k)}$ of the codes in Γ or it is in none of the codes in Γ.

(b) Using the fact that the nonzero n-tuples of weight $d - 1$ or less can be in at most

$$2^{(k-1)(n-k)} \sum_{i=1}^{d-1} \binom{n}{i}$$

(n, k) systematic binary linear codes, show that there exists an (n, k) linear code with minimum distance of at least d if the following bound is satisfied:

$$\sum_{i=1}^{d-1} \binom{n}{i} < 2^{n-k}.$$

(c) Show that there exists an (n, k) binary linear code with minimum distance at least d that satisfies the following inequality:

$$2^{n-k} \le \sum_{i=0}^{d} \binom{n}{i}.$$

This provides a lower bound on the minimum distance attainable with an (n, k) linear code known as the **Gilbert-Varshamov bound**.

3.27 Define a linear (5,3) code over $GF(4)$ by the generator matrix

$$G = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 2 \\ 0 & 0 & 1 & 1 & 3 \end{bmatrix}.$$

(a) Find the parity-check matrix.
(b) Prove that this is a single-error-correcting code.
(c) Prove that it is a double-erasure-correcting code.
(d) Prove that it is a perfect code.

3.28 [203] Let H be the parity check matrix for an (n, k) linear code $\mathcal{C}$. Let $\mathcal{C}'$ be the extended code whose parity check matrix H' is formed by

$$H' = \left[\begin{array}{c|ccc} 0 & & & \\ 0 & & \text{H} & \\ \vdots & & & \\ 0 & & & \\ \hline 1 & 1 & 1 \cdots & 1 \end{array}\right].$$

(a) Show that every codeword of $\mathcal{C}'$ has even weight.
(b) Show that $\mathcal{C}'$ can be obtained from $\mathcal{C}$ by adding an extra parity bit called the overall parity bit to each codeword.

3.29 The $[\mathbf{u}|\mathbf{u}+\mathbf{v}]$ construction: Let $\mathcal{C}_i$, $i = 1, 2$ be linear binary (n, k_i) block codes with generator matrix G_i and minimum distance d_i. Define the code $\mathcal{C}$ by

$$\mathcal{C} = |\mathcal{C}_1|\mathcal{C}_1 + \mathcal{C}_2| = \{[\mathbf{u}|\mathbf{u}+\mathbf{v}] : \mathbf{u} \in \mathcal{C}_1, \mathbf{v} \in \mathcal{C}_2\}.$$

(a) Show that $\mathcal{C}$ has the generator

$$G = \begin{bmatrix} G_1 & G_1 \\ 0 & G_2 \end{bmatrix}.$$

(b) Show that the minimum distance of $\mathcal{C}$ is

$$d_{\min} = \min(2d_1, d_2).$$

3.12 References

The definitions of generator, parity check matrix, distance, and standard arrays are standard; see, for example, [203, 373]. The MacWilliams identity appeared in [219]. Extensions to nonlinear codes appear in [220]. The discussion of probability of error in Section 3.7 is drawn closely from [373]. Our discussion on modifications follows [373], which, in turn, draws from [25]. Our analysis of soft-input decoding was drawn from [15]. Classes of perfect codes are in [337].

Chapter 4

Cyclic Codes, Rings, and Polynomials

4.1 Introduction

We have seen that linear block codes can be corrected using the standard array, but that for long codes the storage and computation time can be prohibitive. Furthermore, we have not yet seen any mechanism by which the generator or parity check matrix can be *designed* to achieve a specified minimum distance or other criteria. In this chapter, we introduce *cyclic codes*, which have additional algebraic structure to make encoding and decoding more efficient. Following the introduction in this chapter, additional algebraic tools and concepts are presented in Chapter 5, which will provide for design specifications and lead to efficient algebraic decoding algorithms.

Cyclic codes are based on polynomial operations. A natural algebraic setting for the operations on polynomials is the algebraic structure of a *ring*.

4.2 Basic Definitions

Given a vector $\mathbf{c} = (c_0, c_1, \ldots, c_{n-2}, c_{n-1}) \in GF(q)^n$, the vector

$$\mathbf{c}' = (c_{n-1}, c_0, c_1, \ldots, c_{n-2})$$

is said to be a *cyclic shift* of $\mathbf{c}$ to the right. A shift by r places to the right produces the vector $(c_{n-r}, c_{n-r+1}, \ldots, c_{n-1}, c_0, c_1, \ldots, c_{n-r-1})$.

Definition 4.1 An (n, k) block code $\mathcal{C}$ is said to be **cyclic** if it is linear and if for every codeword $\mathbf{c} = (c_0, c_1, \ldots, c_{n-1})$ in $\mathcal{C}$, its right cyclic shift $\mathbf{c}' = (c_{n-1}, c_0, \ldots, c_{n-2})$ is also in $\mathcal{C}$. □

Example 4.1 We observed in Section 1.9.2 that the Hamming (7,4) code is cyclic; see the codeword list in (1.35). □

The operations of shifting and cyclic shifting can be conveniently represented using polynomials. The vector

$$\mathbf{c} = (c_0, c_1, \ldots, c_{n-1})$$

is represented by the polynomial

$$c(x) = c_0 + c_1 x + \cdots + c_{n-1} x^{n-1},$$

using the obvious one-to-one correspondence. We write this correspondence as

$$(c_0, c_1, \ldots, c_{n-1}) \leftrightarrow c_0 + c_1 x + \cdots + c_{n-1} x^{n-1}.$$

Box 4.1: The Division Algorithm

Let $p(x)$ be a polynomial of degree n and let $d(x)$ be a polynomial of degree m. That is, $\deg(p(x)) = n$ and $\deg(d(x)) = m$. Then the "division algorithm" for polynomials asserts that there exist polynomials $q(x)$ (the quotient) and $r(x)$ (the remainder), where $0 \leq \deg(r(x)) < m$ and

$$p(x) = q(x)d(x) + r(x).$$

The actual "algorithm" is polynomial long division with remainder. We say that $p(x)$ is equivalent to $r(x)$ modulo $d(x)$ and write this as

$$p(x) \equiv r(x) \mod d(x)$$

or

$$p(x) \pmod{d(x)} = r(x).$$

If $r(x) = 0$, then $d(x)$ divides $p(x)$, which we write as $d(x) \mid p(x)$. If $d(x)$ does not divide $p(x)$ this is denoted as $d(x) \nmid p(x)$.

A (noncyclic) shift is represented by polynomial multiplication:

$$xc(x) = c_0x + c_1x^2 + \cdots + c_{n-1}x^n$$

so

$$(0, c_0, c_1, \ldots, c_{n-1}) \leftrightarrow c_0x + c_1x^2 + \cdots + c_{n-1}x^n.$$

To represent the *cyclic* shift, we move the coefficient of x^n to the constant coefficient position by taking this product modulo $x^n - 1$. Dividing $xc(x)$ by $x^n - 1$ using the usual polynomial division with remainder (i.e., the "division algorithm;" see Box 4.1), we obtain

$$xc(x) = \underbrace{c_{n-1}}_{\text{quotient}}(x^n - 1) + \underbrace{(c_0x + c_1x^2 + \cdots + c_{n-2}x^{n-1} + c_{n-1})}_{\text{remainder}}$$

so that the remainder upon dividing by $x^n - 1$ is

$$xc(x) \pmod{x^n - 1} = c_{n-1} + c_0x + \cdots + c_{n-2}x^{n-1}.$$

4.3 Rings

We now introduce an algebraic structure, the **ring**, which is helpful in our study of cyclic codes. We have met the concept of a group in Chapter 2 . Despite their usefulness in a variety of areas, groups are still limited because they have only one operation associated with them. Rings, on the other hand, have two operations associated with them.

Definition 4.2 A **ring** $\langle R, +, \cdot \rangle$ is a set R with two binary operations $+$ (addition) and $\cdot$ (multiplication) defined on R such that:

R1 $\langle R, + \rangle$ is an Abelian (commutative) group. We typically denote the additive identity as 0.

R2 The multiplication operation $\cdot$ is associative: $(a \cdot b) \cdot c = a \cdot (b \cdot c)$ for all $a, b, c \in R$.

R3 The left and right distributive laws hold:

$$a(b + c) = ab + ac,$$
$$(a + b)c = (ac) + (bc).$$

A ring is said to be a **commutative ring** if $a \cdot b = b \cdot a$ for every $a, b \in R$.

The ring $\langle R, +, \cdot \rangle$ is frequently referred to simply as R.

A ring is said to be a **ring with identity** if $\cdot$ has an identity element. This is typically denoted as 1. □

Notice that we do not require that the multiplication operation form a group: there may not be multiplicative inverses in a ring (even if it has an identity). Nor is the multiplication operation necessarily commutative. All of the rings that we deal with in this book are rings with identity.

Some of the elements of a ring *may* have a multiplicative inverse. An element a in a ring having a multiplicative inverse is said to be a **unit**.

Example 4.2 The set of 2×2 matrices under usual definitions of addition and multiplication form a ring. (This ring is not commutative, nor does every element have an inverse.) □

Example 4.3 $\langle \mathbb{Z}_6, +, \cdot \rangle$ forms a ring.

+	0	1	2	3	4	5
0	0	1	2	3	4	5
1	1	2	3	4	5	0
2	2	3	4	5	0	1
3	3	4	5	0	1	2
4	4	5	0	1	2	3
5	5	0	1	2	3	4

·	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	1	2	3	4	5
2	0	2	4	0	2	4
3	0	3	0	3	0	3
4	0	4	2	0	4	2
5	0	5	4	3	2	1

It is clear that multiplication under $\mathbb{Z}_6$ does *not* form a group. But $\mathbb{Z}_6$ still satisfies the requirements to be a ring. □

Definition 4.3 Let R be a ring and let $a \in R$. For an integer n, let na denote $a + a + \cdots + a$ with n arguments. If a positive integer exists such that $na = 0$ for all $a \in R$, then the *smallest* such positive integer is the **characteristic of the ring** R. If no such positive integer exists, the R is said to be a ring of characteristic 0. □

Example 4.4 In the ring $\mathbb{Z}_6$, the characteristic is 6. In the ring $\langle \mathbb{Z}_n, +, \cdot \rangle$, the characteristic is n. In the ring $\mathbb{Q}$, the characteristic is 0. □

4.3.1 Rings of Polynomials

Let R be a ring. A polynomial $f(x)$ of degree n with coefficients in R is

$$f(x) = \sum_{i=0}^{n} a_i x^i,$$

where $a_n \neq 0$. The symbol x is said to be an *indeterminate*.

Definition 4.4 The set of all polynomials with an indeterminate x with coefficients in a ring R, using the usual operations for polynomial addition and multiplication, forms a ring called the **polynomial ring**. It is denoted as $R[x]$. □

Example 4.5 Let $R = \langle \mathbb{Z}_6, +, \cdot \rangle$ and let $S = R[x] = \mathbb{Z}_6[x]$. Then some elements in S are: $0, 1, x, 1 + x, 4 + 2x, 5 + 4x$, etc. Example operations are

$$(4 + 2x) + (5 + 4x) = 3$$
$$(4 + 2x)(5 + 4x) = 2 + 2x + 2x^2.$$

□

Example 4.6 $\mathbb{Z}_2[x]$ is the ring of polynomials with coefficients that are either 0 or 1 with operations modulo 2. As an example of arithmetic in this ring,

$$(1 + x)(1 + x) = 1 + x + x + x^2 = 1 + x^2,$$

since $x + x = 0$ in $\mathbb{Z}_2$. □

It is clear that polynomial multiplication does not, in general, have an inverse. For example, in the ring of polynomials with real coefficients $\mathbb{R}[x]$, there is no polynomial solution $f(x)$ to

$$f(x)(x^2 + 3x + 1) = x^3 + 2x + 1.$$

Polynomials can represent a sequence of numbers in a single collective object. One reason polynomials are of interest is that polynomial multiplication is equivalent to convolution. The convolution of the sequence

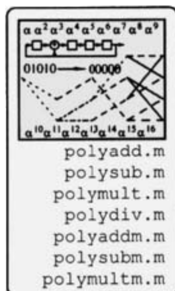
$$\mathbf{a} = \{a_0, a_1, a_2, \ldots, a_n\}$$

with the sequence

$$\mathbf{b} = \{b_0, b_1, b_2, \ldots, b_m\}$$

can be accomplished by forming the polynomials

$$a(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n$$
$$b(x) = b_0 + b_1 x + b_2 x^2 + \cdots + b_m x^m$$



and multiplying them

$$c(x) = a(x)b(x).$$

Then the coefficients of

$$c(x) = c_0 + c_1 x + c_2 x^2 + \cdots + c_{n+m} x^{n+m}$$

are equal to the values obtained by convolving $\mathbf{a} * \mathbf{b}$.

4.4 Quotient Rings

Recall the idea of factor groups introduced in Section 2.2.5: Given a group and a subgroup, a set of cosets was formed by "translating" the subgroup. We now do a similar construction over a ring of polynomials. We assume that the underlying ring is commutative (to avoid certain technical issues). We begin with a particular example, then generalize.

Consider the ring of polynomials $GF(2)[x]$ (polynomials with binary coefficients) and a polynomial $x^3 - 1$.[1] Let us divide the polynomials up into equivalence classes depending on their remainder modulo $x^3 + 1$. For example, the polynomials in

$$S_0 = \{0, x^3 + 1, x^4 + x, x^5 + x^2, x^6 + x^3, \ldots\}$$

[1] In a ring of characteristic 2, $x^n - 1 = x^n + 1$. However, in other rings, the polynomial should be of the form $x^n - 1$.

all have remainder 0 when divided by x^3+1. We write $S_0 = \langle x^3+1 \rangle$, the set generated by x^3+1. The polynomials in

$$S_1 = \{1, x^3, x^4+x+1, x^5+x^2+1, x^6+x^3+1, \ldots\}$$

all have remainder 1 when divided by x^3+1. We can write

$$S_1 = 1 + S_0 = 1 + \langle x^3+1 \rangle.$$

Similarly, the other equivalence classes are

$$\begin{aligned}
S_2 &= \{x, x^3+x+1, x^4, x^5+x^2+x, x^6+x^3+x, \ldots\} \\
&= x + S_0 \\
S_3 &= \{x+1, x^3+x, x^4+1, x^5+x^2+x+1, x^6+x^3+x+1, \ldots\} \\
&= x + 1 + S_0 \\
S_4 &= \{x^2, x^3+x^2+1, x^4+x^2+x, x^5, x^6+x^3+x^2, \ldots\} \\
&= x^2 + S_0 \\
S_5 &= \{x^2+1, x^3+x^2, x^4+x^2+x+1, x^5+1, x^6+x^3+x^2+1, \ldots\} \\
&= x^2 + 1 + S_0 \\
S_6 &= \{x^2+x, x^3+x^2+x+1, x^4+x^2, x^5+x, x^6+x^3+x^2+x, \ldots\} \\
&= x^2 + x + S_0 \\
S_7 &= \{x^2+x+1, x^3+x^2+x, x^4+x^2+1, x^5+x+1, x^6+x^3+x^2+x+1, \ldots\} \\
&= x^2 + x + 1 + S_0
\end{aligned}$$

Thus, $S_0, S_1, \ldots, S_7$ form the cosets of $\langle GF(2)[x], + \rangle$ modulo the subgroup $\langle x^3+1 \rangle$. These equivalence classes exhaust all possible remainders after dividing by x^3+1. It is clear that every polynomial in $GF(2)[x]$ falls into one of these eight sets.

Just as we defined an induced group operation for the cosets of Section 2.2.5 to create the factor group, so we can define induced ring operations for both $+$ and $\cdot$ for the equivalence classes of polynomials modulo x^3+1 by operation on representative elements. This gives us the following addition and multiplication tables.

$+$	S_0	S_1	S_2	S_3	S_4	S_5	S_6	S_7
S_0	S_0	S_1	S_2	S_3	S_4	S_5	S_6	S_7
S_1	S_1	S_0	S_3	S_2	S_5	S_4	S_7	S_6
S_2	S_2	S_3	S_0	S_1	S_6	S_7	S_4	S_5
S_3	S_3	S_2	S_1	S_0	S_7	S_6	S_5	S_4
S_4	S_4	S_5	S_6	S_7	S_0	S_1	S_2	S_3
S_5	S_5	S_4	S_7	S_6	S_1	S_0	S_3	S_2
S_6	S_6	S_7	S_4	S_5	S_2	S_3	S_0	S_1
S_7	S_7	S_6	S_5	S_4	S_3	S_2	S_1	S_0

$\cdot$	S_0	S_1	S_2	S_3	S_4	S_5	S_6	S_7
S_0	S_0	S_0	S_0	S_0	S_0	S_0	S_0	S_0
S_1	S_0	S_1	S_2	S_3	S_4	S_5	S_6	S_7
S_2	S_0	S_2	S_4	S_6	S_1	S_3	S_5	S_7
S_3	S_0	S_3	S_6	S_5	S_5	S_6	S_3	S_0
S_4	S_0	S_4	S_1	S_5	S_2	S_6	S_3	S_7
S_5	S_0	S_5	S_3	S_6	S_6	S_3	S_5	S_0
S_6	S_0	S_6	S_5	S_3	S_3	S_5	S_6	S_0
S_7	S_0	S_7	S_7	S_0	S_7	S_0	S_0	S_7

Let $R = \{S_0, S_1, \ldots, S_7\}$. From the addition table, $\langle R, + \rangle$ clearly forms an Abelian group, with S_0 as the identity. For the multiplicative operation, S_1 clearly acts as an identity. However, not every element has a multiplicative inverse, so $\langle R \setminus S_0, \cdot \rangle$ does not form a group. However, $\langle R, +, \cdot \rangle$ does define a **ring**. The ring is denoted as $GF(2)[x]/\langle x^3+1 \rangle$ or sometimes by $GF(2)[x]/(x^3+1)$, the *ring of polynomials in* $GF(2)[x]$ *modulo* x^3+1.

We denote the ring $GF(2)[x]/\langle x^n - 1\rangle$ by R_n. We denote the ring $\mathbb{F}_q[x]/\langle x^n - 1\rangle$ as $R_{n,q}$.

Each equivalence class can be identified uniquely by its element of lowest degree.

$$\begin{array}{ll} S_0 \leftrightarrow 0 & S_1 \leftrightarrow 1 \\ S_2 \leftrightarrow x & S_3 \leftrightarrow x+1 \\ S_4 \leftrightarrow x^2 & S_5 \leftrightarrow x^2+1 \\ S_6 \leftrightarrow x^2+x & S_7 \leftrightarrow x^2+x+1 \end{array}$$

Let $\mathcal{R} = \{0, 1, x, x+1, x^2, x^2+1, x^2+x, x^2+x+1\}$. Define the addition operation in $\mathcal{R}$ as conventional polynomial addition, and the multiplication operation as polynomial multiplication, followed by computing the remainder modulo x^3+1. Then $\langle \mathcal{R}, +, \cdot \rangle$ forms a ring.

Definition 4.5 Two rings $\langle R, +, \cdot\rangle$ and $\langle \mathcal{R}, +, \cdot\rangle$ are said to be (ring) **isomorphic** if there exists a bijective function $\phi : G \to \mathcal{G}$ called the **isomorphism** such that for every $a, b \in R$,

$$\phi(\underbrace{a+b}_{\substack{\text{operation}\\ \text{in } R}}) = \underbrace{\phi(a)+\phi(b)}_{\substack{\text{operation}\\ \text{in } \mathcal{R}}} \qquad \phi(\underbrace{a\cdot b}_{\substack{\text{operation}\\ \text{in } R}}) = \underbrace{\phi(a)\cdot\phi(b)}_{\substack{\text{operation}\\ \text{in } \mathcal{R}}} \tag{4.1}$$

Ring homomorphism is similarly defined: the function ϕ no longer needs to be bijective, but (4.1) still applies. □

Clearly the rings R (where operation is by representative elements, defined in the tables above) and $\mathcal{R}$ (defined by polynomial operations modulo x^3+1) are isomorphic.

Note that we can factor $x^3+1 = (x+1)(x^2+x+1)$. Also note from the table that in R, $S_3S_7 = S_0$. Equivalently, in $\mathcal{R}$,

$$(x+1)(x^2+x+1) = 0.$$

This is clearly true, since to multiply, we compute the conventional product $(x+1)(x^2+x+1) = x^3+1$, then compute the remainder modulo x^3+1, which is 0. We shall make use of analogous operations in computing syndromes.

More generally, for a field $\mathbb{F}$, the ring of polynomials $\mathbb{F}[x]$ can be partitioned by a polynomial $f(x)$ of degree m into a ring consisting of q^m different equivalence classes, with one equivalence class for each remainder modulo $f(x)$, where $q = |\mathbb{F}|$. This ring is denoted as $\mathbb{F}[x]/\langle f(x)\rangle$ or $\mathbb{F}[x]/f(x)$. A question that arises is under what conditions this ring is, in fact, a field? As we will develop much more fully in Chapter 5, the ring $\mathbb{F}[x]/f(x)$ is a field if and only if $f(x)$ cannot be factored over $\mathbb{F}[x]$. In the example above we have

$$x^3+1 = (x+1)(x^2+x+1),$$

so x^3+1 is reducible and we do not get a field.

4.5 Ideals in Rings

Definition 4.6 Let R be a ring. A nonempty subset $I \subseteq R$ is an **ideal** if it satisfies the following conditions:

I1 I forms a group under the addition operation in R.

I2 For any $a \in I$ and any $r \in R$, $ar \in I$.

□

Example 4.7

1. For any ring R, 0 and R are (trivial) ideals in R.
2. The set $I = \{0, x^5+x^4+x^3+x^2+x+1\}$ forms an ideal in R_6. For example, let $1+x+x^2 \in R_6$. Then

$$\begin{aligned}(1+x+x^2)(x^5+x^4+x^3+x^2+x+1) &= x^7+x^5+x^4+x^3+x^2+1 \pmod{x^6+1}\\ &= x^5+x^4+x^3+x^2+x+1 \in I.\end{aligned}$$

□

Example 4.8 Let R be a ring and let $R[x_1, x_2, \ldots, x_n]$ be the ring of polynomials in the n indeterminates $x_1, x_2, \ldots, x_n$.

Ideals in a the polynomial ring $R[x_1, \ldots, x_n]$ are often generated by a finite number of polynomials. Let $f_1, f_2, \ldots, f_s$ be polynomials in $R[x_1, \ldots, x_n]$. Let $\langle f_1, f_2, \ldots, f_s\rangle$ be the set

$$\langle f_1, f_2, \ldots, f_s\rangle = \left\{\sum_{i=1}^{s} h_i f_i : h_1, \ldots, h_s \in R[x_1, \ldots, x_n]\right\}.$$

That is, it is the set of all polynomials which are linear combinations of the $\{f_i\}$. The set $\langle f_1, \ldots, f_s\rangle$ is an ideal.

Thus, an ideal is similar to a subspace, generated by a set of basis vectors, except that to create a subspace, the coefficients are scalars, whereas for an ideal, the coefficients are polynomials. □

The direction toward which we are working is the following:

Cyclic codes form ideals in a ring of polynomials.

In fact, for cyclic codes the ideals are principal, as defined by the following.

Definition 4.7 An ideal I in a ring R is said to be **principal** if there exists some $g \in I$ such that every element $a \in I$ can be expressed as a product $a = mg$ for some $m \in R$. For a principal ideal, such an element g is called the **generator element.** The ideal generated by g is denoted as $\langle g\rangle$:

$$\langle g\rangle = \{hg : h \in R\}.$$

□

Theorem 4.1 *Let I be an ideal in $\mathbb{F}_q[x]/(x^n - 1)$. Then*

1. *There is a unique monic polynomial $g(x) \in I$ of minimal degree.*[2]
2. *I is principal with generator $g(x)$.*
3. *$g(x)$ divides $(x^n - 1)$ in $\mathbb{F}_q[x]$.*

[2] A polynomial is monic if the coefficient of the *leading term* — the term of highest degree — is equal to 1.

Proof There is at least one ideal (so the result is not vacuous, since the entire ring is an ideal). There is a lower bound on the degrees of polynomials in the ideal. Hence there must be at least one polynomial in the ideal of minimal degree, which may be normalized to be monic. Now to show uniqueness, let $g(x)$ and $f(x)$ be monic polynomials in I of minimal degree with $f \neq g$. Then $h(x) = g(x) - f(x)$ must be in I since I forms a group under addition, and $h(x)$ must be of lower degree, contradicting the minimality of the degree of g and f.

To show that I is principal, we assume (to the contrary) that there is an $f(x) \in I$ that is not a multiple of $g(x)$. Then by the division algorithm

$$f(x) = m(x)g(x) + r(x)$$

with $\deg(r) < \deg(g)$. But $m(x)g(x) \in I$ (definition of an ideal) and $r = f - mg \in I$ (definition of ideal), contradicting the minimality of the degree of g, unless $r = 0$.

To show that $g(x)$ divides $(x^n - 1)$, we assume to the contrary that $g(x)$ does not divide $(x^n - 1)$. By the division algorithm

$$x^n - 1 = h(x)g(x) + r(x)$$

with $0 \leq \deg(r) < \deg(g)$. But $h(x)g(x) \in I$ and $r(x) = (x^n - 1) - h(x)g(x)$ is the additive inverse of $h(x)g(x) \in I$, and so is in I, contradicting the minimality of the degree of g. □

If a monic polynomial $g(x)$ divides $(x^n - 1)$, then it can be used to generate an ideal: $I = \langle g(x) \rangle$.

In the ring $\mathbb{F}_q[x]/(x^n - 1)$, different ideals can be obtained by selecting different divisors $g(x)$ of $x^n - 1$.

Example 4.9 By multiplication, it can be shown that in $GF(2)[x]$,

$$x^7 + 1 = (x + 1)(x^3 + x + 1)(x^3 + x^2 + 1).$$

In the ring $GF(2)[x]/(x^7 + 1)$, there are ideals corresponding to the different factorizations of $x^7 + 1$, so there are the following nontrivial ideals:

$$\langle x + 1 \rangle \qquad \langle x^3 + x + 1 \rangle \qquad \langle x^3 + x^2 + 1 \rangle$$

$$\langle (x + 1)(x^3 + x + 1) \rangle \qquad \langle (x + 1)(x^3 + x^2 + 1) \rangle \qquad \langle (x^3 + x + 1)(x^3 + x^2 + 1) \rangle.$$

□

4.6 Algebraic Description of Cyclic Codes

Let us return now to cyclic codes. As mentioned in Section 4.1, cyclic shifting of a polynomial $c(x)$ can be represented by $xc(x)$ modulo $x^n - 1$. Now think of $c(x)$ as an element of $GF(q)[x]/(x^n - 1)$. Then *in that ring*, $xc(x)$ is a cyclic shift, since operations in the ring are defined modulo $x^n - 1$. Any power of x times a codeword yields a codeword so that, for example,

$$(c_{n-1}, c_0, c_1, \ldots, c_{n-2}) \leftrightarrow xc(x)$$

$$(c_{n-2}, c_{n-1}, c_0, \ldots, c_{n-3}) \leftrightarrow x^2c(x)$$

$$\vdots$$

$$(c_1, c_2, \ldots, c_{n-1}, c_0) \leftrightarrow x^{n-1}c(x),$$

where the arithmetic on the right is done in the ring $GF(q)[x]/(x^n - 1)$. Furthermore, multiples of these codewords are also codewords, so that $a_1 x c(x)$ is a codeword for $a_1 \in GF(q)$, $a_2 x^2 c(x)$ is a codeword for $a_2 \in GF(q)$, etc. Furthermore, any linear combination of such codewords must be a codeword (since the code is linear. Let $\mathcal{C}$ be a cyclic code over $GF(q)$ and let $c(x) \in GF(q)[x]/(x^n - 1)$ be a polynomial representing a codeword in $\mathcal{C}$. If we take a polynomial $a(x) \in GF(q)[x]/(x^n - 1)$ of the form

$$a(x) = a_0 + a_1 x + \cdots + a_{n-1} x^{n-1}$$

then

$$c(x)a(x)$$

is simply a linear combination of cyclic shifts of $c(x)$, which is to say, a linear combination of codewords in $\mathcal{C}$. Thus $c(x)a(x)$ is also a codeword. Since linear codes form a group under addition we see that **a cyclic code is an ideal in** $GF(q)[x]/(x^n - 1)$. From Theorem 4.1, we can immediately make some observations about cyclic codes:

- An (n, k) cyclic code has a unique minimal monic polynomial $g(x)$, which is the generator of the ideal. This is called the **generator polynomial** for the code. Let the degree of g be $n - k$,

  $$g(x) = g_0 + g_1 x + g_2 x^2 + \cdots + g_{n-k} x^{n-k},$$

  and let $r = n - k$ (the redundancy of the code).
- Every code polynomial in the code can be expressed as a multiple of the generator

  $$c(x) = m(x)g(x),$$

  where $m(x)$ is a *message polynomial*. The degree of $m(x)$ is (strictly) less than k,

  $$m(x) = m_0 + m_1 x + \cdots + m_{k-1} x^{k-1}.$$

  There are k independently selectable coefficients in $m(x)$, so the dimension of the code is k. Then $c(x) = m(x)g(x)$ has degree $\leq n - 1$, so that n coded symbols can be represented:

  $$\begin{aligned} c(x) &= c_0 + c_1 x + c_2 x^2 + \cdots + c_{n-1} x^{n-1} \\ &= (g_0 + g_1 x + \cdots + g_{n-k} x^{n-k})(m_0 + m_1 x + m_2 x^2 + \cdots + m_{k-1} x^{k-1}). \end{aligned}$$
- The generator is a factor of $x^n - 1$ in $GF(q)[x]$.

Example 4.10 We consider cyclic codes of length 15 with binary coefficients. By multiplication it can be verified that

$$x^{15} - 1 = (1 + x)(1 + x + x^2)(1 + x + x^4)(1 + x + x^2 + x^3 + x^4)(1 + x^3 + x^4).$$

So there are polynomials of degrees 1, 2, 4, 4, and 4 which can be used to construct generators. The product of any combination of these can be used to construct a generator polynomial. If we want a generator of, say, degree 10, we could take

$$g(x) = (1 + x + x^2)(1 + x + x^4)(1 + x + x^2 + x^3 + x^4).$$

If we want a generator of degree 5 we could take

$$g(x) = (1 + x)(1 + x + x^4)$$

or

$$g(x) = (1+x)(1+x+x^2+x^3+x^4).$$

In fact, in this case, we can get generator polynomials of any degree from 1 to 15. So we can construct the (n, k) codes

$$(15, 1), (15, 2), \ldots, (15, 15).$$

□

4.7 Nonsystematic Encoding and Parity Check

A message vector $\mathbf{m} = \begin{bmatrix} m_0 & m_1 & \ldots & m_{k-1} \end{bmatrix}$ corresponds to a message polynomial

$$m(x) = m_0 + \cdots + m_{k-1}x^{k-1}.$$

Then the code polynomial corresponding to $m(x)$ is obtained by the **encoding operation** of polynomial multiplication:

$$\begin{aligned} c(x) &= m(x)g(x) \\ &= (m_0 g(x) + m_1 x g(x) + \cdots m_{k-1} x^{k-1} g(x)). \end{aligned}$$

This is not a systematic encoding operation; systematic encoding is discussed below. The encoding operation can be written as

$$c(x) = \begin{bmatrix} m_0 & m_1 & m_2 & \cdots & m_{k-1} \end{bmatrix} \begin{bmatrix} g(x) \\ xg(x) \\ x^2 g(x) \\ \vdots \\ x^{k-1} g(x) \end{bmatrix}.$$

This can also be expressed as

$$\mathbf{c}_m = \begin{bmatrix} m_0, m_1, \ldots, m_{k-1} \end{bmatrix} \begin{bmatrix} g_0 & g_1 & \cdots & g_r & & & & & \\ & g_0 & g_1 & \cdots & g_r & & & & \\ & & g_0 & g_1 & \cdots & g_r & & & \\ & & & \ddots & \ddots & & \ddots & & \\ & & & & g_0 & g_1 & \cdots & g_r & \\ & & & & & g_0 & g_1 & \cdots & g_r \end{bmatrix},$$

(where empty locations are equal to 0) or

$$\mathbf{c} = \mathbf{m}G,$$

where G is a $k \times n$ matrix. A matrix such as this which is constant along the diagonals is said to be a **Toeplitz** matrix.

Example 4.11 Let $n = 7$ and let

$$g(x) = (x^3 + x + 1)(x + 1) = 1 + x^2 + x^3 + x^4,$$

so that the code is a (7, 3) code. Then a generator matrix for the code can be expressed as

$$G = \begin{bmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix}.$$

The codewords in the code are as shown in Table 4.1.

□

Table 4.1: Codewords in the Code Generated by $g(x) = 1 + x^2 + x^3 + x^4$

m	$m(x)g(x)$	code polynomial	codeword
(0,0,0)	$0g(x)$	0	0000000
(1,0,0)	$1g(x)$	$1 + x^2 + x^3 + x^4$	1011100
(0,1,0)	$xg(x)$	$x + x^3 + x^4 + x^5$	0101110
(1,1,0)	$(x + 1)g(x)$	$1 + x + x^2 + x^5$	1110010
(0,0,1)	$x^2g(x)$	$x^2 + x^4 + x^5 + x^6$	0010111
(1,0,1)	$(x^2 + 1)g(x)$	$1 + x^3 + x^5 + x^6$	1001011
(0,1,1)	$(x^2 + x)g(x)$	$x + x^2 + x^3 + x^6$	0111001
(1,1,1)	$(x^2 + x + 1)g(x)$	$1 + x + x^4 + x^6$	1100101

For a cyclic code of length n with generator $g(x)$, there is a corresponding polynomial $h(x)$ of degree k satisfying $h(x)g(x) = x^n - 1$. This polynomial is called the **parity check polynomial**. Since codewords are exactly the multiples of $g(x)$, then for a codeword,

$$c(x)h(x) = m(x)g(x)h(x) = m(x)(x^n - 1) \equiv 0 \quad (\text{in } GF(q)[x]/(x^n - 1)).$$

Thus a polynomial $r(x)$ can be examined to see if it is a codeword: $r(x)$ is a codeword if and only if $r(x)h(x) \pmod{x^n - 1}$ is equal to 0.

As for linear block codes, we can define a *syndrome*. This can be accomplished several ways. One way is to define the **syndrome polynomial** corresponding to the received data $r(x)$ as

$$s(x) = r(x)h(x) \pmod{x^n - 1}. \tag{4.2}$$

$s(x)$ is identically zero if and only if $r(x)$ is a codeword.

Let us construct a parity check matrix corresponding to the parity check polynomial $h(x)$. Let $c(x)$ represent a code polynomial in $\mathcal{C}$, so $c(x) = m(x)g(x)$ for some message $m(x) = m_0 + m_1x + \cdots m_{k-1}x^{k-1}$. Then

$$c(x)h(x) = m(x)g(x)h(x) = m(x)(x^n - 1) = m(x) - m(x)x^n.$$

Since $m(x)$ has degree less than k, then powers $x^k, x^{k+1}, \ldots, x^{n-1}$ do not appear[3] in $m(x) - m(x)x^n$. Thus the coefficients of $x^k, x^{k+1}, \ldots, x^{n-1}$ in the product $c(x)h(x)$ must be 0. Thus

$$\sum_{i=0}^{k} h_i c_{l-i} = 0 \text{ for } l = k, k+1, \ldots, n-1. \tag{4.3}$$

This can be expressed as

$$\begin{bmatrix} h_k & h_{k-1} & h_{k-2} & \cdots & h_0 & & & & & \\ & h_k & h_{k-1} & h_{k-2} & \cdots & h_0 & & & & \\ & & h_k & h_{k-1} & h_{k-2} & \cdots & h_0 & & & \\ & & & \ddots & & & & \ddots & & \\ & & & & h_k & h_{k-1} & h_{k-2} & \cdots & h_0 \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ c_{n-1} \end{bmatrix} = \mathbf{0}. \tag{4.4}$$

[3]This "trick" of observing which powers are absent is a very useful one, and we shall see it again.

Thus the parity check matrix H can be expressed as the $(n-k)\times n$ Toeplitz matrix

$$H = \begin{bmatrix} h_k & h_{k-1} & h_{k-2} & \cdots & h_0 & & & & & \\ & h_k & h_{k-1} & h_{k-2} & \cdots & h_0 & & & & \\ & & h_k & h_{k-1} & h_{k-2} & \cdots & h_0 & & & \\ & & & \ddots & & & & \ddots & & \\ & & & & h_k & h_{k-1} & h_{k-2} & \cdots & h_0 \end{bmatrix}.$$

Example 4.12 For the (7,4) cyclic code of Example 4.11 generated by $g(x) = x^4 + x^3 + x^2 + 1$, the parity check polynomial is

$$h(x) = \frac{x^7+1}{x^4+x^3+x^2+1} = x^3 + x^2 + 1.$$

The parity check matrix is

$$H = \begin{bmatrix} 1 & 1 & 0 & 1 & & & \\ & 1 & 1 & 0 & 1 & & \\ & & 1 & 1 & 0 & 1 & \\ & & & 1 & 1 & 0 & 1 \end{bmatrix}.$$

It can be verified that $GH^T = \mathbf{0}$ (in $GF(2)$). □

4.8 Systematic Encoding

With only a little more effort, cyclic codes can be encoded in systematic form. We take the message vector and form a message polynomial from it,

$$\mathbf{m} = (m_0, m_1, \ldots, m_{k-1}) \leftrightarrow m(x) = m_0 + m_1 x + \cdots + m_{k-1}x^{k-1}.$$

Now take the message polynomial and shift it to the right $n-k$ positions:

$$x^{n-k}m(x) = m_0 x^{n-k} + m_1 x^{n-k+1} + \cdots + m_{k-1}x^{n-1}.$$

Observe that the vector corresponding to this is

$$(\underbrace{0, 0, \ldots, 0}_{n-k}, m_0, m_1, \ldots, m_{k-1}) \leftrightarrow x^{n-k}m(x).$$

Now divide $x^{n-k}m(x)$ by the generator $g(x)$ to obtain a quotient and remainder

$$x^{n-k}m(x) = q(x)g(x) + d(x),$$

where $q(x)$ is the quotient and $d(x)$ is the remainder, having degree less than $n-k$. We use the notation $R_{g(x)}[\cdot]$ to denote the operation of computing the remainder of the argument when dividing by $g(x)$. Thus we have

$$d(x) = R_{g(x)}[x^{n-k}m(x)].$$

By the degree of $d(x)$, it corresponds to the code sequence

$$(d_0, d_1, \ldots, d_{n-k-1}, 0, 0, \ldots, 0) \leftrightarrow d(x).$$

Now form

$$x^{n-k}m(x) - d(x) = q(x)g(x).$$

Since the left-hand side is a multiple of $g(x)$, it must be a codeword. It has the vector representation

$$(-d_0, -d_1, \ldots, -d_{n-k-1}, m_0, m_1, \ldots, m_{k-1}) \leftrightarrow x^{n-k}m(x) - d(x).$$

The message symbols appear explicitly in the last k positions of the vector. Parity symbols appear in the first $n-k$ positions. This gives us a systematic encoding.

Example 4.13 We demonstrate systematic coding in the (7, 3) code from Example 4.11. Let $\mathbf{m} = (1, 0, 1) \leftrightarrow m(x) = 1 + x^2$.

1. Compute $x^{n-k}m(x) = x^4 m(x) = x^4 + x^6$.
2. Employ the division algorithm:
$$x^4 + x^6 = (1 + x + x^2)(1 + x^2 + x^3 + x^4) + (1 + x).$$
The remainder is $(1 + x)$.
3. Then the code polynomial is
$$c(x) = x^{n-k}m(x) - d(x) = (1 + x) + (x^4 + x^6) \leftrightarrow (1, 1, 0, 0, \underbrace{1, 0, 1}_{\mathbf{m}}).$$

□

A systematic representation of the generator matrix is also readily obtained. Dividing x^{n-k+i} by $g(x)$ using the division algorithm we obtain

$$x^{n-k+i} = q_i(x)g(x) + b_i(x), \qquad i = 0, 1, \ldots, k-1,$$

where $b_i(x) = b_{i,0} + b_{i,1}x + \cdots + b_{i,n-k-1}x^{n-k-1}$ is the remainder. Equivalently,

$$x^{n-k+i} - b_i(x) = q_i(x)g(x),$$

so $x^{n-k+i} - b_i(x)$ is a multiple of $g(x)$ and must be a codeword. Using these codewords for $i = 0, 1, \ldots, k-1$ to form the rows of the generator matrix, we obtain

$$G = \begin{bmatrix} -b_{0,0} & -b_{0,1} & \cdots & -b_{0,n-k-1} & 1 & 0 & 0 & \cdots & 0 \\ -b_{1,0} & -b_{1,1} & \cdots & -b_{1,n-k-1} & 0 & 1 & 0 & \cdots & 0 \\ -b_{2,0} & -b_{2,1} & \cdots & -b_{2,n-k-1} & 0 & 0 & 1 & \cdots & 0 \\ \vdots & & & & & & & & \\ -b_{k-1,0} & -b_{k-1,1} & \cdots & -b_{k-1,n-k-1} & 0 & 0 & 0 & \cdots & 1 \end{bmatrix}.$$

The corresponding parity check matrix is

$$H = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 & b_{0,0} & b_{1,0} & b_{2,0} & \cdots & b_{k-1,0} \\ 0 & 1 & 0 & \cdots & 0 & b_{0,1} & b_{1,1} & b_{2,1} & \cdots & b_{k-1,1} \\ 0 & 0 & 1 & \cdots & 0 & b_{0,2} & b_{1,2} & b_{2,2} & \cdots & b_{k-1,2} \\ \vdots & & & & & & & & & \\ 0 & 0 & 0 & \cdots & 1 & b_{0,n-k-1} & b_{1,n-k-1} & b_{2,n-k-1} & \cdots & b_{k-1,n-k-1} \end{bmatrix}.$$

Example 4.14 Let $g(x) = 1 + x + x^3$. The $b_i(x)$ polynomials are obtained as follows:

$$\begin{array}{lll} i = 0: \; x^3 & = g(x) + (1 + x) & b_0(x) = 1 + x \\ i = 1: \; x^4 & = xg(x) + (x + x^2) & b_1(x) = x + x^2 \\ i = 2: \; x^5 & = (x^2 + 1)g(x) + (1 + x + x^2) & b_2(x) = 1 + x + x^2 \\ i = 3: \; x^6 & = (x^3 + x + 1)g(x) + (1 + x^2) & b_3(x) = 1 + x^2 \end{array}$$

The generator and parity matrices are

$$G = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix} \qquad H = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix}.$$

□

For systematic encoding, error detection can be readily accomplished. Consider the systematically-encoded codeword

$$\mathbf{c} = (-d_0, -d_1, \ldots, -d_{n-k-1}, m_0, m_1, \ldots, m_{k-1}) = (-\mathbf{d}, \mathbf{m}).$$

We can perform error *detection* as follows:

1. Estimate a message based on the systematic message part of $\mathbf{r}$. Call this $\mathbf{m}'$.
2. Encode $\mathbf{m}'$. Compare the parity bits from this to the received parity bits. If they don't match, then an error is detected.

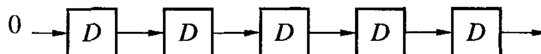
4.9 Some Hardware Background

One of the justifications for using cyclic codes, and using the polynomial representation in general, is that there are efficient hardware configurations for performing the encoding operation. In this section we present circuits for computing polynomial multiplication and division. In Section 4.10, we put this to work for encoding operations. Some of these architectures are also used in conjunction with the convolutional codes, to be introduced in Chapter 12.

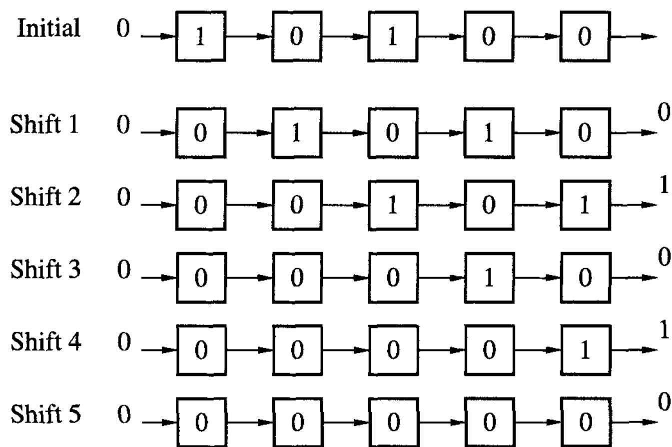
4.9.1 Computational Building Blocks

The building blocks employed here consist of three basic elements. We express the operations over an arbitrary field $\mathbb{F}$.

One-bit memory storage The symbol [D] is a storage element which holds one symbol in the field $\mathbb{F}$. (Most typically, in the field $GF(2)$, it is one bit of storage, like a D flip-flop.) The [D] holds its symbol of information (either a 0 or a 1) until a clock signal (not portrayed in the diagrams) is applied. Then the signal appearing at the input is "clocked" through to the output and also stored internally. In all configurations employed here, all of the [D] elements are clocked simultaneously. As an example, consider the following system of five [D] elements:



This cascaded configuration is called a *shift register*. In this example, the connection on the left end is permanently attached to a "0". If the storage elements are initially loaded with the contents $(1, 0, 1, 0, 0)$, then as the memory elements are clocked, the contents of the shift register change as shown here:



This is frequently represented in tabular form:

Initial:	1	0	1	0	0
Shift 1:	0	1	0	1	0
Shift 2:	0	0	1	0	1
Shift 3:	0	0	0	1	0
Shift 4:	0	0	0	0	1
Shift 5:	0	0	0	0	0

Further clockings of the system result in no further changes: the *state* (the contents of the memory elements) of the system remains in the all-zero state.

Adder The symbol $\oplus$ has two inputs and one output, which is computed as the sum of the inputs (in the field $\mathbb{F}$).

Multiplication The symbol (g_i) has one input and one output, which is computed as the product of the input and the number g_i (in the field $\mathbb{F}$). For the binary field the coefficients are either 0 or 1, represented by either no connection or a connection, respectively.

4.9.2 Sequences and Power series

In the context of these implementations, we represent a sequence of numbers $\{a_0, a_1, a_2, \ldots, a_n\}$ by a polynomial $y(x) = a_0 + a_1x + \cdots + a_nx^n = \sum_{i=0}^{n} a_ix^i$. Multiplication by x yields

$$xy(x) = a_0x + a_1x^2 + \cdots + a_nx^{n+1},$$

which is a representation of the sequence $\{0, a_0, a_1, \ldots, a_n\}$ — a right-shift or delay of the sequence. The x may thus be thought of as a "delay" operator (just as z^{-1} in the context of Z-transforms). Such representations are sometimes expressed using the variable D (for "delay") as $y(D) = a_0 + a_1D + \cdots + a_nD^n$. This polynomial representation is sometimes referred to as the D-transform. Multiplication by D $(= x)$ represents a delay operation.)

There are two different kinds of circuit representations presented below for polynomial operations. In some operations, it is natural to deal with the *last* element of a sequence first. That is, for a sequence $\{a_0, a_1, \ldots, a_k\}$, represented by $a(x) = a_0 + a_1x + \cdots + a_kx^k$, first

a_k enters the processing, then a_{k-1}, and so forth. This seems to run counter to the idea of x as a "delay," where temporally a_0 would seem to come first. But when dealing with a block of data, it is not a problem to deal with any element in the block and it is frequently more convenient to use this representation.

On the other hand, when dealing with a stream of data, it may be more convenient to deal with the elements "in order," first a_0, then a_1, and so forth.

The confusion introduced by these two different orders of processing is exacerbated by the fact that two different kinds of realizations are frequently employed, each of which presents its coefficients in opposite order from the other. For (it is hoped) clarity, representations for both last-element-first and first-element-first realizations are presented here for many of the operations of interest.

4.9.3 Polynomial Multiplication

Last-Element-First Processing

Let $a(x) = a_0 + a_1x + \cdots a_kx^k$ and let $h(x) = h_0 + h_1x + \cdots + h_rx^r$. The product

$$\begin{aligned} b(x) &= a(x)h(x) \\ &= a_0h_0 + (a_0h_1 + a_1h_0)x + \cdots + (a_kh_{r-1} + a_{k-1}h_r)x^{r+k-1} + a_kh_rx^{r+k} \end{aligned}$$

can be computed using a circuit as shown in Figure 4.1. (This circuit should be familiar to readers acquainted with signal processing, since it is simply an implementation of a finite impulse response filter.) The operation is as follows: The registers are first cleared. The last symbol a_k is input first. The first output is a_kh_r, which is the last symbol of the product $a(x)h(x)$. At the next step, a_{k-1} arrives and the output is $a_{k-1}h_r + a_kh_{r-1}$. At the next step, a_{k-2} arrives and the output is $(a_{k-2}h_r + a_{k-1}h_{r-1} + a_kh_{r-2})$, and so forth. After a_0 is clocked in, the system is clocked r times more to produce a total of $k + r + 1$ outputs.

A second circuit for multiplying polynomials is shown in Figure 4.2. This circuit has the

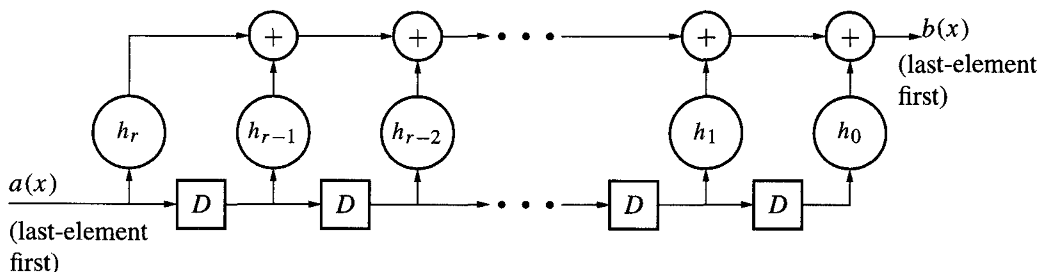


Figure 4.1: A circuit for multiplying two polynomials, last-element first.

advantage for hardware implementation that the addition is not cascaded through a series of addition operators. Hence this configuration is suitable for higher-speed operation.

First-Element-First Processing

The circuits in this section are used for filtering streams of data, such as for the convolutional codes described in Chapter 12.

Figure 4.3 shows a circuit for multiplying two polynomials, first-element first. Note that the coefficients are reversed relative to Figure 4.1. In this case, a_0 is fed in first, resulting

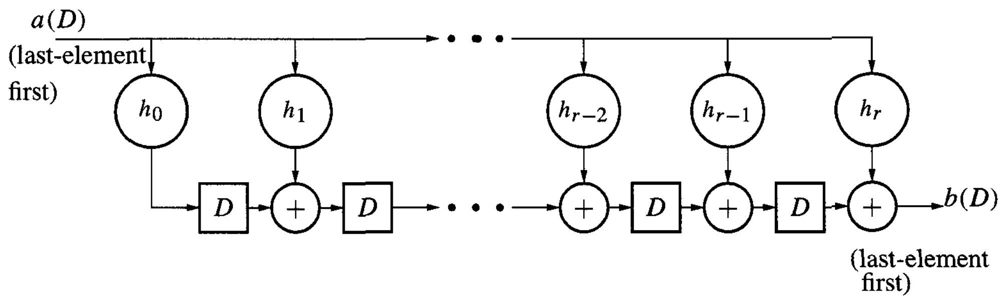


Figure 4.2: A circuit for multiplying two polynomials, last-element first, with high-speed operation.

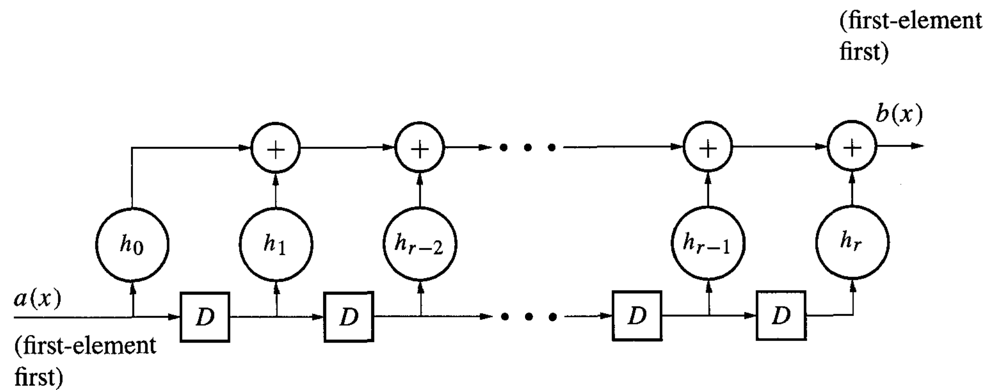


Figure 4.3: A circuit for multiplying two polynomials, first-element first.

in the output a_0h_0 at the first step. At the next step, a_1 is fed in, resulting in the output $a_0h_1 + a_1h_0$, and so forth.

Figure 4.4 shows another high speed circuit for multiplying two polynomials, first-element first.

It may be observed that these filters are FIR (finite impulse response) filters.

4.9.4 Polynomial division

Last-Element-First Processing

Computing quotients of polynomials, and more importantly, the remainder after division, plays a significant role in encoding cyclic codes. The circuits of this section will be applied to that end.

Figure 4.5 illustrates a device for computing the quotient and remainder of the polynomial division

$$\frac{d(x)}{g(x)},$$

where the dividend $d(x)$ represents a sequence of numbers

$$d(x) = d_0 + d_1x + d_2x^2 + \cdots + d_nx^n,$$

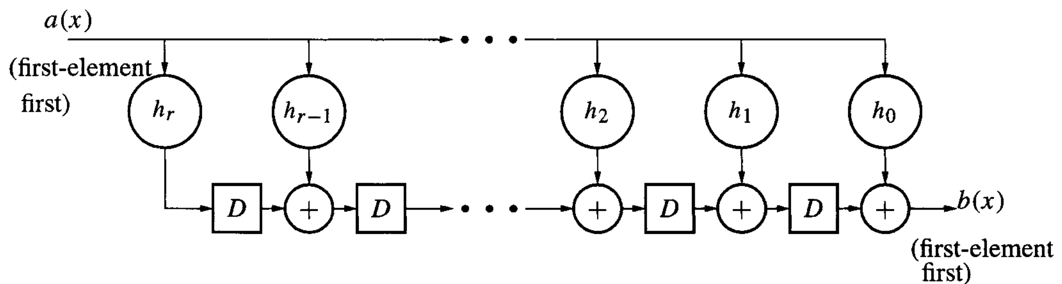


Figure 4.4: A circuit for multiplying two polynomials, first-element first, with high-speed operation.

and the divisor $g(x)$ represents a sequence of numbers

$$g(x) = g_0 + g_1 x + g_2 x^2 + \cdots + g_p x^p.$$

The coefficient g_p is nonzero; for binary polynomials the coefficient $-g_p^{-1}$ has the value of 1. The polynomial $g(x)$ is sometimes called the **connection polynomial**. The remainder $r(x)$ must be of degree $\leq p-1$, since the divisor has degree p:

$$r(x) = r_0 + r_1 x + \cdots + r_{p-1} x^{p-1},$$

and the quotient $q(x)$ can be written

$$q(x) = q_0 + q_1 x + \cdots + q_{n-p} x^{n-p}.$$

Readers familiar with signal processing will recognize the device of Figure 4.5 as an implementation of an all-pole filter.

The division device of Figure 4.5 operates as follows:

1. All the memory elements are initially cleared to 0.
2. The coefficients of $d(x)$ are clocked into the left register for p steps, starting with d_n, the coefficient of x^n in $d(x)$. This initializes the registers and has no direct counterpart in long division as computed by hand.
3. The coefficients of $d(x)$ continue to be clocked in on the left. The bits which are shifted out on the right represent the coefficients of the quotient $d(x)/g(x)$, starting from the highest-order coefficient.
4. After all the coefficients of $d(x)$ have been shifted in, the contents of the memory elements represent the remainder of the division, with the highest-order coefficient r_{p-1} on the right.

Example 4.15 Consider the division of the polynomial $d(x) = x^8 + x^7 + x^5 + x + 1$ by $g(x) =$

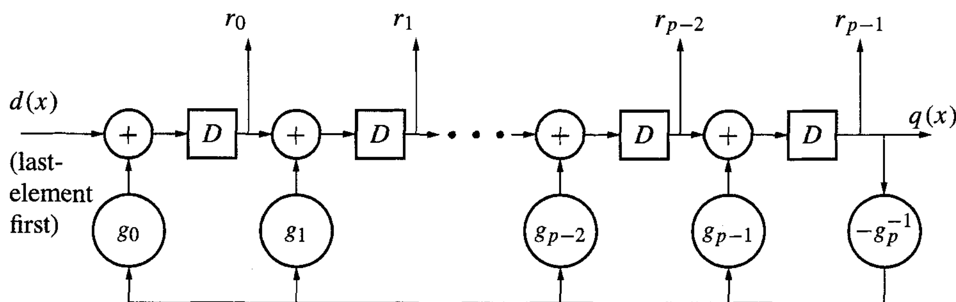


Figure 4.5: A circuit to perform polynomial division.

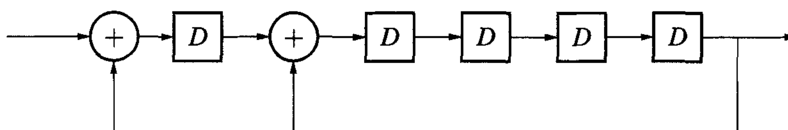


Figure 4.6: A circuit to divide by $g(x) = x^5 + x + 1$.

$x^5 + x + 1$. The polynomial long division is

$$
\begin{array}{r|l}
x+1 & x^8 + x^7 + x^5{}_{\mathsf{A}} + \qquad\qquad\qquad x+1 \\
 & x^8 + \qquad\qquad x^4 + x^3 \\ \hline
 & \quad x^7 + x^5 + x^4 + x^3{}_{\mathsf{B}} + \qquad x+1 \\
 & \quad x^7 + \qquad\qquad x^3 + x^2 \\ \hline
 & \qquad x^5 + x^4 + \qquad x^2{}_{\mathsf{C}} + x + 1 \\
 & \qquad x^5 + \qquad\qquad\qquad x+1 \\ \hline
 & \qquad\qquad x^4 + \qquad x^2{}_{\mathsf{D}}
\end{array}
\tag{4.5}
$$

The circuit for performing this division is shown in Figure 4.6. The operation of the circuit is detailed in Table 4.2. The shaded components of the table correspond to the shaded functions in the long division in (4.5). The Input column of the table shows the coefficient of the dividend polynomial $d(x)$, along with the monomial term x^i that is represented, starting with the coefficient of x^8. The Register column shows the shift register contents, along with the polynomial represented. As the algorithm progresses, the degree of the polynomial represented by the shift register decreases down to a maximum degree of $p - 1$.

Initially, the shift register is zeroed out. After 5 shifts, the shift registers hold the top coefficients of $d(x)$, indicated by A in the table, and also shown highlighted in the long division. The shift register holds the coefficient of the highest power on the right, while the long division has the highest power on the left. With the next shift, the divisor polynomial $g(x)$ is subtracted (or added) from the dividend. The shift registers then hold the results B. The operations continue until the last coefficient of $d(x)$ is clocked in. After completion, the shift registers contain the remainder, $r(x)$, shown as D. Starting from step 5, the right register output represents the coefficients of the quotient. □

Table 4.2: Computation Steps for Long Division Using a Shift Register Circuit

	Input Symbol on jth Shift		Shift Register Contents After j Shifts		Output Symbol on jth Shift	
j	bit	polynomial term	bits	polynomial representation	bit	polynomial term
0	–	–	0 0 0 0 0			
1	1	(x^8)	1 0 0 0 0			
2	1	(x^7)	1 1 0 0 0			
3	0	(x^6)	0 1 1 0 0			
4	1	(x^5)	1 0 1 1 0			
5	0	(x^4)	0 1 0 1 1	A: $x^5+x^7+x^8$	1	x^3
6	0	(x^3)	1 1 1 0 1	B: $x^3+x^4+x^5+x^7$	1	x^2
7	0	(x^2)	1 0 1 1 0	C: $x^2+x^4+x^5$	0	x^1
8	1	(x^1)	1 1 0 1 1	$x+x^2+x^4+x^5$	1	x^0
9	1	1	0 0 1 0 1	D: x^2+x^4		

4.9.5 Simultaneous Polynomial Division and Multiplication

First-Element-First Processing

Figure 4.7 shows a circuit that computes the output

$$b(x) = a(x)\frac{h(x)}{g(x)},$$

where

$$\frac{h(x)}{g(x)} = \frac{h_0 + h_1 x + \cdots + h_r x^r}{g_0 + g_1 x + \cdots + g_r x^r}$$

with $g_0 = 1$. (If $g_0 = 0$, then a non-causal filter results. If $g_0 \neq 0$ and $g_0 \neq 1$, then a constant can be factored out of the denominator.) This form is referred to as the *controller canonical form* or the *first companion form* in the controls literature [109, 181]. Figure 4.8 also computes the output

$$b(x) = a(x)\frac{h(x)}{g(x)}.$$

This form is referred to as the *alternative first companion form* or the *observability form* in the controls literature.

Example 4.16 Figure 4.9 shows the controller form for a circuit implementing the transfer function

$$H(x) = \frac{1+x}{1+x^3+x^4}.$$

For the input sequence $a(x) = 1 + x + x^2$ the output can be computed as

$$b(x) = a(x)\frac{1+x}{1+x^3+x^4} = \frac{(1+x+x^2)(1+x)}{1+x^3+x^4} = \frac{1+x^3}{1+x^3+x^4}$$
$$= 1 + x^4 + x^7 + x^8 + x^{10} + \cdots,$$

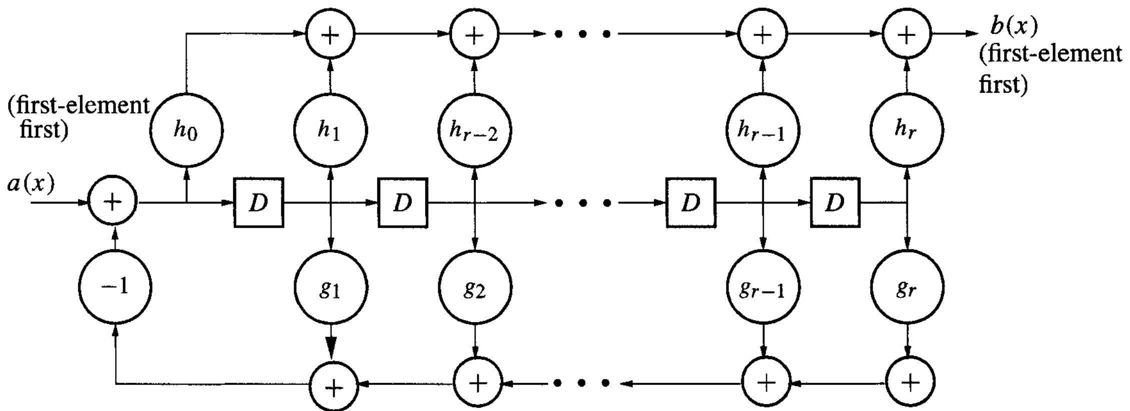


Figure 4.7: Realizing $h(x)/g(x)$ (first-element first), controller canonical form.

as can be verified using the long division $1 + x^3 + x^4 \overline{)1 + x^3}$. The operation of the circuit with this input is detailed in the following table. The column labeled 'b' shows the signal at the point 'b' in Figure 4.9.

k	a_k	b	output	next state	k	a_k	b	output	next state
				0000	5	0	0	0 (x^5)	0011
0	1	1	1	1000	6	0	0	0 (x^6)	0001
1	1	1	0 (x)	1100	7	0	1	1 (x^7)	1000
2	1	1	0 (x^2)	1110	8	0	0	1 (x^8)	0100
3	0	1	0 (x^3)	1111	9	0	0	0 (x^9)	0010
4	0	0	1 (x^4)	0111	10	0	1	1 (x^{10})	1001

Figure 4.10 shows a circuit in the observability form. The following table details its operation for the same input sequence $a(x) = 1 + x + x^2$.

k	a_k	output	next state	k	a_k	output	next state
			0000	5	0	0 (x^5)	0110
0	1	1	1101	6	0	0 (x^6)	0011
1	1	0 (x)	0111	7	0	1 (x^7)	1101
2	1	0 (x^2)	0010	8	0	1 (x^8)	1010
3	0	0 (x^3)	0001	9	0	0 (x^9)	0101
4	0	1 (x^4)	1100	10	0	1 (x^{10})	1110

□

Circuits for simultaneous multiplication and division last-element-first can be obtained by combining the circuit of Figure 4.1 with the circuit of Figure 4.5. An example is given in Section 4.12.

4.10 Cyclic Encoding

Let $g(x) = 1 + g_1x + \cdots + g_{n-k-1}x^{n-k-1} + x^{n-k}$ be the generator for a cyclic code. Nonsystematic encoding of the message polynomial $m(x) = m_0 + m_1x + \cdots + m_{k-1}x^{k-1}$

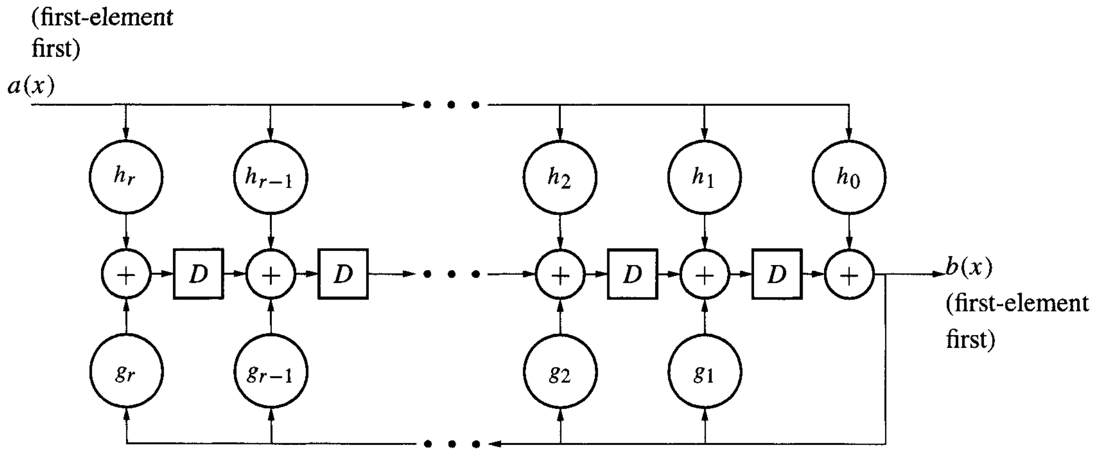


Figure 4.8: Realizing $h(x)/g(x)$ (first-element first), observability form.

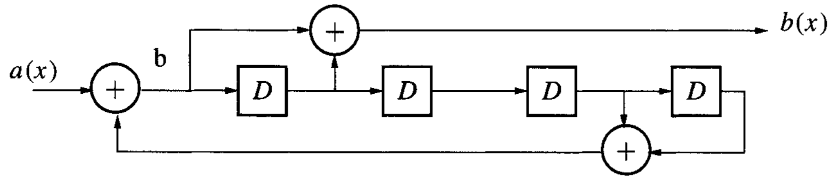


Figure 4.9: Circuit realization of $H(x) = (1 + x)/(1 + x^3 + x^4)$, controller form.

can be accomplished by shifting $m(x)$ (starting from the high-order symbol m_{k-1}) into either of the circuits shown in Figures 4.1 or 4.2, redrawn with the coefficients of $g(x)$ in Figure 4.11.

To compute a systematic encoding, the steps are:

1. Compute $x^{n-k}m(x)$
2. Divide by $g(x)$ and compute the remainder, $d(x)$.
3. Compute $x^{n-k}m(x) - d(x)$.

Figure 4.12 shows a block diagram of a circuit that accomplishes these steps. The connection structure is the same as the polynomial divider in Figure 4.5. However, instead of feeding the signal in from the left end, the signal is fed into the *right* end, corresponding to a shift of

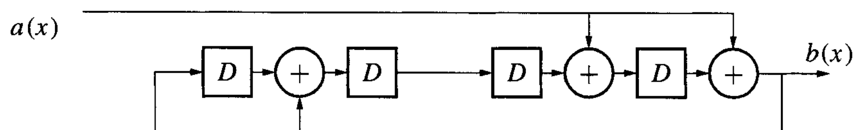


Figure 4.10: Circuit realization of $H(x) = (1 + x)/(1 + x^3 + x^4)$, observability form.

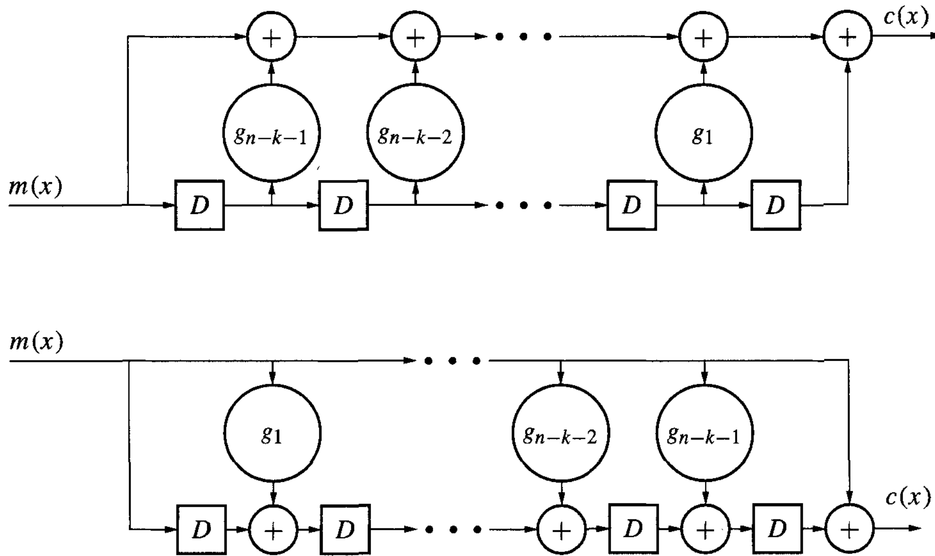


Figure 4.11: Nonsystematic encoding of cyclic codes.

x^{n-k}. This shifted signal is then divided by the feedback structure. The steps of operation are as follows:

1. With the gate "open" (allowing the signal to pass through) and the switch in position A, the message symbols $m_{k-1}, m_{k-2}, \ldots, m_0$ are fed (in that order) into the feedback system and simultaneously into the communication channel. When the message has been shifted in, the $n - k$ symbols in the register form the remainder — they are the parity symbols.
2. The gate is "closed," removing the feedback. The switch is moved to position B. (For binary field, the -1 coefficients are not needed.)
3. The system is clocked $n - k$ times more to shift the parity symbols into the channel.

Example 4.17 For the (7, 4) binary Hamming code with generator $g(x) = 1 + x + x^3$, the systematic encoder circuit is shown in Figure 4.13. For the message $\mathbf{m} = (0, 1, 1, 1)$ with polynomial $m(x) = x + x^2 + x^3$, the contents of the registers are shown here.

Input	Register contents			
	0	0	0	(initial state)
1	1	1	0	
1	1	0	1	
1	0	1	0	
0	0	0	1	(parity bits, $d(x) = x^2$)

The sequence of output bits is

$$\mathbf{c} = (0, 0, 1, \ 0, 1, 1, 1).$$

□

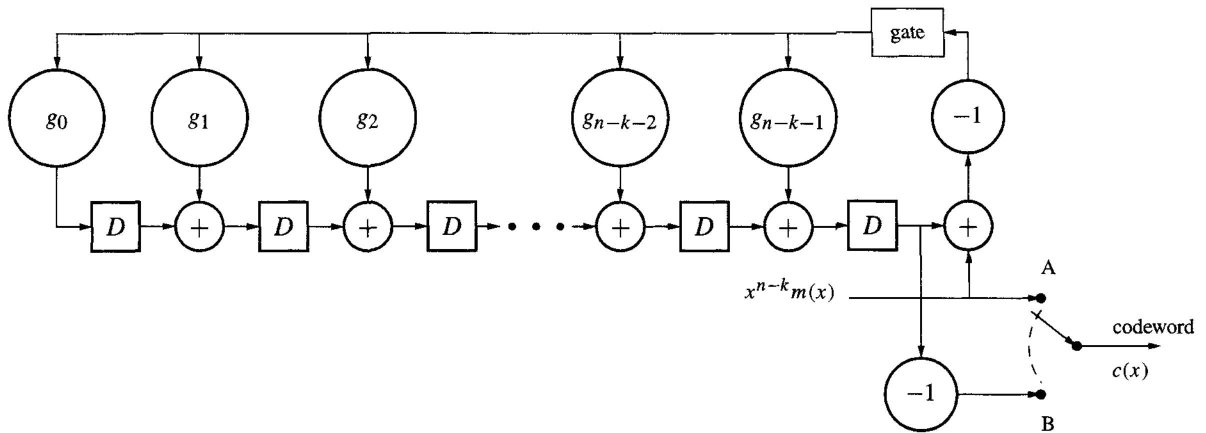


Figure 4.12: Circuit for systematic encoding using $g(x)$.

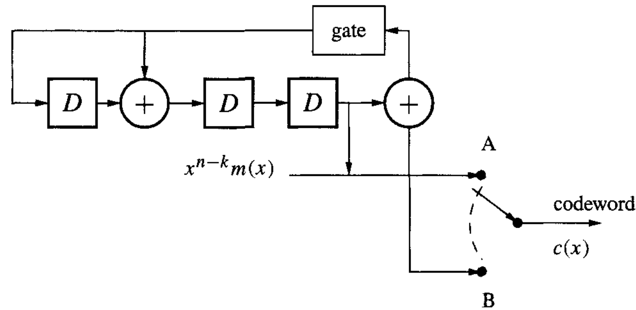


Figure 4.13: Systematic encoder for the (7, 4) code with generator $g(x) = 1 + x + x^3$.

Systematic encoding can also be accomplished using the parity check polynomial $h(x) = h_0 + h_1x + \cdots + h_kx^k$. Since $h_k = 1$, we can write the condition (4.3) as

$$c_{l-k} = -\sum_{i=0}^{k-1} h_i c_{l-i} \qquad l = k, k+1, \ldots, n-1. \tag{4.6}$$

Given the systematic part of the message $c_{n-k} = m_0, c_{n-k+1} = m_1, \ldots, c_{n-1} = m_{k-1}$, the parity check bits $c_0, c_1, \ldots, c_{n-k-1}$ can be found from (4.6). A circuit for doing the computations is shown in Figure 4.14. The operation is as follows.

1. With gate 1 open (passing message symbols) and gate 2 closed and with the syndrome register cleared to 0, the message $m(x) = m_0 + m_1x + \cdots + m_{k-1}x^{k-1}$ is shifted into simultaneously the registers and into the channel, starting with the symbol m_{k-1}. At the end of k shifts, the registers contain the symbols $m_0, m_1, \ldots, m_{k-1}$, reading from left to right.
2. Then gate 1 is closed and gate 2 is opened. The first parity check digit

$$\begin{aligned} c_{n-k-1} &= -(h_0c_{n-1} + h_1c_{n-2} + \cdots + h_{k-1}c_{n-k}) \\ &= -(m_{k-1} + h_1m_{k-2} + \cdots + h_{k-1}m_0) \end{aligned}$$

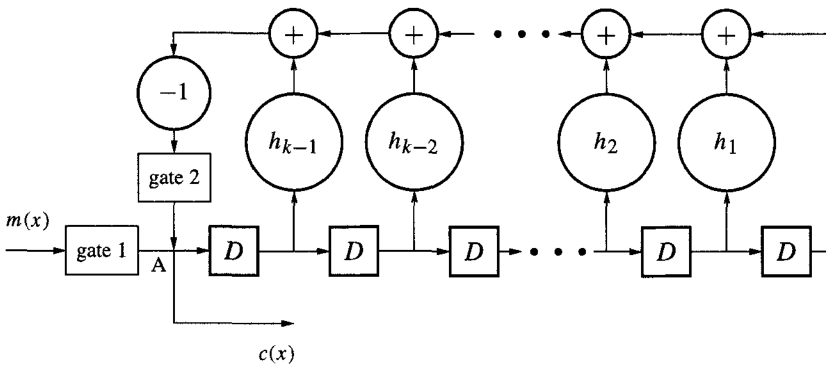


Figure 4.14: A systematic encoder using the parity check polynomial.

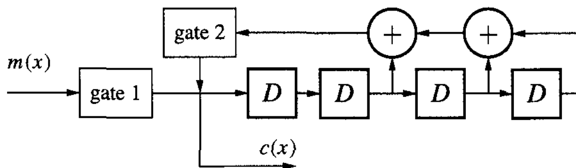


Figure 4.15: A systematic encoder for the Hamming code using $h(x)$.

is produced and appears at the point labeled A. c_{n-k-1} is simultaneously clocked into the channel and into the buffer register (through gate 2).

3. The computation continues until all $n-k$ parity check symbols have been produced.

Example 4.18 For the $(7, 4)$ code generator $g(x) = x^3 + x + 1$, the parity check polynomial is

$$h(x) = \frac{x^7 - 1}{x^3 + x + 1} = x^4 + x^2 + x + 1.$$

Figure 4.15 shows the systematic encoder circuit. (The -1 coefficient is removed because of the binary field.) Suppose $m(x) = x + x^2 + x^3$. The bits (0,1,1,1) are shifted in (with the 1 bit shifted first). Then the contents of the registers are shown here.

Registers				Output
0	1	1	1	(initial)
$\underline{1}$	0	1	1	1
$\underline{0}$	1	0	1	0
$\underline{0}$	0	1	0	0

The sequence of output bits is

$$\mathbf{c} = (0, 0, 1,\ 0, 1, 1, 1),$$

which is the same as produced by the encoding in Example 4.17. □

4.11 Syndrome Decoding

We now examine the question of decoding binary cyclic codes. Recall that for any linear code, we can form a standard array, or we can use the reduced standard array using syn-

dromes. For cyclic codes it is possible to exploit the cyclic structure of the codes to further decrease the memory requirements.

Recall that the syndrome was initially defined as $s(x) = r(x)h(x) \pmod{x^n - 1}$. However, we can define the syndrome an alternative way. Since a codeword must be a multiple of $g(x)$, when we divide $r(x)$ by $g(x)$, the remainder is zero exactly when $r(x)$ is a codeword. Thus we can employ the division algorithm to obtain a syndrome. We write

$$r(x) = q(x)g(x) + s(x),$$

where $q(x)$ is the quotient (which is usually not used for decoding) and $s(x)$ is the remainder polynomial having degree less than the degree of $g(x)$:

$$s(x) = s_0 + s_1 x + \cdots + s_{n-k-1}x^{n-k-1}.$$

Thus, to compute the syndrome we can use polynomial division. A circuit such as that in Figure 4.5 can be used to compute the remainder.

We have the following useful result about cyclic codes and syndromes.

Theorem 4.2 *Let $s(x)$ be the syndrome corresponding to $r(x)$, so $r(x) = q(x)g(x)+s(x)$. Let $r^{(1)}(x)$ be the polynomial obtained by cyclically right-shifting $r(x)$ and let $s^{(1)}(x)$ denote its syndrome. Then $s^{(1)}(x)$ is the remainder obtained when dividing $xs(x)$ by $g(x)$. In other words, syndromes of shifts of $r(x)$ $(\bmod\, x^n - 1)$ are shifts of $s(x)$ $(\bmod\, g(x))$.*

Proof With $r(x) = r_0 + r_1 x + \cdots r_{n-1}x^{n-1}$ the cyclic shift $r^{(1)}(x)$ is

$$r^{(1)}(x) = r_{n-1} + r_0 x + \cdots + r_{n-2}x^{n-1},$$

which can be written as

$$r^{(1)}(x) = xr(x) - r_{n-1}(x^n - 1).$$

Using the division algorithm and the fact that $x^n - 1 = g(x)h(x)$,

$$q^{(1)}(x)g(x) + s^{(1)}(x) = x[q(x)g(x) + s(x)] - r_{n-1}g(x)h(x),$$

where $s^{(1)}(x)$ is the remainder from dividing $r^{(1)}(x)$ by $g(x)$. Rearranging, we have

$$xs(x) = [q^{(1)}(x) + r_{n-1}h(x) - xq(x)]g(x) + s^{(1)}(x).$$

Thus $s^{(1)}(x)$ is the remainder from dividing $xs(x)$ by $g(x)$, as claimed. □

By induction, the syndrome $s^{(i)}(x)$ that corresponds to cyclically shifting $r(x)$ i times to produce $r^{(i)}(x)$ is obtained from the remainder of $x^i s(x)$ when divided by $g(x)$. This can be accomplished in hardware simply by clocking the circuitry that computes the remainder $s(x)$ i times: the shift register motion corresponds to multiplication by x, while the feedback corresponds to computing the remainder upon division by $g(x)$.

Example 4.19 For the (7,4) code with generator $g(x) = x^3 + x + 1$, let $r(x) = x + x^2 + x^4 + x^5 + x^6$ be the received vector. That is, $\mathbf{r} = (0, 1, 1, 0, 1, 1, 1)$. Then the cyclic shifts of $r(x)$ and their corresponding syndromes are shown here.

Polynomial	Syndrome
$r(x) = x + x^2 + x^4 + x^5 + x^6$	$s(x) = x$
$r^{(1)}(x) = 1 + x^2 + x^3 + x^5 + x^6$	$s^{(1)}(x) = x^2$
$r^{(2)}(x) = 1 + x + x^3 + x^4 + x^6$	$s^{(2)}(x) = 1 + x$
$r^{(3)}(x) = 1 + x + x^2 + x^4 + x^5$	$s^{(3)}(x) = x + x^2$
$r^{(4)}(x) = x + x^2 + x^3 + x^5 + x^6$	$s^{(4)}(x) = 1 + x + x^2$
$r^{(5)}(x) = 1 + x^2 + x^3 + x^4 + x^6$	$s^{(5)}(x) = 1 + x^2$
$r^{(6)}(x) = 1 + x + x^3 + x^4 + x^5$	$s^{(6)}(x) = 1$

Figure 4.16 shows the circuit which divides by $g(x)$, producing the remainder $s(x) = s_0 + s_1 x + s_2 x^2$ in its registers. Suppose the gate is initially open and $r(x)$ is clocked in, producing the syndrome $s(x)$. Now the gate is closed and the system is clocked 6 more times. The registers contain successively the syndromes $s^{(i)}(x)$ corresponding to the cyclically shifted polynomials $r^{(i)}(x)$, as shown in Table 4.3. □

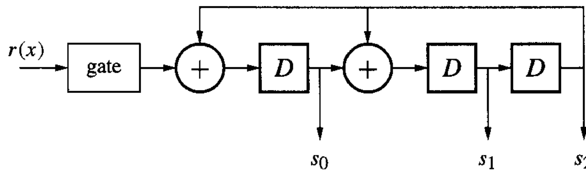


Figure 4.16: A syndrome computation circuit for a cyclic code example.

Table 4.3: Computing the Syndrome and Its Cyclic Shifts

Clock	Input	Registers			Syndrome
Initial:		0	0	0	
1	1	1	0	0	
2	1	1	1	0	
3	1	1	1	1	
4	0	1	0	1	
5	1	0	0	0	
6	1	1	0	0	
7	0	0	1	0	$s(x) = x$
.... (turn off gate)					
8		0	0	1	$s^{(1)}(x) = x^2$
9		1	1	0	$s^{(2)}(x) = 1 + x$
10		0	1	1	$s^{(3)}(x) = x + x^2$
11		1	1	1	$s^{(4)}(x) = 1 + x + x^2$
12		1	0	1	$s^{(5)}(x) = 1 + x^2$
13		0	0	0	$s^{(6)}(x) = 0$ (syndrome adjustment)

We only need to compute one syndrome **s** for an error **e** and all cyclic shifts of **e**, so the size of the syndrome table can be reduced by n. Furthermore, we can compute the shifts necessary using the same circuit that computes the syndrome in the first place.

This observation also indicates a means of producing error correcting hardware. Consider the decoder shown in Figure 4.17. This decoder structure is called a **Meggitt decoder**.

The operation of the circuit is as follows. The error pattern detection circuit is a combinatorial logic circuit that examine the syndrome bits and outputs a 1 if the syndrome corresponds to an error in the *highest* bit position, $e_{n-1} = 1$.

- With gate 1 open and gate 2 closed and with the syndrome register cleared to 0, the received vector is shifted into the buffer register and the syndrome register for n clocks. At the end of this, the syndrome register contains the syndrome for $r(x)$.

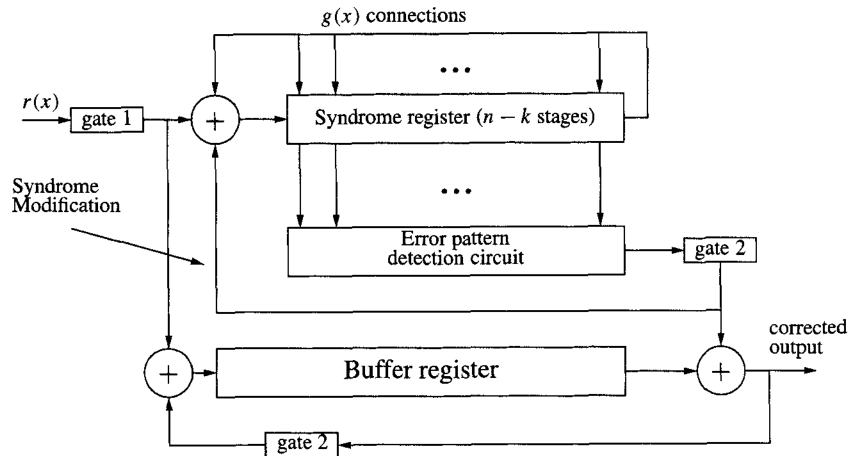


Figure 4.17: Cyclic decoder when $r(x)$ is shifted in the left end of the syndrome register.

- Now gate 1 is closed and gate 2 is opened. The error pattern detection circuit outputs $e_{n-1} = 1$ if it has determined that the (current) highest bit position is in error, so that $e(x) = x^{n-1}$. The modified polynomial, denoted by $r_1(x)$, is $r_1(x) = r_0 + r_1 x + \cdots r_{n-2}x^{n-2} + (r_{n-1} + e_{n-1})x^{n-1}$. Now cyclically shift $r_1(x)$ to produce $r_1^{(1)}(x) = (r_{n-1} + e_{n-1}) + r_0 x + \cdots + r_{n-2}x^{n-1}$. The corresponding syndrome $s_1^{(1)}(x)$ is the remainder of $r_1^{(1)}(x)$ divided by $g(x)$. Since the remainder of $xr(x)$ is $s^{(1)}(x)$ and the remainder of xx^{n-1} is 1, the new syndrome is

$$s_1^{(1)}(x) = s^{(1)}(x) + 1.$$

  Therefore, the syndrome register can be adjusted so that it reflects the modification made to $r(x)$ by adding a 1 to the left end of the register. (If only single error correction is possible, then this update is unnecessary.)

  The modified value is output and is also fed back around through gate 2.

- Decoding now proceeds similarly on the other bits of $r(x)$. As each error is detected, the corresponding bit is complemented and the syndrome register is updated to reflect the modification. Operation continues until all the bits of the buffer register have been output.

At the end of the decoding process, the buffer register contains the corrected bits.

The key to decoding is designing the error pattern detection circuit.

Example 4.20 Consider again the decoder for the code with generator $g(x) = x^3 + x + 1$. The following table shows the error vectors and their corresponding syndrome vectors and polynomials.

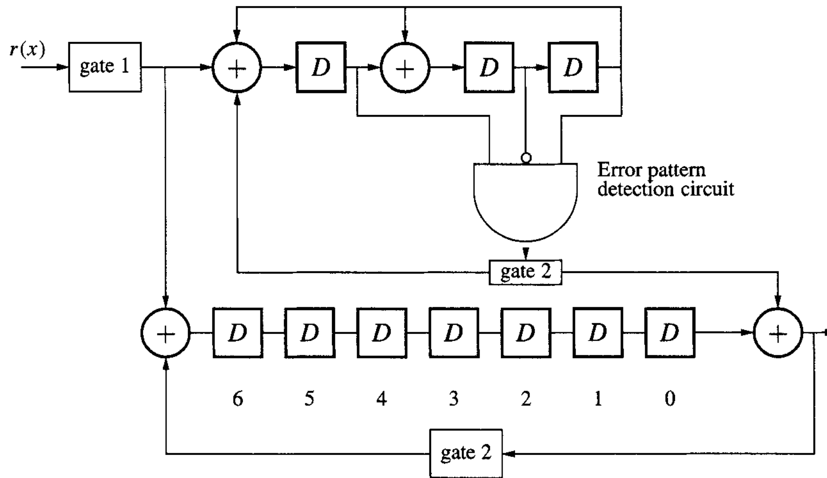


Figure 4.18: Decoder for a (7,4) Hamming code, input on the left.

error	error polynomial	syndrome	syndrome polynomial
0000000	$e(x) = 0$	000	$s(x) = 0$
1000000	$e(x) = 1$	100	$s(x) = 1$
0100000	$e(x) = x$	010	$s(x) = x$
0010000	$e(x) = x^2$	001	$s(x) = x^2$
0001000	$e(x) = x^3$	110	$s(x) = 1 + x$
0000100	$e(x) = x^4$	011	$s(x) = x + x^2$
0000010	$e(x) = x^5$	111	$s(x) = 1 + x + x^2$
0000001	$e(x) = x^6$	101	$s(x) = 1 + x^2$

(From this table, we recognize that the received polynomial $r(x)$ in Example 4.19 has an error in the second bit, since $s(x) = x$ is the computed syndrome). However, what is of immediate interest is the error in the *last* position, $\mathbf{e} = (0000001)$ or $e(x) = x^6$, with its syndrome $s(x) = 1 + x^2$. In the decoder of Figure 4.18, the pattern is detected with a single 3-input and gate with the middle input inverted. When this pattern is detected, the outgoing right bit of the register is complemented and the input bit of the syndrome register is complemented. The decoding circuit is thus as shown in Figure 4.18.

Suppose now that $r(x) = x + x^2 + x^4 + x^5 + x^6$, as in Example 4.19. As this is shifted in, the syndrome $s(x) = x$ is computed. Now the register contents are clocked out, producing in succession the syndromes shown in Table 4.3. At clock tick 12 (which is 5 ticks after the initial the pattern was shifted in), $s^{(5)}(x) = 1 + x^2$ appears in the syndrome register, signaling an error in the right bit of the register. The bit of the buffer register is complemented on its way to output, which corresponds to the second bit of the received codeword. The next syndrome becomes 0, corresponding to a vector with no errors. The corrected codeword is thus

$$c(x) = x^2 + x^4 + x^5 + x^6,$$

corresponding to a message polynomial $m(x) = x + x^2 + x^3$.

The overall operation of the Meggitt decoder of Figure 4.18 is shown in Table 4.4. The input is shifted into the syndrome register and the buffer register. (The erroneous bit is indicated underlined.) After being shifted in, the syndrome register is clocked (with no further input) while the buffer register is cyclically shifted. At step 12, the syndrome pattern is detected as corresponding to an error in the right position. This is corrected. The syndrome is simultaneously adjusted, so that no further changes

are made in the last two steps.

Table 4.4: Operation of the Meggitt decoder, Input from the Left

step	input	syndrome register	buffer register
1	1	100	1000000
2	1	110	1100000
3	1	111	1110000
4	0	101	0111000
5	1	000	1011100
6	1	100	1101110
7	0	010	0110111
8		001	1011011
9		110	1101101
10		011	1110110
11		111	0111011
12		101	1011101 (error corrected)
13		000	0101110
14		000	0010111

□

In some cases, the Meggitt decoder is implemented with the received polynomial shifted in to the *right* of the syndrome register, as shown in Figure 4.19. Since shifting $r(x)$ into the right end of the syndrome register is equivalent to multiplying by x^{n-k}, the syndrome after $r(x)$ has been shifted in is $s^{(n-k)}(x)$, the syndrome corresponding to $r^{(n-k)}(x)$. Now decoding operates as before: if $s^{(n-k)}(x)$ corresponds to an error pattern with $e(x)$ with $e_{n-1} = 1$, then bit r_{n-1} is corrected. The effect of the error must also be removed from the syndrome. The updated syndrome, denoted $s_1^{(n-k)}(x)$ is the sum of $s^{(n-k)}(x)$ and the remainder resulting from dividing x^{n-k-1} by $g(x)$. Since x^{n-k-1} has degree less than the degree of $g(x)$, this remainder is, in fact, equal to x^{n-k-1}. The updated syndrome is thus

$$s_1^{(n-k)}(x) = s^{(n-k)}(x) + x^{(n-k-1)}.$$

This corresponds to simply updating the right coefficient of the syndrome register.

Example 4.21 When the error pattern $e(x) = x^6$ is fed into the right-hand side of the syndrome register of a (7, 4) Hamming code, it appears as $x^3x^6 = x^9$. The remainder upon dividing x^9 by $g(x)$ is $s^{(3)}(x) = R_{g(x)}[x^9] = x^2$. Thus, the syndrome to look for in the error pattern detection circuit is x^2. Figure 4.20 shows the corresponding decoder circuit. □

If this decoder is used with the received polynomial $r(x) = x + x^2 + x^4 + x^5 + x^6$ (as before), then the syndrome register and buffer register contents are as shown in Table 4.5. Initially the received polynomial is shifted in. As before, the erroneous bit is shown underlined. After step $n = 7$, the syndrome register is clocked with no further input. At step 12, the syndrome pattern detects the error in the right position. This is corrected in the buffer register adjusted in the syndrome register.

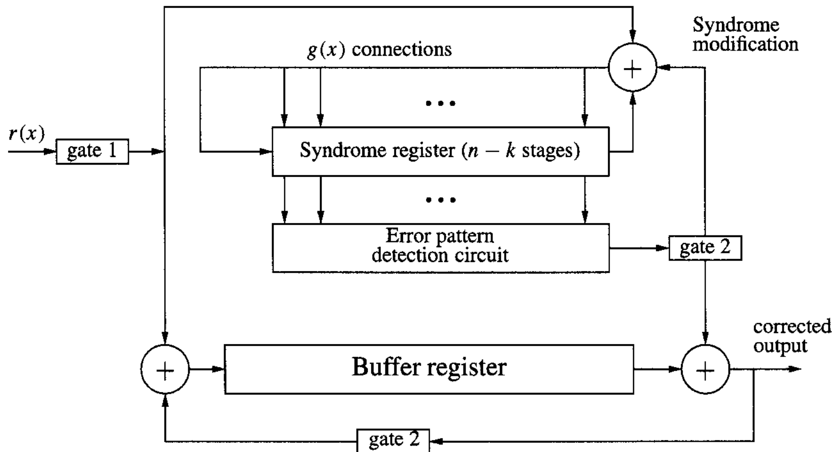


Figure 4.19: Cyclic decoder when $r(x)$ is shifted into the right end of the syndrome register.

Example 4.22 We present decoders for the (31,26) Hamming code generated by $g(x) = 1+x^2+x^5$.

Figure 4.21(a) shows the decoder when the received polynomial is shifted in on the left. The error pattern $e(x) = x^{30}$ results in the syndrome $s(x) = R_{g(x)}[x^{30}] = x^4 + x$;

Figure 4.21(b) shows the decoder when the received polynomial is shifted in on the right. The error pattern $e(x) = x^{30}$ results in the shifted syndrome

$$s(x) = R_{g(x)}[x^{30}x^5] = R_{g(x)}[x^{35}] = x^4.$$

□

4.12 Shortened Cyclic Codes

Shortened block codes were introduced in Section 3.9. In this section we deal in particular about shortened cyclic codes [204]. Let $\mathcal{C}$ be an (n, k) cyclic code and let $\mathcal{C}' \subset \mathcal{C}$ be the set of codewords for which the l high-order message symbols are equal to 0. That is, the symbols $m_{k-l}, m_{k-l+1}, \ldots, m_{k-2}, m_{k-1}$ are all set to 0, so all messages are of the form

$$m(x) = m_0 + m_1 x + \cdots + m_{k-l-1}x^{k-l-1}.$$

There are 2^{k-l} codewords in $\mathcal{C}'$, forming a linear $(n-l, k-l)$ subcode of $\mathcal{C}$. The minimum distance of $\mathcal{C}'$ is at least as large as that of $\mathcal{C}$. $\mathcal{C}'$ is called a shortened cyclic code.

The shortened cyclic code $\mathcal{C}'$ is not, in general, cyclic. However, since $\mathcal{C}$ is cyclic, the encoding and decoding of $\mathcal{C}'$ can be accomplished using the same cyclic-oriented hardware as for $\mathcal{C}$, since the deleted message symbols do not affect the parity-check or syndrome computations. However, care must be taken that the proper number of cyclic shifts is used.

Let $r(x) = r_0 + r_1 x + \cdots + r_{n-l-1}x^{n-l-1}$ be the received polynomial. Consider a decoder in which $r(x)$ is clocked into the right end of the syndrome register, as in Figure 4.19. Feeding $r(x)$ into the right end of the corresponds to multiplying $r(x)$ by x^{n-k}. However, since the code is of length $n-l$, what is desired is multiplication by $x^{n-(k-l)} = x^{n-k+l}$. Thus, the syndrome register must be cyclically clocked another l times after $r(x)$ has been shifted into the register. While this is feasible, it introduces an additional decoder latency of l clock steps. We now show two different methods to eliminate this latency.

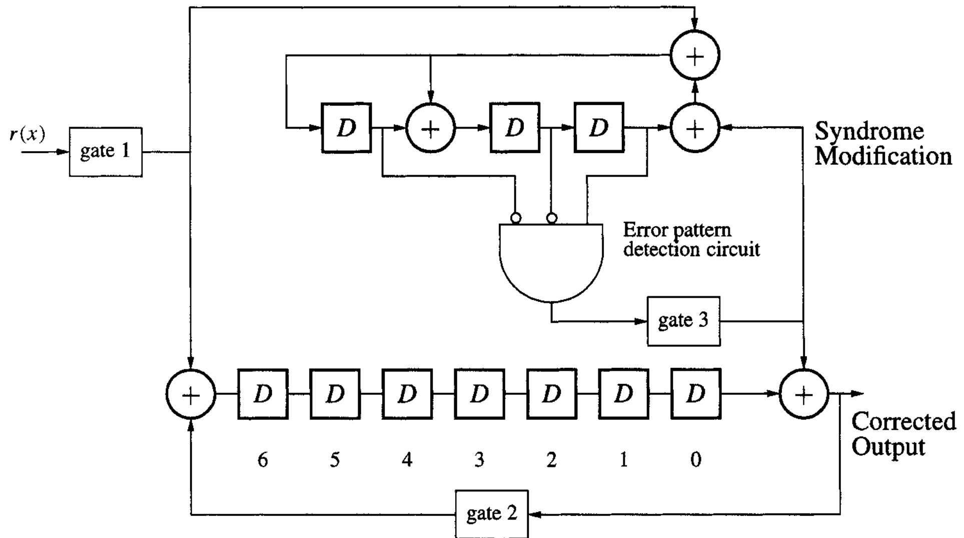


Figure 4.20: Hamming decoder with input fed into the right end of the syndrome register.

Method 1: Simulating the Extra Clock Shifts

In this method, $r(x)$ is fed into the syndrome computation register in such a way in $n-k$ shifts the effect of $n-k+l$ shifts is obtained.

Using the division algorithm to divide $x^{n-k+l}r(x)$ by $g(x)$ we obtain

$$x^{n-k+l}r(x) = q_1(x)g(x) + s^{(n-k+l)}(x), \tag{4.7}$$

where $s^{(n-k+l)}(x)$ is the remainder and is the desired syndrome for decoding the digit r_{n-l-1}. Now divide x^{n-k+l} by $g(x)$,

$$x^{n-k+l} = q_2(x)g(x) + \rho(x),$$

where $\rho(x) = \rho_0 + \rho_1 x + \cdots + \rho_{n-k-1}x^{n-k-1}$ is the remainder. This can also be expressed as

$$\rho(x) = x^{n-k+l} + q_2(x)g(x) \tag{4.8}$$

(for binary operations). Multiply (4.8) by $r(x)$ and use the (4.7) to write

$$\rho(x)r(x) = [q_1(x) + q_2(x)r(x)]g(x) + s^{(n-k+l)}(x).$$

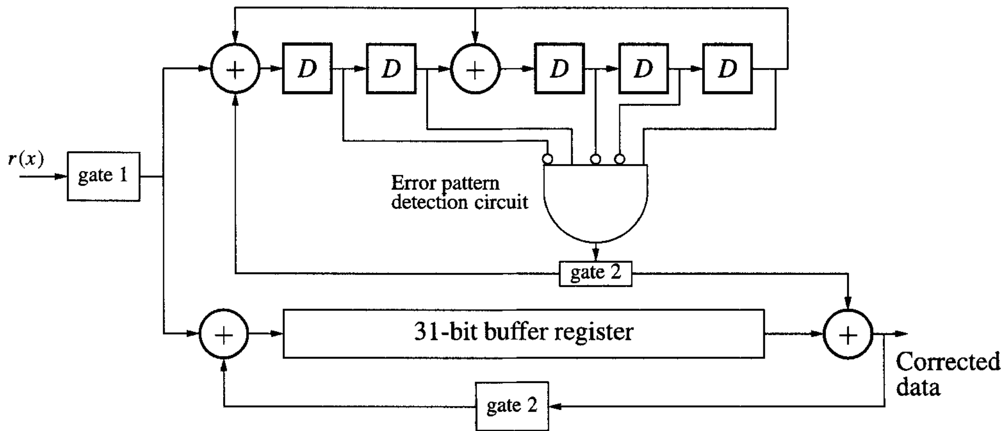
From this equation, it is seen that the desired syndrome $s^{(n-k+l)}(x)$ can be obtained by multiplying $r(x)$ by $\rho(x)$ then computing the remainder modulo $g(x)$. Combining the first-element-first circuits of Figures 4.1 and 4.5 we obtain the circuit shown in Figure 4.22.

The error pattern detection circuit for this implementation is the same as for the unshortened code.

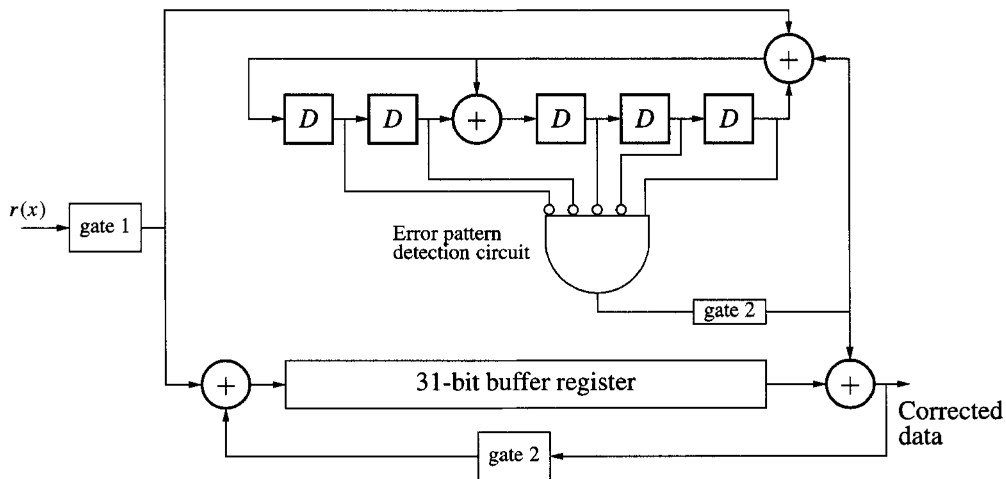
Example 4.23 Consider the Hamming (7, 4) code generated by $g() = 1 + x + x^3$ whose decoder is shown in Figure 4.20. Shortening this code by $l = 2$, a (5,2) code is obtained. To find $\rho(x)$ we have

$$\rho(x) = R_{g(x)}[x^{n-k+l}] = R_{g(x)}[x^5] = x^2 + x + 1.$$

Figure 4.23 shows the decoder for this code. □



(a) Input from the left end.



(b) Input from the right end.

Figure 4.21: Meggitt decoders for the (31,26) Hamming code.

Table 4.5: Operation of the Meggitt Decoder, Input from the Right

step	input	syndrome register	buffer register
1	1	110	1000000
2	1	101	1100000
3	1	010	1110000
4	0	001	0111000
5	1	000	1011100
6	1	110	1101110
7	0	011	0110111
8		111	1011011
9		101	1101101
10		100	1110110
11		010	0111011
12		001	1011101 (error corrected)
13		000	0101110
14		000	0010111

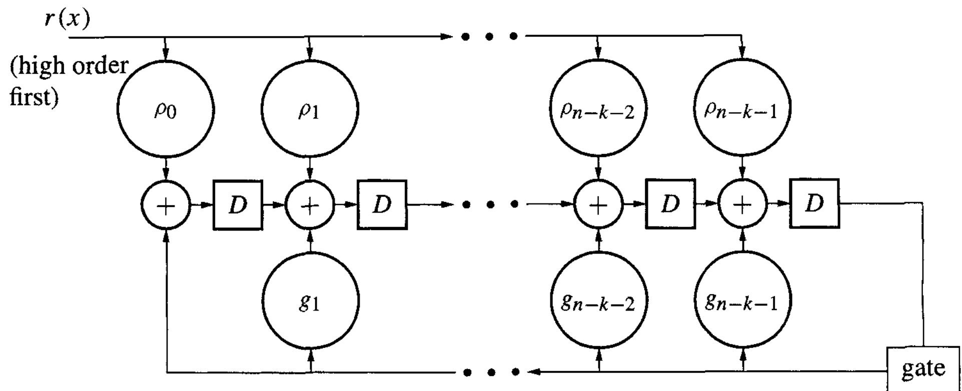


Figure 4.22: Multiply $r(x)$ by $\rho(x)$ and compute the remainder modulo $g(x)$.

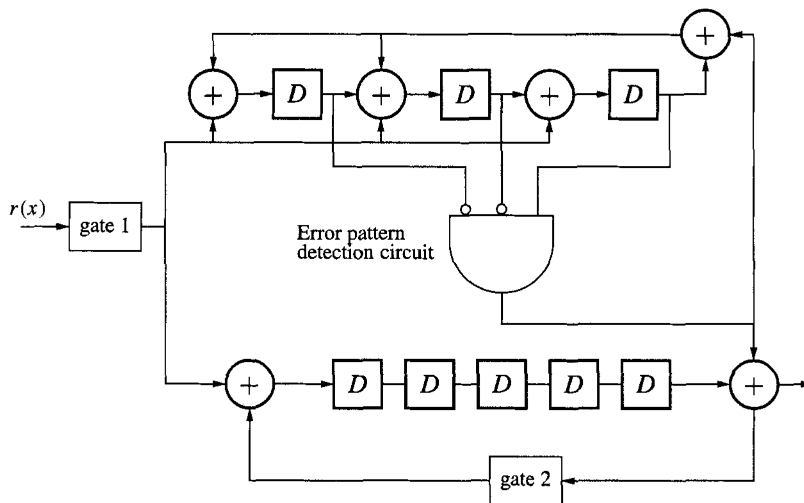


Figure 4.23: Decoder for a shortened Hamming code.

Method 2: Changing the Error Pattern Detection Circuit

Another way to modify the decoder is to change the error pattern detection circuit so that it looks for patterns corresponding to the shifted input, but still retains the usual syndrome computation circuit. The error pattern detection circuit is designed to produce a 1 when the syndrome register corresponds to a correctable error pattern $e(x)$ with an error at the right-position, that is, at position x^{n-l-1}. When this happens, the received digit r_{n-l-1} is corrected and the effect of the error digit e_{n-l-1} is removed from the syndrome register via syndrome modification.

Let $e(x) = x^{n-l-1}$. Since this is input on the right end, this is equivalent to $x^{n-l-1}x^{n-k} = x^{2n-l-k-1}$. The syndrome pattern to watch for is obtained by $\sigma(x) = R_{g(x)}[x^{2n-l-k-1}]$.

Example 4.24 Consider again the (7,4) Hamming code shortened to a (5,2) code. The error pattern at position $x^{n-l-1} = x^4$ appearing on the right-hand side as $x^{2n-l-k-1} = x^7$. The syndrome to watch for is

$$\sigma(x) = R_{g(x)}[x^7] = 1.$$

□

4.13 Binary CRC Codes

The term Cyclic Redundancy Check (CRC) code has come to be jargon applied to cyclic codes used as error *detection* codes: they indicate when error patterns have occurred over a sequence of bits, but not where the errors are nor how to correct them. They are commonly used in networking in conjunction with protocols which call for retransmission of erroneous data packets. Typically CRCs are *binary* codes, with operations taking place in $GF(2)$. A CRC is a cyclic code, that is, the code polynomials are multiples of a generator polynomial $g(x) \in GF(2)[x]$.

CRCs are simply cyclic codes, so the same encoding and decoding concepts as any other cyclic code applies. In this section, however, we will introduce an efficient byte-oriented

algorithm for computing syndromes.

We use the notation $R_{g(x)}[\cdot]$ to denote the operation of computing the remainder of the argument when dividing by $g(x)$. The entire cyclic encoding operation can thus be written, as described in Section 4.8, as

$$c(x) = x^r m(x) + R_{g(x)}[x^r m(x)].$$

Example 4.25 Let $g(x) = x^{16}+x^{15}+x^2+1$ and $m(x) = x^{14}+x^{13}+x^{11}+x^{10}+x^8+x^5+x^2+x+1$ corresponding to the message bits

$$\begin{aligned}\mathbf{m} &= [0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1]\\ &= [m_{15}, m_{14}, \cdots, m_1, m_0].\end{aligned}$$

The vector $\mathbf{m}$ is written here with m_0 on the right. Since $\deg(g(x)) = n - k = 16$, to encode we first multiply $m(x)$ by x^{16}:

$$x^{16}m(x) = x^{30} + x^{29} + x^{27} + x^{26} + x^{24} + x^{21} + x^{18} + x^{17} + x^{16}, \tag{4.9}$$

then divide by $g(x)$ to obtain the remainder

$$d(x) = x^{14} + x^{13} + x^{11} + x^{10} + x^9 + x^7 + x^6 + x^4 + x^2. \tag{4.10}$$

The code polynomial is

$$\begin{aligned}c(x) &= x^{16}m(x) + d(x)\\ &= x^{30} + x^{29} + x^{27} + x^{26} + x^{24} + x^{21} + x^{18} + x^{17} + x^{16}\\ &\quad + x^{14} + x^{13} + x^{11} + x^{10} + x^9 + x^7 + x^6 + x^4 + x^2\end{aligned}$$

The operation can also be represented using bit vectors instead of polynomials. From (4.9),

$$x^{16}m(x) \leftrightarrow [0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1|0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]$$

(with the highest power of x corresponding to the bit on the left of this vector) and from (4.10),

$$d(x) \leftrightarrow [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0|0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0]$$

Adding these two vectors we find

$$\mathbf{c} = [0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1|0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0].$$

The message vector $\mathbf{m}$ is clearly visible in the codeword $\mathbf{c}$. □

Suppose now that the effect of the channel is represented by

$$r(x) = c(x) + e(x).$$

To see if any errors occurred in transmission over the channel, $r(x)$ is divided by $g(x)$ to find $s(x) = R_{g(x)}[r(x)]$. The polynomial $s(x)$ is the syndrome polynomial. Note that

$$s(x) = R_{g(x)}[r(x)] = R_{g(x)}[c(x) + e(x)] = R_{g(x)}[c(x)] + R_{g(x)}[e(x)] = R_{g(x)}[e(x)],$$

since $R_{g(x)}[c(x)] = 0$ for any code polynomial $c(x)$.

If $s(x) \neq 0$, then $e(x) \neq 0$, that is, one or more errors have occurred and they have been detected. If $s(x) = 0$, then it is concluded $r(x)$ has not been corrupted by errors, so that the original message $m(x)$ may be immediately extracted from $r(x)$.

Note, however, that if an error pattern occurs which is exactly one of the code polynomials, say $e(x) = c_1(x)$ for some code polynomial $c_1(x)$, then

$$s(x) = R_{g(x)}[c(x) + c_1(x)] = R_{g(x)}[c(x)] + R_{g(x)}[c_1(x)] = 0.$$

In other words, there are error patterns that can occur which are not detected by the code: an error pattern is undetectable if and only if $e(x)$ is a code polynomial.

Let us consider how many such undetected error patterns there are.

- Suppose there is a single bit in error, $e(x) = x^i$ for $0 \le i \le n-1$. If the polynomial $g(x)$ has more than one nonzero term it cannot divide x^i evenly, so there is a nonzero remainder. Thus all single-bit errors can be detected.
- Suppose that $g(x)$ has $(1 + x)$ as a factor. Then it can be shown that all codewords have even parity, so that *any* odd number of bit errors can be detected.
- A *burst error* of length B is any error pattern for which the number of bits between the first and last errors (inclusive) is B. For example, the bit sequence ...,0,0,1,1,0,1, 1,0,1,0,...has a burst error of length 7.

  Let $e(x)$ be an error burst of length $r = n - k$ or less. Then

  $$e(x) = x^i(1 + e_1 x + \cdots + e_{n-k-1}x^{n-k-1})$$

  for some i, $0 \le i \le k$. Since $g(x)$ is of degree $n - k$ and has a non-zero constant term, that is

  $$g(x) = 1 + g_1 x + \cdots + g_{n-k-1}x^{n-k-1} + x^{n-k},$$

  then $R_{g(x)}[e(x)]$ cannot be zero, so the burst can be detected.
- Consider now a burst of errors of length $n - k + 1$, with error polynomial $e(x) = x^i(1 + e_1 x + \cdots e_{n-k-1}x^{n-k-1} + x^{n-k})$. There are 2^{n-k-1} possible error patterns of this form for each value of i. Of these, all but error bursts of the form $e(x) = x^i g(x)$ are detectable. The fraction of undetectable bursts of length $n - k + 1$ is therefore $2^{-(n-k-1)}$.
- For bursts of length $l > n - k + 1$ starting at position i, all 2^{l-2} of the bursts are detectable except those of the form

  $$e(x) = x^i a(x) g(x)$$

  for some $a(x) = a_0 + a_1 x + \cdots a_{l-n+k-1}x^{l-n+k-1}$ with $a_0 = a_{l-n+k-1} = 1$. The number of undetectable bursts is $2^{l-n+k-2}$, so the fraction of undetectable bursts is 2^{-n+k}.

Example 4.26 Let $g(x) = x^{16} + x^{15} + x^2 + 1$. This can be factored as $g(x) = (1+x)(1+x+x^{15})$, so the CRC is capable of detecting any odd number of bit errors. It can be shown that the smallest integer m such that $g(x)$ divides $1 + x^m$ is $m = 32767$, So by Exercise 4.37, the CRC is able to detect any pattern of two errors — a double error pattern — provided that the code block length $n \le 32767$. All burst errors of length 16 or less are detectable. Bursts of length 17 are detectable with probability 0.99997. Bursts of length ≥ 18 are detectable with probability 0.99998. □

Table 4.6 [373, p. 123], [281] lists commonly used generator polynomials for CRC codes of various lengths.

Table 4.6: CRC Generators

CRC Code	Generator Polynomial
CRC-4	$g(x) = x^4 + x^3 + x^2 + x + 1$
CRC-7	$g(x) = x^7 + x^6 + x^4 + 1$
CRC-8	$g(x) = x^8 + x^7 + x^6 + x^4 + x^2 + 1$
CRC-12	$g(x) = x^{12} + x^{11} + x^3 + x^2 + x + 1$
CRC-ANSI	$g(x) = x^{16} + x^{15} + x^2 + 1$
CRC-CCITT	$g(x) = x^{16} + x^{12} + x^5 + 1$
CRC-SDLC	$g(x) = x^{16} + x^{15} + x^{13} + x^7 + x^4 + x^2 + x + 1$
CRC-24	$g(x) = x^{24} + x^{23} + x^{14} + x^{12} + x^8 + 1$
CRC-32a	$g(x) = x^{32} + x^{30} + x^{22} + x^{15} + x^{12} + x^{11} + x^7 + x^6 + x^5 + x$
CRC-32b	$g(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 +$ $x^7 + x^5 + x^4 + x^2 + x + 1$

4.13.1 Byte-Oriented Encoding and Decoding Algorithms

The syndrome computation algorithms described above are well-adapted to bit-oriented hardware implementations. However, CRCs are frequently used to check the integrity of files or data packets on computer systems which are intrinsically byte-oriented. An algorithm is presented here which produces the same result as a bit-oriented algorithm, but which operates on a byte at a time. The algorithm is faster because it deals with larger pieces of data and also because it makes use of parity information which is computed in advance and stored. It thus has higher storage requirements than the bitwise encoding algorithm but lower operational complexity; for a degree 16 polynomial, 256 two-byte integers must be stored.

Consider a block of N bytes of data, as in a file or a data packet. We think of the first byte of data as corresponding to higher powers of x in its polynomial representation, since polynomial division requires dealing first with highest powers of x. This convention allows the file to be processed in storage order. The data are stored in bytes, as in

$$d_0, d_1, \ldots, d_{N-1},$$

where d_i represents an 8-bit quantity. For a byte of data d_i, let $d_{i,7}, d_{i,6}, \ldots, d_{i,0}$ denote the bits, where $d_{i,0}$ is the least-significant bit (LSB) and $d_{i,7}$ is the most significant bit (MSB). The byte d_i has a corresponding polynomial representation

$$b_{i+1}(x) = d_{i,7}x^7 + d_{i,6}x^6 + \cdots + d_{i,1}x + d_{i,0}.$$

The algorithm described below reads in a byte of data and computes the CRC parity check information for all of the data up to that byte. It is described in terms of a CRC polynomial $g(x)$ of degree 16, but generalization to other degrees is straightforward. For explicitness of examples, the generator polynomial $g(x) = x^{16} + x^{15} + x^2 + 1$ (CRC-ANSI) is used throughout. Let $m^{[i]}(x)$ denote the message polynomial formed from the $8i$ bits of the first i data bytes $\{d_0, d_1, \ldots, d_{i-1}\}$,

$$m^{[i]}(x) = d_{0,7}x^{8i-1} + d_{0,6}x^{8i-2} + \cdots + d_{i-1,1}x + d_{i-1,0},$$

and let $p^{[i]}(x)$ denote the corresponding parity polynomial of degree ≤ 15,

$$p^{[i]}(x) = p_{15}^{[i]}x^{15} + p_{14}^{[i]}x^{14} + \cdots + p_1^{[i]}x + p_0^{[i]}.$$

By the operation of cyclic encoding,

$$p^{[i]}(x) = R_{g(x)}[x^{16}m^{[i]}(x)],$$

that is,

$$x^{16}m^{[i]}(x) = q(x)g(x) + p^{[i]}(x) \tag{4.11}$$

for some quotient polynomial $q(x)$.

Let us now augment the message by one more byte. This is done by shifting the current message polynomial eight positions (bits) and inserting the new byte in the empty bit positions. We can write

$$m^{[i+1]}(x) = \underbrace{x^8 m^{[i]}(x)}_{\text{Shift 8 positions}} + \underbrace{b_{i+1}(x)}_{\text{add new byte}},$$

where $b_{i+1}(x)$ represents the new data. The new parity polynomial is computed by

$$p^{[i+1]}(x) = R_{g(x)}[x^{16}m^{[i+1]}(x)] = R_{g(x)}[x^{16}(x^8 m^{[i]}(x) + b_{i+1}(x))]. \tag{4.12}$$

Using (4.11), we can write (4.12) as

$$p^{[i+1]}(x) = R_{g(x)}[x^8 g(x)q(x) + x^8 p^{[i]}(x) + x^{16}b_{i+1}(x)]$$

This can be expanded as

$$p^{[i+1]}(x) = R_{g(x)}[x^8 g(x)q(x)] + R_{g(x)}[x^8 p^{[i]}(x) + x^{16}b_{i+1}(x)],$$

or

$$p^{[i+1]}(x) = R_{g(x)}[x^8 p^{[i]}(x) + x^{16}b_{i+1}(x)]$$

since $g(x)$ evenly divides $x^8 g(x)q(x)$. The argument can be expressed in expanded form as

$$\begin{aligned} x^8 p^{[i]}(x) + x^{16}b_{i+1}(x) &= p_{15}^{[i]}x^{23} + p_{14}^{[i]}x^{22} + \cdots + p_1^{[i]}x^9 + p_0^{[i]}x^8 \\ &\quad + d_{i,7}x^{23} + d_{i,6}x^{22} + \cdots + d_{i,1}x^{17} + d_{i,0}x^{16} \\ &= (d_{i,7} + p_{15}^{[i]})x^{23} + (d_{i,6} + p_{14}^{[i]})x^{22} + \cdots + (d_{i,0} + p_8^{[i]})x^{16} \\ &\quad + p_7^{[i]}x^{15} + p_6^{[i]}x^{14} + \cdots + p_0^{[i]}x^8. \end{aligned}$$

Now let $t_j = d_{i,j} + p_{j+8}^{[i]}$ for $j = 0, 1, \ldots, 7$. Then

$$x^8 p^{[i]}(x) + x^{16}b_{i+1}(x) = t_7 x^{23} + t_6 x^{22} + \cdots + t_0 x^{16} + p_7^{[i]}x^{15} + p_6^{[i]}x^{14} + \cdots + p_0^{[i]}x^8.$$

The updated parity is thus

$$\begin{aligned} p^{[i+1]}(x) &= R_{g(x)}[t_7 x^{23} + t_6 x^{22} + \cdots + t_0 x^{16} + p_7^{[i]}x^{15} + p_6^{[i]}x^{14} + \cdots + p_0^{[i]}x^8] \\ &= R_{g(x)}[t_7 x^{23} + t_6 x^{22} + \cdots + t_0 x^{16}] + R_{g(x)}[p_7^{[i]}x^{15} + p_6^{[i]}x^{14} + \cdots + p_0^{[i]}x^8] \\ &= R_{g(x)}[t_7 x^{23} + t_6 x^{22} + \cdots + t_0 x^{16}] + p_7^{[i]}x^{15} + p_6^{[i]}x^{14} + \cdots + p_0^{[i]}x^8, \end{aligned}$$

where the last equality follows since the degree of the argument of the second $R_{g(x)}$ is less than the degree of $g(x)$.

Table 4.7: Lookup Table for CRC-ANSI. Values for t and $R(t)$ are expressed in hex.

t	$R(t)$	t	$R(t)$	t	$R(t)$	t	$R(t)$	t	$R(t)$	t	$R(t)$	t	$R(t)$	t	$R(t)$
0	0000	1	8005	2	800f	3	000a	4	801b	5	001e	6	0014	7	8011
8	8033	9	0036	a	003c	b	8039	c	0028	d	802d	e	8027	f	0022
10	8063	11	0066	12	006c	13	8069	14	0078	15	807d	16	8077	17	0072
18	0050	19	8055	1a	805f	1b	005a	1c	804b	1d	004e	1e	0044	1f	8041
20	80c3	21	00c6	22	00cc	23	80c9	24	00d8	25	80dd	26	80d7	27	00d2
28	00f0	29	80f5	2a	80ff	2b	00fa	2c	80eb	2d	00ee	2e	00e4	2f	80e1
30	00a0	31	80a5	32	80af	33	00aa	34	80bb	35	00be	36	00b4	37	80b1
38	8093	39	0096	3a	009c	3b	8099	3c	0088	3d	808d	3e	8087	3f	0082
40	8183	41	0186	42	018c	43	8189	44	0198	45	819d	46	8197	47	0192
48	01b0	49	81b5	4a	81bf	4b	01ba	4c	81ab	4d	01ae	4e	01a4	4f	81a1
50	01e0	51	81e5	52	81ef	53	01ea	54	81fb	55	01fe	56	01f4	57	81f1
58	81d3	59	01d6	5a	01dc	5b	81d9	5c	01c8	5d	81cd	5e	81c7	5f	01c2
60	0140	61	8145	62	814f	63	014a	64	815b	65	015e	66	0154	67	8151
68	8173	69	0176	6a	017c	6b	8179	6c	0168	6d	816d	6e	8167	6f	0162
70	8123	71	0126	72	012c	73	8129	74	0138	75	813d	76	8137	77	0132
78	0110	79	8115	7a	811f	7b	011a	7c	810b	7d	010e	7e	0104	7f	8101
80	8303	81	0306	82	030c	83	8309	84	0318	85	831d	86	8317	87	0312
88	0330	89	8335	8a	833f	8b	033a	8c	832b	8d	032e	8e	0324	8f	8321
90	0360	91	8365	92	836f	93	036a	94	837b	95	037e	96	0374	97	8371
98	8353	99	0356	9a	035c	9b	8359	9c	0348	9d	834d	9e	8347	9f	0342
a0	03c0	a1	83c5	a2	83cf	a3	03ca	a4	83db	a5	03de	a6	03d4	a7	83d1
a8	83f3	a9	03f6	aa	03fc	ab	83f9	ac	03e8	ad	83ed	ae	83e7	af	03e2
b0	83a3	b1	03a6	b2	03ac	b3	83a9	b4	03b8	b5	83bd	b6	83b7	b7	03b2
b8	0390	b9	8395	ba	839f	bb	039a	bc	838b	bd	038e	be	0384	bf	8381
c0	0280	c1	8285	c2	828f	c3	028a	c4	829b	c5	029e	c6	0294	c7	8291
c8	82b3	c9	02b6	ca	02bc	cb	82b9	cc	02a8	cd	82ad	ce	82a7	cf	02a2
d0	82e3	d1	02e6	d2	02ec	d3	82e9	d4	02f8	d5	82fd	d6	82f7	d7	02f2
d8	02d0	d9	82d5	da	82df	db	02da	dc	82cb	dd	02ce	de	02c4	df	82c1
e0	8243	e1	0246	e2	024c	e3	8249	e4	0258	e5	825d	e6	8257	e7	0252
e8	0270	e9	8275	ea	827f	eb	027a	ec	826b	ed	026e	ee	0264	ef	8261
f0	0220	f1	8225	f2	822f	f3	022a	f4	823b	f5	023e	f6	0234	f7	8231
f8	8213	f9	0216	fa	021c	fb	8219	fc	0208	fd	820d	fe	8207	ff	0202

There are $2^8 = 256$ possible remainders of the form

$$R_{g(x)}[t_7 x^{23} + t_6 x^{22} + \cdots + t_0 x^{16}]. \tag{4.13}$$

For each 8-bit combination $t = (t_7, t_6, \ldots, t_0)$, the remainder in (4.13) can be computed and stored in advance. For example, when $t = 1$ (i.e., $t_0 = 1$ and other t_i are 0) we find

$$R_{g(x)}[x^{16}] = x^{15} + x^2 + 1,$$

which has the representation in bits [1,0,0,0, 0,0,0,0, 0,0,0,0, 0,1,0,1], or in hex, 8005. Table 4.7 shows the remainder values for all 256 possible values of t, where the hex number $R(t)$ represents the bits of the syndrome. Let $\tilde{t}_t(x) = t_7 x^{23} + t_6 x^{22} + \cdots + t_0 x^{16}$ and let $R(t) = R_{g(x)}[\tilde{t}(x)]$ (i.e., the polynomial represented by the data in Table 4.7). The encoding update rule is summarized as

$$p^{[i+1]}(x) = R(t) + p_7^{[i]} x^{15} + p_6^{[i]} x^{14} + \cdots + p_0^{[i]} x^8.$$

The algorithm described above in terms of polynomials can be efficiently implemented in terms of byte-oriented arithmetic on a computer. The parity check information is represented in two bytes, `crc1` and `crc0`, with `crc1` representing the high-order parity byte. Together, `[crc1, crc0]` forms the two-byte (16 bit) parity. Also, let $R(t)$ denote the 16-bit parity corresponding to the t, as in Table 4.7. The operation $\oplus$ indicates bitwise modulo-2 addition (i.e., exclusive or). The fast CRC algorithm is summarized in Algorithm 4.1.

Algorithm 4.1 Fast CRC encoding for a stream of bytes

Input: A sequence of bytes $d_0, d_1, \ldots, d_N$.
1 Initialization: Clear the parity information: Set [crc1, crc0] = [0,0].
2 For $i = 0$ to N:
3 Compute $t = d_{N-i} \oplus$ crc1
4 [crc1, crc0] = [crc0,0] $\oplus R(t)$
5 End
6 Output: Return the 16-bit parity [crc1, crc0].

Example 4.27 A file consists of two bytes of data, $d_0 = 39$ and $d_1 = 109$, or in hexadecimal notation, $d_0 = 27_{16}$ and $d_1 = 6D_{16}$. This corresponds to the bits

$$\underbrace{0110\,1101}_{d_1}\ \underbrace{0010\,0111}_{d_0},$$

with the least-significant bit on the right, or, equivalently, the polynomial

$$x^{14} + x^{13} + x^{11} + x^{10} + x^8 + x^5 + x^2 + x + 1.$$

(It is the same data as in Example 4.25.) The steps of the algorithm are as follows:
1 crc1, crc0 = [0,0]
2 $i = 0$:
3 $t = d_1 \oplus$ crc1 $= 6D_{16} + 0 = 6D_{16}$
4 [crc1, crc0] = [crc0, 0] $\oplus R(t) = [0, 0] \oplus 816D_{16}$
2 $i = 1$:
3 $t = d_0 \oplus$ crc1 $= 27_{16} \oplus 81_{16} = A6_{16}$
4 [crc1, crc0] = [crc0, 0] $\oplus R(t) = [6D_{16}, 0] \oplus 03D4_{16} = 6ED4_{16}$
6 Return $6ED4_{16}$.

The return value corresponds to the bits

$$0110\,1110\,1101\,0100$$

which has polynomial representation

$$p(x) = x^{14} + x^{13} + x^{11} + x^{10} + x^9 + x^7 + x^6 + x^4 + x^2.$$

This is the same parity as obtained in Example 4.25. □

4.13.2 CRC Protecting Data Files or Data Packets

When protecting data files or data packets using CRC codes, the CRC codeword length is selected in bytes. As suggested in Example 4.26, for the CRC-16 code the number of bits in the codeword should be less than 32767, so the number of bytes in the codeword should be less than 4095. That is, the number of message bytes should be less than 4093. Let K denote the number of message bytes and let N denote the number of code bytes, for example, $N = K + 2$.

The encoded file is, naturally, longer than the unencoded file, since parity bytes is included. In encoding a file, the file is divided into blocks of length K. Each block of data is written to an encoded file, followed by the parity bytes. At the end of the file, if the number of bytes available is less than K, a shorter block is written out, followed by its parity bytes.

In decoding (or checking) a file, blocks of N bytes are read in and the parity for the block is computed. If the parity is not zero, one or more error has been detected in that block. At the end of the file, if the block size is shorter, the appropriate block length read in is used.

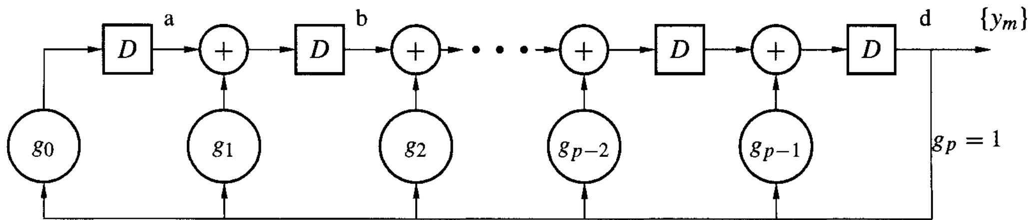


Figure 4.24: Linear feedback shift register.

Appendix 4.A Linear Feedback Shift Registers

Closely related to polynomial division is the linear feedback shift register (LFSR). This is simply a divider with no input — the output is computed based on the initial condition of its storage elements. With proper feedback connections, the LFSR can be used to produce a sequence with many properties of random noise sequences (for example, the correlation function approximates a δ function). These pseudonoise sequences are widely used in spread spectrum communication and as synchronization sequences in common modem protocols. The LFSR can also be used to provide an important part of the representation of Galois fields, which are fundamental to many error correction codes (see Chapter 5). The LFSR also re-appears in the context of decoding algorithms for BCH and Reed-Solomon codes, where an important problem is to determine a shortest LFSR and its initial condition which could produce a given output sequence. (See Chapter 6).

Appendix 4.A.1 Basic Concepts

A binary linear feedback shift register (LFSR) circuit is built using a polynomial division circuit with no input. Eliminating the input to the division circuit of Figure 4.5, we obtain the LFSR shown in Figure 4.24. Since there is no input, the output generated is due to the initial state of the registers. Since there are only a finite number of possible states for this digital device, the circuit must eventually return to a previous state. The number of steps before a state reappears is called the **period** of the sequence generated by the circuit. A binary LFSR with p storage elements has 2^p possible states. Since the all-zero state never changes it is removed from consideration, so the longest possible period is $2^p - 1$.

Example 4.28 Figure 4.25 illustrates the LFSR with connection polynomial $g(x) = 1+x+x^2+x^4$. Table 4.8 shows the sequence of states and the output of the LFSR when it is loaded with the initial condition $(1, 0, 0, 0)$. The sequence of states repeats after 7 steps, so the output sequence is periodic with period 7. Table 4.9 shows the sequence of states for the same connection polynomial when the LFSR is loaded with the initial condition $(1, 1, 0, 0)$, which again repeats after 7 steps. Of the 15 possible nonzero states of the LFSR, these two sequences exhaust all but one of the possible states. The sequence for the last remaining state, corresponding to an initial condition $(1, 0, 1, 1)$, is shown in Table 4.10; this repeats after only one step. □

Example 4.29 Figure 4.26 illustrates the LFSR with connection polynomial $g(x) = 1 + x + x^4$. Table 4.11 shows the sequence of states and the output of the LFSR when it is loaded with the initial condition $(1, 0, 0, 0)$. In this case, the shift register sequences through 15 states before it repeats. □

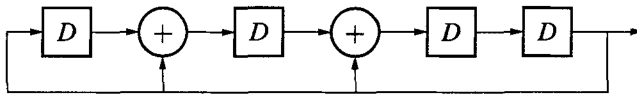
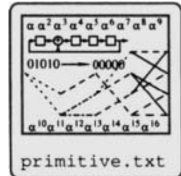


Figure 4.25: Linear feedback shift register with $g(x) = 1 + x + x^2 + x^4$.

Table 4.8: LFSR Example with $g(x) = 1 + x + x^2 + x^4$ and Initial State 1.

Count	State				Output
0	1	0	0	0	0
1	0	1	0	0	0
2	0	0	1	0	0
3	0	0	0	1	1
4	1	1	1	0	0
5	0	1	1	1	1
6	1	1	0	1	1
7	1	0	0	0	0

Definition 4.8 A sequence generated by a connection polynomial $g(x)$ of degree n is said to be a **maximal length** sequence if the period of the sequence is $2^n - 1$. □



Thus, the output sequence of Example 4.29 is a maximal-length sequence, while the output sequences of Example 4.28 are not. A connection polynomial which produces a maximal-length sequence is a **primitive polynomial**. A program to exhaustively search for primitive polynomials modulo p for arbitrary (small) p is `primfind`.

The sequence of outputs of the LFSR satisfy the equation

$$y_m = \sum_{j=0}^{p-1} g_j y_{m-p+j}. \tag{4.14}$$

This may be seen as follows. Denote the output sequence of the LFSR in Figure 4.24 by $\{y_m\}$. For immediate convenience, assume that the sequence is infinite $\{\ldots, y_{-2}, y_{-1}, y_0, y_1, y_2, \ldots\}$ and represent this sequence as a formal power series

$$y(x) = \sum_{i=-\infty}^{\infty} y_i x^i.$$

Table 4.9: LFSR Example with $g(x) = 1 + x + x^2 + x^4$ and Initial State $1 + x$

Count	State				Output
0	1	1	0	0	0
1	0	1	1	0	0
2	0	0	1	1	1
3	1	1	1	1	1
4	1	0	0	1	1
5	1	0	1	0	0
6	0	1	0	1	1
7	1	1	0	0	0

Table 4.10: LFSR Example with $g(x) = 1 + x + x^2 + x^4$ and Initial State $1 + x^2 + x^3$

Count	State	Output
0	1 0 1 1	1
1	1 0 1 1	1

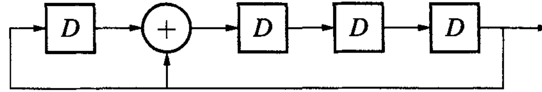


Figure 4.26: Linear feedback shift register with $g(x) = 1 + x + x^4$.

Consider the output at point 'a' in Figure 4.24. Because of the delay x, at point 'a' the signal is

$$g_0 x y(x),$$

where the factor x represents the delay through the memory element. At point 'b', the signal is

$$g_0 x^2 y(x) + g_1 x y(x)$$

Continuing likewise through the system, at the output point 'd' the signal is

$$(g_0 x^p + g_1 x^{p-1} + \cdots g_{p-1} x) y(x),$$

which is the same as the output signal:

$$(g_0 x^p + g_1 x^{p-1} + \cdots g_{p-1} x) y(x) = y(x). \tag{4.15}$$

Equation (4.15) can be true only if coefficients of corresponding powers of x match. This produces the relationship

$$y_i = \sum_{j=0}^{p-1} g_j y_{i-p+j}. \tag{4.16}$$

Letting $g_j^* = g_{p-j}$, (4.16) can be written in the somewhat more familiar form as a convolution,

$$y_i = \sum_{j=1}^{p} g_j^* y_{i-j}. \tag{4.17}$$

Equation (4.17) can also be re-written as

$$\sum_{j=0}^{p} g_j^* y_{i-j} = 0 \tag{4.18}$$

with the stipulation that $g_0^* = 1$.

The polynomial $g^*(x)$ with coefficients $g_j^* = g_{p-j}$ is sometimes referred to as the **reciprocal polynomial**. That is, $g^*(x)$ has its coefficients in the reverse order from $g(x)$. (The term "reciprocal" does not mean that $h(x)$ is a multiplicative inverse of $g(x)$; it is just a conventional name.) The reciprocal polynomial of $g(x)$ is denoted as $g^*(x)$. If $g(x)$ is

Table 4.11: LFSR example with $g(x) = 1 + x + x^4$ and initial state 1

Count	State				Output
0	1	0	0	0	0
1	0	1	0	0	0
2	0	0	1	0	0
3	0	0	0	1	1
4	1	1	0	0	0
5	0	1	1	0	0
6	0	0	1	1	1
7	1	1	0	1	1
8	1	0	1	0	0
9	0	1	0	1	1
10	1	1	1	0	0
11	0	1	1	1	1
12	1	1	1	1	1
13	1	0	1	1	1
14	1	0	0	1	1
15	1	0	0	0	0

a polynomial of degree p with non-zero constant term (i.e., $g_0 = 1$ and $g_p = 1$), then the reciprocal polynomial can be obtained by

$$g^*(x) = x^p g(1/x).$$

It is clear in this case that the reciprocal of the reciprocal is the same as the original polynomial. However, if $g_0 = 0$, then the degree of $x^p g(1/x)$ is less than the degree of $g(x)$ and this latter statement is not true.

With the understanding that the output sequence is periodic with period $2^p - 1$, so that $y_{-1} = y_{2^p-2}$, (4.18) is true for all $i \in \mathbb{Z}$. Because the sum in (4.18) is equal to 0 for all i, the polynomial $g^*(x)$ is said to be an *annihilator* of $y(x)$.

Example 4.30 For the coefficient polynomial $g(x) = 1 + x + x^2 + x^4$, the reciprocal polynomial is $g^*(x) = 1 + x^2 + x^3 + x^4$ and the LFSR relationship is

$$y_i = y_{i-2} + y_{i-3} + y_{i-4} \qquad \text{for all } i \in \mathbb{Z}. \tag{4.19}$$

For the output sequence of Table 4.8 $\{0, 0, 0, 1, 0, 1, 1, 0, \ldots\}$, it may be readily verified that (4.19) is satisfied. □

The LFSR circuit diagram is sometimes expressed in terms of the reciprocal polynomials, as shown in Figure 4.27. It is important to be careful of the conventions used.

Appendix 4.A.2 Connection With Polynomial Division

The output sequence produced by an LFSR has a connection with polynomial long division. To illustrate this, let us take $g(x) = 1 + x + x^2 + x^4$, as in Example 4.28. The reciprocal polynomial is $g^*(x) = 1 + x^2 + x^3 + x^4$. Let the dividend polynomial be $d(x) = x^3$. (The relationship between the sequence and the dividend are explored in Exercise 4.43.) The

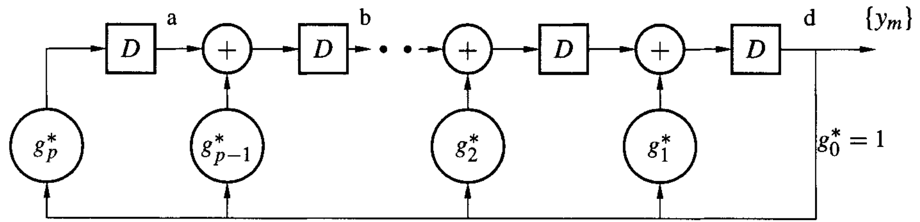


Figure 4.27: Linear feedback shift register, reciprocal polynomial convention.

power series obtained by dividing $d(x)$ by $g^*(x)$, with $g^*(x)$ written in order of *increasing* degree, is obtained by formal long division:

$$
\begin{array}{r|l}
 & x^3 + x^5 + x^6 \quad \cdots \\
\hline
1 + x^2 + x^3 + x^4 & x^3 \\
 & \underline{x^3 + x^5 + x^6 + x^7} \\
 & \quad x^5 + x^6 + x^7 \\
 & \quad \underline{x^5 + x^7 + x^8 + x^9} \\
 & \qquad x^6 + x^8 + x^9 \\
 & \qquad \underline{x^6 + x^8 + x^9 + x^{10}} \\
 & \qquad\qquad\qquad\qquad x^{10}
\end{array}
$$

The quotient polynomial corresponds to the sequence $\{0, 0, 0, 1, 0, 1, 1, \ldots\}$, the same as the output sequence shown in Table 4.8.

Let $y_0, y_1, \ldots$ be an infinite sequence produced by an LFSR, which we represent with $y(x) = y_0 + y_1 x + y_2 x^2 + \cdots = \sum_{n=0}^{\infty} y_n x^n$. Furthermore, represent the initial state of the shift register as $y_{-1}, y_{-2}, \ldots, y_{-p}$. Using the recurrence relation (4.17) we have

$$
\begin{aligned}
y(x) &= \sum_{n=0}^{\infty} \sum_{j=1}^{p} g_j^* y_{n-j} x^n = \sum_{j=1}^{p} g_j^* x^j \sum_{n=0}^{\infty} y_{n-j} x^{n-j} \\
&= \sum_{j=1}^{p} g_j^* x^j \left[(y_{-j} x^{-j} + \cdots + y_{-1} x^{-1}) + \sum_{n=0}^{\infty} y_n x^n \right] \\
&= \sum_{j=1}^{p} g_j^* x^j \left[(y_{-j} x^{-j} + \cdots + y_{-1} x^{-1}) + y(x) \right]
\end{aligned}
$$

so that

$$
\begin{aligned}
y(x) &= \frac{\sum_{j=1}^{p} g_j^* x^j (y_{-j} x^{-j} + \cdots + y_{-1} x^{-1})}{1 - \sum_{i=1}^{p} g_j^* x^j} \\
&= \frac{\sum_{j=1}^{p} g_j^* (y_{-j} + y_{-j+1} x + \cdots + y_{-1} x^{j-1})}{g^*(x)}.
\end{aligned}
\tag{4.20}
$$

Example 4.31 Returning to Example 4.28, with the periodic sequence $y_{-1} = 1$, $y_{-2} = 1$, $y_{-3} = 0$

and $y_{-4} = 1$. From (4.20) we find

$$y(x) = \frac{x^3}{g^*(x)} = \frac{x^3}{1 + x^2 + x^3 + x^4},$$

as before. □

Theorem 4.3 *Let $y(x)$ be produced by a LFSR with connection polynomial $g(x)$ of degree p. If $y(x)$ is periodic with period N then $g^*(x) \mid (x^N - 1)d(x)$, where $d(x)$ is a polynomial of degree $< p$.*

Proof By the results above, $y(x) = \frac{d(x)}{g^*(x)}$ for a polynomial $d(x)$ with $\deg(d(x)) < p$. If $y(x)$ is periodic then

$$\begin{aligned} y(x) &= (y_0 + y_1 x + \cdots + y_{N-1}x^{N-1}) + x^N(y_0 + y_1 x + \cdots + y_{N-1}x^{N-1}) \\ &\quad + x^{2N}(y_0 + y_1 x + \cdots + y_{N-1}x^{N-1}) + \cdots \\ &= (y_0 + y_1 x + \cdots + y_{N_1}x^{N-1})(1 + x^N + x^{2N} + \cdots) \\ &= -\frac{(y_0 + y_1 x + \cdots + y_{N-1}x^{N-1})}{x^N - 1}. \end{aligned}$$

So

$$\frac{d(x)}{g^*(x)} = -\frac{(y_0 + y_1 x + \cdots + y_{N-1}x^{N-1})}{x^N - 1}$$

or $g^*(x)(y_0 + y_1 x + \cdots + y_{N-1}x^{N-1}) = -d(x)(x^N - 1)$, establishing the result. □

For a given $d(x)$, the period is the smallest N such that $g^*(x) \mid (x^N - 1)d(x)$.

Example 4.32 The polynomial $g^*(x) = 1 + x^2 + x^3 + x^4$ can be factored as

$$g^*(x) = (1 + x)(1 + x + x^3).$$

Taking $N = 1$ and $d(x) = 1 + x + x^3$ we see that $y(x)$ has period 1. This is the sequence shown in Table 4.10. We note that $g^*(x) \mid x^7 - 1$, so that any $d(x)$ of appropriate degree will serve. □

As a sort of converse to the previous theorem, we have the following.

Theorem 4.4 *If $g^*(x) \mid x^N - 1$, then $y(x) = \frac{1}{g^*(x)}$ is periodic with period N or some divisor of N.*

Proof Let $q(x) = \frac{x^N - 1}{g^*(x)} = y_0 + y_1 x + \cdots y_{N-1}x^{N-1}$. Then

$$\begin{aligned} \frac{1}{g^*(x)} &= \frac{y_0 + y_1 + \cdots y_{N-1}x^{N-1}}{1 - x^N} = (y_0 + y_1 x + \cdots y_{N-1}x^{N-1})(1 + x^N + x^{2N} + \cdots) \\ &= (y_0 + y_1 x + \cdots y_{N-1}x^{N-1}) + x^N(y_0 + y_1 x + \cdots y_{N-1}x^{N-1}) + \cdots, \end{aligned}$$

which represents a periodic sequence. □

Theorem 4.5 *If the sequence $y(x)$ produced by the connection polynomial $g(x)$ of degree p has period $2^p - 1$ — that is, $y(x)$ a maximal-length sequence — then $g^*(x)$ is irreducible.*

Proof Since the shift register moves through $2^p - 1$ states before repeating, the shift register must progress through all possible nonzero conditions. Therefore, there is some "initial condition" corresponding to $d(x) = 1$. Without loss of generality we can take $y(x) = 1/g^*(x)$.

Suppose that $g^*(x)$ factors as $a^*(x)b^*(x)$, where $\deg(a^*(x)) = p_1$ and $\deg(b^*(x)) = p_2$, with $p_1 + p_2 = p$. Then

$$y(x) = \frac{1}{g^*(x)} = \frac{1}{a^*(x)b^*(x)} = \frac{c(x)}{a^*(x)} + \frac{d(x)}{b^*(x)}$$

by partial fraction expansion. $c(x)/a^*(x)$ represents a series with period at most $2^{r_1} - 1$ and $d(x)/b^*(x)$ represents a series with period at most $2^{r_2} - 1$. The period of the sum $\frac{c(x)}{a^*(x)} + \frac{d(x)}{b^*(x)}$ is at most the least common multiple of these periods, which must be less than the product of the periods:

$$(2^{p_1} - 1)(2^{p_2} - 1) = 2^p - 3.$$

But this is less than the period $2^p - 1$, so $g^*(x)$ must not have such factors. □

As mentioned above, irreducibility does not imply maximal-length. The polynomial $g^*(x) = 1 + x + x^2 + x^3 + x^4$ divides $x^5 + 1$. But by Theorem 4.4, $y(x) = 1/g^*(x)$ has period 5, instead of the period 15 that a maximal-length sequence would have. What is needed for the polynomial to be primitive.

Appendix 4.A.3 Some Algebraic Properties of Shift Sequences

Let $y(x)$ be a sequence with period N. Then $y(x)$ can be considered an element of $R_N = GF(2)[x]/(x^N - 1)$. Let $g(x)$ be a connection polynomial and $g^*(x)$ be its reciprocal. Let $w(x) = g^*(x)y(x)$, where computation occurs in the ring R_N, and let $w(x) = w_0 + w_1x + \cdots + w_{N-1}x^{N-1}$. The coefficient w_i of this polynomial is computed by

$$w_i = \sum_{j=0}^{p} g_p^* y_{i-j}.$$

However, by (4.18), this is equal to 0. That is, $g^*(x)y(x) = 0$ in the ring R_N. In this case, we say that $g^*(x)$ *annihilates* the sequence $y(x)$. Let $V(g^*)$ be the set of sequences annihilated by $g^*(x)$. We observe that $V(g^*)$ is an ideal in the ring R_n and has a generator $h^*(x)$ which must divide $x^N - 1$. The generator $h^*(x)$ is the polynomial factor of $(x^N - 1)/g^*(x)$ of smallest positive degree. If $(X^N - 1)/g^*(x)$ is irreducible, then $h^*(x) = (X^N - 1)/g^*(x)$.

Example 4.33 Let $g(x) = 1 + x + x^2 + x^4$, as in Example 4.28. Then $g^*(x) = 1 + x^2 + x^3 + x^4$. This polynomial divides $x^7 + 1$:

$$h^*(x) = \frac{x^7 + 1}{g^*(x)} = 1 + x^2 + x^3.$$

The polynomial $y(x) = h^*(x)$ corresponds to the output sequence $1, 0, 1, 1, 0, 0, 0$ and its cyclic shifts, which appears in Table 4.8.

The polynomial $y(x) = (1+x)h^*(x) = 1+x+x^2+x^4$ corresponds to the sequence $1, 1, 1, 0, 1$ and its cyclic shifts, which appears in Table 4.9.

The polynomial $y(x) = (1 + x + x^3) * h^*(x) = 1 + x + x^2 + x^3x^4 + x^5 + x^6$ corresponds to the sequence $1, 1, 1, 1, 1, 1$ and its cyclic shifts. This sequence appears in Table 4.10. This sequence also happens to have period 2. □

Example 4.34 For the generator polynomial $g(x) = 1+x+x^4$ and its reciprocal $g^*(x) = 1+x^3+x^4$. This polynomial divides $x^{15}+1$:

$$h^*(x) = \frac{x^{15}+1}{g^*(x)} = 1+x^3+x^4+x^6+x^8+x^9+x^{10}+x^{11}.$$

The polynomial $y(x) = h^*(x)$ corresponds to the sequence $1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1$, which appears in Table 4.11. □

Programming Laboratory 2: Polynomial Division and Linear Feedback Shift Registers

Objective

Computing quotients and remainders in polynomial division is an important computational step for encoding and decoding cyclic codes. In this lab, you are to create a C++ class which performs these operations for binary polynomials. You will also create an LFSR class, which will be used in the construction of a Galois field class.

Preliminary Exercises

Reading: Section 4.9, Appendix 4.A.1.

1) Let $g(x) = x^4+x^3+x+1$ and $d(x) = x^8+x^7+x^5+x^4+x^3+x+1$.

> **a)** Perform polynomial long division of $d(x)$ and $g(x)$, computing the quotient and remainder, as in Example 4.15.
>
> **b)** Draw the circuit configuration for dividing by $g(x)$.
>
> **c)** Trace the operation of the circuit for the $g(x)$ and $d(x)$ given, identifying the polynomials represented by the shift register contents at each step of the algorithm, as in Table 4.2. Also, identify the quotient and the remainder produced by the circuit.

2) For the connection polynomial $g(x) = x^4+x^3+x+1$, trace the LFSR when the initial register contents are $(1, 0, 0, 0)$, as in Example 4.28. Also, if this does not exhaust all possible 15 states of the LFSR, determine other initial states and the sequences they generate.

Programming Part: `BinLFSR`

Create a C++ class `BinLFSR` which implements an LFSR for a connection polynomial of degree < 32. Create a constructor with arguments

```
BinLFSR(int g, int n, int initstate=1);
```

The first argument `g` is a representation of the connection polynomial. For example, `g = 0x17` represents the bits `10111`, which represents the polynomial $g(x) = x^4+x^2+x+1$. The second argument `n` is the degree of the connection polynomial. The third argument has a default value, corresponding to the initial state (1,0,0,...,0). Use a single `unsigned int` internally to hold the state of the shift register. The class should have member functions as follows:

```
BinLFSR(void) { g=n=state=mask=mask1=0;}
     // default constructor
BinLFSR(int g, int n, int initstate=1);
     // constructor
~BinLFSR() {};
     // destructor
void setstate(int state);
     // Set the initial state of the LFSR
unsigned char step(void);
     // Step the LFSR one step,
     // and return 1-bit output
unsigned char step(int &state);
     // Step the LFSR one step,
     // return 1-bit output
     // and the new state
void steps(int nstep, unsigned char *outputs);
     // Step the LFSR nstep times,
     // returning the array of 1-bit outputs
```

Test the class as follows:

1) Use the LFSR class to generate the three sequences of Example 4.28.

2) Use the LFSR class to generate the output sequence and the sequence of states shown in Table 4.11.

Resources and Implementation Suggestions

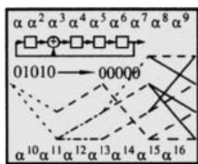
The storage of the polynomial divider and LFSR could be implemented with a character array, as in

```
unsigned char *storage = new unsigned char[n];
```

Shifting the registers would require a `for` loop. However, since the degree of the coefficient polynomial is of degree < 32, all the memory can be contained in a single 4-byte integer, and the register shift can be accomplished with a single bit shift operation.

- The operator << shifts bits left, shifting 0 into the least significant bits. Thus, if `a=3`, then `a<<2` is equal to 12. The number `1<<m` is equal to 2^m for $m \geq 0$.

- The operator >> shifts bits right, shifting in 0 to the most significant bit. Thus, if `a=13`, then `a»2` is equal to 3.
- Hexadecimal constants can be written using `0xnnnn`, as in `0xFF` (the number 255), or `0x101` (the number 257). Octal constants can be written using `0nnn`, as in 0123, which has the bit pattern 001 010 011.
- The bitwise `and` operator `&` can be used to mask bits off. For example, if `a = 0x123`, then in `b = a & 0xFF;`, `b` is equal to `0x23`. To retain the lowest `m` bits of a number, mask it with `((1«m)-1)`.
- The algorithms can be implemented either by shifting right using >> or by shifting left using <<. For a few reasons, it makes sense to shift left, so that the input comes into the least significant bit and the output comes out of the most significant bit. This may be initially slightly confusing, since the pictures portray shifts to the right.
- **As a tutorial, the code for the LFSR is explicitly portrayed.**



Algorithm 4.2 BinLFSR
File: `BinLFSR.h`
`BinLFSR.cc`
`testBinLFSR.cc`
`MakeLFSR`

The class declarations are given in `BinLFSR.h`. The class definitions are given in `BinLFSR.cc`. In this case, the definitions are short enough that it would make sense to merge the `.cc` file into the `.h` file, but they are separated for pedagogical reasons.[4] A simple test program is `testBinLFSR.cc`. A very simple makefile (if you choose to use `make`) is in `MakeLFSR`.

Programming Part: `BinPolyDiv`

Create a C++ class `BinPolyDiv` which implements a polynomial divisor/remainder circuit, where the degree of $g(x)$ is < 32. The constructor has arguments representing the divisor polynomial and its degree:

```
BinPolyDiv(unsigned char *g, int p);
```

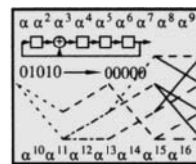
The class should have member functions `div` and `remainder` which compute, respectively, the quotient and the remainder, with arguments as follows:

```
int div(unsigned char *d,    // dividend
        int ddegree,
        unsigned char *q,
        int &quotientdegree,
        int &remainderdegree);
```

```
int remainder(unsigned char *d,
        int n,
        int &remainderdegree);
```

The dividend `d` is passed in as an `unsigned char` array, one bit per character, so that arbitrarily long dividend polynomials can be accommodated. The remainder is returned as a single integer whose bits represent the storage register, with the least-significant bit representing the coefficient of smallest degree of the remainder. Internally, the remainder should be stored in a single `unsigned int`.

Test your function on the polynomials $g(x) = x^4 + x^3 + x + 1$ and $d(x) = x^8 + x^7 + x^5 + x^4 + x^3 + x + 1$ from the Preliminary Exercises. Also test your function on the polynomials from Example 4.15.



Algorithm 4.3 BinPolyDiv
File: `BinPolyDiv.h`
`BinPolyDiv.cc`
`testBinPolyDiv.cc`

Follow-On Ideas and Problems

A binary $\{0, 1\}$ sequence $\{y_n\}$ can be converted to a binary ± 1 sequence z_n by $z_n = (-1)^{y_n}$. For a binary ± 1 sequence $\{z_0, z_1, \ldots, z_{N-1}\}$ with period N, define the cyclic autocorrelation by

$$r_z(\tau) = \frac{1}{N}\sum_{i=0}^{N-1} z_i z_{((i+\tau))},$$

where $((i + \tau))$ denotes $i + \tau$ modulo N.

Using your LFSR class, generate the sequence with connection polynomial $g(x) = 1 + x + x^4$ and compute and plot $r_z(\tau)$ for $\tau = 0, 1, \ldots, 15$. (You may want to make the plots by saving the computed data to a file, then plotting using some convenient plotting tool such as Matlab.) You should observe that there is a single point with correlation 1 (at $\tau = 0$) and that the correlations at all other lags has correlation $-1/N$.

The shape of the correlation function is one reason that maximal length sequences are called pseudonoise sequences: the correlation function approximates a δ function (with the approximation improving for longer N).

As a comparison, generate a sequence with period 7 using $g(x) = 1+x+x^2+x^4$ and plot $r_z(\tau)$ for this sequence.

[4] The *comment* at the end of this code is parsed by the `emacs` editor in the `c++` mode. This comment can be used by the `compile` command in `emacs` to run the compiler inside the editor.

Programming Laboratory 3: CRC Encoding and Decoding

Objective

In this lab, you become familiar with cyclic encoding and decoding, both in bit-oriented and byte-oriented algorithms.

Preliminary

Reading: Section 4.13.

Verify that the remainder $d(x)$ in (4.10) is correct by dividing $x^{16}m(x)$ by $g(x)$. (You may want to do this both by hand and using a test program invoking `BinPolyDiv`, as a further test on your program.)

Programming Part

1) Write a C++ class `CRC16` which computes the 16-bit parity bits for a stream of data, where $g(x)$ is a generator polynomial of degree 16. The algorithm should use the polynomial division idea (that is, a bit-oriented algorithm. You may probably want to make use of a `BinPolyDiv` object from Lab 2 in your class). Here is a class declaration you might find useful:

```
class CRC16 {
protected:
    BinPolyDiv div;     // the divider object
public:
    CRC16(int crcpoly); // constructor
    int CRC(unsigned char *data, int len);
    // Compute the CRC for the data
    // data=data to be encoded
    // len = number of bytes to be encoded
    // Return value: the 16 bits of
    // parity
    // (data[0] is associated with the
    // highest power of x^n)
};
```

Test your program first using Example 4.25.

2) Write a standalone program `crcenc` which encodes a file, making use of your `CRC16` class. Use $g(x) = x^{16} + x^{15} + x^2 + 1$. The program should accept three command line arguments:

```
crcenc K filein fileout
```

where `K` is the message block length (in bytes), `filein` is the input file, and `fileout` is the encoded file.

3) Write a standalone program `crcdec` which decodes a file, making use of your `CRC` class. The program should accept three arguments:

```
crcdec K filein fileout
```

where `K` is the message block length (in bytes), `filein` is an encoded file, and `fileout` is a decoded file.

4) Test `crcenc` and `crcdec` by first encoding then decoding a file, then comparing the decoded file with the original. (A simple compare program is `cmpsimple`.) The decoded file should be the same as the original file. Use a message block length of 1024 bytes. Use a file of 1,000,000 random bytes created using the `makerand` program for the test.

5) Test your programs further by passing the encoded data through a binary symmetric channel using the `bsc` program. Try channel crossover probabilities of 0.00001, 0.001, 0.01, and 0.1. Are there any blocks of data that have errors that are not detected?

6) Write a class `FastCRC16` which uses the byte-oriented algorithm to compute the parity bits for a generator $g(x)$ of degree 16. A sample class definition follows:

```
class FastCRC16 {
protected:
    static int *crctable;
    unsigned char crc0, crc1;
    // the two bytes of parity
public:
    FastCRC16(int crcpoly);  // constructor
    int CRC(unsigned char *data, int len);
    // Compute the CRC for the data
    // data[0] corresponds to the
    // highest powers of x
};
```

The table of parity values (as in Table 4.7) should be stored in a static class member variable (see the discussion below about static variables). The constructor for the class should allocate space for the table and fill the table, if it has not already been built.

7) Test your program using the data in Example 4.27.

8) Write a standalone program `fastcrcenc` which encodes a file using `FastCRC16`. Use $g(x) = x^{16} + x^{15} + x^2 + 1$. The program should have the same arguments as the program `crcenc`. Test your program by encoding some data and verify that the encoded file is the same as for a file encoded using `crcenc`.

9) Write a decoder program `fastcrcdec` which decodes using `FastCRC16`. Verify that it decodes correctly.

10) Compare the encoding rates of `crcenc` and `fastcrcenc`. How much faster is the byte-oriented algorithm?

Resources and Implementation Suggestions

Static Member Variables A static member variable of a class is a variable that is associated with the class. However, all instances of the class share that same data, so the data is not really part of any particular object. To see why these might be used, suppose that you want to build a system that has two `FastCRC16` objects in it:

```
FastCRC16 CRC1(g);   // instantiate two objects
FastCRC16 CRC2(g);
```

The `FastCRC16` algorithm needs the data from Table 4.7. This data could be represented using member data as in

```
class FastCRC16 {
protected:
   int *crctable;
   unsigned char crc0, crc1;
   // the two bytes of parity
public:
   FastCRC16(int crcpoly);  // constructor
   int CRC(unsigned char *data, int len);
};
```

However, there are two problems with this:

1) Each object would have its own table. This wastes storage space.
2) Each object would have to construct its table, as part of the constructor routine. This wastes time.

As an alternative, the lookup table could be stored in a static member variable. Then it would only need to be constructed once (saving computation time) and only stored once (saving memory). The tradeoff is that it is not possible by this arrangement to have two or more different lookup tables in the same system of software at the same time. (There are ways to work around this problem, however. You should try to think of a solution on your own.)

The declaration `static int *crctable;` which appears in the `.h` file does not define the variable. There must be a definition somewhere, in a C++ source file that is only compiled once. Also, since it is a static object, in a sense external to the class, its definition must be fully scoped. Here is how it is defined:

```
// File: FastCRC.cc

// ...
#include "FastCRC.h"

int *FastCRC16::crctable=0;
```

This defines the pointer and initializes it to 0. allocation of space for the table and computation of its contents is accomplished by the constructor:

```
// Constructor for FastCRC16 object
FastCRC16::FastCRC16(int crcpoly)
{
   if(FastCRC16::crctable==0) {
      // the table has not been allocated yet
      FastCRC16::crctable = new int[256];
      // Now build the tables
      // ...
   }
   // ...
}
```

Static member variables do not necessarily disappear when an object goes out of scope. We shall use static member variables again in the Galois field arithmetic implementation.

Command Line Arguments For operating systems which provide a command-line interface, reading the command line arguments into a program is very straightforward. The arguments are passed in to the `main` routine using the variables `argc` and `argv`. These may then be parsed and used. `argc` is the total number of arguments on the command line, including the program name. If there is only the program name (with no other arguments), then `argc==1`. `argv` is an array of pointers to the string commands. `argv[0]` is the name of the program being run.

As an example, to read the arguments for `crcenc K filein fileout`, you could use the following code:

```
// Program crcenc

// ...

main(int argc, char *argv[])
{
   int K;
   char *infname, *outfname;

   // ...
   if(argc!=4) {
   // check number of arguments is as expected
      cout << "Usage: " << argv[0] <<
         "K infile outfile" << endl;
      exit(-1);
   }
   K = atoi(argv[1]);
   // read blocksize as an integer
   infname = argv[2];
   // pointer to input file name
   outfname = argv[3];
   // pointer to output file name
   // ...
}
```

Picking Out All the Bits in a File To write the bit-oriented decoder algorithm, you need to pick out all the bits in an array of data. Here is some sample code:

```
// d is an array of unsigned characters
// with 'len' elements
unsigned char bits[8];
// an array that hold the bits of one byte of d

for(int i = 0; i < len; i++) {
   // work through all the bytes of data
   for(int j = 7; j >= 0; j--) {
     // work through the bits in each byte
     bits[j] = (data[i]&(1<<j)) != 0;
   }
   // bits now has the bits of d[i] in it
   // ...
}
```

4.14 Exercises

4.1 List the codewords for the (7, 4) Hamming code. Verify that the code is cyclic.

4.2 In a ring with identity (that is, multiplicative identity), denote this identity as 1. Prove:

(a) The multiplicative identity is unique.

(b) If an element a has both a right inverse b (i.e., an element b such that $ab = 1$) and a left inverse c (i.e., an element c such that $ca = 1$), then $b = c$. In this case, the element a is said to have an inverse (denoted by a^{-1}). Show that the inverse of an element a, when it exists, is unique.

(c) If a has a multiplicative inverse a^{-1}, then $(a^{-1})^{-1} = a$.

(d) The set of units of a ring forms a group under multiplication. (Recall that a unit of a ring is an element that has a multiplicative inverse).

(e) If $c = ab$ and c is a unit, then a has a right inverse and b has a left inverse.

(f) In a ring, a nonzero element a such that $ax = 0$ for $x \neq 0$ is said to be a zero divisor. Show that if a has an inverse, then a is not a zero divisor.

4.3 Construct the ring $R_4 = GF(2)[x]/(x^4 + 1)$. That is, construct the addition and multiplication tables for the ring. Is R_4 a field?

4.4 Let R be a commutative ring and let $a \in R$. Let $I = \{b \in R : ab = 0\}$. Show that I is an ideal of R.

4.5 An element a of a ring R is nilpotent if $a^n = 0$ for some positive integer n. Show that the set of all nilpotent elements in a commutative ring R is an ideal.

4.6 Let A and B be ideals in a ring R. The sum $A + B$ is defined as $A + B = \{a + b : a \in A, b \in B\}$. Show that $A + B$ is an ideal in R. Show that $A \subset A + B$.

4.7 Show that in the ring $\mathbb{Z}_{15}$ the polynomial $p(x) = x^2 - 1$ has more than two zeros. In a field there would be only two zeros. What may be lacking in a ring that leads to "too many" zeros?

4.8 In the ring $R_4 = GF(2)[x]/(x^4 + 1)$, multiply $a(x) = 1 + x^2 + x^3$ and $b(x) = x + x^2$. Also, cyclically convolve the sequences $\{1, 0, 1, 1\}$ and $\{0, 1, 1\}$. What is the relationship between these two results?

4.9 For the (15,11) binary Hamming code with generator $g(x) = x^4 + x + 1$:

(a) Determine the parity check polynomial $h(x)$.

(b) Determine the generator matrix G and the parity check matrix H for this code in nonsystematic form.

(c) Determine the generator matrix G and the parity check matrix H for this code in systematic form.

(d) Let $m(x) = x + x^2 + x^3$. Determine the code polynomial $c(x) = g(x)m(x)$.

(e) Let $m(x) = x + x^2 + x^3$. Determine the systematic code polynomial $c(x) = x^{n-k}m(x) + R_{g(x)}[x^{n-k}m(x)]$, where $R_{g(x)}[\,]$ computes the remainder after division by $g(x)$.

(f) For the codeword $c(x) = 1 + x + x^3 + x^4 + x^5 + x^9 + x^{10} + x^{11} + x^{13}$, determine the message if nonsystematic encoding is employed.

(g) For the codeword $c(x) = 1 + x + x^3 + x^4 + x^5 + x^9 + x^{10} + x^{11} + x^{13}$, determine the message if systematic encoding is employed.

(h) Let $r(x) = x^{14} + x^{10} + x^5 + x^3$. Determine the syndrome for $r(x)$.

(i) Draw the systematic encoder circuit for this code using the $g(x)$ feedback polynomial.

(j) Draw the decoder circuit for this circuit with $r(x)$ input on the left of the syndrome register. Determine in particular the error pattern detection circuit.

(k) Draw the decoder circuit for this circuit with $r(x)$ input on the right of the syndrome register. Determine in particular the error pattern detection circuit.

(l) Let $r(x) = x^{13} + x^{10} + x^9 + x^5 + x^2 + 1$. Trace the execution of the Meggitt decoder with the input on the left, analogous to Table 4.4.

(m) Let $r(x) = x^{13} + x^{10} + x^9 + x^5 + x^2 + 1$. Trace the execution of the Meggitt decoder with the input on the right, analogous to Table 4.5.

4.10 Let $f(x)$ be a polynomial of degree m in $\mathbb{F}[x]$, where $\mathbb{F}$ is a field. Show that if a is a root of $f(x)$ (so that $f(a) = 0$), then $(x - a) \mid f(x)$. *Hint*: Use the division algorithm. Inductively, show that $f(x)$ has at most m roots in $\mathbb{F}$.

4.11 The following are code polynomials from binary cyclic codes. Determine the highest-degree generator $g(x)$ for each code.

(a) $c(x) = 1 + x^4 + x^5$

(b) $c(x) = 1 + x + x^2 + x^3 + x^4 + x^5 + x^6 + x^7 + x^8 + x^9 + x^{10} + x^{11} + x^{12} + x^{13} + x^{14}$

(c) $c(x) = x^{13} + x^{12} + x^9 + x^5 + x^4 + x^3 + x^2 + 1$

(d) $c(x) = x^8 + 1$

(e) $c(x) = x^{10} + x^7 + x^5 + x^4 + x^3 + x^2 + x + 1$

4.12 Let $g(x) = g_0 + g_1 x + \cdots + g_{n-k} x^{n-k}$ be the generator for a cyclic code. Show that $g_0 \neq 0$.

4.13 Show that $g(x) = 1 + x + x^4 + x^5 + x^7 + x^8 + x^9$ generates a binary (21,12) cyclic code. Devise a syndrome computation circuit for this code. Let $r(x) = 1 + x^4 + x^{16}$ be a received polynomial. Compute the syndrome of $r(x)$. Also, show the contents of the syndrome computation circuit as each digit of $r(x)$ is shifted in.

4.14 [204] Let $g(x)$ be the generator for a binary (n, k) cyclic code $\mathcal{C}$. The reciprocal of $g(x)$ is defined as

$$g^*(x) = x^{n-k} g(1/x).$$

(In this context, "reciprocal" does *not* mean multiplicative inverse.)

(a) As a particular example, let $g(x) = 1 + x^2 + x^4 + x^6 + x^7 + x^{10}$. Determine $g^*(x)$. The following subproblems deal with arbitrary cyclic code generator polynomials $g(x)$.

(b) Show that $g^*(x)$ also generates an (n, k) cyclic code.

(c) Let $\mathcal{C}^*$ be the code generated by $g^*(x)$. Show that $\mathcal{C}$ and $\mathcal{C}^*$ have the same weight distribution.

(d) Suppose $\mathcal{C}$ has the property that whenever $c(x) = c_0 + c_1 x + c_{n-1} x^{n-1}$ is a codeword, so is its reciprocal $c^*(x) = c_{n-1} + c_{n-2} x + \cdots + c_0 x^{n-1}$. Show that $g(x) = g^*(x)$. Such a code is said to be a *reversible* cyclic code.

4.15 [204] Let $g(x)$ be the generator polynomial of a binary cyclic code of length n.

(a) Show that if $g(x)$ has $x + 1$ as a factor then the code contains no codevectors of odd weight. *Hint*: The following is true for any ring $\mathbb{F}[x]$:

$$\boxed{1 - x^{n-1} = (1 - x)(1 + x + x^2 + \cdots + x^{n-1})}.$$

(b) Show that if n is odd and $x + 1$ is *not* a factor of $g(x)$, then the code contains the all-one codeword.

(c) Show that the code has minimum weight 3 if n is the smallest integer such that $g(x)$ divides $x^n - 1$.

4.16 Let $A(z)$ be the weight enumerator for a binary cyclic code $\mathcal{C}$. with generator $g(x)$. Suppose furthermore that $x + 1$ is not a factor of $g(x)$. Show that the code generated by $\tilde{g}(x) = (x + 1)g(x)$ has weight enumerator $\tilde{A}(z) = \frac{1}{2}[A(z) + A(-z)]$.

4.17 Let $g(x)$ be the generator polynomial of an (n, k) cyclic code $\mathcal{C}$. Show that $g(x^\lambda)$ generates an $(\lambda n, \lambda k)$ cyclic code that has the same minimum weight as the code generated by $g(x)$.

4.18 Let $\mathcal{C}$ be a $(2^m - 1, 2^m - m - 1)$ Hamming code. Show that if a Meggitt decoder with input on the right-hand side is used, as in Figure 4.19, then the syndrome to look for to correct the digit r_{n-1} is $s(x) = x^{m-1}$. *Hint*: $g(x)$ divides $x^{2^m-1} + 1$. Draw the Meggitt decoder for this Hamming code decoder.

4.19 [33] The code of length 15 generated by $g(x) = 1 + x^4 + x^6 + x^7 + x^8$ is capable of correcting 2 errors. (It is a (15,7) BCH code.) Show that there are 15 correctable error patterns in which the highest-order bit is equal to 1. Devise a Meggitt decoder for this code with the input applied to the right of the syndrome register. Show that the number of syndrome patterns to check can be reduced to 8.

4.20 Let $g(x)$ be the generator of a $(2^m - 1, 2^m - m - 1)$ Hamming code and let $\tilde{g}(x) = (1 + x)g(x)$. Show that the code generated by $\tilde{g}(x)$ has minimum distance exactly 4.

(a) Show that there exist distinct integers i and j such that $x^i + x^j$ is not a codeword generated by $g(x)$. Write $x^i + x^j = q_1(x)g(x) + r_1(x)$.

(b) Choose an integer k such that the remainder upon dividing x^k by $g(x)$ is not equal to $r_1(x)$. Write $x^i + x^j + x^k = q_2(x)g(x) + r_2(x)$

(c) Choose an integer l such that when x^l is divided by $g(x)$ the remainder is $r_2(x)$. Show that l is not equal to i, j, or k.

(d) Show that $x^i + x^j + x^k + x^l = [q_2(x) + q_3(x)]g(x)$ and that $x^i + x^j + x^k + x^l$ is a multiple of $(x + 1)g(x)$

4.21 [204] An error pattern of the form $e(x) = x^i + x^{i+1}$ is called a double-adjacent-error pattern. Let $\mathcal{C}$ be the $(2^m - 1, 2^m - m - 2)$ cyclic code generated by $g(x) = (x + 1)p(x)$, where $p(x)$ is a primitive polynomial of degree m.

Show that no two double-adjacent-error patterns can be in the same coset of a standard array for $\mathcal{C}$. Also show that no double-adjacent error pattern and single error pattern can be in the same coset of the standard array. Conclude that the code is capable of correcting all the single-error patterns and all the double-adjacent-error patterns.

4.22 [204] Let $c(x)$ be a code polynomial in a cyclic code of length n and let $c^{(i)}(x)$ be its ith cyclic shift. Let l be the smallest positive integer such that $c^{(l)}(x) = c(x)$. Show that l is a factor of n.

4.23 Verify that the circuit shown in Figure 4.1 computes the product $a(x)h(x)$.

4.24 Verify that the circuit shown in Figure 4.2 computes the product $a(x)h(x)$.

4.25 Verify that the circuit shown in Figure 4.3 computes the product $a(x)h(x)$.

4.26 Verify that the circuit shown in Figure 4.4 computes the product $a(x)h(x)$.

4.27 Let $h(x) = 1 + x^2 + x^3 + x^4$. Draw the multiplier circuit diagrams as in Figures 4.1, 4.2, 4.3, and 4.4.

4.28 Let $r(x) = 1 + x^3 + x^4 + x^5$ be the input to the decoder in Figure 4.20. Trace the execution of the decoder by following the contents of the registers. If the encoding is systematic, what was the transmitted message?

4.29 Cyclic code dual.

(a) Let $\mathcal{C}$ be a cyclic code. Show that the dual code $\mathcal{C}^\perp$ is also a cyclic code.

(b) Given a cyclic code $\mathcal{C}$ with generator polynomial $g(x)$, describe how to obtain the generator polynomial for the dual code $\mathcal{C}^\perp$.

4.30 As described in Section 3.6, the dual to a Hamming code is a $(2^m - 1, m)$ maximal-length code. Determine the generator matrix for a maximal-length code of length $2^m - 1$.

4.31 Let C be an (n, k) binary cyclic code with minimum distance $d_{\min}$ and let $C' \subset C$ be the shortened code for which the l high-order message bits are equal to 0. Show that C' has 2^{k-l} codewords and is a linear code. Show that the minimum distance $d'_{\min}$ of C' is at least as large as $d_{\min}$.

4.32 For the binary $(31, 26)$ Hamming code generated using $g(x) = 1+x^2+x^5$ shortened to a (28,23) Hamming code.

(a) Draw the decoding circuit for the (28,23) shortened code using the method of simulating extra clock shifts.

(b) Draw the decoding circuit for the (28,23) shortened code using the method of changing the error pattern detection circuit.

4.33 Explain why CRC codes can be thought of as shortened cyclic codes.

4.34 Let $g(x) = 1 + x^2 + x^4 + x^5$. Determine the fraction of all burst errors of the following lengths that can be detected by a cyclic code using this generator: (a) burst length= 4; (b) burst length= 5; (c) burst length = 6; (d) burst length = 7; (e) burst length = 8.

4.35 Let $g(x) = x^8 + x^7 + x^6 + x^4 + x^2 + 1$ be the generator polynomial for a CRC code.

(a) Let $m(x) = 1 + x + x^3 + x^6 + x^7 + x^{12} + x^{16} + x^{21}$. Determine the CRC encoded message $c(x)$.

(b) The CRC-encoded polynomial $r(x) = x^2 + x^3 + x^5 + x^6 + x^7 + x^8 + x^{10} + x^{11} + x^{14} + x^{17} + x^{20} + x^{23} + x^{26} + x^{28}$ is received. Has an error been made in transmission?

4.36 Verify the entry for $t = 3$ in Table 4.7. Also verify the entry for t =dc.

4.37 A double error pattern is one of the form $e(x) = x^i + x^j$ for $0 \leq i < j \leq n - 1$. If $g(x)$ does not have x as a factor and does not evenly divide $1 + x^{j-i}$, show that any double error pattern is detectable.

4.38 A file containing the two bytes $d_0 = 56 = 38_{16}$ and $d_1 = 125 = 7D_{16}$ is to be CRC encoded using the CRC-ANSI generator polynomial $g(x) = x^{16} + x^{15} + x^2 + 1$.

(a) Convert these data to a polynomial $m(x)$.

(b) Determine the CRC-encoded data $c(x) = x^r m(x) + R_{g(x)}[x^r m(x)]$ and represent the encoded data as a stream of bits.

(c) Using the fast CRC encoding of Algorithm 4.1, encode the data. Verify that it corresponds to the encoding obtained previously.

4.39 The output sequence of an LFSR with connection polynomial $g(x)$ can be obtained by formal division of some dividend $d(x)$ by $g^*(x)$. Let $g(x) = 1 + x + x^4$. Show by computational examples that when the connection polynomial is reversed (i.e., reciprocated), the sequence generated by it is reversed (with possibly a different starting point in the sequence). Verify this result analytically.

4.40 Show that (4.16) follows from (4.15). Show that (4.17) follows from (4.16).

4.41 Show that the circuit in Figure 4.28 produces the same sequence as that of Figure 4.24. (Of the two implementations, the one in Figure 4.24 is generally preferred, since the cascaded summers of Figure 4.28 result in propagation delays which inhibit high-speed operations.)

4.42 Figure 4.29 shows an LFSR circuit with the outputs of the memory elements labeled as state variables x_1 through x_p. Let

$$\mathbf{x}[k] = \begin{bmatrix} x_1[k] \\ x_2[k] \\ \vdots \\ x_p[k] \end{bmatrix}.$$

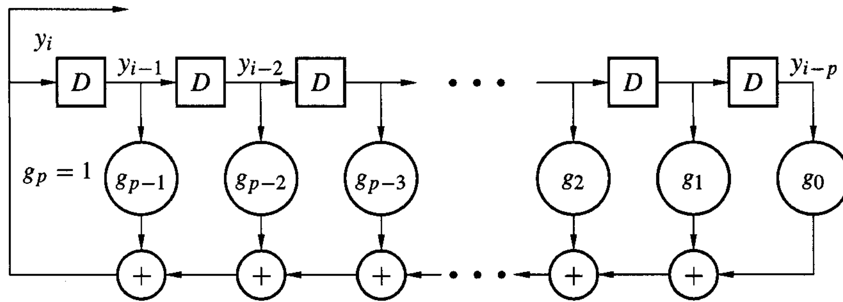


Figure 4.28: Another LFSR circuit.

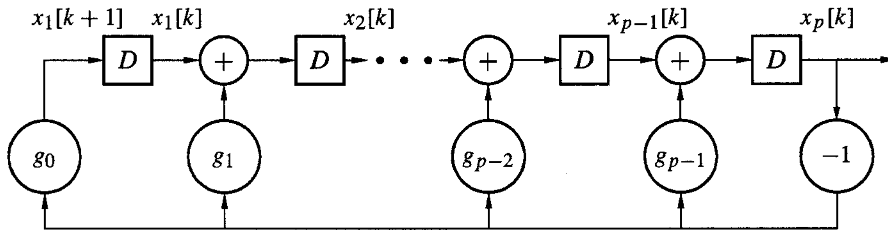


Figure 4.29: An LFSR with state labels.

(a) Show that for the state labels as in Figure 4.29 that the state update equation is

$$\mathbf{x}[k+1] = M\mathbf{x}[k],$$

where M is the companion matrix

$$M = \begin{bmatrix} 0 & 0 & 0 & 0 & \cdots & 0 & -g_0 \\ 1 & 0 & 0 & 0 & \cdots & 0 & -g_1 \\ 0 & 1 & 0 & 0 & \cdots & 0 & -g_2 \\ 0 & 0 & 1 & 0 & \cdots & 0 & -g_3 \\ \vdots & & & & & & \\ 0 & 0 & 0 & 0 & \cdots & 1 & -g_{p-1} \end{bmatrix}.$$

(b) The characteristic polynomial of a matrix is $p(x) = \det(xI - M)$. Show that

$$p(x) = g_0 + g_1 x + g_2 x^2 + \cdots + g^p = g(x).$$

(c) It is a fact that every matrix satisfies its own characteristic polynomial. That is, $p(M) = 0$. (This is the Cayley-Hamilton theorem.) Use this to show that if $g(x) \mid (1-x^k)$ then $M^k = I$.

(d) The period k of an LFSR with initial vector $\mathbf{x}[0]$ is the smallest k such that $M^k = I$. Interpret this in light of the Cayley-Hamilton theorem, if $p(x)$ is irreducible.

(e) A particular output sequence $\{x_p[0], x_p[1], \ldots, \}$ is to be produced from this LFSR. Determine what the initial vector $\mathbf{x}[0]$ should be to obtain this sequence. (That is, what is the initial value of the LFSR register?)

4.43 Given a sequence $y(x)$ produced by dividing by the reciprocal polynomial of $g(x)$,

$$y(x) = \frac{d(x)}{g^*(x)},$$

determine what $d(x)$ should be to obtain the given $y(x)$.

The sequence $\{0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, \ldots\}$ is generated by the polynomial $g^*(x) = 1 + x^3 + x^4$. Determine the numerator polynomial $d(x)$.

4.44 Show that the set of sequences annihilated by a polynomial $g^*(x)$ is an ideal.

4.45 [199] A Barker code is a binary-valued sequence $\{b_n\}$ of length n whose autocorrelation function has values of 0, 1, and n. Only nine such sequences are known, shown in Table 4.12.

(a) Compute the autocorrelation value for the Barker sequence $\{b_5\}$.

(b) Contrast the autocorrelation function for a Barker code with that of a maximal-length LFSR sequence.

Table 4.12: Barker Codes

n	$\{b_n\}$
2	[1,1]
2	[-1,1]
3	[1,1,-1]
4	[1,1,-1,1]
4	[1,1,1,-1]
5	[1,1,1,-1,1]
7	[1,1,1,-1,-1,1,-1]
11	[1,1,1,-1,-1,-1,1,-1,-1,1,-1]
13	[1,1,1,1,1,-1,-1,1,1,-1,1,-1,1]

4.15 References

Cyclic codes were explored by Prange [271, 272, 273]. Our presentation owes much to Wicker [373], who promotes the idea of cyclic codes as ideals in a ring of polynomials. The Meggitt decoder is described in [237]. Our discussion of the Meggitt decoder closely follows [203]; many of the exercises were also drawn from that source.

The tutorial paper [281] provides an overview of CRC codes, comparing five different implementations and also providing references to more primary literature.

Much of the material on polynomial operations was drawn from [262]. The table of primitive polynomials is from [386], which in turn was drawn from [262]. An early but still important and thorough work on linear feedback shift registers is [120]. See also [119, 118]. The circuit implementations presented here can take other canonical forms. For other realizations, consult a book on digital signal processing, such as [253], or controls [109, 181]. The paper [387] has some of the fundamental algebraic results in it. An example of maximal length sequences to generate modem synchronization sequences is provided in [160]. The paper [301] has descriptions of correlations of maximal length sequences under decimation.

Chapter 5

Rudiments of Number Theory and Algebra

5.1 Motivation

We have seen that the cyclic structure of a code provides a convenient way to encode and reduces the complexity of decoders for some simple codes compared to linear block codes. However, there are several remaining questions to be addressed in approaching practical long code designs and effective decoding algorithms.

1. The cyclic structure means that the error pattern detection circuitry must only look for errors in the last digit. This reduces the amount of storage compared to the syndrome decoding table. However, for long codes, the complexity of the error pattern detection circuitry may still be considerable. It is therefore of interest to have codes with additional *algebraic* structure, in addition to the cyclic structure, that can be exploited to develop efficient decoding algorithms.
2. The decoders presented in chapter 4 are for *binary* codes: knowing the location of errors is sufficient to decode. However, there are many important nonbinary codes, for which both the error locations and values must be determined. We have presented no theory yet for how to do this.
3. We have seen that generator polynomials $g(x)$ must divide $x^n - 1$. Some additional algebraic tools are necessary to describe how to find such factorizations over arbitrary finite fields.
4. Finally, we have not presented yet a *design* methodology, by which codes having a specified minimum distance might be designed.

This chapter develops mathematical tools to address these issues. In reality, the amount of algebra presented in this chapter is both more and less than is needed. It is more than is needed, in that concepts are presented which are not directly called for in later chapters (even though their presence helps puts other algebraic notions more clearly in perspective). It is less than is needed, in that the broad literature of coding theory uses all of the algebraic concepts presented here, and much more. An attempt has been made to strike a balance in presentation.

Example 5.1 We present another example motivating the use of the algebra over finite fields [25]. This example will preview many of the concepts to be developed in this chapter, including modulo operations, equivalence, the Euclidean algorithm, irreducibility, and operations over a finite field.

We have seen in Section 1.9.2 that the decoding algorithm for the Hamming code can be expressed purely in an algebraic way: finding the (single) error can be expressed as finding the solution to a single algebraic equation. It is possible to extend this to a two-error-correcting code whose solution

is found by solving two polynomial equations in two unknowns. We demonstrate this by a particular example, starting from a Hamming $(31, 26)$ code having a parity check matrix

$$H = \begin{bmatrix} 0 & 0 & 0 & \cdots & 1 & 1 \\ 0 & 0 & 0 & \cdots & 1 & 1 \\ 0 & 0 & 0 & \cdots & 1 & 1 \\ 0 & 1 & 1 & \cdots & 1 & 1 \\ 1 & 0 & 1 & \cdots & 0 & 1 \end{bmatrix}.$$

The 5-tuple in the ith column is obtained from the binary representation of the integer i. As in Section 1.9.2, we represent the 5-tuple in the ith column as a single "number," denoted by γ_i, so we write

$$H = \begin{bmatrix} \gamma_1 & \gamma_2 & \gamma_3 & \cdots & \gamma_{30} & \gamma_{31} \end{bmatrix}.$$

Let us now attempt to move beyond a single error correction code by appending 5 additional rows to H. We will further assume that the 5 new elements in each column are some function of column number. That is, we assume that we can write H as

$$H = \begin{bmatrix} 0 & 0 & 0 & \cdots & 1 & 1 \\ 0 & 0 & 0 & \cdots & 1 & 1 \\ 0 & 0 & 0 & \cdots & 1 & 1 \\ 0 & 1 & 1 & \cdots & 1 & 1 \\ 1 & 0 & 1 & \cdots & 0 & 1 \\ f_1(1) & f_1(2) & f_1(3) & \cdots & f_1(30) & f_1(31) \\ f_2(1) & f_2(2) & f_2(3) & \cdots & f_2(30) & f_2(31) \\ f_3(1) & f_3(2) & f_3(3) & \cdots & f_3(30) & f_3(31) \\ f_4(1) & f_4(2) & f_4(3) & \cdots & f_4(30) & f_4(31) \\ f_5(1) & f_5(2) & f_4(3) & \cdots & f_5(30) & f_5(31) \end{bmatrix}. \tag{5.1}$$

The function $\mathbf{f}(i) = [f_1(i), f_2(i), f_3(i), f_4(i), f_5(i)]^T$ has binary components, so $f_l(i) \in \{0, 1\}$. This function tells what binary pattern should be associated with each column. Another way to express this is to note that $\mathbf{f}$ maps binary 5-tuples to binary 5-tuples. We can also use our shorthand notation. Let $f(\gamma)$ be the symbol represented by the 5-tuple $(f_1(i), f_2(i), f_3(i), f_4(i), f_5(i))$, where i is the integer corresponding to $\gamma_i = \gamma$. Using our shorthand notation we could write (5.1) as

$$H = \begin{bmatrix} \gamma_1 & \gamma_2 & \gamma_3 & \cdots & \gamma_{30} & \gamma_{31} \\ f(\gamma_1) & f(\gamma_2) & f(\gamma_3) & \cdots & f(\gamma_{30}) & f(\gamma_{31}) \end{bmatrix}.$$

The problem now is to select a function $\mathbf{f}$ so that H represents a code capable of correcting two errors, and does so in such a way that an algebraic solution is possible. To express the functions $\mathbf{f}$ we need some way of dealing with these γ_i 5-tuples as algebraic objects in their own right, with arithmetic defined to add, subtract, multiply, and divide. That is, the γ_i need to form a *field*, as defined in Section 2.3, or (since there are only finitely many of them) a *finite field*. Addition in the field is straightforward: we could define addition element-by-element. But how do we multiply in a meaningful, nontrivial way? How do we divide?

The key is to think of each 5-tuple as corresponding to a polynomial of degree ≤ 4. For example:

$$\begin{aligned} (0,0,0,0,0) &\leftrightarrow 0 \\ (0,0,0,0,1) &\leftrightarrow 1 \\ (0,0,0,1,0) &\leftrightarrow x \\ (0,0,1,0,0) &\leftrightarrow x^2 \\ (1,0,1,0,1) &\leftrightarrow x^4 + x^2 + 1. \end{aligned}$$

Note that each coefficient of the polynomials is binary; we assume that addition is modulo 2 (i.e., over GF(2)). Clearly, addition of polynomials accomplishes exactly the same thing as addition of the vectors. (They are **isomorphic.**)

How can we multiply? We want our polynomials representing the 5-tuples to have degree ≤ 4, and yet when we multiply the degree may exceed that. For example,

$$(x^3 + x + 1)(x^4 + x^3 + x + 1) = x^7 + x^6 + x^5 + x^4 + x^2 + 1.$$

To reduce the degree, we choose some polynomial $M(x)$ of degree 5, and **reduce the product modulo** $M(x)$. That is, we divide by $M(x)$ and take the remainder. Let us take $M(x) = x^5 + x^2 + 1$. When we divide $x^7 + x^6 + x^5 + x^4 + x^2 + 1$ by $M(x)$ we get a quotient of $x^2 + x + 1$, and a remainder of $x^3 + x^2 + x$. We use the remainder:

$$\begin{aligned}(x^3 + x + 1)(x^4 + x^3 + x + 1) &= x^7 + x^6 + x^5 + x^4 + x^2 + 1 \\ &\equiv x^3 + x^2 + x \pmod{x^5 + x^2 + 1}.\end{aligned}$$

Our modulo operations allows us now to add, subtract, and multiply these 5-tuples, considered as polynomials modulo some $M(x)$. Can we divide? More fundamentally, given a polynomial $a(x)$, is there some other polynomial $s(x)$ — we may consider it a multiplicative inverse or a reciprocal — such that

$$a(x)s(x) \equiv 1 \mod M(x).$$

The answer lies in the oldest algorithm in the world, the Euclidean algorithm. (More details later!) For now, just be aware that if $M(x)$ is **irreducible** — it cannot be factored — then we can define division so that all of the 5-tuples γ_i have a multiplicative inverse except (0, 0, 0, 0, 0).

Let us return now to the problem of creating a two-error-correcting code. Suppose that there are two errors, occurring at positions i_i and i_2. Since the code is linear, it is sufficient to consider

$$\mathbf{r} = (0, 0, \ldots, \underbrace{1}_{i_1}, \ldots, \underbrace{1}_{i_2}, 0, \ldots, 0)$$

We find

$$\mathbf{r}H^T = (s_1, s_2)$$

with

$$\begin{aligned}\gamma_{i_1} + \gamma_{i_2} &= s_1 \\ f(\gamma_{i_1}) + f(\gamma_{i_2}) &= s_2.\end{aligned} \tag{5.2}$$

If the two equations in (5.2) are functionally independent, then we have two equations in two unknowns, which we could solve for γ_{i_1} and γ_{i_2} which, in turn, will determine the error locations i_1 and i_2.

Let us consider some possible simple functions. One might be a simple multiplication: $f(\gamma) = a\gamma$. But this would lead to the two equations

$$\gamma_{i_1} + \gamma_{i_2} = s_1 \qquad a\gamma_{i_1} + a\gamma_{i_2} = s_2,$$

representing the dependency $s_2 = as_1$; the new parity check equations would tell us nothing new.

We could try $f(\gamma) = \gamma + a$; This would not help, since we would always have $s_2 = s_1$.

Let us try some powers. Say $f(\gamma) = \gamma^2$. We would then obtain

$$\gamma_{i_1} + \gamma_{i_2} = s_1 \qquad \gamma_{i_1}^2 + \gamma_{i_2}^2 = s_2.$$

These looks like independent equations, but we have to remember that we are dealing with operations modulo 2. Notice that

$$s_1^2 = (\gamma_{i_1} + \gamma_{i_2})^2 = \gamma_{i_1}^2 + \gamma_{i_2}^2 + 2\gamma_{i_1}\gamma_{i_2} = \gamma_{i_1}^2 + \gamma_{i_2}^2 = s_2.$$

We have only the redundant $s_1^2 = s_2$: the second equation is the square of the first and still conveys no new information.

Try $f(\gamma) = \gamma^3$. Now the decoder equations are

$$\gamma_{i_1} + \gamma_{i_2} = s_1 \qquad \gamma_{i_1}^3 + \gamma_{i_2}^3 = s_2.$$

These are independent!

Now let's see what we can do to solve these equations algebraically. In a finite field, we can do conventional algebraic manipulation, keeping in the back of our mind how we do multiplication and division.

We can write

$$s_2 = \gamma_{i_1}^3 + \gamma_{i_2}^3 = (\gamma_{i_1} + \gamma_{i_2})(\gamma_{i_1}^2 - \gamma_{i_1}\gamma_{i_2} + \gamma_{i_2}^2) = s_1(\gamma_{i_1}^2 + \gamma_{i_1}\gamma_{i_2} + \gamma_{i_2}^2) = s_1(\gamma_{i_1}\gamma_{i_2} - s_1^2)$$

(where the signs have changed with impunity because these values are based on $GF(2)$). Hence we have the two equations

$$\gamma_{i_1} + \gamma_{i_2} = s_1 \qquad \gamma_{i_1}\gamma_{i_2} = s_1^2 + \frac{s_2}{s_1}$$

if $s_1 \neq 0$. We can combine these two equations into a quadratic:

$$\gamma_{i_1}(s_1 + \gamma_{i_1}) = s_1^2 + \frac{s_2}{s_1},$$

or

$$\gamma_{i_1}^2 + s_1\gamma_{i_1} + \left(s_1^2 + \frac{s_2}{s_1}\right) = 0$$

or

$$1 + s_1\gamma_{i_1}^{-1} + \left(s_1^2 + \frac{s_2}{s_1}\right)\gamma_{i_1}^{-2} = 0.$$

For reasons to be made clear later, it is more useful to deal with the reciprocals of the roots. Let $z = \gamma_{i_1}^{-1}$. We then have the equation

$$q(z) = 1 + s_1 z + \left(s_1^2 + \frac{s_2}{s_1}\right)z^2 = 0.$$

The polynomial $q(z)$ is said to be an *error locator polynomial*: the reciprocals of its roots tell the γ_{i_1} and γ_{i_2}, which, in turn, tell the locations of the errors.

If there is only one error, then $\gamma_{i_1} = s_1$ and $\gamma_{i_1}^3 = s_2$ and we end up with the equation $1 + s_1\gamma^{-1} = 0$. If there are no errors, then $s_1 = s_2 = 0$.

Let us summarize the steps we have taken. First, we have devised a way of operating on 5-tuples as single algebraic objects, defining addition, subtraction, multiplication, and division. This required finding some irreducible polynomial $M(x)$ which works behind the scenes. Once we have got this, the steps are as follows:

1. We compute the syndrome $\mathbf{r}H^T$.
2. From the syndrome, we set up the error locator polynomial. We note that there must be some relationship between the sums of the powers of roots and the coefficients.
3. We then find the roots of the polynomial, which determine the error locations.

For binary codes, knowing where the error is suffices to correct the error. For nonbinary codes, there is another step: knowing the error location, we must also determine the error value at that location. This involves setting up another polynomial, the error evaluation polynomial, whose roots determine the error values.

The above steps establish the outline for this and the next chapters. Not only will we develop more fully the arithmetic, but we will be able to generalize to whole families of codes, capable of correcting many errors. However, the concepts are all quite similar to those demonstrated here.

(It is historically interesting that it took roughly ten years of research to bridge the gap between Hamming and the code presented above. Once this was accomplished, other generalizations followed quickly.) □

5.2 Number Theoretic Preliminaries

We begin with some notation and concepts from elementary number and polynomial theory.

5.2.1 Divisibility

Definition 5.1 An integer b is **divisible** by a nonzero integer a if there is an integer c such that $b = ac$. This is indicated notationally as $a \mid b$ (read "a divides b"). If b is not divisible by a we write $a \nmid b$. Let $a(x)$ and $b(x)$ be polynomials in $F[x]$ (that is, the ring of polynomials with coefficients in F) where F is a field and assume that $a(x)$ is not identically 0. Then $b(x)$ is divisible by a polynomial $a(x)$ if there is some polynomial $c(x) \in F[x]$ such that $b(x) = a(x)c(x)$; this is indicated by $a(x) \mid b(x)$. □

Example 5.2 For $a(x)$ and $b(x)$ in $\mathbb{R}[x]$, with

$$b(x) = 112 + 96x + 174x^2 + 61x^3 + 42x^4 \quad \text{and} \quad a(x) = \frac{3}{4}x^2 + \frac{5}{7}x + 2$$

we have $a(x) \mid b(x)$ since $b(x) = 28(2 + x + 2x^3)a(x)$. □

The following properties of divisibility of integers are straightforward to show.

Lemma 5.1 *[250] For integers,*

1. *$a \mid b$ implies $a \mid bc$ for any integer c.*
2. *$a \mid b$ and $b \mid c$ imply $a \mid c$.*
3. *$a \mid b$ and $a \mid c$ imply $a \mid (bs + ct)$ for any integers s and t.*
4. *$a \mid b$ and $b \mid a$ imply $a = \pm b$.*
5. *$a \mid b$, $a > 0$ and $b > 0$ imply $a \leq b$.*
6. *if $m \neq 0$, then $a \mid b$ if and only if $ma \mid mb$*
7. *if $ac \mid bc$ then $a \mid b$.*
8. *if $a \mid b$ and $c \mid d$ then $ac \mid bd$.*

These properties apply with a few modifications to polynomials. Property (4) is different for polynomials: if $a(x) \mid b(x)$ and $b(x) \mid a(x)$ then $a(x) = cb(x)$, where c is a nonzero element of the field of coefficients. Property (5) is also different for polynomials: $a(x) \mid b(x)$ implies $\deg(a(x)) \leq \deg(b(x))$.

An important fact regarding division is expressed in the following theorem.

Theorem 5.2 **(Division algorithm)** *For any integers a and b with $a > 0$, there exist* unique *integers q and r such that*

$$b = qa + r,$$

where $0 \le r < a$. *The number* q *is the quotient and* r *is the remainder.*

For polynomials, for $a(x)$ *and* $b(x)$ *in* $F[x]$, F *a field, there is a unique representation*

$$b(x) = q(x)a(x) + r(x),$$

where $\deg(r(x)) < \deg(a(x))$.

Proof [250, p. 5] We provide a partial proof for integers. Form the arithmetic progression

$$\ldots, b-3a, b-2a, b-a, b, b+a, b+2a, b+3a, \ldots$$

extending indefinitely in both directions. In this sequence select the smallest non-negative element and denote it by r; this satisfies the inequality $0 \le r < a$ and implicitly defines q by $r = b - qa$. □

Example 5.3 With $b = 23$ and $a = 7$ we have

$$23 = 3 \cdot 7 + 2.$$

The quotient is 3 and the remainder is 2. □

Example 5.4 With $b(x) = 2x^3 + 3x + 2$ and $a(x) = x^2 + 7$ in $\mathbb{R}[x]$,

$$b(x) = (2x)(x^2 + 7) + (-11x + 2).$$

□

Definition 5.2 If $d \mid a$ and $d \mid b$ then d is said to be a **common divisor** of a and b.

A common divisor $g > 0$ such that every common divisor of a and b divides g is called the **greatest common divisor** (GCD) and is denoted by (a, b).

Integers a and b with a greatest common divisor equal to 1 are said to be **relatively prime**. The integers $a_1, a_2, \ldots, a_k$ are **pairwise** relatively prime if $(a_i, a_j) = 1$ for $i \ne j$.

If $d(x) \mid a(x)$ and $d(x) \mid b(x)$ then $d(x)$ is said to be a **common divisor** of $a(x)$ and $b(x)$. If either $a(x)$ or $b(x)$ is not zero, the common divisor $g(x)$ such that every common divisor of $a(x)$ and $b(x)$ divides $g(x)$ is referred to as the **greatest common divisor** (GCD) of $a(x)$ and $b(x)$ and is denoted by $(a(x), b(x))$.

The GCD of polynomials $(a(x), b(x))$ is, by convention, normalized so that it is a *monic* polynomial.

If the greatest common divisor of $a(x)$ and $b(x)$ is a constant (which can be normalized to 1), then $a(x)$ and $b(x)$ are said to be **relatively prime**. □

Example 5.5 If $a = 24$ and $b = 18$ then, clearly, $(a, b) = (24, 18) = 6$. □

Example 5.6 By some trial and error (to be reduced to an effective algorithm), we can determine that $(851, 966) = 23$. □

Example 5.7 With $a(x) = 4x^3 + 10x^2 + 8x + 2$ and $b(x) = 8x^3 + 14x^2 + 7x + 1$ in $\mathbb{R}[x]$, it can be shown that

$$(a(x), b(x)) = x^2 + \frac{3}{2}x + \frac{1}{2}.$$

□

Useful properties of the greatest common divisor:

Theorem 5.3

1. *For any positive integer m,* $(ma, mb) = m(a, b)$.
2. *As a consequence of the previous result, if* $d \mid a$ *and* $d \mid b$ *and* $d > 0$ *then*
$$\left(\frac{a}{d}, \frac{b}{d}\right) = \frac{1}{d}(a, b).$$
3. *If* $(a, b) = g$ *then* $(a/g, b/g) = 1$.
4. *If* $(a, c) = (b, c) = 1$*, then* $(ab, c) = 1$
5. *If* $c \mid ab$ *and* $(b, c) = 1$ *then* $c \mid a$.
6. *Every divisor d of a and b divides* (a, b)*. This follows immediately from (3) in Lemma 5.1 (or from the definition).*
7. $(a, b) = |a|$ *if and only if* $a \mid b$.
8. $(a, (b, c)) = ((a, b), c)$ *(associativity).*
9. $(ac, bc) = |c|(a, b)$ *(distributivity).*

5.2.2 The Euclidean Algorithm and Euclidean Domains

The Euclidean algorithm is perhaps the oldest algorithm in the world, being attributed to Euclid over 2000 years ago and appearing in his *Elements*. It was formulated originally to find the greatest common divisor of two integers. It has since been generalized to apply to elements in an algebraic structure known as a *Euclidean domain*. The powerful algebraic consequences include a method for solving a key step in the decoding of Reed-Solomon and BCH codes.

To understand the Euclidean algorithm, it is perhaps most helpful to first see the Euclidean algorithm in action, without worrying formally yet about how it works. The Euclidean algorithm works by simple repeated division: Starting with two numbers, a and b, divide a by b to obtain a remainder. Then divide b by the remainder, to obtain a new remainder. Proceed in this manner, dividing the last divisor by the most recent remainder, until the remainder is 0. Then the last nonzero remainder is the greatest common divisor (a, b).

Example 5.8 Find $(966, 851)$. Let $a = 966$ and $b = 851$. Divide a by b and express in terms of quotient and remainder. The results are expressed in equation and "long division"form:

$$966 = 851 \cdot 1 + 115$$

$$\begin{array}{r} 1 \\ 851\,\overline{)966} \\ \underline{851} \\ 115 \end{array}$$

Now take the divisor (851) and divide it by the remainder (115):

$$851 = 115 \cdot 7 + 46$$

$$\begin{array}{rr} 7 & 1 \\ 115\,\overline{)851} & \overline{)966} \\ \underline{805} & \underline{851} \\ 46 & 115 \end{array}$$

Now take the divisor (115) and divide it by the remainder (46):

$$115 = 46 \cdot 2 + 23$$

```
           2    7    1
    46 | 115 | 851 | 966
          92   805   851
         ---   ---   ---
          23    46   115
```

Now take the divisor (46) and divide it by the remainder (23):

$$46 = \mathbf{23} \cdot 2 + 0$$

```
         2    2    7    1
    23 | 46 | 115 | 851 | 966
         23    92   805   851
        ---   ---   ---   ---
          0    23    46   115
```

The remainder is now 0; the last nonzero remainder 23 is the GCD:

$$(966, 851) = 23.$$

□

Example 5.9 In this example, we perform computations over $\mathbb{Z}_5[x]$, that is, operations modulo 5. Determine $(a(x), b(x)) = (x^7 + 3x^6 + 4x^4 + 2x^3 + x^2 + 4, x^6 + 3x^3 + 2x + 4)$, where $a(x), b(x) \in \mathbb{Z}_5[x]$.

$$\begin{aligned}(x^7 + 3x^6 + 4x^4 + 2x^3 + x^2 + 4) &= (x + 3)(x^6 + 3x^3 + 2x + 4) + (x^4 + 3x^3 + 4x^2 + 2)\\(x^6 + 3x^3 + 2x + 4) &= (x^2 + 2x)(x^4 + 3x^3 + 4x^2 + 2) + (3x^2 + 3x + 4)\\(x^4 + 3x^3 + 4x^2 + 2) &= (2x^2 + 4x + 3)(3x^2 + 3x + 4) + 0\end{aligned} \tag{5.3}$$

With the degree of the last remainder equal to zero, we take the last nonzero remainder, $3x^2 + 3x + 4$ and normalize it to obtain the GCD:

$$g(x) = 3^{-1}(3x^2 + 3x + 4) = 2(3x^2 + 3x + 4) = x^2 + x + 3.$$

□

The Euclidean algorithm is established with the help of the following theorems and lemmas.

Theorem 5.4 *If $g = (a, b)$ then there exist integers s and t such that*

$$g = (a, b) = as + bt.$$

For polynomials, if $g(x) = (a(x), b(x))$, then there are polynomials $s(x)$ and $t(x)$ such that

$$g(x) = a(x)s(x) + b(x)t(x).$$

Proof [250] We provide the proof for the integer case; modification for the polynomial case is straightforward.

Consider the linear combinations $as + bt$ where s and t range over all integers. The set of integers $E = \{as + bt, s \in \mathbb{Z}, t \in \mathbb{Z}\}$ contains positive and negative values and 0. Choose s_0 and t_0 so that $as_0 + bt_0$ is the smallest positive integer in the set: $l = as_0 + bt_0 > 0$. We now establish that $l \mid a$; showing that $l \mid b$ is analogous. By the division algorithm, $a = lq + r$ with $0 \leq r < l$. Hence $r = a - ql = a - q(as_0 + bt_0) = a(1 - qs_0) + b(-qt_0)$, so r

itself is in the set E. However, since l is the smallest positive integer in R, r must be 0, so $a = lq$, or $l \mid a$.

Since g is the GCD of a and b, we may write $a = gm$ and $b = gn$ for some integers m and n. Then $l = as_0 + bt_0 = g(ms_0 + nt_0)$, so $g \mid l$. Since it cannot be that $g < l$, since g is the *greatest* common divisor, it must be that $g = l$. □

From the proof of this theorem, we make the following important observation: the greatest common divisor $g = (a, b)$ is the smallest positive integer value of $as + bt$ as s and t range over all integers.

Lemma 5.5 *For any integer n, $(a, b) = (a, b + an)$.*
For any polynomial $n(x) \in F[x]$, $(a(x), b(x)) = (a(x), b(x) + a(x)n(x))$.

Proof Let $d = (a, b)$ and $g = (a, b + an)$. By Theorem 5.4 there exist s_0 and t_0 such that $d = as_0 + bt_0$. Write this as

$$d = a(s_0 - nt_0) + (b + an)t_0 = as_1 + (b + an)t_0.$$

It follows (from Lemma 5.1 part (3)) that $g \mid d$. We now show that $d \mid g$. Since $d \mid a$ and $d \mid b$ we have that $d \mid (an + b)$. Since g is the GCD of a and $an + b$ and any divisor of a and $an + b$ must divide the GCD, it follows that $d \mid g$. Since $d \mid g$ and $g \mid d$, we must have $g = d$.

(For polynomials, the proof is almost exactly the same, except that it is possible that $g(x) = d(x)$ only if both are monic.) □

We demonstrate the use of this theorem and lemma by an example.

Example 5.10 Determine $g = (966, 851)$; this is the same as in Example 5.8, but now we keep track of a few more details. By the division algorithm,

$$966 = 1 \cdot 851 + 115. \tag{5.4}$$

By Lemma 5.5,

$$g = (851, 966) = (851, 966 - 1 \cdot 851) = (851, 115) = (115, 851).$$

Thus the problem has been reduced using the lemma to one having smaller numbers than the original, but with the same GCD. Applying the division algorithm again,

$$851 = 7 \cdot 115 + 46 \tag{5.5}$$

hence, again applying Lemma 5.5,

$$(115, 851) = (115, 851 - 7 \cdot 115) = (115, 46) = (46, 115).$$

Again, the GCD problem is reduced to one with smaller numbers. Proceeding by application of the division algorithm and the property, we obtain successively

$$\begin{aligned} 115 &= 2 \cdot 46 + 23 \\ (46, 115) &= (46, 115 - 2 \cdot 46) = (46, 23) = (23, 46) \\ 46 &= 2 \cdot 23 + 0 \\ (23, 46) &= 23. \end{aligned} \tag{5.6}$$

Chaining together the equalities we obtain

$$(966, 851) = 23.$$

We can find the s and t in the representation suggested by Theorem 5.4,

$$(966, 851) = 966s + 851t,$$

by working the equations backward, substituting in for the remainders from each division in reverse order

$$\begin{aligned} 23 &= 115 - 2 \cdot 46 && \text{"23" from (5.6)} \\ &= 115 - 2 \cdot (851 - 7 \cdot 115) = -2 \cdot 851 + 15 \cdot 115 && \text{"46" from (5.5)} \\ &= -2 \cdot 851 + 15(966 - 1 \cdot 851) = 15 \cdot 966 - 17 \cdot 851 && \text{"115" from (5.4)} \end{aligned}$$

so $s = 15$ and $t = -17$. □

Example 5.11 It can be shown that for the polynomials in Example 5.9,

$$s(x) = 3x^2 + x \qquad t(x) = 2x^3 + 2x + 2.$$

□

Having seen the examples and the basic theory, we can now be a little more precise. In fullest generality, the Euclidean algorithm applies to algebraic structures known as Euclidean domains:

Definition 5.3 [106, p. 301] A **Euclidean domain** is a set D with operations $+$ and $\cdot$ satisfying:

1. D forms a commutative ring with identity. That is, D has an operation $+$ such that $\langle D, + \rangle$ is a commutative group. Also, there is a commutative operation "multiplication," denoted using $\cdot$ (or merely juxtaposition), such that for any a and b in D, $a \cdot b$ is also in D. The distributive property also applies: $a \cdot (b + c) = a \cdot b + a \cdot c$ for any $a, b, c \in D$. Also, there is an element 1, the multiplicative identity, in D such that $a \cdot 1 = 1 \cdot a = a$.
2. Multiplicative cancellation holds: if $ab = cb$ and $b \neq 0$ then $a = c$.
3. Every $a \in D$ has a **valuation** $v(a) : D \to \mathbb{N} \cup \{-\infty\}$ such that:
   (a) $v(a) \geq 0$ for all $a \in D$.
   (b) $v(a) \leq v(ab)$ for all $a, b \in D, b \neq 0$.
   (c) For all $a, b \in D$ with $v(a) > v(b)$ there is a $q \in D$ (quotient) and $r \in D$ (remainder) such that

   $$a = qb + r$$

   with $v(r) < v(b)$ or $r = 0$. $v(b)$ is never $-\infty$ except possibly when $b = 0$. The valuation v is also called a Euclidean function.

□

We have seen two examples of Euclidean domains:

1. The ring of integers under integer addition and multiplication, where the valuation is $v(a) = |a|$ (the absolute value). Then the statement

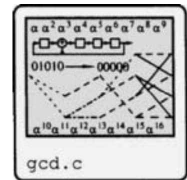
   $$a = qb + r$$

   is obtained simply by integer division with remainder (the division algorithm).

2. Let F be a field. Then $F[x]$ is a Euclidean domain with valuation function $v(a(x)) = \deg(a(x))$ (the degree of the polynomial $a(x) \in F[x]$). It is conventional for this domain to take $v(0) = -\infty$. Then the statement
$$a(x) = q(x)b(x) + r(x)$$
follows from polynomial division.

The Euclidean algorithm can be stated in two versions. The first simply computes the GCD.

Theorem 5.6 *(The Euclidean Algorithm) Let a and b be nonzero elements in a Euclidean domain. Then by repeated application of the division algorithm in the Euclidean domain, we obtain a series of equations:*
$$\begin{aligned}
a &= bq_1 + r_1 & r_1 &\neq 0 \text{ and } v(r_1) < v(b)\\
b &= r_1 q_2 + r_2 & r_2 &\neq 0 \text{ and } v(r_2) < v(r_1)\\
r_1 &= r_2 q_3 + r_3 & r_3 &\neq 0 \text{ and } v(r_3) < v(r_2)\\
&\vdots\\
r_{j-2} &= r_{j-1} q_j + r_j & r_j &\neq 0 \text{ and } v(r_j) < v(r_{j-1})\\
r_{j-1} &= r_j q_{j+1} + 0 & & (r_{j+1} = 0).
\end{aligned}$$
Then $(a, b) = r_j$, the last nonzero remainder of the division process.


gcd.c

That the theorem stops after a finite number of steps follows since every remainder must be smaller (in valuation) than the preceding remainder and the (valuation of the) remainder must be nonnegative. That the final nonzero remainder is the GCD follows from property Lemma 5.5.

This form of the Euclidean algorithm is very simple to code. Let $\lfloor a/b \rfloor$ denote the "quotient" without remainder of a/b, that is, $a = \lfloor a/b \rfloor b + r$. Then recursion in the Euclidean algorithm may be expressed as
$$\begin{aligned}
q_i &= \lfloor r_{i-2}/r_{i-1} \rfloor\\
r_i &= r_{i-2} - r_{i-1} q_i
\end{aligned} \tag{5.7}$$
for $i = 1, 2, \ldots$ (until termination) with $r_{-1} = a$ and $r_0 = b$.

The second version of the Euclidean algorithm, sometimes called the **extended Euclidean algorithm**, computes $g = (a, b)$ and also the coefficients s and t of Theorem 5.4 such that
$$as + bt = g.$$
The values for s and t are computed by finding intermediate quantities s_i and t_i satisfying
$$as_i + bt_i = r_i \tag{5.8}$$
at every step of the algorithm. The formula to update s_i and t_i is (see Exercise 5.18)
$$\begin{aligned}
s_i &= s_{i-2} - q_i s_{i-1}\\
t_i &= t_{i-2} - q_i t_{i-1},
\end{aligned} \tag{5.9}$$
for $i = 1, 2, \ldots$ (until termination), with
$$\begin{aligned}
s_{-1} &= 1 & s_0 &= 0\\
t_{-1} &= 0 & t_0 &= 1.
\end{aligned} \tag{5.10}$$
The Extended Euclidean Algorithm is as shown in Algorithm 5.1.

Algorithm 5.1 Extended Euclidean Algorithm

```
1  Initialization: Set s and t as in (5.10).
2  Let r_{-1} = a, r_0 = b, s_{-1} = 1, s_0 = 0, t_{-1} = 0, t_0 = 1, i = 0
3  while(r_i ≠ 0) {     Repeat until remainder is 0
4     i = i + 1
5     q_i = ⌊r_{i-2}/r_{i-1}⌋     Compute quotient
6     r_i = r_{i-2} - q_i r_{i-1}     Compute remainder
7     s_i = s_{i-2} - q_i s_{i-1}     Compute s and t values
8     t_i = t_{i-2} - q_i t_{i-1}
9  }
10 Return: s = s_{i-1}, t = t_{i-1}, g = r_{i-1}
```

The following are some facts about the GCD which are proved using the Euclidean algorithm. Analogous results hold for polynomials. (It is helpful to verify these properties using small integer examples.)

Lemma 5.7

1. *For integers, (a, b) is the smallest positive value of $as + bt$, where s and t range over all integers.*
2. *If $as + bt = 1$ for some integers s and t, then $(a, b) = 1$; that is, a and b are relatively prime. Thus a and b are relatively prime if and only if there exist s and t such that $as + bt = 1$.*

5.2.3 An Application of the Euclidean Algorithm: The Sugiyama Algorithm

The Euclidean algorithm, besides computing the GCD, has a variety of other applications. Here, the Euclidean algorithm is put to use as a means of solving the problem of finding the shortest LFSR which produces a given output. This problem, as we shall see, is important in decoding BCH and Reed-Solomon codes. (The Berlekamp-Massey algorithm is another way of arriving at this solution.)

We introduce the problem as a prediction problem. Given a set of $2p$ data points $\{b_t, t = 0, 1, \ldots, 2p-1\}$ satisfying the LFSR equation[1]

$$b_k = -\sum_{j=1}^{p} t_j b_{k-j}, \qquad k = p, p+1, \ldots, 2p-1 \tag{5.11}$$

we want to find the coefficients $\{t_j\}$ so that (5.11) is satisfied. That is, we want to find coefficients to predict b_k using prior values. Furthermore, we want the number of nonzero coefficients p to be as small as possible, so that $t(x)$ has the smallest degree possible consistent with (5.11). Equation (5.11) can also be written as

$$\sum_{j=0}^{p} t_j b_{k-j} = 0, \qquad k = p, p+1, \ldots, 2p-1, \tag{5.12}$$

[1]Comparison with (4.17) shows that this equation has a $-$ where (4.17) does not. This is because (4.17) is expressed over $GF(2)$.

where $t_0 = 1$. One way to find the coefficients $\{t_j\}$, given a set of measurements $\{b_j\}$, is to explicitly set up and solve the Toeplitz matrix equation

$$\begin{bmatrix} b_{p-1} & b_{p-2} & b_{p-3} & \cdots & b_0 \\ b_p & b_{p-1} & b_{p-2} & \cdots & b_1 \\ b_{p+1} & b_p & b_{p-1} & \cdots & b_2 \\ \vdots \\ b_{2p-2} & b_{2p-3} & b_{2p-4} & \cdots & b_{p-1} \end{bmatrix} \begin{bmatrix} t_1 \\ t_2 \\ t_3 \\ \vdots \\ t_p \end{bmatrix} = \begin{bmatrix} -b_p \\ -b_{p+1} \\ -b_{p+2} \\ \vdots \\ -b_{2p-1} \end{bmatrix}.$$

There is no guarantee, however, that solution of this set of equations will yield $t(x)$ of shortest degree. The Sugiyama algorithm is an efficient way of solving this equation which guarantees that $t(x)$ has minimal degree. Put another way, the Sugiyama algorithm provides a means of synthesizing LFSR coefficients, given a sequence of its outputs.

The convolution (5.12) can be written in terms of polynomials. Let

$$b(x) = \sum_{i=0}^{2p-1} b_i x^i \quad \text{and} \quad t(x) = 1 + \sum_{i=1}^{p} t_i x^i.$$

Then the condition (5.12) is equivalent to saying that the kth coefficient of the polynomial product $b(x)t(x)$ is equal to zero for $k = p, p+1, \ldots, 2p-1$. Another way of saying this is that

$$b(x)t(x) = r(x) - x^{2p}s(x), \tag{5.13}$$

where $r(x)$ is a polynomial with $\deg(r(x)) < p$ and $x^{2p}s(x)$ is a polynomial whose first term has degree at least $2p$.

Example 5.12 In this example, computations are done in $\mathbb{Z}_5$. The sequence $\{2, 3, 4, 2, 2, 3\}$, corresponding to the polynomial $b(x) = 2 + 3x + 4x^2 + 2x^3 + 2x^4 + 3x^5$, can be generated using the coefficients $t_1 = 3, t_2 = 4, t_3 = 2$, so that $t(x) = 1 + 3x + 4x^2 + 2x^3$. We have $p = 3$. Then in $\mathbb{Z}_5[x]$,

$$b(x)t(x) = 2 + 4x + x^2 + x^6 + x^7 + x^8 = (2 + 4x + x^2) + x^6(1 + x + x^2). \tag{5.14}$$

Note that the terms x^3, x^4 and x^5 are missing. We identify

$$r(x) = 2 + 4x + x^2 \qquad s(x) = -(1 + x + x^2).$$

□

Equation (5.13) can be written as

$$x^{2p}s(x) + b(x)t(x) = r(x). \tag{5.15}$$

The problem can now be stated as: given a sequence of $2p$ observations $\{b_0, b_1, b_2, \ldots, b_{2p-1}\}$ and its corresponding polynomial representation $b(x)$, find a solution to (5.15). When stated this way, the problem appears underdetermined: all we know is $b(x)$ and p. However, the Euclidean algorithm provides a solution, under the constraints that $\deg(t(x)) \leq p$ and $\deg(r(x)) < p$. We start the Euclidean algorithm with $r_{-1}(x) = x^{2p}$ and $r_0(x) = b(x)$. The algorithm iterates until the first i such that

$$\deg(r_i(x)) < p.$$

Then by the definition of the Euclidean algorithm, it must be the case that the $s_i(x)$ and $t_i(x)$ solve (5.15). The algorithm then concludes by normalizing $t_i(x)$ so that the constant term is 1. While we don't prove this here, it can be shown that this procedure will find a solution minimizing the degree of $t(x)$.

Example 5.13 Given the sequence $\{2, 3, 4, 2, 2, 3\}$, where the coefficients are in $\mathbb{Z}_5$, calling the gcd function with $a(x) = x^6$ and $b(x) = 2 + 3x + 4x^2 + 2x^3 + 2x^4 + 3x^5$ results after three iterations in

$$r_i(x) = 3 + x + 4x^2 \qquad s_i(x) = 1 + x + x^2 \qquad t_i(x) = 4 + 2x + x^2 + 3x^3.$$

Normalizing $t_i(x)$ by scaling by $4^{-1} = 4$ we find

$$t(x) = 1 + 3x + 4x^2 + 2x^3$$

$$r(x) = 2 + 4x + x^2$$

$$s(x) = 4 + 4x + 4x^2 = -(1 + x + x^2).$$

These correspond to the polynomials in (5.14). □

One of the useful attributes of the Sugiyama algorithm is that it determines the coefficients $\{t_1, \dots, t_p\}$ satisfying (5.12) with the *smallest* value of p. Put another way, it determines the $t(x)$ of smallest degree satisfying (5.13).

Example 5.14 To see this, consider the sequence $\{2, 3, 2, 3, 2, 3\}$. This can be generated by the polynomial $t_1(x) = 1 + 3x + 4x^2 + 2x^3$, since

$$b(x)t_1(x) = 2 + 4x + 4x^2 + 3x^6 + x^7 + x^8 = (2 + 4x + 4x^2) + x^6(3 + x + x^2).$$

However, as a result of calling the Sugiyama algorithm, we obtain the polynomial

$$t(x) = 1 + x,$$

so

$$b(x)t(x) = 2 + 3x^6.$$

It may be observed (in retrospect) that the sequence of coefficients in b happen to satisfy $b_k = -b_{k-1}$, consistent with the $t(x)$ obtained. □

5.2.4 Congruence

Operations modulo an integer are fairly familiar. We frequently deal with operations on a clock modulo 24, "If it is 10:00 now, then in 25 hours it will be 11:00," or on a week modulo 7, "If it is Tuesday, then in eight days it will be Wednesday." The concept of congruence provides a notation to capture the idea of modulo operations.

Definition 5.4 If an integer $m \neq 0$ divides $a - b$, then we say that a is **congruent** to b modulo m and write $a \equiv b \pmod{m}$. If a polynomial $m(x) \neq 0$ divides $a(x) - b(x)$, then we say that $a(x)$ is congruent to $b(x)$ modulo $m(x)$ and write $a(x) \equiv b(x) \pmod{m(x)}$.

In summary:

$$\boxed{a \equiv b \pmod{m} \text{ if and only if } m \mid (a - b).} \tag{5.16}$$

□

Example 5.15

1. $7 \equiv 20 \pmod{13}$.
2. $7 \equiv -6 \pmod{13}$.

□

Congruences have the following basic properties.

Theorem 5.8 *[250, Theorem 2.1, Theorem 2.3, Theorem 2.4] For integers* a, b, c, d, x, y, m*:*

1. $a \equiv b \pmod{m} \Leftrightarrow b \equiv a \pmod{m} \Leftrightarrow b - a \equiv 0 \pmod{m}$.
2. *If* $a \equiv b \pmod{m}$ *and* $b \equiv c \pmod{m}$ *then* $a \equiv c \pmod{m}$.
3. *If* $a \equiv b \pmod{m}$ *and* $c \equiv d \pmod{m}$ *then* $ax + cy \equiv bx + dy \pmod{m}$.
4. *If* $a \equiv b \pmod{m}$ *and* $c \equiv d \pmod{m}$ *then* $ac \equiv bd \pmod{m}$. *From this it follows that if* $a \equiv b \pmod{m}$ *then* $a^n \equiv b^n \pmod{m}$.
5. *If* $a \equiv b \pmod{m}$ *and* $d \mid m$ *and* $d > 0$ *then* $a \equiv b \pmod{d}$.
6. *If* $a \equiv b \pmod{m}$ *then for* $c > 0$, $ac \equiv bc \pmod{mc}$.
7. $ax \equiv ay \pmod{m}$ *if and only if* $x \equiv y \pmod{m/(a, m)}$.
8. *If* $ax \equiv ay \pmod{m}$ *and* $(a, m) = 1$ *then* $x \equiv y \pmod{m}$.
9. *If* $a \equiv b \pmod{m}$ *then* $(a, m) = (b, m)$.

From the definition, we note that if $n \mid a$, then $a \equiv 0 \pmod{n}$.

5.2.5 The ϕ Function

Definition 5.5 The **Euler totient function** $\phi(n)$ is the number of positive integers less than n that are relatively prime to n. This is also called the **Euler** ϕ function, or sometimes just the ϕ function. □

Example 5.16

1. $\phi(5) = 4$ (the numbers 1,2,3,4 are relatively prime to 5).
2. $\phi(4) = 2$ (the numbers 1 and 3 are relatively prime to 4).
3. $\phi(6) = 2$ (the numbers 1 and 5 are relatively prime to 6).

□

It can be shown that the ϕ function can be written as

$$\phi(n) = n \prod_{p \mid n} \left(1 - \frac{1}{p}\right) = n \prod_{p \mid n} \frac{p-1}{p},$$

where the product is taken over all primes p dividing n.

Example 5.17

$$\phi(189) = \phi(3 \cdot 3 \cdot 3 \cdot 7) = 189(1 - 1/3)(1 - 1/7) = 108.$$
$$\phi(64) = \phi(2^6) = 64(1 - 1/2) = 32.$$

□

We observe that:

1. $\phi(p) = p - 1$ if p is prime.
2. For distinct primes p_1 and p_2,

$$\phi(p_1 p_2) = (p_1 - 1)(p_2 - 1). \tag{5.17}$$

3. $\phi(p^m) = p^{m-1}(p-1)$ for p prime.
4. $\phi(p^m q^n) = p^{m-1}q^{n-1}(p-1)(q-1)$ for distinct primes p and q.
5. For positive integers m and n with $(m, n) = 1$,

$$\phi(mn) = \phi(m)\phi(n). \tag{5.18}$$

5.2.6 Some Cryptographic Payoff

With all the effort so far introducing number theory, it is interesting to put it to work on a problem of practical interest: public key cryptography using the RSA algorithm. This is really a topic distinct from error correction coding, but the application is important in modern communication and serves to motivate some of these theoretical ideas.

In a symmetric public key encryption system, a user B has a private "key" which is only known to B and a public "key" which may be known to any interested party, C. A message encrypted by one key (either the public or private) can be decrypted by the other.

For example, if C wants to send a sealed letter so that only B can read it, C encrypts using B's public key. Upon reception, B can read it by deciphering using his private key. Or, if B wants to send a letter that is known to come from only him, B encrypts with his private key. Upon receipt, C can successfully decrypt only using B's public key.

Public key encryption relies upon a "trapdoor": an operation which is exceedingly difficult to compute unless some secret information is available. For the RSA encryption algorithm, the secret information is number theoretic: it relies upon the difficulty of factoring very large integers.

Fermat's Little Theorem

Theorem 5.9

1. *(Fermat's little theorem)*[2] *If p is a prime and if a is an integer such that $(a, p) = 1$ (i.e., p does not divide a), then p divides $a^{p-1} - 1$. Stated another way, if $a \not\equiv 0 \pmod{p}$,*

$$a^{p-1} \equiv 1 \pmod{p}.$$

2. *(Euler's generalization of Fermat's little theorem) If n and a are integers such that $(a, n) = 1$, then*

$$a^{\phi(n)} \equiv 1 \pmod{n},$$

where ϕ is the Euler ϕ function. For any prime p, $\phi(p) = p - 1$ and we get Fermat's little theorem.

Example 5.18

1. Let $p = 7$ and $a = 2$. Then $a^p - 1 = 63$ and $p \mid 2^6 - 1$.
2. Compute the remainder of 8^{103} when divided by 13. Note that

$$103 = 8 \cdot 12 + 7.$$

Then with all computations modulo 13,

$$8^{103} = (8^{12})^8(8^7) \equiv (1^8)(8^7) \equiv (-5)^7 \equiv (-5)^6(-5) \equiv (25)^3(-5) \equiv (-1)^3(-5) \equiv 5.$$

□

Proof of Theorem 5.9.

[2] Fermat's little theorem should not be confused with "Fermat's last theorem," proved by A. Wiles, which states that $x^n + y^n = z^n$ has no solution over the integers if $n > 2$.

1. The nonzero elements in the group $\mathbb{Z}_p$, $\{1, 2, \ldots, p-1\}$ form a group of order $p-1$ under multiplication. By Lagrange's theorem (Theorem 2.3), the order of any element in a group divides the order of the group, so for $a \in \mathbb{Z}_p$ with $a \neq 0$, $a^{p-1} = 1$ in $\mathbb{Z}_p$. If $a \in \mathbb{Z}$ and $a \notin \mathbb{Z}_p$, write $a = (\tilde{a} + kp)$ for some $k \in \mathbb{Z}$ and for $0 \leq \tilde{a} < p$, then reduce modulo p.
2. Let G_n be the set of elements in $\mathbb{Z}_n$ that are relatively prime to n. Then (it can be shown that) G_n forms a group under multiplication. Note that the group G_n has $\phi(n)$ elements in it. Now let $a \in \mathbb{Z}_n$ be relatively prime to n. Then a is in the group G_n. Since the order of an element divides the order of the group, we have $a^{\phi(n)} \equiv 1 \pmod{n}$. If $a \notin \mathbb{Z}_n$, write $a = (\tilde{a} + kn)$ where $\tilde{a} \in \mathbb{Z}_n$. Then reduce modulo n.

□

RSA Encryption

Named after its inventors, Ron Rivest, Adi Shamir, and Leonard Adleman [293], the RSA encryption algorithm gets its security from the difficulty of factoring *large* numbers. The steps in setting up the system are:

- Choose two distinct random prime numbers p and q (of roughly equal length for best security) and compute $n = pq$. Note that $\phi(n) = (p-1)(q-1)$.
- Randomly choose an encryption key e, an integer e such that the GCD $(e, (p-1)(q-1)) = 1$. By the extended Euclidean algorithm, there are numbers d and f such that $de - f(p-1)(q-1) = 1$, or
$$de = 1 + (p-1)(q-1)f.$$
That is, $d = e^{-1} \pmod{(p-1)(q-1)}$.
- Publish the pair of numbers $\{e, n\}$ as the public key. Retain the pair of numbers $\{d, n\}$ as the private key. The factors p and q of n are never disclosed.

To **encrypt** (say, using the public key), break the message m (as a sequence of numbers) into blocks m_i of length less than the length of n. Furthermore, assume that $(m_i, n) = 1$ (which is highly probable, since n has only two factors). For each block m_i, compute the encrypted block c_i by

$$c_i = m_i^e \pmod{n}.$$

(If $e < 0$, then find the inverse modulo n.) To **decrypt** (using the corresponding private key) compute

$$c_i^d \,(\text{mod } n) = m_i^{de} \,(\text{mod } n) = m_i^{f(p-1)(q-1)+1} \,(\text{mod } n) = m_i m_i^{f(p-1)(q-1)} \,(\text{mod } n).$$

Since $(m_i, n) = 1$,

$$m_i^{f(p-1)(q-1)} = (m_i^f)^{(p-1)(q-1)} = (m_i^f)^{\phi(n)} \equiv 1 \pmod{n},$$

so that

$$c_i^d \,(\text{mod } n) = m_i,$$

as desired.

To crack this, a person knowing n and e would have to factor n to find d. While multiplication (and computing powers) is easy and straightforward, factoring very large integers is very difficult.

Example 5.19 Let $p = 47$ and $q = 71$ and $n = 3337$. (This is clearly too short to be of cryptographic value.) Then

$$(p-1)(q-1) = 3220.$$

The encryption key e must be relatively prime to 3220; take $e = 79$. Then we find by the Euclidean algorithm that $d = 1019$. The public key is (79, 3337). The private key is (1019, 3337).

To encode the message block $m_1 = 688$, we compute

$$c_1 = (688)^{79} \pmod{3337} = 1570.$$

To decrypt this, exponentiate

$$c_1^d = (1570)^{1019} \pmod{3337} = 688.$$

□

In practical applications, primes p and q are usually chosen to be at least several hundred digits long. This makes factoring $n = pq$ exceedingly difficult!

5.3 The Chinese Remainder Theorem

The material from this section is not used until Section 7.4.2. In its simplest interpretation, the Chinese Remainder Theorem (CRT) is a method for finding the *simultaneous* solution to the set of congruences

$$x \equiv a_1 \pmod{m_1} \qquad x \equiv a_2 \pmod{m_2} \qquad \ldots \qquad x \equiv a_r \pmod{m_r}. \tag{5.19}$$

However, the CRT applies not only to integers, but to other Euclidean domains, including rings of polynomials. The CRT provides an interesting isomorphism between rings which is useful in some decoding algorithms.

One approach to the solution of (5.19) would be to find the solution *set* to each congruence separately, then determine if there is a point in the intersection of these sets. The following theorem provides a more constructive solution.

Theorem 5.10 *If $m_1, m_2, \ldots, m_r$ are pairwise relatively prime elements with positive valuation in a Euclidean domain R, and $a_1, a_2, \ldots, a_r$ are any elements in the Euclidean domain, then the set of congruences in (5.19) have common solutions. Let $m = m_1 m_2 \cdots m_r$. If x_0 is a solution, then so is $x = x_0 + km$ for any $k \in R$.*

Proof Let $m = \prod_{j=1}^{r} m_j$. Observe that $(m/m_j, m_j) = 1$ since the m_is are relatively prime. By Theorem 5.4, there are unique elements s and t such that

$$(m/m_j)s + m_j t = 1,$$

which is to say that

$$(m/m_j)s \equiv 1 \pmod{m_j}.$$

Let $b_j = s$ in this expression, so that we can write

$$(m/m_j)b_j \equiv 1 \pmod{m_j}. \tag{5.20}$$

Also $(m/m_j)b_j \equiv 0 \pmod{m_i}$ if $i \neq j$ since $m_i \mid m$. Let

$$x_0 = \sum_{j=1}^{r} (m/m_j) b_j a_j. \tag{5.21}$$

Then

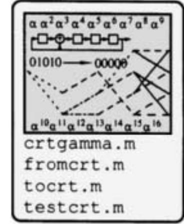
$$x_0 \equiv (m/m_i) b_i a_i \equiv a_i \pmod{m_i}.$$

Uniqueness is straightforward to verify, as is the fact that $x = x_0 + km$ is another solution. □

It is convenient to introduce the notation $M_i = m/m_i$. The solution (5.21) can be written as $x_0 = \sum_{j=1}^{r} \gamma_j a_j$, where

$$\gamma_j = \frac{m}{m_j} b_j = M_j b_j \tag{5.22}$$

with b_j determined by the solution to (5.20). Observe that γ_j depends only upon the set of moduli $\{m_i\}$ and not upon x. If the γ_js are precomputed, then the synthesis of x from the $\{a_i\}$ is a simple inner product.



Example 5.20 Find a solution x to the set of congruences

$$x \equiv 0 \pmod{4} \qquad x \equiv 2 \pmod{27} \qquad x \equiv 3 \pmod{25}.$$

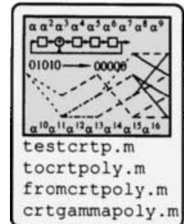
Since the moduli m_i are powers of distinct primes, they are pairwise relative prime. Then $m = m_1 m_2 m_3 = 2700$. Using the Euclidean algorithm it is straightforward to show that $(m/4)b_1 \equiv 1 \pmod{4}$ has solution $b_1 = -1$. Similarly

$$b_2 = 10 \qquad b_3 = -3.$$

The solution to the congruences is given by

$$x = (m/4)(-1)(0) + (m/27)(10)(2) + (m/25)(-3)(3) = 1028.$$

□



Example 5.21 Suppose the Euclidean domain is $\mathbb{R}[x]$ and the polynomials are

$$m_1(x) = (x-1) \qquad m_2(x) = (x-2)^2 \qquad m_3(x) = (x-3)^3.$$

These are clearly pairwise relatively prime. We find that

$$m(x) = m_1(x)m_2(x)m_3(x) = x^6 - 14x^5 + 80x^4 - 238x^3 + 387x^2 - 324x + 108.$$

If

$$f(x) = x^5 + 4x^4 + 5x^3 + 2x^2 + 3x + 2,$$

then we obtain

$$a_1(x) = 17 \qquad a_2(x) = 279x - 406 \qquad a_3(x) = 533x^2 - 2211x + 2567.$$

□

The CRT provides a means of representing integers in the range $0 \le x < m$, where $m = m_1 m_2 \cdots m_r$ and the m_i are pairwise relatively prime. Let $R/\langle m \rangle$ denote the ring of integers modulo m and let $R/\langle m_i \rangle$ denote the ring of integers modulo m_i. Given a number $x \in R/\langle m \rangle$, it can be decomposed into an r-tuple $[x_1, x_2, \ldots, x_r]$ by

$$x \equiv x_i \pmod{m_i}, \qquad i = 1, 2, \ldots, r,$$

where $x_i \in R/\langle m_i \rangle$. Going the other way, an r-tuple of numbers $[x_1, x_2, \ldots, x_r]$ with $0 \le x_i < m_i$ can be converted into the number x they represent using (5.21). If we let $\underline{x} = [x_1, x_2, \ldots, x_r]$, then the correspondence between a number x and its representation using the CRT can be represented as

$$x \leftrightarrow \underline{x}.$$

We also denote this as

$$\underline{x} = \text{CRT}(x) \qquad x = \text{CRT}^{-1}(\underline{x}).$$

Ring operations can be equivalently computed in the original ring $R/\langle m\rangle$, or in each of the rings $R/\langle m_i\rangle$ separately.

Example 5.22 Let $m_1 = 4, m_2 = 27, m_3 = 25$. Let $x = 25$; then $\underline{x} = [1, 25, 0]$. Let $y = 37$; then $\underline{y} = [1, 10, 12]$. The sum $z = x + y = 62$ has $\underline{z} = [2, 8, 12]$, which represents $\underline{x} + \underline{y}$, added element by element, with the first component modulo 4, the second component modulo 27, and the third component modulo 25.

The product $z = x \cdot y = 925$ has $\underline{z} = [1, 7, 0]$, corresponding to the element-by-element product (modulo 4, 27, and 25, respectively). □

More generally, we have a ring isomorphism by the CRT. Let $\pi_i : R/\langle m\rangle \to R/\langle m_i\rangle, i = 1, 2, \ldots, r$ denote the ring homomorphism defined by $\pi_i(a) = a \pmod{m_i}$. We define the homomorphism $\chi : R/\langle m\rangle \to R/m_1 \times R/m_2 \times \cdots \times R/m_r$ by $\chi = \pi_1 \times \pi_2 \times \cdots \times \pi_r$, that is,

$$\chi(a) = (a \bmod m_1, a \bmod m_2, \ldots, a \bmod m_r) \tag{5.23}$$

Then χ defines a ring isomorphism: both the additive and multiplicative structure of the ring are preserved, and the mapping is bijective.

Example 5.23 Using the same polynomials as in Example 5.21, let

$$f_1(x) = x^5 + 4x^4 + 5x^3 + 2x^2 + 3x + 2.$$

Then using the CRT,

$$f_1(x) \leftrightarrow (17,\ 279x - 406,\ 533x^2 - 2211 + 2567) = \underline{f}_1(x).$$

Also let

$$f_2(x) = x^3 + 2x^2 \leftrightarrow (3,\ 20x - 24,\ 11x^2 - 27x + 27) = \underline{f}_2(x).$$

Then

$$f_1(x) + f_2(x) = x^5 + 4x^4 + 6x^3 + 4x^2 + 3x + 2 \leftrightarrow (20,\ 299x - 430,\ 544x^2 - 2238x + 2594).$$

□

5.3.1 The CRT and Interpolation

The Evaluation Homomorphism

Let $\mathbb{F}$ be a field and let $R = \mathbb{F}[x]$. Let $f(x) \in R$ and let $m_1(x) = x - u_1$. Then computing the remainder of $f(x)$ modulo $x - u_1$ gives exactly $f(u_1)$.

Example 5.24 Let $f(x) = x^4 + 3x^3 + 2x^2 + 4 \in \mathbb{R}[x]$ and let $m_1(x) = x - 3$. Then computing $f(x)/m_1(x)$ by long division, we obtain the quotient and remainder

$$f(x) = (x - 3)(x^3 + 6x^2 + 20x + 60) + 184.$$

But we also find that

$$f(3) = 184.$$

So $f(x) \bmod (x - 3) = 184 = f(3)$. □

Thus we can write

$$f(x) \bmod (x-u) = f(u).$$

The mapping $\pi_i : \mathbb{F}[x] \to \mathbb{F}$ defined by

$$\pi_i(f(x)) = f(u_i) = f(x) \pmod{(x-u_i)}$$

is called the **evaluation homomorphism**. It can be shown that it is, in fact, a homomorphism: for two polynomials $f(x), g(x) \in \mathbb{F}[x]$,

$$\pi_i(f(x)+g(x)) = \pi_i(f(x)) + \pi_i(g(x)) \qquad \pi_i(f(x)g(x)) = \pi_i(f(x))\pi_i(g(x)).$$

The Interpolation Problem

Suppose we are given the following problem: Given a set of points (u_i, a_i), $i = 1, 2, \ldots, r$, determine an **interpolating** polynomial $f(x) \in \mathbb{F}[x]$ of degree $< r$ such that

$$f(u_1) = a_1, f(u_2) = a_2, \ldots, f(u_r) = a_r. \tag{5.24}$$

Now let $f(x) = f_0 + f_1 x + \cdots + f_{r-1}x^{r-1}$. Since $\deg(f(x)) < r$, we can think of $f(x)$ as being in $\mathbb{F}[x]/(x^r - 1)$. Also let $m_i(x) = x - u_i \in \mathbb{F}[x]$ for $i = 1, 2, \ldots, r$ where the $u_1, u_2, \ldots, u_r \in F$ are pairwise *distinct*. Then the $m_i(x)$ are pairwise relatively prime.

By the evaluation homomorphism, the set of constraints (5.19) can be expressed as

$$f(x) \equiv a_1 \pmod{m_1(x)} \qquad f(x) \equiv a_2 \pmod{m_2(x)} \qquad f(x) \equiv a_r \pmod{m_r(x)}.$$

So solving the interpolation problem simply becomes an instance of solving a Chinese Remainder problem.

The interpolating polynomial is found using the CRT. Let $m(x) = \prod_{i=1}^{r}(x - u_i)$. By the proof of Theorem 5.10, we need functions $b_j(x)$ such that

$$(m(x)/m_j(x))b_j(x) \equiv 1 \pmod{m_j}$$

and

$$(m(x)/m_j(x))b_j(x) \equiv 0 \pmod{m_k}.$$

That is,

$$\left[\prod_{i=1, i\neq j}^{r} (x-u_i)\right] b_j(x) \equiv 1 \pmod{m_j},$$

and

$$\left[\prod_{i=1, i\neq j}^{r} (x-u_i)\right] b_j(x) \equiv 0 \pmod{m_k}$$

for $k \neq j$. Let

$$b_j(x) = \prod_{i=1, i\neq j}^{r} \frac{1}{(u_j - u_i)}$$

and let

$$l_j(x) = (m(x)/m_j(x))b_j(x) = \prod_{i=1, i\neq j}^{r} \frac{(x-u_i)}{(u_j-u_i)}. \tag{5.25}$$

Since

$$l_j(u_j) = 1 \quad \text{and} \quad l_j(u_k) = 0, \quad j \neq k,$$

we see that $b_j(x)$ satisfies the necessary requirements. By (5.21), the interpolating polynomial is then simply

$$f(x) = \sum_{j=1}^{r} (m(x)/m_j(x)) b_j(x) a_j = \sum_{j=1}^{r} a_j l_j(x). \tag{5.26}$$

This form of an interpolating polynomial is called a *Lagrange interpolator.* The basis functions $l_i(x)$ are called *Lagrange interpolants.* By the CRT, this interpolating polynomial is unique modulo $m(x)$.

The Lagrange interpolator can be expressed in another convenient form. Let

$$m(x) = \prod_{i=1}^{r} (x - u_i).$$

Then the derivative[3] is

$$m'(x) = \sum_{k=1}^{r} \prod_{i \neq k} (x - u_i)$$

so that

$$m'(u_j) = \prod_{i \neq j} (u_j - u_i).$$

(See also Definition 6.5.) The interpolation formula (5.26) can now be written as

$$f(x) = \sum_{i=1}^{r} a_i \frac{m(x)}{(x - u_i)} \frac{1}{m'(u_i)}. \tag{5.27}$$

Example 5.25 An important instance of interpolation is the discrete Fourier transform (DFT). Let $f(x) = f_0 + f_1 x + \cdots + f_{N-1} x^{N-1}$, with $x = z^{-1}$, be the Z-transform of a complex cyclic sequence. Then $f(x) \in \mathbb{C}[x]/(x^N - 1)$, since it is a polynomial of degree $\leq N - 1$. Let $m(x) = x^N - 1$. The N roots of $m(x)$ are the complex numbers $e^{-ij2\pi/N}$, $i = 0, 1, \ldots, N - 1$. Let $\omega = e^{-j2\pi/N}$; this is a primitive N-th root of unity. Then the factorization of $m(x)$ can be written as

$$m(x) = x^N - 1 = \prod_{i=0}^{N-1} (x - \omega^i),$$

where the factors are pairwise relative prime. Define the evaluations

$$F_k = \pi_k(f(x)) = f(\omega^k) = \sum_{i=0}^{N-1} f_i \omega^{ik}, \qquad k = 0, 1, \ldots, N - 1.$$

Expressed another way, $F_k = f(x) \pmod{x - \omega^k}$. The coefficients $\{f_k\}$ may be thought of as existing in a "time domain," while the coefficients $\{F_k\}$ may be thought of as existing in a "frequency domain."

For functions in the "time domain," multiplication is polynomial multiplication (modulo $x^N - 1$). That is, for polynomials $f(x)$ and $g(x)$, multiplication is $f(x)g(x) \pmod{x^N - 1}$, which amounts to cyclic convolution.

[3] Or formal derivative, if the field of operations is not real or complex

For functions in the "transform domain," multiplication is element by element. That is, for sequences $(F_0, F_1, \ldots, F_{N-1})$ and $(G_0, G_1, \ldots, G_{N-1})$, multiplication is element by element as complex numbers:

$$(F_0G_0, F_1G_1, \ldots, F_{N-1}G_{N-1}).$$

Thus, the ring isomorphism validates the statement: (cyclic) convolution in the time domain is equivalent to multiplication in the frequency domain.

□

5.4 Fields

Fields were introduced in Section 2.3. We review the basic requirements here, in comparison with a ring. In a ring, not every element has a multiplicative inverse. In a field, the familiar arithmetic operations that take place in the usual real numbers are all available: $\langle F, +\rangle$ is an Abelian group. (Denote the additive identity element by 0.) The set $F \setminus \{0\}$ (the set F with the additive identity removed) forms a **commutative** group under multiplication. Denote the multiplicative identity element by 1. Finally, as in a ring the operations $+$ and $\cdot$ distribute: $a \cdot (b + c) = a \cdot b + a \cdot c$ for all $a, b, c \in F$.

In a field, all the elements except the additive identity form a group, whereas in a ring, there may not even be a multiplicative identity, let alone an inverse for every element. Every field is a ring, but not every ring is a field.

Example 5.26 $\langle \mathbb{Z}_5, +, \cdot\rangle$ forms a field; every nonzero element has a multiplicative inverse. So this set forms not only a ring but also a group. Since this field has only a finite number of elements in it, it is said to be a **finite field**.

However, $\langle \mathbb{Z}_6, +, \cdot\rangle$ does not form a field, since not every element has a multiplicative inverse. □

One way to obtain finite fields is described in the following.

Theorem 5.11 *The ring $\langle \mathbb{Z}_p, +, \cdot\rangle$ is a field if and only if p is a prime.*

Before proving this, we need the following definition and lemma.

Definition 5.6 In a ring R, if $a, b \in R$ with both a and b not equal to zero but $ab = 0$, then a and b are said to be **zero divisors**. □

Lemma 5.12 *In a ring $\mathbb{Z}_n$, the zero divisors are precisely those elements that are not relatively prime to n.*

Proof Let $a \in \mathbb{Z}_n$ be not equal to 0 and be not relatively prime to n. Let d be the greatest common divisor of n and a. Then $a(n/d) = (a/d)n$, which, being a multiple of n, is equal to 0 in $\mathbb{Z}_n$. We have thus found a number $b = n/d$ such that $ab = 0$ in $\mathbb{Z}_n$, so a is a zero divisor in $\mathbb{Z}_n$.

Conversely, suppose that there is an $a \in \mathbb{Z}_n$ relatively prime to n such that $ab = 0$. Then it must be the case that

$$ab = kn$$

for some integer k. Since n has no factors in common with a, then it must divide b, which means that $b = 0$ in $\mathbb{Z}_n$. □

Observe from this lemma that if p is a prime, there are *no* divisors of 0 in $\mathbb{Z}_p$. We now turn to the proof of Theorem 5.11.

Proof of Theorem 5.11.

We have already shown that if p is not prime, then there are zero divisors and hence $\langle \mathbb{Z}_p, +, \cdot \rangle$ cannot form a field. Let us now show that if p is prime, $\langle \mathbb{Z}_p, +, \cdot \rangle$ is a field.

We have already established that $\langle \mathbb{Z}_p, + \rangle$ is a group. The key remaining requirement is to establish that $\langle \mathbb{Z}_p \backslash \{0\}, \cdot \rangle$ forms a group. The multiplicative identity is 1 and multiplication is commutative. The key remaining requirement is to establish that every nonzero element in $\mathbb{Z}_p$ has a multiplicative inverse.

Let $\{1, 2, \ldots, p-1\}$ be a list of the nonzero elements in $\mathbb{Z}_p$, and let $a \in \mathbb{Z}_p$ be nonzero. Form the list

$$\{1a, 2a, \ldots, (p-1)a\}. \tag{5.28}$$

Every element in this list is distinct, since if any two were identical, say $ma = na$ with $m \neq n$, then $a(m-n) = 0$, which is impossible since there are no zero divisors in $\mathbb{Z}_p$. Thus the list (5.28) contains all nonzero elements in $\mathbb{Z}_p$ and is a permutation of the original list. Since 1 is in the original list, it must appear in the list in (5.28). □

5.4.1 An Examination of $\mathbb{R}$ and $\mathbb{C}$

Besides the finite fields $\langle \mathbb{Z}_p, +, \cdot \rangle$ with p prime, there are other finite fields. These fields are *extension fields* of $\mathbb{Z}_p$. However, before introducing them, it is instructive to take a look at how the field of complex numbers $\mathbb{C}$ can be constructed as a field extension from the field of real numbers $\mathbb{R}$

Recall that there are several representations for complex numbers. Sometimes it is convenient to use a "vector" notation, in which a complex number is represented as (a, b). Sometimes it is convenient to use a "polynomial" notation $a + bi$, where i is taken to be a root of the polynomial $x^2 + 1$. However, since there is some preconception about the meaning of the symbol i, we replace it with the symbol α, which doesn't carry the same connotations (yet). In particular, α is not (yet) the symbol for $\sqrt{-1}$. You may think of $a + b\alpha$ as being a polynomial of degree ≤ 1 in the "indeterminate" α. There is also a polar notation for complex numbers, in which the complex number is written as $a + ib = re^{i\theta}$ for the appropriate r and θ. Despite the differences in notation, it should be borne in mind that they all represent the same number.

Given two complex numbers we *define* the addition component-by-component in the vector notation (a, b) and (c, d), where a, b, c and d are all in $\mathbb{R}$, based on the addition operation of the underlying field $\mathbb{R}$. The set of complex number thus forms a two-dimensional *vector space* of real numbers. We define

$$(a, b) + (c, d) = (a + c, b + d). \tag{5.29}$$

It is straightforward to show that this addition operation satisfies the group properties for addition, based on the group properties it inherits from $\mathbb{R}$.

Now consider the "polynomial notation." Using the conventional rules for adding polynomials, we obtain

$$a + b\alpha + c + d\alpha = (a + c) + (b + d)\alpha,$$

which is equivalent to (5.29).

How, then, to define multiplication in such a way that all the field requirements are satisfied? If we simply multiply using the conventional rules for polynomial multiplication,

$$(a + b\alpha)(c + d\alpha) = ac + (ad + bc)\alpha + bd\alpha^2, \tag{5.30}$$

we obtain a quadratic polynomial, whereas complex numbers are represented as polynomials having degree ≤ 1 in the variable α.

Polynomial multiplication must be followed by another step, computing the *remainder* modulo some other polynomial. Let us pick the polynomial

$$g(\alpha) = 1 + \alpha^2$$

to divide by. Dividing the product in (5.30) by $g(\alpha)$

$$\begin{array}{r|l} & bd \\ \cline{2-2} \alpha^2 + 1 & bd\alpha^2 + (ad + bc)\alpha + ac \\ & \underline{bd\alpha^2 + \qquad\qquad\qquad bd} \\ & \qquad (ad + bc)\alpha + ac - bd \end{array}$$

we obtain the remainder $(ac - bd) + (ad + bc)\alpha$. Summarizing this, we *define* the product of $(a + b\alpha)$ by $(c + d\alpha)$ by the following two steps:

1. Multiply $(a + b\alpha)$ by $(c + d\alpha)$ as polynomials.
2. Compute the remainder of this product when divided by $g(\alpha) = \alpha^2 + 1$.

That is, the multiplication is defined in the ring $\mathbb{R}[\alpha]/g(\alpha)$, as described in Section 4.4.

Of course, having established the pattern, it is not necessary to carry out the actual polynomial arithmetic: by this two-step procedure we have obtained the familiar formula

$$(a + b\alpha) \cdot (c + d\alpha) = (ac - bd) + (ad + bc)\alpha$$

or, in vector form,

$$(a, b) \cdot (c, d) = (ac - bd, ad + bc).$$

As an important example, suppose we want to multiply the complex numbers (in vector form) (0, 1) times (0, 1), or (in polynomial form) α times α. Going through the steps of computing the product and the remainder we find

$$\alpha \cdot \alpha = -1. \tag{5.31}$$

In other words, in the arithmetic that we have defined, the element α satisfies the equation

$$\alpha^2 + 1 = 0. \tag{5.32}$$

In other words, the indeterminate α *acts like the number* $\sqrt{-1}$. This is a result of the fact that multiplication is computed modulo the polynomial $g(\alpha) = \alpha^2 + 1$: the symbol α is (now by construction) a root of the polynomial $g(x)$. To put it another way, the remainder of a polynomial $\alpha^2 + 1$ divided by $\alpha^2 + 1$ is exactly 0. So, by this procedure, any time $\alpha^2 + 1$ appears in any computation, it may be replaced with 0.

Let us take another look at the polynomial multiplication in (5.30):

$$(a + b\alpha)(c + d\alpha) = ac + (ad + bc)\alpha + bd\alpha^2. \tag{5.33}$$

Using (5.31), we can replace α^2 in (5.33) wherever it appears with expressions involving lower powers of α. We thus obtain

$$(a + b\alpha)(c + d\alpha) = ac + (ad + bc)\alpha + bd(-1) = (ac - bd) + (ad + bc)\alpha,$$

as expected. If we had an expression involving α^3 it could be similarly simplified and expressed in terms of lower powers of α:

$$\alpha^3 = \alpha \cdot \alpha^2 = \alpha \cdot (-1) = -\alpha.$$

Using the addition and multiplication as defined, it is (more or less) straightforward to show that we have created a field which is, in fact, the field of complex numbers $\mathbb{C}$.

As is explored in the exercises, it is important that the polynomial $g(\alpha)$ used to define the multiplication operation *not* have roots in the base field $\mathbb{R}$. If $g(\alpha)$ were a polynomial so that $g(b) = 0$ for some $b \in \mathbb{R}$, then the multiplication operation defined would not satisfy the field requirements, as there would be zero divisors. A polynomial $g(x)$ that cannot be factored into polynomials of lower degree is said to be *irreducible*. By the procedure above, we have taken a polynomial equation $g(\alpha)$ which has no real roots (it is irreducible) and *created* a new element α which is the root of $g(\alpha)$, defining along the way an arithmetic system that is mathematically useful (it is a field). The new field $\mathbb{C}$, with the new element α in it, is said to be an **extension field** of the base field $\mathbb{R}$.

At this point, it might be a tempting intellectual exercise to try to extend $\mathbb{C}$ to a bigger field. However, we won't attempt this because:

1. The extension created is sufficient to demonstrate the operations necessary to extend a finite field to a larger finite field; and (more significantly)
2. It turns out that $\mathbb{C}$ does not have any further extensions: it already contains the roots of all polynomials in $\mathbb{C}[x]$, so there are no other polynomials by which it could be extended. This fact is called the fundamental theorem of algebra.

There are a couple more observations that may be made about operations in $\mathbb{C}$. First, we point out again that addition in the extension field is easy, being simply element by element addition of the vector representation. Multiplication has its own special rules, determined by the polynomial $g(\alpha)$. However, if we represent complex numbers in polar form,

$$a + b\alpha = r_1 e^{j\theta_1} \qquad c + d\alpha = r_2 e^{j\theta_2},$$

then multiplication is also easy: simply multiply the magnitudes and add the angles:

$$r_1 e^{j\theta_1} \cdot r_2 e^{j\theta_2} = r_1 r_2 e^{j(\theta_1+\theta_2)}.$$

Analogously, we will find that addition in the Galois fields we construct is achieved by straightforward vector addition, while multiplication is achieved either by some operation which depends on a polynomial g, or by using a representation loosely analogous to the polar form for complex numbers, in which the multiplication is more easily computed.

5.4.2 Galois Field Construction: An Example

A subfield of a field is a subset of the field that is also a field. For example, $\mathbb{Q}$ is a subfield of $\mathbb{R}$. A more potent concept is that of an extension field. Viewed one way, it simply turns the idea of a subfield around: an extension field E of a field F is a field in which F is a subfield. The field F in this case is said to be the **base field**. But more importantly is the way that the extension field is constructed. Extension fields are constructed to create roots of irreducible polynomials that do not have roots in the base field.

Definition 5.7 A nonconstant polynomial $f(x) \in R[x]$ is **irreducible over** R if $f(x)$ cannot be expressed as a product $g(x)h(x)$ where both $g(x)$ and $h(x)$ are polynomials of degree less than the degree of $f(x)$ and $g(x) \in R[x]$ and $h(x) \in R[x]$. □

Box 5.1: Èveriste Galois (1811–1832)

The life of Galois is a study in brilliance and tragedy. At an early age, Galois studied the works in algebra and analysis of Abel and Lagrange, convincing himself (justifiably) that he was a mathematical genius. His mundane schoolwork, however, remained mediocre. He attempted to enter the Ècole Polytechnique, but his poor academic performance resulted in rejection, the first of many disappointments. At the age of seventeen, he wrote his discoveries in algebra in a paper which he submitted to Cauchy, who lost it. Meanwhile, his father, an outspoken local politician who instilled in Galois a hate for tyranny, committed suicide after some persecution. Some time later, Galois submitted another paper to Fourier. Fourier took the paper home and died shortly thereafter, thereby resulting in another lost paper. As a result of some outspoken criticism against its director, Galois was expelled from the normal school he was attending. Yet another paper presenting his works in finite fields was a failure, being rejected by the reviewer (Poisson) as being too incomprehensible.

Disillusioned, Galois joined the National Guard, where his outspoken nature led to some time in jail for a purported insult against Louis Philippe. Later he was challenged to a duel — probably a setup — to defend the honor of a woman. The night before the duel, Galois wrote a lengthy letter describing his discoveries. The letter was eventually published in *Revue Encylopèdique*. Alas, Galois was not there to read it: he was shot in the stomach in the duel and died the following day of peritonitis at the tender age of twenty.

In this definition, the ring (or field) in which the polynomial is irreducible makes a difference. For example, the polynomial $f(x) = x^2 - 2$ is irreducible over $\mathbb{Q}$, but over the real numbers we can write

$$f(x) = (x + \sqrt{2})(x - \sqrt{2}),$$

so $f(x)$ is reducible over $\mathbb{R}$.

We have already observed that $\langle \mathbb{Z}_p, +, \cdot \rangle$ forms a field when p is prime. It turns out that *all* finite fields have order equal to some power of a prime number, p^m. For $m > 1$, the finite fields are obtained as extension fields to $\mathbb{Z}_p$ using an irreducible polynomial in $\mathbb{Z}_p[x]$ of degree m. These finite fields are usually denoted by $GF(p^m)$ or $GF(q)$ where $q = p^m$, where GF stands for "Galois field," named after the French mathematician Èveriste Galois.

We demonstrate the extension process by constructing the operations for the field $GF(2^4)$, analogous to the way the complex field was constructed from the real field. Any number in $GF(2^4)$ can be represented as a 4-tuple (a, b, c, d), where $a, b, c, d \in GF(2)$. Addition of these numbers is defined to be element-by-element, modulo 2: For $(a_1, a_2, a_3, a_4) \in GF(2^4)$ and $(b_1, b_2, b_3, b_3) \in GF(2^4)$, where $a_i \in GF(2)$ and $b_i \in GF(2)$,

$$(a_1, a_2, a_3, a_4) + (b_1, b_2, b_3, b_4) = (a_1 + b_1, a_2 + b_2, a_3 + b_3, a_4 + b_4).$$

Example 5.27 Add the numbers $(1, 0, 1, 1) + (0, 1, 0, 1)$. Recall that in $GF(2)$, $1 + 1 = 0$, so that we obtain

$$(1, 0, 1, 1) + (0, 1, 0, 1) = (1, 1, 1, 0).$$

□

To define the multiplicative structure, we need an irreducible polynomial of degree 4. The polynomial $g(x) = 1 + x + x^4$ is irreducible over $GF(2)$. (This can be verified since $g(0) = 1$ and $g(1) = 1$, which eliminates linear factors and it can be verified by exhaustion that the polynomial cannot be factored into quadratic factors.) In the extension field $GF(2^4)$, define α to be root of g:

$$\alpha^4 + \alpha + 1 = 0,$$

or

$$\alpha^4 = 1 + \alpha. \tag{5.34}$$

A 4-tuple (a, b, c, d) representing a number in $GF(2^4)$ has a representation in polynomial form

$$a + b\alpha + c\alpha^2 + d\alpha^3.$$

Now take successive powers of α beyond α^4:

$$\begin{aligned} \alpha^4 &= 1 + \alpha, \\ \alpha^5 &= \alpha(\alpha^4) = \alpha + \alpha^2, \\ \alpha^6 &= \alpha^2(\alpha^4) = \alpha^2 + \alpha^3, \\ \alpha^7 &= \alpha^3(\alpha^4) = \alpha^3(1 + \alpha) = \alpha^3 + 1 + \alpha, \end{aligned} \tag{5.35}$$

and so forth. In fact, because of the particular irreducible polynomial $g(x)$ which we selected, powers of α up to α^{14} are all distinct and $\alpha^{15} = 1$. Thus all 15 of the nonzero elements of the field can be represented as powers of α. This gives us something analogous to a "polar" form; we call it the "power" representation. The relationship between the vector representation, the polynomial representation, and the "power" representation for $GF(2^4)$ is shown in Table 5.1. The fact that a 4-tuple has a corresponding representation as a power of α is denoted using $\leftrightarrow$. For example,

$$(0, 1, 0, 0) \leftrightarrow \alpha \qquad (0, 1, 1, 0) \leftrightarrow \alpha^5.$$

The Vector Representation (integer) column of the table is obtained from the Vector Representation column by binary-to-decimal conversion, with the least-significant bit on the left.

Example 5.28 In $GF(2^4)$ multiply the Galois field numbers $1 + \alpha + \alpha^3$ and $\alpha + \alpha^2$. Step 1 is to multiply these "as polynomials" (where the arithmetic of the coefficients takes place in $GF(2)$):

$$(1 + \alpha + \alpha^3) \cdot (\alpha + \alpha^2) = \alpha + \alpha^3 + \alpha^4 + \alpha^5.$$

Step 2 is to reduce using Table 5.1 or, equivalently, to compute the remainder modulo $\alpha^4 + \alpha + 1$:

$$\alpha + \alpha^3 + \alpha^4 + \alpha^5 = \alpha + \alpha^3 + (1 + \alpha) + (\alpha + \alpha^2) = 1 + \alpha + \alpha^2 + \alpha^3.$$

So in $GF(2^4)$,

$$(1 + \alpha + \alpha^3) \cdot (\alpha + \alpha^2) = 1 + \alpha + \alpha^2 + \alpha^3.$$

In vector notation, we could also write

$$(1, 1, 0, 1) \cdot (0, 1, 1, 0) = (1, 1, 1, 1).$$

This product could also have been computed using the power representation. Since $(1, 1, 0, 1) \leftrightarrow \alpha^7$ and $(0, 1, 1, 0) \leftrightarrow \alpha^5$, we have

$$(1, 1, 0, 1) \cdot (0, 1, 1, 0) \leftrightarrow \alpha^7\alpha^5 = \alpha^{12} \leftrightarrow (1, 1, 1, 1).$$

The product is computed simply using the laws of exponents (adding the exponents). □

Table 5.1: Power, Vector, and Polynomial Representations of $GF(2^4)$ as an Extension Using $g(\alpha) = 1 + \alpha + \alpha^4$

Polynomial Representation	Vector Representation	Vector Representation (integer)	Power Representation α^n	Logarithm n	Zech Logarithm $z(n)$
0	0 0 0 0	0	–	–	–
1	1 0 0 0	1	$1 = \alpha^0$	0	–
α	0 1 0 0	2	α	1	4
α^2	0 0 1 0	4	α^2	2	8
α^3	0 0 0 1	8	α^3	3	14
$1+\alpha$	1 1 0 0	3	α^4	4	1
$\alpha+\alpha^2$	0 1 1 0	6	α^5	5	10
$\alpha^2+\alpha^3$	0 0 1 1	12	α^6	6	13
$1+\alpha+\alpha^3$	1 1 0 1	11	α^7	7	9
$1+\alpha^2$	1 0 1 0	5	α^8	8	2
$\alpha+\alpha^3$	0 1 0 1	10	α^9	9	7
$1+\alpha+\alpha^2$	1 1 1 0	7	α^{10}	10	5
$\alpha+\alpha^2+\alpha^3$	0 1 1 1	14	α^{11}	11	12
$1+\alpha+\alpha^2+\alpha^3$	1 1 1 1	15	α^{12}	12	11
$1+\alpha^2+\alpha^3$	1 0 1 1	13	α^{13}	13	6
$1+\alpha^3$	1 0 0 1	9	α^{14}	14	3

The Zech logarithm is explained in Section 5.6.

Example 5.29 In $GF(2^4)$, compute $\alpha^{12} \cdot \alpha^5$. In this case, we would get

$$\alpha^{12} \cdot \alpha^5 = \alpha^{17}.$$

However since $\alpha^{15} = 1$,

$$\alpha^{12} \cdot \alpha^5 = \alpha^{17} = \alpha^{15}\alpha^2 = \alpha^2.$$

□

We compute $\alpha^a \alpha^b = \alpha^c$ by finding $c = (a + b) \pmod{p^m - 1}$.

Since the exponents are important, a nonzero number is frequently represented by the exponent. The exponent is referred to as the *logarithm* of the number.

It should be pointed that this power (or logarithm) representation of the Galois field exists because of the particular polynomial $g(\alpha)$ which was chosen. The polynomial is not only irreducible, it is also *primitive* which means that successive powers of α up to $2^m - 1$ are all unique, just as we have seen.

While different irreducible polynomials can be used to construct the field there is, in fact, only one field with q elements in it, up to isomorphism.

Tables which provide both addition and multiplication operations for $GF(2^3)$ and $GF(2^4)$ are provided inside the back cover of this book.

5.4.3 Connection with Linear Feedback Shift Registers

The generation of a Galois field can be represented using an LFSR with $g(x)$ as the connection polynomial by labeling the registers as 1, α, α^2 and α^3, as shown in Figure 5.1. Then

as the LFSR is clocked, successive powers of α are represented by the state of the LFSR. Compare the vector representation in Table 5.1 with the LFSR sequence in Table 4.11. The state contents provide the vector representation, while the count provides the exponent in the power representation.

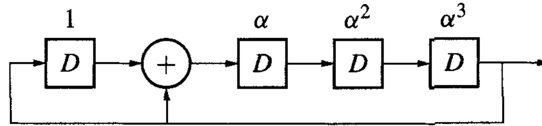


Figure 5.1: LFSR labeled with powers of α to illustrate Galois field elements.

5.5 Galois Fields: Mathematical Facts

Having presented an example of constructing a Galois field, we now lay out some aspects of the theory.

We first examine the additive structure of finite fields, which tells us what size any finite field can be. Recalling Definition 4.3, that the characteristic is the smallest positive integer m such that $m(1) = 1 + 1 + \cdots + 1 = 0$, we have the following.

Lemma 5.13 *The characteristic of a field must be either 0 or a prime number.*

Proof If the field has characteristic 0, the field must be infinite. Otherwise, suppose that the characteristic is a finite number k. Assume k is a composite number. Then $k(1) = 0$ and there are integers $m \neq 1$ and $n \neq 1$ such that $k = mn$. Then

$$0 = k(1) = (mn)(1) = m(1)n(1) = 0.$$

But a field has no zero divisors, so either m or n is the characteristic, violating the minimality of the characteristic. □

It can be shown that any field of characteristic 0 contains the field $\mathbb{Q}$.

On the basis of this lemma, we can observe that in a finite field $GF(q)$, there are p elements (p a prime number) $\{0, 1, 2 = 2(1), \ldots, (p-1) = (p-1)(1)\}$ which behave as a field (i.e., we can define addition and multiplication on them as a field). Thus $\mathbb{Z}_p$ (or something isomorphic to it, which is the same thing) is a subfield of every Galois field $GF(q)$. In fact, a stronger assertion can be made:

Theorem 5.14 *The order q of every finite field $GF(q)$ must be a power of a prime.*

Proof By Lemma 5.13, every finite field $GF(q)$ has a subfield of prime order p. We will show that $GF(q)$ acts like a vector space over its subfield $GF(p)$.

Let $\beta_1 \in GF(q)$, with $\beta_1 \neq 0$. Form the elements $a_1\beta_1$ as a_1 varies over the elements $\{0, 1, \ldots, p-1\}$ in $GF(p)$. The product $a_1\beta_1$ takes on p *distinct* values. (For if $x\beta_1 = y\beta_1$ we must have $x = y$, since there are no zero divisors in a field.) If by these p products we have "covered" all the elements in the field, we are done: they form a vector space over $GF(p)$.

If not, let β_2 be an element which has not been covered yet. Then form $a_1\beta_1 + a_2\beta_2$ as a_1 and a_2 vary independently. This must lead to p^2 distinct values in $GF(q)$. If still not done, then continue, forming the linear combinations

$$a_1\beta_1 + a_2\beta_2 + \cdots a_m\beta_m$$

until all elements of $GF(q)$ are covered. Each combination of coefficients $\{a_1, a_2, \ldots, a_m\}$ corresponds to a distinct element of $GF(q)$. Therefore, there must be p^m elements in $GF(q)$. □

This theorem shows that all finite fields have the structure of a vector space of dimension m over a finite field $\mathbb{Z}_p$. For the field $GF(p^m)$, the subfield $GF(p)$ is called the **ground field.**

This proof raises an important point about the representation of a field. In the construction of $GF(2^4)$ in Section 5.4.2, we formed the field as a vector space over the basis vectors $1, \alpha, \alpha^2$ and α^3. (Or, more generally, to form $GF(p^m)$, we would use the elements $\{1, \alpha, \alpha^2, \ldots, \alpha^{m-1}\}$ as the basis vectors.) However, another set of basis vectors could be used. Any set of m linearly independent nonzero elements of $GF(p^m)$ can be used as a basis set. For example, for $GF(2^4)$ we could construct the field as all linear combinations of $\{1+\alpha, \alpha+\alpha^2, 1+\alpha^3, \alpha+\alpha^2+\alpha^3\}$. The multiplicative relationship prescribed by the irreducible polynomial still applies. While this is not as convenient a construction for most purposes, it is sometimes helpful to think of representations of a field in terms of different bases.

Theorem 5.15 *If x and y are elements in a field of characteristic p,*

$$(x+y)^p = x^p + y^p.$$

This rule is sometimes called "freshman exponentiation," since it is erroneously employed by some students of elementary algebra.

Proof By the binomial theorem,

$$(x+y)^p = \sum_{i=0}^{p} \binom{p}{i} x^i y^{p-i}.$$

For a prime p and for any integer $i \not{0}$ and $i \neq p$, $p \mid \binom{p}{i}$ so that $\binom{p}{i} \equiv 0 \pmod{p}$. Thus all the terms in the sum except the first and the last are p times some quantity, which are equal to 0 since the characteristic of the field is p. □

This theorem extends by induction in two ways: both to the number of summands and to the exponent: If $x_1, x_2, \ldots, x_k$ are in a field of characteristic p, then

$$\left(\sum_{i=1}^{k} x_i\right)^{p^r} = \sum_{i=1}^{k} x_i^{p^r} \tag{5.36}$$

for all $r \geq 0$.

We now consider some multiplicative questions related to finite fields.

Definition 5.8 Let $\beta \in GF(q)$. The **order**[4] of β, written $\text{ord}(\beta)$ is the smallest positive integer n such that $\beta^n = 1$. □

[4]The nomenclature is unfortunate, since we already have defined the order of a group and the order of an element within a group.

Definition 5.9 An element with order $q-1$ in $GF(q)$ (i.e., it generates all the nonzero elements of the field) is called a **primitive element**. □

In other words, a primitive element has the highest possible order.

We saw in the construction of $GF(2^4)$ that the element we called α has order 15, making it a primitive element in the field. We also saw that the primitive element enables the "power representation" of the field, which makes multiplication particularly easy. The questions addressed by the following lemmas are: Does a Galois field always have a primitive element? How many primitive elements does a field have?

Lemma 5.16 *If* $\beta \in GF(q)$ *and* $\beta \neq 0$ *then* $\text{ord}(\beta) \mid (q-1)$.

Proof Let $t = \text{ord}(\beta)$. The set $\{\beta, \beta^2, \ldots, \beta^t = 1\}$ forms a subgroup of the nonzero elements in $GF(q)$ under multiplication. Since the order of a subgroup must divide the order of the group (Lagrange's theorem, Theorem 2.3), the result follows. □

Example 5.30 In the field $GF(2^4)$, the element α^3 has order 5, since

$$(\alpha^3)^5 = \alpha^{15} = 1,$$

and $5 \mid 15$. In fact, we have the sequence

$$\alpha^3, \quad (\alpha^3)^2 = \alpha^6, \quad (\alpha^3)^3 = \alpha^9, \quad (\alpha^3)^4 = \alpha^{12}, \quad (\alpha^3)^5 = \alpha^{15} = 1.$$

□

Lemma 5.17 *Let* $\beta \in GF(q)$. $\beta^s = 1$ *if and only if* $\text{ord}(\beta) \mid s$.

Proof Let $t = \text{ord}(\beta)$. Let s be such that $\beta^s = 1$. Using the division algorithm, write $s = at + r$ where $0 \leq r < t$. Then $1 = \beta^s = \beta^{at}\beta^r = \beta^r$. By the minimality of the order (it must be the *smallest* positive integer), we must have $r = 0$.

Conversely, if $\text{ord}(\beta) \mid s$, then $\beta^s = \beta^{qt} = (\beta^t)^q = 1$, where $t = \text{ord}(\beta)$ and $q = s/t$. □

Lemma 5.18 *If* α *has order* s *and* β *has order* t *and* $(s,t) = 1$, *then* $\alpha\beta$ *has order* st.

Example 5.31 In $GF(2^4)$, α^3 and α^5 have orders that are relatively prime, being 5 and 3 respectively. It may be verified that $\alpha^3\alpha^5 = \alpha^8$ has order 15 (it is primitive). □

Proof First,

$$(\alpha\beta)^{st} = (\alpha^s)^t(\beta^t)^s = 1.$$

Might there be a smaller value for the order than st?

Let k be the order of $\alpha\beta$. Since $(\alpha\beta)^k = 1$, $\alpha^k = \beta^{-k}$. Since $\alpha^s = 1$, $\alpha^{sk} = 1$, and hence $\beta^{-sk} = 1$. Furthermore, $\alpha^{tk} = \beta^{-tk} = 1$. By Lemma 5.17, $s \mid tk$. Since $(s,t) = 1$, k must be a multiple of s.

Similarly, $\beta^{-sk} = 1$ and so $t \mid sk$. Since $(s,t) = 1$, k must be a multiple of t.

Combining these, we see that k must be a multiple of st. In light of the first observation, we have $k = st$. □

Lemma 5.19 *In a finite field, if* $\text{ord}(\alpha) = t$ *and* $\beta = \alpha^i$*, then*

$$\text{ord}(\beta) = \frac{t}{(i, t)}.$$

Proof If $\text{ord}(\alpha) = t$, then $\alpha^s = 1$ if and only if $t \mid s$ (Lemma 5.17).

Let $\text{ord}(\beta) = u$. Note that $i/(i, t)$ is an integer. Then

$$\beta^{t/(i,t)} = (\alpha^i)^{t/(i,t)} = (\alpha^t)^{i/(i,t)} = 1.$$

Thus $u \mid t/(i, t)$. We also have

$$(\alpha^i)^u = 1$$

so $t \mid iu$. This means that $t/(i, t) \mid u$. Combining the results, we have $u = t/(i, t)$. □

Theorem 5.20 *For a Galois field* $GF(q)$*, if* $t \mid q - 1$ *then there are* $\phi(t)$ *elements of order* t *in* $GF(q)$*, where* $\phi(t)$ *is the Euler totient function.*

Proof Observe from Lemma 5.16 that if $t \not| q - 1$, then there are no elements of order t in $GF(q)$. So assume that $t \mid q - 1$; we now determine how many elements of order t there are.

Let α be an element with order t. Then by Lemma 5.19, if $\beta = \alpha^i$ for some i such that $(i, t) = 1$, then β also has order t. The number of such is is $\phi(t)$.

Could there be other elements not of the form α^i having order t? Any element having order t is a root of the polynomial $x^t - 1$. Each element in the set $\{\alpha, \alpha^2, \alpha^3, \ldots, \alpha^t\}$ is a solution to the equation $x^t - 1 = 0$. Since a polynomial of degree t over a field has no more than t roots (see Theorem 5.27 below), there are no elements of order t not in the set. □

The following theorem is a corollary of this result:

Theorem 5.21 *There are* $\phi(q - 1)$ *primitive elements in* $GF(q)$.

Example 5.32 In $GF(7)$, the numbers 5 and 3 are primitive:

$$5^1 = 5, \quad 5^2 = 4, \quad 5^3 = 6, \quad 5^4 = 2, \quad 5^5 = 3, \quad 5^6 = 1.$$

$$3^1 = 3, \quad 3^2 = 4, \quad 3^3 = 6, \quad 3^4 = 4, \quad 3^5 = 5, \quad 3^6 = 1.$$

We also have $\phi(q - 1) = \phi(6) = 2$, so these are the only primitive elements. □

Because primitive elements exist, the nonzero elements of the field $GF(q)$ can always be written as powers of a primitive element. If $\alpha \in GF(q)$ is primitive, then

$$\{\alpha, \alpha^2, \alpha^3, \ldots, \alpha^{q-2}, \alpha^{q-1} = 1\}$$

is the set of all nonzero elements of $GF(q)$. If we let $GF(q)^*$ denote the set of nonzero elements of $GF(q)$, we can write

$$GF(q)^* = \langle \alpha \rangle.$$

If $\beta = \alpha^i$ is also primitive (i.e., $(i, q - 1) = 1$), then the nonzero elements of the field are also generated by

$$\{\beta, \beta^2, \beta^3, \ldots, \beta^{q-2}, \beta^{q-1} = 1\},$$

that is, $GF(q)^* = \langle \beta \rangle$. Despite the fact that these are different generators, these are not different fields, only different representations, so $\langle \alpha \rangle$ is isomorphic to $\langle \beta \rangle$. We thus talk of *the* Galois field with q elements, since there is only one.

Theorem 5.22 *Every element of the field $GF(q)$ satisfies the equation $x^q - x = 0$. Furthermore, they constitute the entire set of roots of this equation.*

Proof Clearly, the equation can be written as $x(x^{q-1} - 1) = 0$. Thus $x = 0$ is clearly a root. The nonzero elements of the field are all generated as powers of a primitive element α. For an element $\beta = \alpha^i \in GF(q)$, $\beta^{q-1} = (\alpha^i)^{q-1} = (\alpha^{q-1})^i = 1$. Since there are q elements in $GF(q)$, and at most q roots of the equation, the elements of $GF(q)$ are all the roots. □

An extension field E of a field $\mathbb{F}$ is a **splitting field** of a nonconstant polynomial $f(x) \in \mathbb{F}[x]$ if $f(x)$ can be factored into linear factors over E, but not in any proper subfield of E. Theorem 5.22 thus says that $GF(q)$ is the splitting field for the polynomial $x^q - q$.

As an extension of Theorem 5.22, we have

Theorem 5.23 *Every element in a field $GF(q)$ satisfies the equation*

$$x^{q^n} - x = 0$$

for every $n \geq 0$.

Proof When $n = 0$ the result is trivial; when $n = 1$ we have Theorem 5.22, giving $x^q = x$. The proof is by induction: Assume that $x^{q^{n-1}} = x$. Then $(x^{q^{n-1}})^q = x^q = x$, or $x^{q^n} = x$. □

A field $GF(p)$ can be extended to a field $GF(p^m)$ for any $m > 1$. Let $q = p^m$. The field $GF(q)$ can be further extended to a field $GF(q^r)$ for any r, by extending by an irreducible polynomial of degree r in $GF(q)[x]$. This gives the field $GF(p^{mr})$.

5.6 Implementing Galois Field Arithmetic

Lab 5 describes one way of implementing Galois field arithmetic in a computer using two tables. In this section, we present a way of computing operations using one table of *Zech logarithms*, as well as some concepts for hardware implementation.

5.6.1 Zech Logarithms

In a Galois field $GF(2^m)$, the Zech logarithm $z(n)$ is defined by

$$\alpha^{z(n)} = 1 + \alpha^n, \qquad n = 1, 2, \ldots, 2^m - 2.$$

Table 5.1 shows the Zech logarithm for the field $GF(2^4)$. For example, when $n = 2$, we have

$$1 + \alpha^2 = \alpha^8$$

so that $z(2) = 8$.

In the Zech logarithm approach to Galois field computations, numbers are represented using the exponent. Multiplication is thus natural. To see how to add, consider the following example:

$$\alpha^3 + \alpha^5.$$

The first step is to factor out the term with the smallest exponent,

$$\alpha^3(1 + \alpha^2).$$

Now the Zech logarithm is used: $1+\alpha^2 = \alpha^{z(2)} = \alpha^8$. So

$$\alpha^3 + \alpha^5 = \alpha^3(1+\alpha^2) = \alpha^3\alpha^8 = \alpha^{11}.$$

The addition requires one table lookup and one multiply operation. It has been found that in many implementations, the use of Zech logarithms can significantly improve execution time.

5.6.2 Hardware Implementations

We present examples for the field $GF(2^4)$ generated by $g(x) = 1 + x + x^4$. Addition is easily accomplished by simple mod-2 addition for numbers in vector representation.

Multiplication of the element $\beta = b_0 + b_1\alpha + b_2\alpha^2 + b_3\alpha^3$ by the primitive element α is computed using $\alpha^4 = 1 + \alpha$ as

$$\alpha\beta = b_0\alpha + b_1\alpha^2 + b_2\alpha^3 + b_3\alpha^4 = b_3 + (b_0 + b_3)\alpha + b_1\alpha^2 + b_2\alpha^3.$$

These computations can be obtained using an LFSR as shown in Figure 5.2. Clocking the registers once fills them with the representation of $\alpha\beta$.

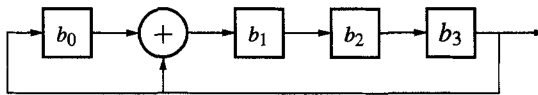


Figure 5.2: Multiplication of β by α.

Multiplication by specific powers of α can be accomplished with dedicated circuits. For example, to multiply $\beta = b_0 + b_1\alpha + b_2\alpha^2 + b_3\alpha^3$ by $\alpha^4 = 1 + \alpha$, we have

$$\beta\alpha^4 = \beta + \alpha\beta = (b_0 + b_3) + (b_0 + b_1 + b_3)\alpha + (b_1 + b_2)\alpha^2 + (b_2 + b_3)\alpha^3,$$

which can be represented as shown in Figure 5.3.

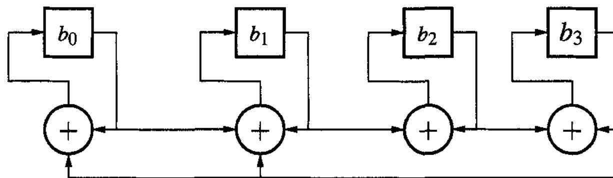


Figure 5.3: Multiplication of an arbitrary β by α^4.

Finally, we present a circuit which multiplies two arbitrary Galois field elements. Let $\beta = b_0 + b_1\alpha + b_2\alpha^2 + b_3\alpha^3$ and let $\gamma = c_0 + c_1\alpha + c_2\alpha^2 + c_3\alpha^3$. Then $\beta\gamma$ can be written in a Horner-like notation as

$$\beta\gamma = (((c_3\beta)\alpha + c_2\beta)\alpha + c_1\beta)\alpha + c_0\beta.$$

Figure 5.4 shows a circuit for this expression. Initially, the upper register is cleared. Then at the first clock the register contains $c_3\beta$. At the second clock the register contains $c_3\beta\alpha + c_2\beta$, where the multiplication by α comes by virtue of the feedback structure. At the next clock the register contains $(c_3\beta\alpha + c_2\beta)\alpha + c_1\beta$. At the final clock the register contains the entire product.

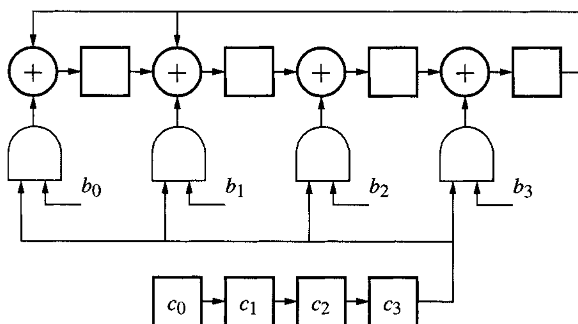


Figure 5.4: Multiplication of β by an arbitrary field element.

5.7 Subfields of Galois Fields

Elements in a base field $GF(q)$ are also elements of its extension field $GF(q^m)$. Given an element $\beta \in GF(q^m)$ in the extension field, it is of interest to know if it is an element in the base field $GF(q)$. The following theorem provides the answer.

Theorem 5.24 *An element $\beta \in GF(q^m)$ lies in $GF(q)$ if and only if $\beta^q = \beta$.*

Proof If $\beta \in GF(q)$, then by Lemma 5.16, $\text{ord}(\beta) \mid (q-1)$, so that $\beta^q = \beta$.

Conversely, assume $\beta^q = \beta$. Then β is a root of $x^q - x = 0$. Now observe that all q elements of $GF(q)$ satisfy this polynomial and it can only have q roots. Hence $\beta \in GF(q)$. □

By induction, it follows that an element $\beta \in GF(q^m)$ lies in the subfield $GF(q)$ if $\beta^{q^n} = \beta$ for any $n \geq 0$.

Example 5.33 The field $GF(4)$ is a subfield of $GF(256)$. Let α be primitive in $GF(256)$. We desire to find an element in $GF(4) \subset GF(256)$. Let $\beta = \alpha^{85}$. Then, invoking Theorem 5.24

$$\beta^4 = \alpha^{85 \cdot 4} = \alpha^{255}\alpha^{85} = \beta.$$

So $\beta \in GF(4)$ and $GF(4)$ has the elements $\{0, 1, \beta, \beta^2\} = \{0, 1, \alpha^{85}, \alpha^{170}\}$. □

Theorem 5.25 *$GF(q^k)$ is a subfield of $GF(q^j)$ if and only if $k \mid j$.*

The proof relies on the following lemma.

Lemma 5.26 *If n, r, and s are positive integers and $n \geq 2$, then $n^s - 1 \mid n^r - 1$ if and only if $s \mid r$.*

Proof of Theorem 5.25. If $k \mid j$, say $j = mk$, then $GF(q^k)$ can be extended using an irreducible polynomial of degree m over $GF(q^k)$ to obtain the field with $(q^k)^m = q^j$ elements.

Conversely, let $GF(q^k)$ be a subfield of $GF(q^j)$ and let β be a primitive element in $GF(q^k)$. Then $\beta^{q^k-1} = 1$. As an element of the field $GF(q^k)$, it must also be true (see, e.g.,

Theorem 5.22) that $\beta^{q^j-1} = 1$. From Lemma 5.17, it must be the case that $q^k - 1 \mid q^j - 1$ and hence, from Lemma 5.26, it follows that $k \mid j$. □

As an example of this, Figure 5.5 illustrates the subfields of $GF(2^{24})$.

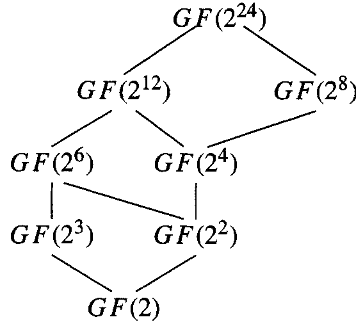


Figure 5.5: Subfield structure of $GF(2^{24})$.

5.8 Irreducible and Primitive polynomials

We first present a result familiar from polynomials over complex numbers.

Theorem 5.27 *A polynomial of degree d over a field $\mathbb{F}$ has at most d roots in any field containing $\mathbb{F}$.*

This theorem seems obvious, but in fact over a *ring* it is not necessarily true! The quadratic polynomial $x^2 - 1$ has four roots in $\mathbb{Z}_{15}$, namely 1, 4, 11 and 14 [25].

Proof Every polynomial of degree 1 (i.e., a linear polynomial) is irreducible. Since the degree of a product of several polynomials is the sum of their degrees, a polynomial of degree d cannot have more than d linear factors. By the division algorithm, $(x - \beta)$ is a factor of a polynomial $f(x)$ if and only if $f(\beta) = 0$ (see Exercise 5.47). Hence $f(x)$ can have at most d roots. □

While any irreducible polynomial can be used to construct the extension field, computation in the field is easier if a primitive polynomial is used. We make the following observation:

Theorem 5.28 *Let p be prime. An irreducible mth-degree polynomial $f(x) \in GF(p)[x]$ divides $x^{p^m-1} - 1$.*

Example 5.34 $(x^3 + x + 1) \mid (x^7 + 1)$ in $GF(2)$ (this can be shown by long division). □

It is important to understand the implication of the theorem: an irreducible polynomial divides $x^{p^m} - 1$, but just because a polynomial divides $x^{p^m} - 1$ does not mean that it is irreducible. (Showing irreducibility is much harder than that!)

Proof Let $GF(q) = GF(p^m)$ be constructed using the irreducible polynomial $f(x)$, where α denotes the root of $f(x)$ in the field: $f(\alpha) = 0$. By Theorem 5.22, α is a root of $x^{p^m-1} - 1$ in $GF(q)$. Using the division algorithm write

$$x^{p^m-1} - 1 = g(x)f(x) + r(x), \tag{5.37}$$

where $\deg(r(x)) < m$. Evaluating (5.37) at $x = \alpha$ in $GF(q)$ we obtain

$$0 = 0 + r(\alpha).$$

But the elements of the field $GF(q)$ are represented as polynomials in α of degree $< m$, so since $r(\alpha) = 0$ it must be that $r(x)$ is the zero polynomial, $r(x) = 0$. □

A slight generalization, proved similarly using Theorem 5.23, is the following:

Theorem 5.29 *If $f[x] \in GF(q)[x]$ is an irreducible polynomial of degree m, then*

$$f(x) \,\big|\, (x^{q^k} - x)$$

for any k such that $m \,|\, k$.

Definition 5.10 An irreducible polynomial $p(x) \in GF(p)[x]$ of degree m is said to be **primitive** if the smallest positive integer n for which $p(x)$ divides $x^n - 1$ is $n = p^m - 1$. □

Example 5.35 Taking $f(x) = x^3 + x + 1$, it can be shown by exhaustive checking that $f(x) \nmid x^4 + 1$, $f(x) \nmid x^5 + 1$, and $f(x) \nmid x^6 + 1$, but $f(x) \,|\, x^7 + 1$. In fact,

$$x^7 - 1 = (x^3 + x + 1)(x^4 + x^2 + x + 1).$$

Thus $f(x)$ is primitive. □

The following theorem provides the motivation for using primitive polynomials.

Theorem 5.30 *The roots of an mth degree primitive polynomial $p(x) \in GF(p)[x]$ are primitive elements in $GF(p^m)$.*

That is, any of the roots can be used to generate the nonzero elements of the field $GF(p^m)$.

Proof Let α be a root of an mth-degree primitive polynomial $p(x)$. We have

$$x^{p^m-1} - 1 = p(x)q(x)$$

for some $q(x)$. Observe that

$$\alpha^{p^m-1} - 1 = p(\alpha)q(\alpha) = 0q(\alpha) = 0,$$

from which we note that

$$\alpha^{p^m-1} = 1.$$

Now the question is, might there be a smaller power t of α such that $\alpha^t = 1$? If this were the case, then we would have

$$\alpha^t - 1 = 0.$$

There would therefore be some polynomial $x^t - 1$ that would have α as a root. However, any root of $x^t - 1$ must also be a root of $x^{p^m-1} - 1$, because $\mathrm{ord}(\alpha) \,|\, p^m - 1$. To see this, suppose (to the contrary) that $\mathrm{ord}(\alpha) \nmid p^m - 1$. Then

$$p^m - 1 = k \,\mathrm{ord}(\alpha) + r$$

for some r with $0 < r < \mathrm{ord}(\alpha)$. Therefore we have

$$1 = \alpha^{p^m-1} = \alpha^{k\,\mathrm{ord}(\alpha)+r} = \alpha^r,$$

which contradicts the minimality of the order.

Thus, all the roots of $x^t - 1$ are the roots of $x^{p^m-1} - 1$, so

$$x^t - 1 \big| x^{p^m-1} - 1.$$

We show below that all the roots of an irreducible polynomial are of the same order. This means that $p(x) \big| x^t - 1$. But by the definition of a primitive polynomial, we must have $t = p^m - 1$. □

All the nonzero elements of the field can be **generated as powers of the roots of the primitive polynomial.**

Example 5.36 The polynomial $p(x) = x^2 + x + 2$ is primitive in $GF(5)$. Let α represent a root of $p(x)$, so that $\alpha^2 + \alpha + 2 = 0$, or $\alpha^2 = 4\alpha + 3$. The elements in $GF(5)$ can be represented as powers of α as shown in the following table.

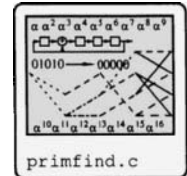
0	$\alpha^0 = 1$	$\alpha^1 = \alpha$	$\alpha^2 = 4\alpha + 3$	$\alpha^3 = 4\alpha + 2$
$\alpha^4 = 3\alpha + 2$	$\alpha^5 = 4\alpha + 4$	$\alpha^6 = 2$	$\alpha^7 = 2\alpha$	$\alpha^8 = 3\alpha + 1$
$\alpha^9 = 3\alpha + 4$	$\alpha^{10} = \alpha + 4$	$\alpha^{11} = 3\alpha + 3$	$\alpha^{12} = 4$	$\alpha^{13} = 4\alpha$
$\alpha^{14} = \alpha + 2$	$\alpha^{15} = \alpha + 3$	$\alpha^{16} = 2\alpha + 3$	$\alpha^{17} = \alpha + 1$	$\alpha^{18} = 3$
$\alpha^{19} = 3\alpha$	$\alpha^{20} = 2\alpha + 4$	$\alpha^{21} = 2\alpha + 1$	$\alpha^{22} = 4\alpha + 1$	$\alpha^{23} = 2\alpha + 2$

As an example of some arithmetic in this field,

$$(3\alpha + 4) + (4\alpha + 1) = 2\alpha$$

$$(3\alpha + 4)(4\alpha + 1) = \alpha^9\alpha^{22} = \alpha^{31} = (\alpha^{24})(\alpha^7) = 2\alpha.$$

□



The program `primfind` Find primitive polynomials in $GF(p)[x]$, where the prime p can be specified. It does this by recursively producing *all* polynomials (or all of those of a weight you might specify) and evaluating whether they are primitive by using them as feedback polynomials in an LFSR. Those which generate maximal length sequences are primitive.

5.9 Conjugate Elements and Minimal Polynomials

From chapter 4, we have seen that cyclic codes have a generator polynomial $g(x)$ dividing $x^n - 1$. Designing cyclic codes with a specified code length n thus requires the facility to factor $x^n - 1$ into factors with certain properties. In this section we explore aspects of this factorization question.

It frequently happens that the structure of a code is defined over a field $GF(q^m)$, but it is desired to employ a generator polynomial $g(x)$ over the base field $GF(q)$. For example, we might want a binary generator polynomial — for ease of implementation — but need to work over a field $GF(2^m)$ for some m. How to obtain polynomials having coefficients in the base field but roots in the larger field is our first concern. The concepts of conjugates and minimal polynomials provide a language to describe the polynomials we need.

We begin with a reminder and analogy from polynomials with real coefficients. Suppose we are given a complex number $x_1 = (2 + 3i)$. Over the (extension) field $\mathbb{C}$, there is a polynomial $x - (2 + 3i) \in \mathbb{C}[x]$ which has x_1 as a root. But suppose we are asked to find the polynomial with *real* coefficients that has x_1 as a root. We are well acquainted with the

fact that the roots of real polynomials come in complex conjugate pairs, so we conclude immediately that a real polynomial with root x_1 must also have a root $x_2 = (2 - 3i)$. We say that x_2 is a *conjugate* root to x_1. A polynomial having these roots is

$$(x - (2 + 3i))(x - (2 - 3i)) = x^2 - 4x + 13.$$

Note in particular that the coefficients of the resulting polynomials are in $\mathbb{R}$, which was the base field for the extension to $\mathbb{C}$.

This concept of conjugacy has analogy to finite fields. Suppose that $f(x) \in GF(q)[x]$ has $\alpha \in GF(q^m)$ as a root. (That is, the polynomial has coefficients in the *base field*, while the root comes from an extension field.) What are the other roots of $f(x)$ in this field?

Theorem 5.31 *Let $GF(q) = GF(p^r)$ for some $r \geq 1$. Let $f(x) = \sum_{j=0}^{d} f_j x^j \in GF(q)[x]$. That is, $f_i \in GF(q)$. Then*

$$f(x^{q^n}) = [f(x)]^{q^n}$$

for any $n \geq 0$.

Proof

$$\begin{aligned}
[f(x)]^{q^n} &= \left[\sum_{j=0}^{d} f_j x^j\right]^{q^n} = \sum_{j=0}^{d} f_j^{q^n} (x^j)^{q^n} && \text{(by (5.36))} \\
&= \sum_{j=0}^{d} f_j (x^{q^n})^j && \text{(by Theorem 5.24)} \\
&= f(x^{q^n}).
\end{aligned}$$

□

Thus, if $\beta \in GF(q^m)$ is a root of $f(x) \in GF(q)[x]$, then β^{q^n} is also a root of $f(x)$. This motivates the following definition.

Definition 5.11 Let $\beta \in GF(q^m)$. The **conjugates** of β with respect to a subfield $GF(q)$ are $\beta, \beta^q, \beta^{q^2}, \beta^{q^3}, \ldots$. (This list must, of course, repeat at some point since the field is finite.)

The conjugates of β with respect to $GF(q)$ form a set called the **conjugacy class** of β with respect to $GF(q)$. □

Example 5.37

1. Let $\alpha \in GF(2^3)$ be primitive. The conjugates of α are

$$\alpha, \alpha^2, (\alpha^{2^2}) = \alpha^4, (\alpha^{2^3}) = \alpha.$$

So the conjugacy class of α is $\{\alpha, \alpha^2, \alpha^4\}$.

Let $\beta = \alpha^3$, an element not in the conjugacy class of α. The conjugates of β are

$$\beta = \alpha^3, \quad (\alpha^3)^2 = \alpha^6, \quad (\alpha^3)^{2^2} = \alpha^{12} = \alpha^7\alpha^5 = \alpha^5, \quad (\alpha^3)^{2^3} = \alpha^{24} = \alpha^{21}\alpha^3 = \alpha^3.$$

So the conjugacy class of β is $\{\alpha^3, \alpha^6, \alpha^5\}$.

The only other elements of $GF(2^3)$ are 1, which always forms its own conjugacy class, and 0, which always forms its own conjugacy class.

We observe that the conjugacy classes of the elements of $GF(2^3)$ form a partition of $GF(2^3)$.

2. Let $\beta \in GF(16)$ be an element such that $\text{ord}(\beta) = 3$. (Check for consistency: since $3 \mid 15$, there are $\phi(3) = 2$ elements of order 3 in $GF(16)$.) The conjugacy class of β is

$$\beta, \beta^2, \beta^{2^2} = \beta^4 = \beta.$$

So there are 2 elements in this conjugacy class, $\{\beta, \beta^2\}$.

3. Find all the conjugacy classes in $GF(2^4)$ with respect to $GF(2)$.

Let $\alpha \in GF(2^4)$ be primitive. Pick α and list its conjugates with respect to $GF(2)$:

$$\alpha, \alpha^2, \alpha^4, \alpha^8, \alpha^{16} = \alpha$$

so the first conjugacy class is $\{\alpha, \alpha^2, \alpha^4, \alpha^8\}$. Now pick an element unused so far. Take α^3 and write its conjugates:

$$\alpha^3, (\alpha^3)^2 = \alpha^6, (\alpha^3)^4 = \alpha^{12}, (\alpha^3)^8 = \alpha^9, (\alpha^3)^{16} = \alpha^3,$$

so the next conjugacy class is $\{\alpha^3, \alpha^6, \alpha^9, \alpha^{12}\}$. Take another unused element, α^5:

$$\alpha^5, (\alpha^5)^2 = \alpha^{10}, (\alpha^5)^4 = \alpha^5$$

so the next conjugacy class is $\{\alpha^5, \alpha^{10}\}$. Take another unused element, α^7:

$$\alpha^7, (\alpha^7)^2 = \alpha^{14}, (\alpha^7)^4 = \alpha^{13}, (\alpha^7)^8 = \alpha^{11}, (\alpha^7)^{16} = \alpha^7,$$

so the next conjugacy class is $\{\alpha^7, \alpha^{14}, \alpha^{13}, \alpha^{11}\}$. The only unused elements now are 0, with conjugacy class $\{0\}$, and 1, with conjugacy class $\{1\}$.

4. Find all the conjugacy classes in $GF(2^4)$ with respect to $GF(4)$.

Let α be primitive in $GF(2^4)$. The conjugacy classes with respect to $GF(4)$ are:

$$\{\alpha, \alpha^4\} \quad \{\alpha^2, \alpha^8\} \quad \{\alpha^3, \alpha^{12}\} \quad \{\alpha^5\} \quad \{\alpha^6, \alpha^9\} \quad \{\alpha^7, \alpha^{13}\} \quad \{\alpha^{10}\} \quad \{\alpha^{11}, \alpha^{14}\}.$$

□

Definition 5.12 [5] The smallest positive integer d such that $n \mid q^d - 1$ is called the **multiplicative order of q modulo n.**

□

Lemma 5.32 *Let $\beta \in GF(q^m)$ have $\text{ord}(\beta) = n$ and let d be the multiplicative order of q modulo n. Then $\beta^{q^d} = \beta$. The d elements $\beta, \beta^q, \beta^{q^2}, \ldots, \beta^{q^{d-1}}$ are all distinct.*

Proof Since $\text{ord}(\beta) = n$ and $n \mid q^d - 1$, $\beta^{q^d - 1} = 1$, so $\beta^{q^d} = \beta$.

To check distinctness, suppose that $\beta^{q^k} = \beta^{q^i}$ for $0 \le i < k < d$. Then $\beta^{q^k - q^i} = 1$, which by Lemma 5.17 implies that $n \mid q^k - q^i$, that is, $q^k \equiv q^i \pmod{n}$. By Theorem 5.8, item 7 it follows that $q^{k-i} \equiv 1 \pmod{n/(n, q^i)}$, that is, $q^{k-i} \equiv 1 \pmod{n}$ (since q is a power of a prime, and $n \mid q^d - 1$). By definition of d, this means that $d \mid k - i$, which is not possible since $i < k < d$.

□

[5] This is yet another usage of the word "order."

5.9.1 Minimal Polynomials

In this section, we examine the polynomial in $GF(q)[x]$ which has an element $\beta \in GF(q^m)$ and all of its conjugates as roots.

Definition 5.13 Let $\beta \in GF(q^m)$. The **minimal polynomial** of β with respect to $GF(q)$ is the smallest-degree, nonzero, monic polynomial $p(x) \in GF(q)[x]$ such that $p(\beta) = 0$. □

Returning to the analogy with complex numbers, we saw that the polynomial with $f(x) = x^2 - 4x + 13$ with *real* coefficients has the *complex* number $x_1 = 2 + 3i$ as a root. Furthermore, it is clear that there is no real polynomial of smaller degree which has x_1 as a root. We would say that $x^2 - 4x + 13$ is the minimal polynomial of $2 + 3i$ with respect to the real numbers.

Some properties for minimal polynomials:

Theorem 5.33 *[373, Theorem 3-2] For each $\beta \in GF(q^m)$ there exists a unique monic polynomial $p(x)$ of minimal degree in $GF(q)[x]$ such that:*

1. $p(\beta) = 0$.
2. *The degree of $p(x) \leq m$.*
3. *If there is a polynomial $f(x) \in GF(q)[x]$ such that $f(\beta) = 0$ then $p(x) \mid f(x)$.*
4. *$p(x)$ is irreducible in $GF(q)[x]$.*

Proof Existence: Given an element $\beta \in GF(q^m)$, write down the $(m + 1)$ elements $1, \beta, \beta^2, \ldots, \beta^m$ which are elements of $GF(q^m)$. Since $GF(q^m)$ is a vector space of dimension m over $GF(q)$, these $m + 1$ elements must be linearly dependent. Hence there exist coefficients $a_i \in GF(q)$ such that $a_0 + a_1\beta + \cdots + a_m\beta^m = 0$; these are the coefficients of a polynomial $f(x) = \sum_{i=0}^{m} a_i x^i$ which has β as the root. (It is straightforward to make this polynomial monic.) This also shows that the degree of $f(x) \leq m$.

Uniqueness: Suppose that there are two minimal polynomials of β, which are normalized to be monic; call them $f(x)$ and $g(x)$. These must both have the same degree. Then there is a polynomial $r(x)$ having $\deg(r(x)) < \deg(f(x))$ such that

$$f(x) = g(x) + r(x).$$

Since β is a root of f and g, we have

$$0 = f(\beta) = g(\beta) + r(\beta).$$

so that $r(\beta) = 0$. Since a minimal polynomial $f(x)$ has the smallest nonzero degree polynomial such that $f(\beta) = 0$, it must be the case that $r(x) = 0$ (i.e., it is the zero polynomial), so $f(x) = g(x)$.

Divisibility: Let $p(x)$ be a minimal polynomial. If there is a polynomial $f(x)$ such that $f(\beta) = 0$, we write using the division algorithm

$$f(x) = p(x)q(x) + r(x),$$

where $\deg(r) < \deg(p)$. But then $f(\beta) = p(\beta)q(\beta) + r(\beta) = 0$, so $r(\beta) = 0$. By the minimality of the degree of $p(x)$, $r(x) = 0$, so $p(x) \mid f(x)$.

Irreducibility: If $p(x)$ factors, so $p(x) = f(x)g(x)$, then either $f(\beta) = 0$ or $g(\beta) = 0$, again a contradiction to the minimality of the degree of $p(x)$. □

We observe that primitive polynomials are the minimal polynomials for primitive elements in a finite field.

Let $p(x) \in GF(q)[x]$ be a minimal polynomial for β. Then $\beta^q, \beta^{q^2}, \ldots, \beta^{q^{d-1}}$ are also roots of $p(x)$. Could there be other roots of $p(x)$? The following theorem shows that the conjugacy class for β contains all the roots for the minimal polynomial of β.

Theorem 5.34 *[25, Theorem 4.410] Let $\beta \in GF(q^m)$ have order n and let d be the multiplicative order of q mod n. Then the coefficients of the polynomial $p(x) = \prod_{i=0}^{d-1}(x - \beta^{q^i})$ are in $GF(q)$. Furthermore, $p(x)$ is irreducible. That is, $p(x)$ is the minimal polynomial for β.*

Proof From Theorem 5.31 we see that $p(\beta) = 0$ implies that $p(\beta^q) = 0$ for $p(x) \in GF(q)[x]$. It only remains to show that the polynomial having the conjugates of β as its roots has its coefficients in $GF(q)$. Write

$$[p(x)]^q = \prod_{i=0}^{d-1}(x - \beta^{q^i})^q = \prod_{i=0}^{d-1}(x^q - \beta^{q^{i+1}}) \qquad \text{(by Theorem 5.15)}$$

$$= \prod_{i=1}^{d}(x^q - \beta^{q^i}) = \prod_{i=0}^{d-1}(x^q - \beta^{q^i}) \qquad (\text{since } \beta^{q^d} = \beta = \beta^{q^0}).$$

Thus $[p(x)]^q = p(x^q)$. Now writing $p(x) = \sum_{i=0}^{d} p_i x^i$ we have

$$[p(x)]^q = \left(\sum_{i=0}^{d} p_i x^i\right)^q = \sum_{i=0}^{d} p_i^q x^{iq} \tag{5.38}$$

and

$$p(x^q) = \sum_{i=0}^{d} p_i x^{iq}. \tag{5.39}$$

The two polynomials in (5.38) and (5.39) are identical, so it must be that $p_i^q = p_i$, so $p_i \in GF(q)$.

If $p(x) = g(x)h(x)$, where $g(x) \in GF(q)[x]$ and $h(x) \in GF(q)[x]$ and are monic, then $p(\beta) = 0$ implies that $g(\beta) = 0$ or $h(\beta) = 0$. If $g(\beta) = 0$, then $g(\beta^q) = g(\beta^{q^2}) = \cdots = g(\beta^{q^{d-1}}) = 0$. g thus has d roots, so $g(x) = p(x)$. Similarly, if $h(\beta) = 0$, then it follows that $h(x) = p(x)$. □

As a corollary, we have the following.

Corollary 5.35 *[373, p. 58] Let $f(x) \in GF(q)[x]$ be irreducible. Then all of the roots of $f(x)$ have the same order.*

Proof Let $GF(q^m)$ be the smallest field containing all the roots of the polynomial $f(x)$ and let $\beta \in GF(q^m)$ be a root of $f(x)$. Then $\text{ord}(\beta) \mid q^m - 1$ (Lemma 5.16). By the theorem, the roots of $f(x)$ are the conjugates of β and so are of the form $\{\beta, \beta^q, \beta^{q^2}, \ldots\}$. Since $q = p^r$ for some r, it follows that $(q, q^m - 1) = 1$. Also, if $t \mid q^m - 1$, then $(q, t) = 1$. By Lemma 5.19 we have

$$\text{ord}(\beta^{q^k}) = \frac{\text{ord}(\beta)}{(q^k, \text{ord}(\beta))} = \text{ord}(\beta).$$

Table 5.2: Conjugacy Classes over $GF(2^3)$ with Respect to $GF(2)$

Conjugacy Class	Minimal Polynomial
$\{0\}$	$M_-(x) = x$
$\{1\}$	$M_0(x) = x + 1$
$\{\alpha, \alpha^2, \alpha^4\}$	$M_1(x) = (x-\alpha)(x-\alpha^2)(x-\alpha^4) = x^3 + x + 1$
$\{\alpha^3, \alpha^6, \alpha^5\}$	$M_3(x) = (x-\alpha^3)(x-\alpha^5)(x-\alpha^6) = x^3 + x^2 + 1.$

Table 5.3: Conjugacy Classes over $GF(2^4)$ with Respect to $GF(2)$

Conjugacy Class	Minimal Polynomial
$\{0\}$	$M_-(x) = x$
$\{1\}$	$M_0(x) = x + 1$
$\{\alpha, \alpha^2, \alpha^4, \alpha^8\}$	$M_1(x) = (x-\alpha)(x-\alpha^2)(x-\alpha^4)(x-\alpha^8)$ $= x^4 + x + 1$
$\{\alpha^3, \alpha^6, \alpha^9, \alpha^{12}\}$	$M_3(x) = (x-\alpha^3)(x-\alpha^6)(x-\alpha^9)(x-\alpha^{12})$ $= x^4 + x^3 + x^2 + x + 1$
$\{\alpha^5, \alpha^{10}\}$	$M_5(x) = (x-\alpha^5)(x-\alpha^{10}) = x^2 + x + 1$
$\{\alpha^7, \alpha^{11}, \alpha^{13}, \alpha^{14}\}$	$M_7(x) = (x-\alpha^7)(x-\alpha^{11})(x-\alpha^{13})(x-\alpha^{14})$ $= x^4 + x^3 + 1$

Since this is true for any k, each root has the same order. □

Example 5.38 According to Theorem 5.34, we can obtain the minimal polynomial for an element β by multiplying the factors $(x - \beta^{q^i})$. In what follows, you may refer to the conjugacy classes determined in Example 5.37.

1. Determine the minimal polynomial for each conjugacy class in $GF(8)$ with respect to $GF(2)$. To do the multiplication, a representation of the field is necessary; we use the representation using primitive polynomial $g(x) = x^3 + x + 1$. Using the conjugacy classes we found before in $GF(8)$, we obtain the minimal polynomials shown in Table 5.2.
2. Determine the minimal polynomial for each conjugacy class in $GF(2^4)$ with respect to $GF(2)$. Use Table 5.1 as a representation. The minimal polynomials are shown in Table 5.3.
3. Determine the minimal polynomial for each conjugacy class in $GF(2^5)$ with respect to $GF(2)$. Using the primitive polynomial $x^5 + x^2 + 1$, it can be shown that the minimal polynomials are as shown in Table 5.4.
4. Determine the minimal polynomials in $GF(4^2)$ with respect to $GF(4)$. Use the representation obtained from the subfield $GF(4) = \{0, 1, \alpha^5, \alpha^{10}\} \subset GF(16)$ from Table 5.1. The result is shown in Table 5.5.

□

As this example suggests, the notation $M_i(x)$ is used to denote the minimal polynomial of the conjugacy class that α^i is in, where i is the smallest exponent in the conjugacy class.

It can be shown that (see [200, p. 96]) for the minimal polynomial $m(x)$ of degree d in a field of $GF(q^m)$ that $d \mid m$.

Table 5.4: Conjugacy Classes over $GF(2^5)$ with Respect to $GF(2)$

Conjugacy Class	Minimal Polynomial
$\{0\}$	$M_-(x) = x$
$\{1\}$	$M_0(x) = x + 1$
$\{\alpha, \alpha^2, \alpha^4, \alpha^8, \alpha^{16}\}$	$M_1(x) = (x-\alpha)(x-\alpha^2)(x-\alpha^4)(x-\alpha^8)(x-\alpha^{16})$ $= x^5 + x^2 + 1$
$\{\alpha^3, \alpha^6, \alpha^{12}, \alpha^{17}, \alpha^{24}\}$	$M_3(x) = (x-\alpha^3)(x-\alpha^6)(x-\alpha^{12})(x-\alpha^{17})(x-\alpha^{24})$ $= x^5 + x^4 + x^3 + x^2 + 1$
$\{\alpha^5, \alpha^9, \alpha^{10}, \alpha^{18}, \alpha^{20}\}$	$M_5(x) = (x-\alpha^5)(x-\alpha^9)(x-\alpha^{10})(x-\alpha^{18})(x-\alpha^{20})$ $= x^5 + x^4 + x^2 + x + 1$
$\{\alpha^7, \alpha^{14}, \alpha^{19}, \alpha^{25}, \alpha^{28}\}$	$M_7(x) = (x-\alpha^7)(x-\alpha^{14})(x-\alpha^{19})(x-\alpha^{25})(x-\alpha^{28})$ $= x^5 + x^3 + x^2 + x + 1$
$\{\alpha^{11}, \alpha^{13}, \alpha^{21}, \alpha^{22}, \alpha^{26}\}$	$M_{11}(x) = (x-\alpha^{11})(x-\alpha^{13})(x-\alpha^{21})(x-\alpha^{22})(x-\alpha^{26})$ $= x^5 + x^4 + x^3 + x + 1$
$\{\alpha^{15}, \alpha^{23}, \alpha^{27}, \alpha^{29}, \alpha^{30}\}$	$M_{15}(x) = (x-\alpha^{15})(x-\alpha^{23})(x-\alpha^{27})(x-\alpha^{29})(x-\alpha^{30})$ $= x^5 + x^3 + 1$

5.10 Factoring $x^n - 1$

We now have the theoretical tools necessary to describe how to factor $x^n - 1$ over arbitrary finite fields for various values of n. When $n = q^m - 1$, from Theorem 5.22, every element of $GF(q^m)$ is a root of $x^{q^m-1} - 1$, so

$$x^{q^m-1} - 1 = \prod_{i=0}^{q^m-1} (x - \alpha^i) \tag{5.40}$$

for a primitive element $\alpha \in GF(q^m)$. To provide a factorization of $x^{q^m-1} - 1$ over the field $GF(q)$, the factors in $(x - \alpha^i)$ (5.40) can be grouped together according to conjugacy classes, which then multiply together to form minimal polynomials. Thus $x^{q^m-1} - 1$ can be expressed as a product of the minimal polynomials of the nonzero elements.

Example 5.39

1. The polynomial $x^7 - 1 = x^{2^3-1} - 1$ can be factored over $GF(2)$ as a product of the minimal polynomials shown in Table 5.2.

$$x^7 - 1 = (x+1)(x^3+x+1)(x^3+x^2+1).$$

2. The polynomial $x^{15} - 1 = x^{2^4-1} - 1$ can be factored over $GF(2)$ as a product of the minimal polynomials shown in Table 5.3.

$$x^{15} - 1 = (x+1)(x^4+x+1)(x^4+x^3+x^2+x+1)(x^2+x+1)(x^4+x^3+1).$$

□

We now pursue the slightly more general problem of factoring $x^n - 1$ when $n \neq q^m - 1$. An element $\beta \neq 1$ such that $\beta^n = 1$ is called an nth root of unity. The first step is to

Table 5.5: Conjugacy Classes over $GF(4^2)$ with Respect to $GF(4)$

Conjugacy Class	Minimal Polynomial
$\{0\}$	$M_-(x) = x$
$\{1\}$	$M_0(x) = x + 1$
$\{\alpha, \alpha^4\}$	$M_1(x) = (x+\alpha)(x+\alpha^4) = x^2 + x + \alpha^5$
$\{\alpha^2, \alpha^8\}$	$M_2(x) = (x+\alpha^2)(x+\alpha^8) = x^2 + x + \alpha^{10}$
$\{\alpha^3, \alpha^{12}\}$	$M_3(x) = (x+\alpha^3)(x+\alpha^{12}) = x^2 + \alpha^{10}x + 1$
$\{\alpha^5\}$	$M_5(x) = x + \alpha^5$
$\{\alpha^6, \alpha^9\}$	$M_6(x) = (x+\alpha^6)(x+\alpha^9) = x^2 + \alpha^5 x + 1$
$\{\alpha^7, \alpha^{13}\}$	$M_7(x) = (x+\alpha^7)(x+\alpha^{13}) = x^2 + \alpha^5 x + \alpha^5$
$\{\alpha^{10}\}$	$M_{10}(x) = x + \alpha^{10}$
$\{\alpha^{11}, \alpha^{14}\}$	$M_{11}(x) = (x+\alpha^{11})(x+\alpha^{14}) = x^2 + \alpha^{10}x + \alpha^{10}$

determine a field $GF(q^m)$ (that is, to determine m) in which nth roots of unity can exist. Once the field is found, factorization is accomplished using minimal polynomials in the field.

Theorem 5.20 tells us that if

$$n \mid q^m - 1, \tag{5.41}$$

then there are $\phi(n)$ elements of order n in $GF(q^m)$. Finding a field $GF(q^m)$ with nth roots of unity thus requires finding an m such that $n \mid q^m - 1$, which is usually done by trial and error.

Example 5.40 Determine an extension field $GF(3^m)$ in which 13th roots of unity exist. We see that $13 \mid 3^3 - 1$, so that 13th roots exist in the field $GF(3^3)$. □

Once the field is found, we let β be an element of order n in the field $GF(q^m)$. Then β is a root of $x^n - 1$ in that field, and so are the elements $\beta^2, \beta^3, \ldots, \beta^{n-1}$. That is,

$$x^n - 1 = \prod_{i=0}^{n-1} (x - \beta^i).$$

The roots are divided into conjugacy classes to form the factorization over $GF(q)$.

Example 5.41 Determine an extension field $GF(2^m)$ in which 5th roots of unity exist and express the factorization in terms of polynomials in $GF(2)[x]$. Using (5.41) we check:

$$5 \nmid (2-1) \quad 5 \nmid (2^2 - 1) \quad 5 \nmid (2^3 - 1) \quad 5 \mid (2^4 - 1).$$

So in $GF(16)$ there are primitive fifth roots of unity. For example, if we let $\beta = \alpha^3$, α primitive, then $\beta^5 = \alpha^{15} = 1$.

The roots of $x^5 - 1 = x^5 + 1$ in $GF(16)$ are

$$1, \beta, \beta^2, \beta^3, \beta^4,$$

which can be expressed in terms of the primitive element α as

$$1, \alpha^3, \alpha^6, \alpha^9, \alpha^{12}.$$

Using the minimal polynomials shown in Table 5.3 we have

$$x^5 + 1 = (x+1)M_3(x) = (x+1)(x^4 + x^3 + x^2 + x + 1).$$

□

Example 5.42 We want to find a field $GF(2^m)$ which has 25th roots of unity. We need

$$25 \mid (2^m - 1).$$

By trial and error we find that when $m = 20$, $25 \mid 2^m - 1$. Now let us divide the roots of $2^{20} - 1$ into conjugacy classes. Let β be a primitive 25th root of unity. The other roots of unity are the powers of β: $\beta^0, \beta^1, \beta^2, \ldots, \beta^{24}$. Let us divide these powers into conjugacy classes:

$$\{1\}$$
$$\{\beta, \beta^2, \beta^4, \beta^8, \beta^{16}, \beta^7, \beta^{14}, \beta^3, \beta^6 \beta^{12}, \beta^{24}, \beta^{23}, \beta^{21}, \beta^{17}, \beta^9, \beta^{18}, \beta^{11}, \beta^{22}, \beta^{19}, \beta^{13}\}$$
$$\{\beta^5, \beta^{10}, \beta^{20}, \beta^{15}\}$$

Letting $M_i(x) \in GF(2)[x]$ denote the minimal polynomial having β^i for the smallest i as a root, we have the factorization $x^{25} + 1 = M_0(x)M_1(x)M_5(x)$. □

Example 5.43 Let us find a field $GF(7^m)$ in which $x^{15} - 1$ has roots; this requires an m such that

$$15 \mid 7^m - 1.$$

$m = 4$ works. Let γ be a primitive 15th root of unity in $GF(7^4)$. Then $\gamma^0, \gamma^1, \ldots, \gamma^{14}$ are roots of unity. Let us divide these up into conjugacy classes with respect to $GF(7)$:

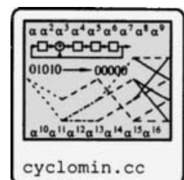
$$\{1\}, \{\gamma, \gamma^7, \gamma^{49} = \gamma^4, \gamma^{7^3} = \gamma^{13}\}, \{\gamma^2, \gamma^{14}, \gamma^8, \gamma^{11}\}, \{\gamma^3, \gamma^6, \gamma^{12}, \gamma^9\}, \{\gamma^5\}, \{\gamma^{10}\}$$

Thus $x^{15} - 1$ factors into six irreducible polynomials in $GF(7)$. □

5.11 Cyclotomic Cosets

Definition 5.14 The cyclotomic cosets modulo n with respect to $GF(q)$ contain the *exponents* of the n distinct powers of a primitive nth root of unity with respect to $GF(q)$, each coset corresponding to a conjugacy class. These cosets provide a shorthand representation for the conjugacy class. □

Example 5.44 For Example 5.43, $n = 15$ and $q = 7$. The cyclotomic cosets and the corresponding conjugacy classes are shown in Table 5.6. □



"Tables" of cyclotomic cosets and minimal polynomials are available using the program `cyclomin`.

Table 5.6: Cyclotomic Cosets modulo 15 with Respect to $GF(7)$

Conjugacy Class		Cyclotomic Cosets
$\{1\}$	$\leftrightarrow$	$\{0\}$
$\{\gamma, \gamma^7, \gamma^4, \gamma^{13}\}$	$\leftrightarrow$	{1,7,4,13}
$\{\gamma^2, \gamma^{14}, \gamma^8, \gamma^{11}\}$	$\leftrightarrow$	{2,14,8,11}
$\{\gamma^3, \gamma^6, \gamma^{12}, \gamma^9\}$	$\leftrightarrow$	{3,6,12,9}
$\{\gamma^5\}$	$\leftrightarrow$	{5}
$\{\gamma^{10}\}$	$\leftrightarrow$	{10}

Appendix 5.A How Many Irreducible Polynomials Are There?

The material in this appendix is not needed later in the book. However, it introduces several valuable analytical techniques and some useful facts.

A finite field $GF(q^m)$ can be constructed as an extension of $GF(q)$ if an irreducible polynomial of degree m over $GF(q)$ exists. The question of the existence of finite fields of order any prime power, then, revolves on the question of the existence of irreducible polynomials of arbitrary degree. Other interesting problems are related to *how many* such irreducible polynomials there are.

To get some insight into the problem, let us first do some exhaustive enumeration of irreducible polynomials with coefficients over $GF(2)$. Let I_n denote the number of irreducible polynomials of degree n. The polynomials of degree 1, x and $x+1$, are both irreducible, so $I_1 = 2$. The polynomials of degree 2 are

$$x^2 \text{ (reducible)} \qquad x^2 + 1 = (x+1)^2 \text{ (reducible)}$$
$$x^2 + x = x(x+1) \text{ (reducible)} \qquad x^2 + x + 1 \text{ (irreducible)}.$$

So $I_2 = 1$.

In general, there are 2^n polynomials of degree n. Each of these can either be factored into products of powers of irreducible polynomials of lower degree, or are irreducible themselves. Let us count how many different ways the set of binary cubics might factor. It can factor into a product of an irreducible polynomial of degree 2 and a polynomial of degree 1 in $I_2 I_1 = 2$ ways:

$$x(x^2 + x + 1) \qquad (x+1)(x^2 + x + 1).$$

It can factor into a product of three irreducible polynomials of degree 1 in four ways:

$$x^3 \qquad x^2(x+1) \qquad x(x+1)^2 \qquad (x+1)^3$$

The remaining cubic binary polynomials,

$$x^3 + x + 1 \quad \text{and} \quad x^3 + x^2 + 1$$

must be irreducible, so $I_3 = 2$.

This sort of counting can continue, but becomes cumbersome without some sort of mechanism to keep track of the various combinations of factors. This is accomplished using a *generating function* approach.

Definition 5.15 A **generating function** of a sequence $A_0, A_1, A_2, \ldots$ is the formal power series

$$A(z) = \sum_{k=0}^{\infty} A_k z^k.$$

□

The generating function is analogous to the z-transform of discrete-time signal processing, allowing us to formally manipulate sequences of numbers by polynomial operations. Generating functions $A(z)$ and $B(z)$ can be added (term by term), multiplied (using polynomial multiplication)

$$A(z)B(z) = \sum_{i=0}^{\infty} \left(\sum_{k=0}^{i} B_k A_{i-k} \right) z^i = \sum_{i=0}^{\infty} \left(\sum_{k=0}^{i} A_k B_{i-k} \right) z^i$$

and (formal) derivatives computed,

$$\text{if } A(z) = \sum_{k=0}^{\infty} A_k z^k \text{ then } A'(z) = \sum_{k=1}^{\infty} k A_k z^{k-1},$$

with operations taking place in some appropriate field.

The key theorem for counting the number of irreducible polynomials is the following.

Theorem 5.36 *Let $f(z)$ and $g(z)$ be relatively prime, monic irreducible polynomials over $GF(q)$ of degrees m and n, respectively. Let C_k be the number of monic polynomials of degree k whose only irreducible factors are $f(x)$ and $g(x)$. Then the moment generating function for C_k is*

$$C(z) = \frac{1}{1-z^m}\frac{1}{1-z^n}.$$

That is, $C_k = \sum_i B_i A_{k-i}$, where A_i is the ith coefficient in the generating function $A(z) = 1/(1-z^m)$ and B_i is the ith coefficient in the generating function $B(z) = 1/(1-z^n)$.

Example 5.45 Let $f(x) = x$ and $g(x) = x + 1$ in $GF(2)[x]$. The set of polynomials whose factors are $f(x)$ and $g(x)$ are those with linear factors, for example,

$$p(x) = (f(x))^a (g(x))^b, \qquad a, b \geq 0.$$

According to the theorem, the weight enumerator for the number of such polynomials is

$$\frac{1}{(1-z)^2}.$$

This can be shown to be equal to

$$\frac{1}{(1-z)^2} = \sum_{k=0}^{\infty} (k+1) z^k = 1 + 2z + 3z^2 + \cdots. \tag{5.42}$$

That is, there are 2 polynomials of degree 1 ($f(x)$ and $g(x)$), 3 polynomials of degree 2 ($f(x)g(x)$, $f(x)^2$ and $g(x)^2$), 4 polynomials of degree 3, and so on. □

Proof Let A_k be the number of monic polynomials in $GF(q)[x]$ of degree k which are powers of $f(x)$. The kth power of $f(x)$ has degree km, so

$$A_k = \begin{cases} 1 & \text{if } m \mid k \\ 0 & \text{otherwise.} \end{cases}$$

We will take $A_0 = 1$ (corresponding to $f(x)^0 = 1$). The generating function for the A_k is

$$A(z) = 1 + z^m + z^{2m} + \cdots = \frac{1}{1-z^m}.$$

$A(z)$ is called the *enumerator by degree* of the powers of $f(z)$.

Similarly, let B_k be the number of monic polynomials in $GF(q)[x]$ of degree k which are powers of $g(x)$; arguing as before we have $B(z) = 1/(1 - z^n)$.

With C_k the number of monic polynomials of degree k whose only factors are $f(x)$ and $g(x)$, we observe that if $\deg(g(x)^b) = nb = i$, then $\deg(f(x)^a) = ma = k - i$ for every $0 \leq i \leq k$. Thus

$$C_k = \sum_{i=0}^{k} B_i A_{k-i}$$

or, equivalently,

$$C(z) = A(z)B(z).$$

□

The theorem can be extended by induction to multiple sets of polynomials, as in the following corollary.

Corollary 5.37 *Let $S_1, S_2, \ldots, S_N$ be sets of polynomials such that any two polynomials in different sets are relatively prime. The set of polynomials which are products of a polynomial from each set has an enumerator by degree $\prod_{i=1}^{N} A_i(z)$, where $A_i(z)$ is the enumerator by degree of the set S_i.*

Example 5.46 For the set of polynomials formed by products of x, $x+1$ and $x^2+x+1 \in GF(2)[x]$, the enumerator by degree is

$$\left(\frac{1}{1-z}\right)^2 \frac{1}{1-z^2} = 1 + 2z + 4z^2 + 6z^3 + 9z^4 + \cdots$$

That is, there are 6 different ways to form polynomials of degree 3, and 9 different ways to form polynomials of degree 4. (Find them!) □

Let I_m be the number of monic irreducible polynomials of degree m. Applying the corollary, the set which includes I_1 irreducible polynomials of degree 1, I_2 irreducible polynomials of degree 2, and so forth, has the enumerator by the degree

$$\prod_{m=1}^{\infty} \left(\frac{1}{1-z^m}\right)^{I_m}.$$

Let us now extend this to a base field $GF(q)$. We observe that the set of all monic polynomials in $GF(q)[z]$ of degree k contains q^k polynomials in it. So the enumerator by degree of the set of polynomials of degree k is

$$\sum_{k=0}^{\infty} q^k z^k = \frac{1}{1-qz}.$$

Furthermore, the set of all products of powers of irreducible polynomials is precisely the set of all monic polynomials. Hence, we have the following.

Theorem 5.38 *[25, Theorem 3.32]*

$$\frac{1}{1-qz} = \prod_{m=1}^{\infty} \left(\frac{1}{1-z^m}\right)^{I_m}. \tag{5.43}$$

Equation (5.43) does not provide a very explicit formula for computing I_m. However, it can be manipulated into more useful forms. Reciprocating both sides we obtain

$$(1 - qz) = \prod_{m=1}^{\infty} (1 - z^m)^{I_m}. \tag{5.44}$$

Taking the formal derivative of both sides and rearranging we obtain

$$\frac{q}{1 - qz} = \sum_{m=1}^{\infty} \frac{m I_m z^{m-1}}{1 - z^m}. \tag{5.45}$$

Multiplying both sides by z and expanding both sides of (5.45) in formal series we obtain

$$\sum_{k=1}^{\infty} (qz)^k = \sum_{m=1}^{\infty} m I_m \sum_{k=1}^{\infty} (z^m)^k = \sum_{m=1}^{\infty} m I_m \sum_{\substack{k:k \neq 0 \\ m|k}} z^k = \sum_{k=1}^{\infty} \sum_{m:m|k} m I_m z^k.$$

Equating the kth terms of the sums on the left and right sides of this equation, we obtain the following theorem.

Theorem 5.39

$$q^k = \sum_{m|k} m I_m, \tag{5.46}$$

where the sum is taken of all m which divide k, including 1 and k.

This theorem has the following interpretation. By Theorem 5.29, in a field of order q, the product of all distinct monic polynomials whose degrees divide k divides $x^{q^k} - x$. The degree of $x^{q^k} - x$ is q^k, the left-hand side of (5.46). The degree of the product of all distinct monic polynomials whose degrees divide k is the sum of the degrees of those polynomials. Since there are I_m distinct monic irreducible polynomials, the contribution to the degree of the product of those polynomials is $m I_m$. Adding all of these up, we obtain the right-hand side of (5.46). This implies the following:

Theorem 5.40 *[25, Theorem 4.415] In a field of order q, $x^{q^k} - x$ factors into the product of all monic irreducible polynomials whose degrees divide k.*

Example 5.47 Let us take $q = 2$ and $k = 3$. The polynomials whose degrees divide $k = 3$ have degree 1 or 3. The product of the binary irreducible polynomials of degree 1 and 3 is

$$x(x + 1)(x^3 + x + 1)(x^3 + x^2 + 1) = x^8 + x.$$

□

Theorem 5.39 allows a sequence of equations to be built up for determining I_m for any m. Take for example $q = 2$:

$$\begin{aligned} k = 1: &\quad 2 = (1)I_1 &&\longrightarrow I_1 = 2 \\ k = 2: &\quad 4 = (1)I_1 + 2I_2 &&\longrightarrow I_2 = 1 \\ k = 3: &\quad 8 = (1)I_1 + 3I_3 &&\longrightarrow I_3 = 2. \\ &\quad \vdots \end{aligned}$$

Appendix 5.A.1 Solving for I_m Explicitly: The Moebius Function

However, equation (5.46) only implicitly determines I_m. An explicit formula can also be found. Equation (5.46) is a special case of a summation of the form

$$f(k) = \sum_{m|k} g(m). \tag{5.47}$$

in which $f(k) = q^k$ and $g(m) = mI_m$. Solving such equations for $g(m)$ can be accomplished using the number-theoretic function known as the Moebius (or Möbius) function μ.

Definition 5.16 The function $\mu(n) : \mathbb{Z}^+ \to \mathbb{Z}^+$ is the **Moebius function**, defined by

$$\mu(n) = \begin{cases} 1 & \text{if } n = 1 \\ (-1)^r & \text{if } n \text{ is the product of } r \text{ distinct primes} \\ 0 & \text{if } n \text{ contains any repeated prime factors.} \end{cases}$$

□

Theorem 5.41 *The Moebius function satisfies the following formula:*

$$\sum_{d|n} \mu(d) = \begin{cases} 1 & \text{if } n = 1 \\ 0 & \text{if } n > 1. \end{cases} \tag{5.48}$$

The proof is developed in Exercise 81. This curious "delta-function-like" behavior allows us to compute an inverse of some number-theoretic sums, as the following theorem indicates.

Theorem 5.42 Moebius inversion formula *If* $f(n) = \sum_{d|n} g(d)$ *then*

$$g(n) = \sum_{d|n} \mu(d) f(n/d).$$

Proof Let $d \mid n$. Then from the definition of $f(n)$, we have

$$f(n/d) = \sum_{k|(n/d)} g(k).$$

Multiplying both sides of this by $\mu(d)$ and summing over divisors d of n we obtain

$$\sum_{d|n} \mu(d) f(n/d) = \sum_{d|n} \sum_{k|(n/d)} \mu(d) g(k).$$

The order of summation can be interchanged as

$$\sum_{d|n} \mu(d) f(n/d) = \sum_{k|n} \sum_{d|(n/k)} \mu(d) g(k) = \sum_{k|n} g(k) \sum_{d|(n/k)} \mu(d).$$

By (5.48), $\sum_{d|(n/k)} \mu(d) = 1$ if $n/k = 1$, that is, if $n = k$, and is zero otherwise. So the double summation collapses down to a $g(n)$. □

Returning now to the problem of irreducible polynomials, (5.46) can be solved for I_m using the Moebius inversion formula of Theorem 5.42,

$$mI_m = \sum_{d|m} \mu(d) q^{(m/d)} \quad \text{or} \quad I_m = \frac{1}{m} \sum_{d|m} \mu(d) q^{(m/d)}. \tag{5.49}$$

Programming Laboratory 4: Programming the Euclidean Algorithm

Objective

The Euclidean algorithm is important both for modular arithmetic in general and also for specific decoding algorithms for BCH/Reed-Solomon codes. In this lab, you are to implement the Euclidean algorithm over both integers and polynomials.

Preliminary Exercises

Reading: Sections 5.2.2, 5.2.3.

1) In $\mathbb{Z}_5[x]$, determine $g(x) = (2x^5 + 3x^4 + 4x^3 + 3x^2 + 2x + 1, x^4 + 2x^3 + 3x^2 + 4x + 3)$ and also $s(x)$ and $t(x)$ such that

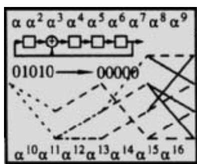
$$a(x)s(x) + b(x)t(x) = (a(x), b(x)).$$

2) Compute $(x^3 + 2x^2 + x + 4, x^2 + 3x + 4)$, operations in $\mathbb{R}[x]$, and also find polynomials $s(x)$ and $t(x)$ such that

$$a(x)s(x) + b(x)t(x) = (a(x), b(x)).$$

Background

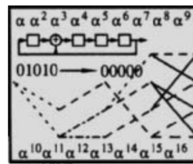
Code is provided which implements modulo arithmetic in the class `ModAr`, implemented in the files indicated in Algorithm 5.2.



Algorithm 5.2 Modulo Arithmetic
File: `ModAr.h`
`ModAr.cc`
`testmodar1.cc`
`ModArnew.h`
`testmodarnew.cc`

Code is also provided which implements polynomial arithmetic in the class `polynomialT`, using the files indicated in Algorithm 5.3. This class is templatized, so that the coefficients can come from a variety of fields or rings. For example, if you want a polynomial with `double` coefficients or `int` coefficients or `ModAr` coefficients, the objects are declared as

```
polynomialT<double> p1;
polynomialT<int> p2;
polynomialT<ModAr> p3;
```



Algorithm 5.3 Templatized Polynomials
File: `polynomialT.h`
`polynomialT.cc`
`testpoly1.cc`

Programming Part

1) Write a C or C++ function that performs the Euclidean algorithm on integers a and b, returning g, s, and t such that $g = as + bt$. The function should have declaration

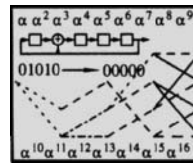
```
void gcd(int a, int b, int &g, int &s, int &t);
```

Test your algorithm on (24,18), (851,966), and other pairs of integers. Verify in each case that $as + bt = g$.

2) Write a function that computes the Euclidean algorithm on `polynomialT<TYPE>`. The function should have declaration

```
template <class T> void
gcd(const polynomialT<T> &a,
  const polynomialT<T> &b, polynomialT<T> &g,
    polynomialT<T> &s, polynomialT<T> &t);
```

Also, write a program to test your function. Algorithm 5.4 shows a test program and the framework for the program, showing how to instantiate the function with `ModAr` and `double` polynomial arguments.



Algorithm 5.4 Polynomial GCD
File: `testpolygcd.cc`
`gcdpoly.cc`

Test your algorithm as follows:

a) Compute $(3x^7 + 4x^6 + 3x^4 + x^3 + 1, 4x^4 + x^3 + x)$ and $t(x)$ and $s(x)$ for polynomials in $\mathbb{Z}_5[x]$. Verify that $a(x)s(x) + b(x)t(x) = g(x)$.

b) Compute $(2x^5 + 3x^4 + 4x^3 + 3x^2 + 2x + 1, x^4 + 2x^3 + 3x^2 + 4x + 3)$ and $s(x)$ and $t(x)$ for polynomials in $\mathbb{Z}_5[x]$. Verify that $a(x)s(x) + b(x)t(x) = g(x)$.

c) Compute $(2 + 8x + 10x^2 + 4x^3, 1 + 7x + 14x^2 + 8x^3)$ and $s(x)$ and $t(x)$ for polynomials in $\mathbb{R}[x]$. For polynomials with real coefficients, extra care must be taken to handle roundoff. Verify that $a(x)s(x) + b(x)t(x) = g(x)$.

3) Write a function which applies the Sugiyama algorithm to a sequence of data or its polynomial representation.

4) Test your algorithm over $\mathbb{Z}_5[x]$ by finding the shortest polynomial generating the sequence $\{3, 2, 3, 1, 4, 0, 4, 3\}$.

Having found $t(x)$, compute $b(x)t(x)$ and identify $r(x)$ and $s(x)$ and verify that they are consistent with the result found by the Sugiyama algorithm.

5) In $\mathbb{Z}_5[x]$, verify that the sequence $\{3, 2, 1, 0, 4, 3, 2, 1\}$ can be generated using the polynomial $t_1(x) = 1 + 2x + 4x^2 + 2x^3 + x^4$. Then use the Sugiyama algorithm to find the shortest polynomial $t(x)$ generating the sequence and verify that it works.

Programming Laboratory 5: Programming Galois Field Arithmetic

Objective

Galois fields are fundamental to algebraic block codes. This lab provides a tool to be used for BCH and Reed-Solomon codes. It builds upon the LFSR code produced in lab 2.

Preliminary Exercises

Reading: Section 5.4.

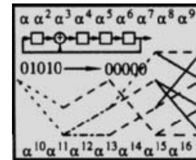
Write down the vector, polynomial, and power representations of the field $GF(2^3)$ generated with the polynomial $g(x) = 1 + x + x^3$. Based on this, write down the tables `v2p` and `p2v` for this field. (See the implementation suggestions for the definition of these tables.)

Programming Part

Create a C++ class `GFNUM2m` with overloaded operators to implement arithmetic over the field $GF(2^m)$ for an arbitrary $m < 32$. This is similar in structure to class `ModAr` class, except that the details of the arithmetic are different.

Test all operations of your class: `+`, `-`, `*`, `/`, `^`, `+=`, `-=`, `*=`, `/=`, `^=`, `==`, `!=` for the field generated by $g(x) = 1 + x + x^4$ by comparing the results the computer provides with results you calculate by hand. Then test for $GF(2^3)$ generated by $g(x) = 1 + x + x^3$.

The class `GFNUM2m` of Algorithm 5.5 provides the declarations and definitions for the class. In this representation, the field elements are represented intrinsically in the *vector* form, with the vector elements stored as the bits in a single `int` variable. This makes addition fast (bit operations). Multiplication of Galois field elements is easier when they are in exponential form and addition is easier when they are in vector form. Multiplication here is accomplished by converting to the power representation, adding the exponents, then converting back to the vector form. In `GFNUM2m`, all of the basic field operations are present except for completing the construction operator `initgf` which builds the tables `v2p` and `p2v`. The main programming task, therefore, is to build these tables. This builds upon the LFSR functions already written.



Algorithm 5.5 $GF(2^m)$
File: `GFNUM2m.h`
`GF2.h`
`GFNUM2m.cc`
`testgfnum.cc`

To make the conversion between the vector and power representations, two arrays are employed. The array `v2p` converts from vector to power representation and the array `p2v` converts from power to vector representation.

Example 5.48 In the field $GF(2^4)$ represented in Table 5.1, the field element $(1, 0, 1, 1)$ has the power representation α^7. The vector $(1, 0, 1, 1)$ can be expressed as an integer using binary-to-decimal conversion (LSB on the right) as 11. We thus think of 11 as the vector representation. The number `v2p[11]` converts from the vector representation, 11, to the exponent of the power representation, 7.

Turned around the other way, the number α^7 has the vector representation (as an integer) of 11. The number `p2v[7]` converts from the exponent of the power representation to the number 11. The conversion tables for the field are

i	v2p[i]	p2v[i]	i	v2p[i]	p2v[i]
0	–	1	8	3	5
1	0	2	9	14	10
2	1	4	10	9	7
3	4	8	11	7	14
4	2	3	12	6	15
5	8	6	13	13	13
6	5	12	14	11	9
7	10	11	15	12	–

□

To get the whole thing working, the arrays `p2v` and `v2p` need to be set up. To this end, a *static* member function `initgf(int m, int g)` is created. Given the degree of the extension `m` and the coefficients of $g(x)$ in the bits of the integer `g`, `initgf` sets up the conversion arrays. This can take advantage of the LFSR programmed in lab 2. Starting with an initial LFSR state of 1, the `v2p` and `p2v` arrays can be obtained by repeatedly clocking the LFSR: the state of the LFSR represents the vector representation of the Galois field numbers, while the number of times the LFSR has been clocked represents the power representation.

There are some features of this class which bear remarking on:

- The output format (when printing) can be specified in either vector or power form. In power form, something like `A^3` is printed; in vector form, an integer like `8` is printed. The format can be specified by invoking the static member function `setouttype`, as in

```
GFNUM2m::setouttype(vector);
   // set vector output format
GFNUM2m::setouttype(power);
   // set power output format
```

- The `v2p` and `p2v` arrays are stored as static arrays. This means that (for this implementation) all field elements must come from the same size field. It is not possible, for example, to have some elements to be $GF(2^4)$ and other elements to be $GF(2^8)$. (You may want to give some thought to how to provide for such flexibility in a memory efficient manner.)
- A few numbers are stored as static data in the class. The variable `gfm` represents the number m in $GF(2^m)$. The variable `gfN` represents the number $2^m - 1$. These should be set up as part of the `initgf` function. These numbers are used in various operators (such as multiplication, division, and exponentiation).
- Near the top of the header are the lines

```
extern GFNUM2m ALPHA;
   // set up a global alpha
extern GFNUM2m& A;
   // and a reference to alpha
   // for shorthand
```

These declare the variables `ALPHA` and `A`, the latter of which is a reference to the former. These variables can be used to represent the variable α in your programs, as in

```
GFNUM2m a;
a = (A^4) + (A^8);//
  // a is alpha^4 + alpha^8
```

The *definitions* of these variables should be provided in the `GFNUM2m.cc` file.

Write your code, then extensively test the arithmetic using $GF(2^4)$ (as shown above) and $GF(2^8)$. For the field $GF(2^8)$, use the primitive polynomial $p(x) = 1 + x^2 + x^3 + x^4 + x^8$:

```
GFNUM2m::initgf(8,0x11D);
   //   1 0001 1101
   //  x^8 + x^4 + x^3 + x^2 + 1
```

5.12 Exercises

5.1 Referring to the computations outlined in Example 5.1:

(a) Write down the polynomial equivalents for $\gamma_1, \gamma_2, \ldots, \gamma_{31}$. (That is, find the binary representation for γ_i and express it as a polynomial.)

(b) Write down the polynomial representation for γ_i^3, using operations modulo $M(x) = 1 + x^2 + x^5$.

(c) Explicitly write down the 10×31 binary parity check matrix H in (5.1).

5.2 Prove the statements in Lemma 5.1 that apply to integers.

5.3 [250] Let s and $g > 0$ be integers. Show that integers x and y exist satisfying $x + y = s$ and $(x, y) = g$ if and only if $g \mid s$.

5.4 [250] Show that if $m > n$, then $(a^{2^n} + 1) \mid (a^{2^m} - 1)$.

5.5 Wilson's theorem: Show that if p is a prime, then $(p-1)! \equiv -1 \pmod{p}$.

5.6 Let $f(x) = x^{10} + x^9 + x^5 + x^4$ and $g(x) = x^2 + x + 1$ be polynomials in $GF(2)[x]$. Write $f(x) = g(x)q(x) + r(x)$, where $\deg(r(x)) < \deg(g(x))$.

5.7 Uniqueness of division algorithm: Suppose that for integers $a > 0$ and b there are two representations

$$b = q_1 a + r_1 \qquad b = q_2 a + r_2,$$

with $0 \le r_1 < a$ and $0 \le r_2 < a$. Show that $r_1 = r_2$.

5.8 Let $R_a[b]$ be the remainder of b when divided by a, where a and b are integers. That is, by the division algorithm, $b = qa + R_a[b]$. Prove the following by relating both sides to the division algorithm.

(a) $R_a[b + c] = R_a[R_a[b] + R_a[c]]$.

(b) $R_a[bc] = R_a[R_a[b]R_a[c]]$.

(c) Do these results extend to polynomials?

5.9 Find the GCD g of 6409 and 42823. Also, find s and t such that $6409s + 42823t = g$.

5.10 Use the extended Euclidean algorithm over $\mathbb{Z}_p[x]$ to find $g(x) = (a(x), b(x))$ and the polynomials $s(x)$ and $t(x)$ such that $a(x)s(t) + b(x)t(x) = g(x)$ for

(a) $a(x) = x^3 + x + 1, b(x) = x^2 + x + 1$ for $p = 2$ and $p = 3$.

(b) $a(x) = x^6 + x^5 + x + 1, b(x) = x^4 + x^3$, $p = 2$ and $p = 3$.

5.11 Let $a \in \mathbb{Z}_n$. Describe how to use the Euclidean algorithm to find an integer $b \in \mathbb{Z}_n$ such that $ab = 1$ in $\mathbb{Z}_n$, if such a b exists and determine conditions when such a b exists.

5.12 Show that all Euclidean domains with a finite number of elements are fields.

5.13 Prove the GCD properties in Theorem 5.3.

5.14 Let a and b be integers. The **least common multiple** (LCM) m of a and b is the smallest positive integer such that $a \mid m$ and $b \mid m$. The LCM of a and b is frequently denoted $[a, b]$ (For polynomials $a(x)$ and $b(x)$, the LCM is the polynomial $m(x)$ of smallest degree such that $a(x) \mid m(x)$ and $b(x) \mid m(x)$.

(a) If s is any common multiple of a and b (that is, $a \mid r$ and $b \mid r$) and $m = [a, b]$ is the least common multiple of a and b, then $m \mid s$. *Hint*: division algorithm.

(b) Show that for $m > 0$, $[ma, mb] = m[a, b]$.

(c) Show that $[a, b](a, b) = |ab|$

5.15 Let $\mathcal{C}_1$ and $\mathcal{C}_2$ be cyclic codes generated by $g_1(x)$ and $g_2(x)$, respectively, with $g_1(x) \neq g_2(x)$. Let $\mathcal{C}_3 = \mathcal{C}_1 \cap \mathcal{C}_2$. Show that $\mathcal{C}_3$ is also a cyclic code and determine its generator polynomial $g_3(x)$. If d_1 and d_2 are the minimum distances of $\mathcal{C}_1$ and $\mathcal{C}_2$, respectively, what can you say about the minimum distance of $\mathcal{C}_3$?

5.16 Show that $\sum_{i=1}^{p-1} i^{-2} \equiv 0 \pmod{p}$, where p is a prime. *Hint*: The sum of the squares of the first n natural numbers is $n(n + 1)(2n + 1)/6$.

5.17 [360] Show that $\{as + bt : s, t \in \mathbb{Z}\} = \{k(a, b) : k \in \mathbb{Z}\}$ for all $a, b \in \mathbb{Z}$.

5.18 Show that the update equations for the extended Euclidean algorithm in (5.9) are correct. That is, show that the recursion (5.9) produces s_i and t_i satisfying the equation $as_i + bt_i = r_i$ for all i. *Hint*: Show for the initial conditions given in (5.10) that (5.8) is satisfied for $i = -1$ and $i = 0$. Then do a proof by induction.

5.19 [33] **A matrix formulation of the Euclidean algorithm**. For polynomials $a(x)$ and $b(x)$, use the notation $a(x) = \left\lfloor \frac{a(x)}{b(x)} \right\rfloor b(x) + r(x)$ to denote the division algorithm, where $q(x) = \left\lfloor \frac{a(x)}{b(x)} \right\rfloor$ is the quotient. Let $\deg(a(x)) > \deg(b(x))$. Let $a^{(0)}(x) = a(x)$ and $b^{(0)}(x) = b(x)$ and

$A^{(0)}(x) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$. Let

$$q^{(k)}(x) = \left\lfloor \frac{a^{(k)}(x)}{b^{(k)}(x)} \right\rfloor$$
$$A^{(k+1)}(x) = \begin{bmatrix} 0 & 1 \\ 1 & -q^{(k)}(x) \end{bmatrix} A^{(k)}(x)$$
$$\begin{bmatrix} a^{(k+1)}(x) \\ b^{(k+1)}(x) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & -q^{(k)}(x) \end{bmatrix} \begin{bmatrix} a^{(k)}(x) \\ b^{(k)}(x) \end{bmatrix}.$$

(a) Show that $\begin{bmatrix} a^{(k+1)}(x) \\ b^{(k+1)}(x) \end{bmatrix} = A^{(k+1)} \begin{bmatrix} a(x) \\ b(x) \end{bmatrix}$.

(b) Show that $b^{(K)}(x) = 0$ for some integer K.

(c) Show that $\begin{bmatrix} a^{(K)}(x) \\ 0 \end{bmatrix} = A^{(K)}(x) \begin{bmatrix} a(x) \\ b(x) \end{bmatrix}$. Conclude that any divisor of both $a(x)$ and $b(x)$ also divides $a^{(K)}(x)$. Therefore $(a(x), b(x)) \mid a^{(K)}(x)$.

(d) Show that

$$\begin{bmatrix} a(x) \\ b(x) \end{bmatrix} = \left[\prod_{k=0}^{K-1} \begin{bmatrix} q^{(k)}(x) & 1 \\ 1 & 0 \end{bmatrix} \right] \begin{bmatrix} a^{(K)}(x) \\ 0 \end{bmatrix}.$$

Hence conclude that $a^{(K)}(x) \mid a(x)$ and $a^{(K)}(x) \mid b(x)$, and therefore that $a^{(K)}(x) \mid (a(x), b(x))$.

(e) Conclude that $a^{(K)}(x) = \gamma(a(x), b(x))$ for some scalar γ. Furthermore show that $a^{(K)}(x) = A_{11}^{(K)}(x)a(x) + A_{12}^{(K)}(x)b(x)$.

5.20 [360] **More on the matrix formulation of the Euclidean algorithm.** Let

$$R^0 = \begin{bmatrix} s_{-1} & t_{-1} \\ s_0 & t_0 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad \text{and} \quad Q^{(i)} = \begin{bmatrix} 0 & 1 \\ 1 & -q^{(i)}(x) \end{bmatrix}$$

where $q^{(i)}(x) = \lfloor r_{i-2}/r_{i-1} \rfloor$ and let $R^{(i+1)} = Q^{(i+1)} R^{(i)}$. Show that:

(a) $\begin{bmatrix} r_{i-1}(x) \\ r_i(x) \end{bmatrix} = R^{(i)} \begin{bmatrix} a(x) \\ b(x) \end{bmatrix}$, $0 \le i \le K$, where K is the last index such that $r_K(x) \neq 0$.

(b) $R^{(i)} = \begin{bmatrix} s_{i-1} & t_{i-1} \\ s_i & t_i \end{bmatrix}$, $0 \le i \le K$.

(c) Show that any common divisor of r_i and r_{i+1} is a divisor of r_K and that $r_K \mid r_i$ and $r_K \mid r_{i+1}$ for $-1 \le i < K$.

(d) $s_i t_{i+1} - s_{i+1} t_i = (-1)^{i+1}$, so that $(s_i, t_i) = 1$. *Hint*: determinant of $R^{(i+1)}$.

(e) $s_i a + t_i b = r_i$, $-1 \le i \le K$.

(f) $(r_i, t_i) = (a, t_i)$

5.21 [234] Properties of the extended Euclidean algorithm. Let q_i, r_i, s_i, and t_i be defined as in Algorithm 5.1. Let n be the values of i such that $r_n = 0$ (the last iteration). Using proofs by

induction, show that the following relationships exist among these quantities:

$$\begin{aligned} t_i r_{i-1} - t_{i-1} r_i &= (-1)^i a & 0 \le i \le n \\ s_i r_{i-1} - s_{i-1} r_i &= (-1)^{i+1} b & 0 \le i \le n \\ s_i t_{i-1} - s_{i-1} t_i &= (-1)^{i+1} & 0 \le i \le n \\ s_i a + t_i b &= r_i & -1 \le i \le n \\ \deg(s_i) + \deg(r_{i-1}) &= \deg(b) & 1 \le i \le n \\ \deg(t_i) + \deg(r_{i-1}) &= \deg(a) & 0 \le i \le n \end{aligned}$$

5.22 Continued fractions and the Euclidean algorithm. Let u_0 and u_1 be in a field $\mathbb{F}$ or ring of polynomials over a field $\mathbb{F}[x]$. A continued fraction representation of the ratio u_0/u_1 is a fraction of the form

$$\frac{u_0}{u_1} = a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cdots \cfrac{1}{a_{j-1} + \cfrac{1}{a_j}}}}. \tag{5.50}$$

For example,

$$\frac{51}{22} = 2 + \cfrac{1}{3 + \cfrac{1}{7}}.$$

The continued fraction (5.50) can be denoted as $\langle a_0, a_1, \ldots, a_j \rangle$.

(a) Given u_0 and u_1, show how to use the Euclidean algorithm to find the $a_0, a_1, \ldots, a_j$ in the continued fraction representation of u_0/u_1. *Hint*: by the division algorithm, $u_0 = u_1 a_0 + u_2$. This is equivalent to $u_0/u_1 = a_0 + 1/(u_1/u_2)$.

(b) Determine the continued fraction expansion for $u_0 = 966$, $u_1 = 815$. Verify that it works.

(c) Let $u_0(x), u_1(x) \in \mathbb{Z}_5[x]$, where $u_0(x) = 1 + 2x + 3x^2 + 4x^3 + 3x^5$ and $u_1(x) = 1 + 3x^2 + 2x^3$. Determine the continued fraction expansion for $u_0(x)/u_1(x)$.

5.23 [234] Padé Approximation and the Euclidean Algorithm. Let $A(x) = a_0 + a_1 x + a_2 x^2 + \cdots$ be a power series with coefficients in a field $\mathbb{F}$. A (μ, ν) Padé approximant to $A(x)$ is a rational function $p(x)/q(x)$ such that $q(x)A(x) \equiv p(x) \pmod{x^{N+1}}$, where $\mu + \nu = N$ and where $\deg(p(x) \le \mu$ and $\deg(q(x)) \le \nu$. That is, $A(x)$ agrees with the expansion $p(x)/q(x)$ for terms up to x^N. The Padé condition can be written as $q(x)A_N(x) \equiv p(x) \pmod{x^{N+1}}$, where $A_N(x)$ is the Nth truncation of $A(x)$,

$$A_N(x) = a_0 + a_1 x + \cdots + a_N x^N.$$

(a) Describe how to use the Euclidean algorithm to obtain a sequence of polynomials $r_j(x)$ and $t_j(x)$ such that $t_i(x)A_N(x) \equiv r_j(x) \pmod{x^{N+1}}$.

(b) Let $\mu + \nu = \deg(a(x)) - 1$, with $\mu \ge \deg((a(x), b(x))$. Show that there exists a unique index j such that $\deg(r_j) \le \mu$ and $\deg(t_j) \le \nu$. *Hint*: See the last property in Exercise 5.21

(c) Let $A(x) = 1+2x+x^3+3x^7+x^9+\cdots$ be a power series. Determine a Padé approximation with $\mu = 5$ and $\nu = 3$; that is, an approximation to the truncated series $A_8(x) = 1 + 2x + x^3 + 3x^7$.

5.24 Let I_1 and I_2 be ideals in $\mathbb{F}[x]$ generated by $g_1(x)$ and $g_2(x)$, respectively.

(a) The least common multiple of $g_1(x)$ and $g_2(x)$ is the polynomial $g(x)$ of smallest degree such that $g_1(x) \mid g(x)$ and $g_2(x) \mid g(x)$. Show that $I_1 \cap I_2$ is generated by the least common multiple of $g_1(x)$ and $g_2(x)$.

(b) Let $I_1 + I_2$ mean the smallest ideal which contains I_1 and I_2. Show that $I_1 + I_2$ is generated by the greatest common divisor of $g_1(x)$ and $g_2(x)$.

5.25 Using the extended Euclidean algorithm, determine the shortest linear feedback shift register that could have produced the sequence $[1, 4, 2, 2, 4, 1]$ with elements in $\mathbb{Z}_5$.

5.26 Prove statements (1) – (9) of Theorem 5.8.

5.27 If x is an even number, then $x \equiv 0 \pmod 2$. What congruence does an odd integer satisfy? What congruence does an integer of the form $x = 7k + 1$ satisfy?

5.28 [250] Write a single congruence equivalent to the pair of congruences $x \equiv 1 \pmod 4$ and $x \equiv 2 \pmod 3$.

5.29 Compute: $\phi(190), \phi(191), \phi(192)$.

5.30 [250] Prove the following divisibility facts:

(a) $n^6 - 1$ is divisible by 7 if $(n, 7) = 1$

(b) $n^7 - n$ is divisible by 42 for any integer n.

(c) $n^{12} - 1$ is divisible by 7 if $(n, 7) = 1$.

(d) $n^{6k} - 1$ is divisible by 7 if $(n, 7) = 1$, k a positive integer.

(e) $n^{13} - n$ is divisible by 2, 3, 5, 7, and 13 for any positive integer n.

5.31 Show that

$$\phi(n) = n \prod_{p \mid n} \left(1 - \frac{1}{p}\right),$$

where the product is taken over all primes p dividing n. *Hint*: write $n = p_1^{e_1} p_2^{e_2} \cdots p_r^{e_r}$.

5.32 In this exercise, you will prove an important property of the Euler ϕ function:

$$\sum_{d \mid n} \phi(d) = n. \tag{5.51}$$

where the sum is over all the numbers d that divide n.

(a) Suppose $n = p^e$, where p is prime. Show that (5.51) is true.

(b) Now proceed by induction. Suppose that (5.51) is true for integers with k or fewer distinct prime factors. Consider any integer N with $k + 1$ distinct prime factors. Let p denote one of the prime factors of N and let p^e be the highest power of p that divides N. Then $N = p^e n$, where n has k distinct prime factors. As d ranges over the divisors of n, the set $d, pd, p^2 d, \ldots, p^e d$ ranges over the divisors of N. Now complete the proof.

5.33 Let G_n be the elements in $\mathbb{Z}_n$ that are relatively prime to n. Show that G_n forms a group under multiplication.

5.34 RSA Encryption: Let $p = 97$ and $q = 149$. Encrypt the message $m = 1234$ using the public key $\{e, n\} = \{35, 14453\}$. Determine the private key $\{d, n\}$. Then decrypt.

5.35 A message is encrypted using the public key $\{e, n\} = \{23, 64777\}$. The encrypted message is $c = 1216$. Determine the original message m. (That is, crack the code.)

5.36 Find all integers that simultaneously satisfy the congruences $x \equiv 2 \pmod 4$, $x \equiv 1 \pmod 9$ and $x \equiv 2 \pmod 5$.

5.37 Find a polynomial $f(x) \in \mathbb{Z}_5[x]$ simultaneously satisfying the three congruences

$$f(x) \equiv 2 \pmod{x - 1} \qquad f(x) \equiv 3 + 2x \pmod{(x-2)^2}$$
$$f(x) \equiv x^2 + 3x + 2 \pmod{(x-3)^3}$$

5.38 Evaluation homomorphism: Show that $f(x) \pmod{x-u} = f(u)$. Show that $\pi : \mathbb{F}[x] \to \mathbb{F}$ defined by $\pi_u(f(x)) = f(u)$ is a ring homomorphism.

5.39 Determine a Lagrange interpolating polynomial $f(x) \in \mathbb{R}[x]$ such that

$$f(1) = 3 \qquad f(2) = 6 \qquad f(3) = 1.$$

5.40 Let $(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)$ be points.

(a) Write down the Lagrange interpolants $l_i(x)$ for these points.

(b) Write down a polynomial $f(x)$ that interpolates through these points.

5.41 An interesting identity. In this exercise you will prove that for $p_1, p_2, \ldots, p_N$ all distinct, the following identity holds:

$$\sum_{i=1}^{N} \prod_{n=1, n\neq i}^{N} \frac{p_i}{p_i - p_n} = 1$$

in any field.

(a) Verify the identity when $N = 2$ and $N = 3$.

(b) Let $f(x) = x^{N-1}$. Find a Lagrange interpolating polynomial $g(x)$ for the points (p_1, p_1^{N-1}), $(p_2, p_2^{N-1}), \ldots, (p_N, p_N^{N-1})$.

(c) Determine the $(N-1)$st derivative of $g(x)$, $g^{(N-1)}(x)$.

(d) Determine the $(N-1)$st derivative of $f(x)$, $f^{(N-1)}(x)$.

(e) Show that the identity is true.

(f) Based on this identity, prove the following facts:

i. $\sum_{i=1}^{N} \prod_{n=1,n\neq i}^{N} \frac{i}{i-n} = 1$.

ii. $\frac{1}{N!} \sum_{i=1}^{N} (-1)^{N-i} \binom{N}{i} i^N = 1$.

iii. $\sum_{i=1}^{N} \prod_{n=1,n\neq i}^{N} \frac{n}{n-i} = 1$.

iv. $\sum_{i=1}^{N} (-1)^{i-1} \binom{N}{i} = 1$.

v. $\sum_{i=1}^{N} \prod_{n=1,n\neq i}^{N} \frac{1}{1-x^{n-i}} = 1, x \neq 1$.

vi. $\sum_{i=1}^{N} \prod_{n=1,n\neq i}^{N} \frac{1}{1-(n/i)^x} = 1$ for all $x \neq 0$.

5.42 Let $l_j(x)$ be a Lagrange interpolant, as in (5.25). Show that $\sum_{j=1}^{r} l_j(x) = 1$.

5.43 Show that $x^5 + x^3 + 1$ is irreducible over $GF(2)$.

5.44 Determine whether each of the following polynomials in $GF(2)[x]$ is irreducible. If irreducible, determine if it is also primitive.

(a) $x^2 + 1$
(b) $x^2 + x + 1$
(c) $x^3 + x + 1$
(d) $x^4 + x^2 + 1$
(e) $x^4 + x^2 + x^2 + x + 1$.
(f) $x^4 + x^3 + x + 1$
(g) $x^5 + x^3 + x^2 + x + 1$
(h) $x^5 + x^2 + 1$
(i) $x^6 + x^5 + x^4 + x + 1$

5.45 Let $p(x) \in GF(2)[x]$ be $p(x) = x^4 + x^3 + x^2 + x + 1$. This polynomial is irreducible. Let this polynomial be used to create a representation of $GF(2^4)$. Using this representation of $GF(2^4)$, do the following:

(a) Let α be a root of $p(x)$. Show that α is not a primitive element.

(b) Show that $\beta = \alpha + 1$ is primitive.

(c) Find the minimal polynomial of $\beta = \alpha + 1$.

5.46 Solve the following set of equations over $GF(2^4)$, using the representation in Table 5.1.

$$\alpha x + \alpha^5 y + z = \alpha$$
$$\alpha^2 x + \alpha^3 y + \alpha^8 z = \alpha^4$$
$$\alpha^2 x + y + \alpha^9 z = \alpha^{10}.$$

5.47 Show that $(x - \beta)$ is a factor of a polynomial $f(x)$ if and only if $f(\beta) = 0$.

5.48 Create a table such as Table 5.1 for the field $GF(2^3)$ generated by the primitive polynomial $x^3 + x + 1$, including the Zech logarithms.

5.49 Construct the field $GF(8)$ using the primitive polynomial $p(x) = 1 + x^2 + x^3$, producing a table similar to Table 5.1. Use β to represent the root of $p(x)$: $\beta^3 + \beta^2 + 1 = 0$.

5.50 Extension of $GF(3)$:

(a) Prove that $p(x) = x^2 + x + 2$ is irreducible in $GF(3)$.

(b) Construct the field $GF(3^2)$ using the primitive polynomial $x^2 + x + 2$.

5.51 Let $g(x) = (x^2 - 3x + 2) \in \mathbb{R}[x]$. Show that in $\mathbb{R}[x]/\langle g(x) \rangle$ there are zero divisors, so that this does not produce a field.

5.52 Let $f(x)$ be a polynomial of degree n over $GF(2)$. The reciprocal of $f(X)$ is defined as

$$f^*(x) = x^n f(1/x)$$

(a) Find the reciprocal of the polynomial

$$f(x) = 1 + x + x^5.$$

(b) Let $f(x)$ be a polynomial with nonzero constant term. Prove that $f(x)$ is irreducible over $GF(2)$ if and only if $f^*(x)$ is irreducible over $GF(2)$.

(c) Let $f(x)$ be a polynomial with nonzero constant term. Prove the $f(x)$ is primitive if and only if $f^*(x)$ is primitive.

5.53 Extending Theorem 5.15. Show that

$$\left(\sum_{i=1}^{k} x_i\right)^p = \sum_{i=1}^{k} x_i^p.$$

Show that

$$(x + y)^{p^r} = x^{p^r} + y^{p^r}.$$

5.54 Prove Lemma 5.26.

5.55 Show that, over any field, $x^s - 1 \mid x^r - 1$ if and only if $s \mid r$.

5.56 [25, p. 29] Let $d = (m, n)$. Show that $(x^m - 1, x^n - 1) = x^d - 1$. *Hint*: Let r_k denote the remainders for the Euclidean algorithm over integers computing (m, n), with $r_{k-2} = q_k r_{k-1} + r_k$, and let $r^{(k)}$ be the remainder for the Euclidean algorithm over polynomials computing $(x^m - 1, x^n - 1)$. Show that

$$x^{r_k} \left[\sum_{i=0}^{q_k - 1} (x^{r_{k-1}})^i \right] (x^{r_{k-1}} - 1) = x^{q_k r_{k-1} + r_k} - x^{r_k},$$

so that

$$x^{r_{k-1}} - 1 = x^{r_k} \left[\sum_{i=0}^{q_k - 1} (x^{r_{k-1}})^i \right] (x^{r_{k-1}} - 1) + x^{r_k} - 1.$$

5.57 The set of Gaussian integers $\mathbb{Z}[i]$ is made by adjoining $i = \sqrt{-1}$ to $\mathbb{Z}$:

$$\mathbb{Z}[i] = \{a + bi : a, b \in \mathbb{Z}\}.$$

(This is analogous to adjoining $i = \sqrt{-1}$ to $\mathbb{R}$ to form $\mathbb{C}$.) For $a \in \mathbb{Z}[i]$, define the valuation function $v(a) = aa^*$, where * denotes complex conjugation. $v(a)$ is also called the **norm**. $\mathbb{Z}[i]$ with this valuation forms a Euclidean domain.

(a) Show that $v(a)$ is a Euclidean function. *Hint*: Let $a = a_1 + a_2 i$ and $b = b_1 + b_2 i$, $a/b = Q + iS$ with $Q, S \in \mathbb{Q}$ and $q = q_1 + iq_2$, $q_1, q_2 \in \mathbb{Z}$ be the nearest integer point to (a/b) and let $r = a - bq$. Show that $v(r) < v(b)$ by showing that $v(r)/v(b) < 1$.

(b) Show that the units in $\mathbb{Z}[i]$ are the elements with norm 1.

(c) Compute the greatest common divisors of 6 and $3 + i$ in $\mathbb{Z}[i]$. Express them as linear combinations of 6 and $3 + i$.

5.58 The **trace** is defined as follows: For $\beta \in GF(p^r)$, $\mathrm{Tr}(\beta) = \beta + \beta^p + \beta^{p^2} + \cdots + \beta^{p^{r-1}}$. Show that the trace has the following properties:

(a) For every $\beta \in GF(p^r)$, $\mathrm{Tr}(\beta) \in GF(p)$.

(b) There is an element $\beta \in GF(p^r)$ such that $\mathrm{Tr}(\beta) \neq 0$.

(c) The trace is a $GF(p)$-linear function. That is, for $\beta, \gamma \in GF(p)$ and $\delta_1, \delta_2 \in GF(p^r)$,

$$\mathrm{Tr}[\beta\delta_1 + \gamma\delta_2] = \beta\,\mathrm{Tr}[\delta_1] + \gamma\,\mathrm{Tr}[\delta_2].$$

5.59 Square roots in finite fields: Show that every element in $GF(2^m)$ has a square root. That is, for every $\beta \in GF(2^m)$, there is an element $\gamma \in GF(2^m)$ such that $\gamma^2 = \beta$.

5.60 [360, p. 238] Let $q = p^m$ and let t be a divisor of $q - 1$, with prime factorization $t = p_1^{e_1} p_2^{e_2} \cdots p_r^{e_r}$. Prove the following:

(a) For $\alpha \in GF(q)$ with $\alpha \neq 0$, $\mathrm{ord}(\alpha) = t$ if and only if $\alpha^t = 1$ and $\alpha^{t/p_i} \neq 1$ for $i = 1, 2, \ldots, r$.

(b) $GF(q)$ contains an element β_i of order $p_i^{e_i}$ for $i = 1, 2, \ldots, r$.

(c) If $\alpha \in GF(q)$ and $\beta \in GF(q)$ have $(\mathrm{ord}(\alpha), \mathrm{ord}(\beta)) = 1$, then $\mathrm{ord}(\alpha\beta) = \mathrm{ord}(\alpha)\,\mathrm{ord}(\beta)$.

(d) $GF(q)$ has an element of order t.

(e) $GF(q)$ has a primitive element.

5.61 Let $f(x) = (x - a_1)^{r_1} \cdots (x - a_l)^{r_l}$. Let $f'(x)$ be the formal derivative of $f(x)$. Show that $(f(x), f'(x)) = (x - a_1)^{r_1 - 1} \cdots (x - a_l)^{r_l - 1}$.

5.62 The polynomial $f(x) = x^2 - 2$ is irreducible over $\mathbb{Q}[x]$ because $\sqrt{2}$ is irrational. Prove that $\sqrt{2}$ is irrational.

5.63 Express the following as products of binary irreducible polynomials over $GF(2)[x]$. **(a)** $x^7 + 1$. **(b)** $x^{15} + 1$.

5.64 Construct all binary cyclic codes of length 7.

5.65 Refer to Theorem 4.1. List all of the distinct ideals in the ring $GF(2)[x]/(x^{15} - 1)$ by their generators.

5.66 [373] List by dimension all of the binary cyclic codes of length 31.

5.67 [373] List by dimension all of the 8-ary cyclic codes of length 33. *Hint*:

$$x^{33} - 1 = (x + 1)(x^2 + x + 1)(x^{10} + x^7 + x^5 + x^3 + 1)(x^{10} + x^9 + x^5 + x + 1)$$
$$(x^{10} + x^9 + x^8 + x^7 + x^6 + x^5 + x^4 + x^3 + x^2 + x + 1).$$

5.68 List the dimensions of all the binary cyclic codes of length 19. *Hint*:

$$x^{19} + 1 = (x + 1) \sum_{i=0}^{18} x^i.$$

5.69 Let β be an element of $GF(2^m)$, with $\beta \neq 0$ and $\beta \neq 1$. Let $q(x)$ be the minimal polynomial of β. What is the shortest cyclic code with $q(x)$ as the generator polynomial?

5.70 Let $\beta \in GF(q)$ have minimal polynomial $m(x)$ of degree d. Show that the reciprocal $x^d m(1/x)$ is the minimal polynomial of β^{-1}.

5.71 Let $\alpha \in GF(2^{10})$ be primitive. Find the conjugates of α with respect to $GF(2)$, $GF(4)$, and $GF(32)$.

5.72 In the field $GF(9)$ constructed with the primitive polynomial $x^2 + x + 2$ (see Exercise 5.50), determine the minimal polynomials of all the elements with respect to $GF(3)$ and determine the cyclotomic cosets.

5.73 [373] Determine the degree of the minimal polynomial of β with respect to $GF(2)$ for a field element β with the following orders: **(a)** $\text{ord}(\beta) = 3$. Example: $\{\beta, \beta^2, \beta^4 = \beta\}$, so there are two conjugates. Minimal polynomial has degree 2. **(b)** $\text{ord}(\beta) = 5$ **(c)** $\text{ord}(\beta) = 7$. **(d)** $\text{ord}(\beta) = 9$

5.74 For each of the following polynomials, determine a field $GF(2^m)$ where the polynomials can be factored into polynomials in $GF(2)[x]$. Determine the cyclotomic cosets in each case and the number of binary irreducible polynomials in the factorization.

(a) $x^9 + 1$
(b) $x^{11} + 1$
(c) $x^{13} + 1$
(d) $x^{17} + 1$
(e) $x^{19} + 1$
(f) $x^{29} + 1$

5.75 Let $g(x)$ be the generator of a cyclic code over $GF(q)$ of length n and let q and n be relatively prime. Show that the vector of all 1s is a codeword if and only if $g(1) \neq 0$.

5.76 Let $g(x) = x^9 + \beta^2 x^8 + x^6 + x^5 + \beta^2 x^2 + \beta^2$ be the generator for a cyclic code of length 15 over $GF(4)$, where $\beta = \beta^2 + 1$ is primitive over $GF(4)$.

(a) Determine $h(x)$.
(b) Determine the dimension of the code.
(c) Determine the generator matrix G and the parity check matrix H for this code.
(d) Let $r(x) = \beta x^3 + \beta^2 x^4$. Determine the syndrome for $r(x)$.
(e) Draw a circuit for a systematic encoder for this code.
(f) Draw a circuit which computes the syndrome for this code.

5.77 [373] Let $\langle f(x), h(x) \rangle$ be the ideal $I \subset GF(2)[x]/(x^n - 1)$ formed by all linear combinations of the form $a(x)f(x) + b(x)g(x)$, where $a(x), b(x) \in GF(2)[x]/(x^n - 1)$. By Theorem 4.1, I is principal. Determine the generator for I.

5.78 Let $A(z) = \sum_{n=0}^{\infty} A_n z^n$ and $B(z) = \sum_{n=0}^{\infty} B_n z^n$ be generating functions and let $C(z) = A(z)B(z)$. Using the property of a formal derivative that $A'(z) = \sum_{n=0}^{\infty} n A_n z^{n-1}$, show that $C'(z) = A'(z)B(z) + A(z)B'(z)$. By extension, show that if $C(z) = \prod_{i=1}^{k} A_i(z)$, show that

$$\frac{\left[\prod_{i=1}^{k} A_i(z)\right]'}{\prod_{i=1}^{k} A_i(z)} = \sum_{i=1}^{k} \frac{[A_i(z)]'}{A_i(z)}.$$

5.79 Show how to make the transition from (5.44) to (5.45).

5.80 Show that:

(a) For an LFSR with an irreducible generator $g(x)$ of degree p, the period of the sequence is a factor of $2^p - 1$.

(b) If $2^p - 1$ is prime, then every irreducible polynomial of degree p produces a maximal-length shift register.

Incidentally, if $2^n - 1$ is a prime number, it is called a **Mersenne prime**. A few values of n which yield Mersenne primes are: $n = 2, 3, 5, 7, 13, 17, 19, 31, 61, 89$.

5.81 Prove Theorem 5.41.

5.82 Using Exercise 5.32 and Theorem 5.42, show that

$$\phi(n) = \sum_{d|n} \mu(d)(n/d).$$

5.83 Let $q = 2$. Use (5.49) to determine I_1, I_2, I_3, I_4, and I_5.

5.13 References

The discussion of number theory was drawn from [250] and [360]. Discussions of the computational complexity of the Euclidean algorithm are in [187] and [360]. The Sugiyama algorithm is discussed in [324] and [234]. Fast Euclidean algorithm implementations are discussed in [34]. Continued fractions are discussed in [250]; Padé approximation appears in [234, 239, 360] A delightful introduction to applications of number theory appears in [304]. Modern algebra can be found in [106, 360, 373, 31, 25]. More advanced treatments can be found in [155, 162], while a thorough treatment of finite fields is in [200, 201]. Zech logarithms and some of their interesting properties are presented in [304]. Our discussion of hardware implementations is drawn from [203], while the format of the add/multiply tables inside the back cover is due to [274]. Discussion of irreducible polynomials was drawn closely from [25]. The RSA algorithm was presented first in [293]; for introductory discussions, see also [360] and [304]. The algebraic approach using the Chinese Remainder Theorem for fast transforms in multiple dimensions is explored in [277].

Chapter 6

BCH and Reed-Solomon Codes: Designer Cyclic Codes

The most commonly used cyclic error correcting codes are the BCH and Reed-Solomon codes. The BCH code is named for Bose, Ray-Chaudhuri, and Hocquenghem (see the references at the end of the chapter), who published work in 1959 and 1960 which revealed a means of designing codes over $GF(2)$ with a specified design distance. Decoding algorithms were then developed by Peterson and others.

The Reed-Solomon codes are also named for their inventors, who published in 1960. It was later realized that Reed-Solomon (RS) codes and BCH codes are related and that their decoding algorithms are quite similar.

This chapter describes the construction of BCH and RS codes and several decoding algorithms. Decoding of these codes is an extremely rich area. Chapter 7 describes other "modern" decoding algorithms.

6.1 BCH Codes

6.1.1 Designing BCH Codes

BCH codes are cyclic codes and hence may be specified by a generator polynomial. A BCH code over $GF(q)$ of length n capable of correcting at least t errors is specified as follows:

1. Determine the smallest m such that $GF(q^m)$ has a primitive nth root of unity β.
2. Select a nonnegative integer b. Frequently, $b = 1$.
3. Write down a list of $2t$ *consecutive* powers of β:
$$\beta^b, \beta^{b+1}, \ldots, \beta^{b+2t-1}.$$
Determine the minimal polynomial with respect to $GF(q)$ of each of these powers of β. (Because of conjugacy, frequently these minimal polynomials are not distinct.)
4. The generator polynomial $g(x)$ is the least common multiple (LCM) of these minimal polynomials. The code is a $(n, n - \deg(g(x)))$ cyclic code.

Because the generator is constructed using minimal polynomials with respect to $GF(q)$, the generator $g(x)$ has coefficients in $GF(q)$, and the code is over $GF(q)$.

Definition 6.1 If $b = 1$ in the construction procedure, the BCH code is said to be **narrow sense**. If $n = q^m - 1$ then the BCH code is said to be **primitive**. □

Two fields are involved in the construction of the BCH codes. The "small field" $GF(q)$ is where the generator polynomial has its coefficients and is the field where the elements of the codewords are. The "big field" $GF(q^m)$ is the field where the generator polynomial has its roots. For encoding purposes, it is sufficient to work only with the small field. However, as we shall see, decoding requires operations in the big field.

Example 6.1 Let $n = 31 = 2^5 - 1$ for a primitive code with $m = 5$ and let β be a root of the primitive polynomial $x^5 + x^2 + 1$ in $GF(2^5)$. (That is, β is an element with order n.)

Let us take $b = 1$ (narrow sense code) and construct a single-error correcting binary BCH code. That is, we have $t = 1$. The $2t$ consecutive powers of β are β, β^2. The minimal polynomials of β and β^2 with respect to $GF(2)$ are the same (they are conjugates). Let us denote this minimal polynomial by $M_1(x)$. Since β is primitive, $M_1(x) = x^5 + x^2 + 1$ (see Table 5.4). Then

$$g(x) = M_1(x) = x^5 + x^2 + 1.$$

Since $\deg(g) = 5$, we have a (31,26) code. (This is, in fact, a Hamming code.) □

Example 6.2 As before, let $n = 31$, but now construct a $t = 2$-error correcting code, with $b = 1$, and let β be a primitive element. We form $2t = 4$ consecutive powers of β: $\beta, \beta^2, \beta^3, \beta^4$. Dividing these into conjugacy classes with respect to $GF(2)$, we have $\{\beta, \beta^2, \beta^4\}, \{\beta^3\}$. Using Table 5.4 we find the minimal polynomials

$$M_1(x) = x^5 + x^2 + 1 \qquad M_3(x) = x^5 + x^4 + x^3 + x^2 + 1$$

so that

$$\begin{aligned} g(x) &= \text{LCM}[M_1(x), M_3(x)] = (x^5 + x^2 + 1)(x^5 + x^4 + x^3 + x^2 + 1) \\ &= x^{10} + x^9 + x^8 + x^6 + x^5 + x^3 + 1. \end{aligned}$$

This gives a $(31, 31 - 10) = (31, 21)$ binary cyclic code. □

Example 6.3 As before, let $n = 31$, but now construct a $t = 3$-error correcting code, with $b = 1$. We form $2t = 6$ consecutive powers of β:

$$\beta, \beta^2, \beta^3, \beta^4, \beta^5, \beta^6.$$

Divided into conjugacy classes with respect to $GF(2)$, we have

$$\{\beta, \beta^2, \beta^4\}, \{\beta^3, \beta^6\}, \{\beta^5\}.$$

Denote the minimal polynomials for these sets as $M_1(x)$, $M_3(x)$ and $M_5(x)$, respectively. Using Table 5.4 we find that the minimal polynomials for elements in these classes are

$$M_1(x) = x^5 + x^2 + 1 \qquad M_3(x) = x^5 + x^4 + x^3 + x^2 + 1 \qquad M_5(x) = x^5 + x^4 + x^2 + x + 1$$

so the generator is

$$\begin{aligned} g(x) &= \text{LCM}[M_1(x), M_3(x), M_5(x)] = M_1(x)M_3(x)M_5(x) \\ &= x^{15} + x^{11} + x^{10} + x^9 + x^8 + x^7 + x^5 + x^3 + x^2 + x + 1 \end{aligned}$$

This gives a $(31, 31 - 15) = (31, 16)$ binary cyclic code. □

Example 6.4 Construct a generator for a quaternary, narrow-sense BCH 2-error correcting code of length $n = 51$. For a quaternary code we have $q = 4$. The smallest m such that $51 \mid q^m - 1$ is $m = 4$, so the arithmetic takes place in the field $GF(4^4) = GF(256)$. Let α be primitive in the field $GF(256)$. The element $\beta = \alpha^5$ is a 51st root of unity. The subfield $GF(4)$ in $GF(256)$ can be represented (see Example 5.33) using the elements $\{0, 1, \alpha^{85}, \alpha^{170}\}$. The $2t$ consecutive powers of β are $\beta, \beta^2, \beta^3, \beta^4$. Partitioned into conjugacy classes with respect to $GF(4)$, these powers of β are $\{\beta, \beta^4\}, \{\beta^2\}, \{\beta^3\}$. The conjugacy classes for these powers of β are

$$\{\beta, \beta^4, \beta^{16}, \beta^{64}\} \qquad \{\beta^2, \beta^8, \beta^{32}, \beta^{128}\} \qquad \{\beta^3, \beta^{12}, \beta^{48}, \beta^{192}\}$$

with corresponding minimal polynomials

$$p_1(x) = x^4 + \alpha^{170}x^3 + x^2 + x + \alpha^{170}$$
$$p_2(x) = x^4 + \alpha^{85}x^3 + x^2 + \alpha^{85}$$
$$p_3(x) = x^4 + \alpha^{170}x^3 + x^2 + \alpha^{170}x + 1.$$

These are in $GF(4)[x]$, as expected. The generator polynomial is

$$\begin{aligned} g(x) &= p_1(x)p_2(x)p_3(x) \\ &= x^{12} + \alpha^{85}x^{11} + \alpha^{170}x^{10} + x^8 + \alpha^{170}x^6 + \alpha^{170}x^5 + \alpha^{170}x^4 + \alpha^{85}x^3 \\ &\quad + \alpha^{85}x^2 + \alpha^{85}x + 1 \end{aligned}$$

□

6.1.2 The BCH Bound

The BCH bound is the proof that the constructive procedure described above produces codes with at least the specified minimum distance.

We begin by constructing a parity check matrix for the code. Let $c(x) = m(x)g(x)$ be a code polynomial. Then, for $i = b, b+1, \ldots, b+2t-1$,

$$c(\beta^i) = m(\beta^i)g(\beta^i) = m(\beta^i)0 = 0,$$

since these powers of β are, by design, the roots of $g(x)$. Writing $c(x) = c_0 + c_1x + \cdots + c_{n-1}x^{n-1}$ we have

$$c_0 + c_1\beta^i + c_2(\beta^i)^2 + \cdots + c_{n-1}(\beta^i)^{n-1} = 0 \qquad i = b, b+1, \ldots, b+2t-1.$$

The parity check conditions can be expressed in the form

$$\begin{bmatrix} 1 & \beta^i & \beta^{2i} & \cdots & \beta^{(n-1)i} \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ c_{n-1} \end{bmatrix} = 0 \qquad i = b, b+1, \ldots, b+2t-1.$$

Let $\delta = 2t + 1$; δ is called the **design distance** of the code. Stacking the row vectors for different values of i we obtain a parity check matrix H,

$$H = \begin{bmatrix} 1 & \beta^b & \beta^{2b} & \cdots & \beta^{(n-1)b} \\ 1 & \beta^{b+1} & \beta^{2(b+1)} & \cdots & \beta^{(n-1)(b+1)} \\ \vdots & & & & \\ 1 & \beta^{b+\delta-3} & \beta^{2(b+\delta-3)} & \cdots & \beta^{(n-1)(b+\delta-3)} \\ 1 & \beta^{b+\delta-2} & \beta^{2(b+\delta-2)} & \cdots & \beta^{(n-1)(b+\delta-2)} \end{bmatrix}. \tag{6.1}$$

With this in place, there is one more important result needed from linear algebra to prove the BCH bound.

A **Vandermonde matrix** is a square matrix of the form

$$V = V(x_0, x_1, \ldots, x_{n-1}) = \begin{bmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^{n-1} \\ 1 & x_1 & x_1^2 & \cdots & x_1^{n-1} \\ 1 & x_2 & x_2^2 & \cdots & x_2^{n-1} \\ \vdots & & & & \\ 1 & x_{n-1} & x_{n-1}^2 & \cdots & x_{n-1}^{n-1} \end{bmatrix}$$

or the transpose of such a matrix.

Lemma 6.1

$$\det(V) = \prod_{0 \le j < k \le n-1} (x_k - x_j).$$

Thus, as long as the x_i are all distinct, the Vandermonde matrix is invertible.

Proof The determinant is a polynomial function of the elements of the matrix of total degree

$$1 + 2 + \cdots + n - 1 = n(n-1)/2.$$

We note that if any $x_i = x_0$, $i \neq 0$, then the determinant is equal to 0, since the determinant of a matrix with two identical columns is equal to 0. This suggests that we can think of the determinant as a polynomial of degree $n-1$ in the variable x_0. Thus we can write

$$\det(V) = (x_0 - x_1)(x_0 - x_2)\cdots(x_0 - x_{n-1})p_0(x_1, x_2, \ldots, x_{n-1}),$$

where p_0 is some polynomial function of its arguments. Similarly, we can think of $\det(V)$ as a polynomial in x_1 of degree $n-1$, having roots at locations $x_0, x_2, \ldots, x_{n-1}$:

$$\det(V) = (x_1 - x_0)(x_1 - x_2)\cdots(x_1 - x_{n-1})p_1(x_0, x_2, \ldots, x_{n-1}).$$

This applies to all the elements, so

$$\det(V) = \prod_{0 \le j < k \le n-1} (x_k - x_j) p(x_0, x_1, \ldots, x_{n-1})$$

for some function p which is polynomial in its arguments. The product

$$\prod_{0 \le j < k \le n-1} (x_k - x_j)$$

has total degree

$$(n-1) + n - 2 + \cdots + 1 = n(n-1)/2.$$

Comparing the degrees of both sides, we see that $p = 1$. □

Theorem 6.2 *(The BCH Bound) Let C be a q-ary (n, k) cyclic code with generator polynomial $g(x)$. Let $GF(q^m)$ be the smallest extension field of $GF(q)$ that contains a primitive nth root of unity and let β be a primitive nth root of unity in that field. Let $g(x)$ be the minimal-degree polynomial in $GF(q)[x]$ having $2t$ consecutive roots of the form $g(\beta^b) = g(\beta^{b+1}) = g(\beta^{b+2}) = \cdots = g(\beta^{b+2t-1})$. Then the minimum distance of the code satisfies $d_{min} \ge \delta = 2t + 1$; that is, the code is capable of correcting at least t errors.*

In designing BCH codes we may, in fact, exceed the design distance, since extra roots as consecutive powers of β are often included with the minimal polynomials. For RS codes, on the other hand, the minimum distance for the code is exactly the design distance.

Proof Let $\mathbf{c} \in C$, with corresponding polynomial representation $c(x)$. As we have seen in (6.1), a parity check matrix for the code can be written as

$$H = \begin{bmatrix} 1 & \beta^b & \beta^{2b} & \cdots & \beta^{(n-1)b} \\ 1 & \beta^{b+1} & \beta^{2(b+1)} & \cdots & \beta^{(n-1)(b+1)} \\ \vdots & & & & \\ 1 & \beta^{b+\delta-3} & \beta^{2(b+\delta-3)} & \cdots & \beta^{(n-1)(b+\delta-3)} \\ 1 & \beta^{b+\delta-2} & \beta^{2(b+\delta-2)} & \cdots & \beta^{(n-1)(b+\delta-2)} \end{bmatrix}.$$

By Theorem 3.3, the minimum distance of a linear code C is equal to the minimum positive number of columns of H which are linearly dependent. Also, the minimum distance of a code is the smallest nonzero weight of the codewords.

We do a proof by contradiction. Suppose there is a codeword $\mathbf{c}$ of weight $w < \delta$. We write the nonzero components of $\mathbf{c}$ as $[c_{i_1}, c_{i_2}, \ldots, c_{i_w}]^T = \mathbf{d}^T$. Then since $H\mathbf{c} = 0$, we have

$$\begin{bmatrix} \beta^{i_1 b} & \beta^{i_2 b} & \beta^{i_3 b} & \cdots & \beta^{i_w b} \\ \beta^{i_1(b+1)} & \beta^{i_2(b+1)} & \beta^{i_3(b+1)} & \cdots & \beta^{i_w(b+1)} \\ \vdots & & & & \\ \beta^{i_1(b+\delta-3)} & \beta^{i_2(b+\delta-3)} & \beta^{i_3(b+\delta-3)} & \cdots & \beta^{i_w(b+\delta-3)} \\ \beta^{i_1(b+\delta-2)} & \beta^{i_2(b+\delta-2)} & \beta^{i_3(b+\delta-2)} & \cdots & \beta^{i_w(b+\delta-2)} \end{bmatrix} \begin{bmatrix} c_{i_1} \\ c_{i_2} \\ \vdots \\ c_{i_{w-1}} \\ c_{i_w} \end{bmatrix} = \mathbf{0}.$$

From the first w rows, we obtain

$$\begin{bmatrix} \beta^{i_1 b} & \beta^{i_2 b} & \beta^{i_3 b} & \cdots & \beta^{i_w b} \\ \beta^{i_1(b+1)} & \beta^{i_2(b+1)} & \beta^{i_3(b+1)} & \cdots & \beta^{i_w(b+1)} \\ \vdots & & & & \\ \beta^{i_1(b+w-2)} & \beta^{i_2(b+w-2)} & \beta^{i_3(b+w-2)} & \cdots & \beta^{i_w(b+w-2)} \\ \beta^{i_1(b+w-1)} & \beta^{i_2(b+w-1)} & \beta^{i_3(b+w-1)} & \cdots & \beta^{i_w(b+w-1)} \end{bmatrix} \begin{bmatrix} c_{i_1} \\ c_{i_2} \\ \vdots \\ c_{i_{w-1}} \\ c_{i_w} \end{bmatrix} = \mathbf{0}.$$

Let H' be the square $w \times w$ matrix on the LHS of this equation. Since $\mathbf{d} \neq 0$, we must have that H' is singular, so that $\det(H') = 0$. Note that

$$\det(H') = \beta^{i_1 b + i_2 b + \cdots + i_w b} \det \begin{bmatrix} 1 & 1 & \cdots & 1 \\ \beta^{i_1} & \beta^{i_2} & \cdots & \beta^{i_w} \\ \vdots & & & \\ \beta^{i_1(w-2)} & \beta^{i_2(w-2)} & \cdots & \beta^{i_w(w-2)} \\ \beta^{i_1(w-1)} & \beta^{i_2(w-1)} & \cdots & \beta^{i_w(w-1)} \end{bmatrix}.$$

But the latter matrix is a Vandermonde matrix: its determinant is zero if and only if $\beta^{i_j} = \beta^{i_k}$ for some j and k, $j \neq k$. Since $0 \leq i_k < n$ and β is of order n, the β^{i_k} elements along the second row are all distinct. There is thus a contradiction. We conclude that a codeword with weight $w < \delta$ cannot exist. $\square$

It should be mentioned that there are extensions to the BCH bound. In the BCH bound, only a single consecutive block of roots of the generator is considered. The Hartman-Tzeng bound and the Roos bound, by contrast, provide bounds on the error correction capability of BCH codes based on *multiple* sets of consecutive roots. For these, see [143, 144, 296].

6.1.3 Weight Distributions for Some Binary BCH Codes

While the weight distributions of most BCH codes are not known, weight distributions for the *duals* of all double and triple-error correcting binary primitive BCH codes have been found [186], [220, p. 451, p. 669], [203, pp. 177–178]. From the dual weight distributions, the weight distributions of these BCH codes can be obtained using the MacWilliams identity.

Example 6.5 In Example 6.2 we found the generator for a double-error correcting code of length $n = 31 = 2^5 - 1$ to be

$$g(x) = x^{10} + x^9 + x^8 + x^6 + x^5 + x^3 + 1.$$

From Table 6.1, we obtain

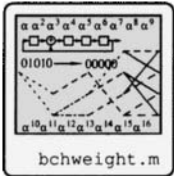
Table 6.1: Weight Distribution of the Dual of a Double-Error-Correcting Primitive Binary BCH Code of Length $n = 2^m - 1, m \geq 3, m$ Odd

Codeword Weight i	Number of Codewords of Weight i, B_i
0	1
$2^{m-1} - 2^{(m-1)/2}$	$(2^{m-2} + 2^{(m-3)/2})(2^m - 1)$
2^{m-1}	$(2^{m-1} + 1)(2^m - 1)$
$2^{m-1} + 2^{(m-1)/2}$	$(2^{m-2} - 2^{(m-3)/2})(2^m - 1)$

Table 6.2: Weight Distribution of the Dual of a Double-Error-Correcting Primitive Binary Narrow-Sense BCH Code, $n = 2^m - 1, m \geq 4, m$ Even

Codeword Weight i	Number of Codewords of Weight i, B_i
0	1
$2^{m-1} - 2^{m/2}$	$\frac{1}{3}2^{(m/2)-2}(2^{(m-2)/2} + 1)(2^m - 1)$
$2^{m-1} - 2^{(m/2)-1}$	$\frac{1}{3}2^{m/2}(2^{m/2} + 1)(2^m - 1)$
2^{m-1}	$(2^{m-2} + 1)(2^m - 1)$
$2^{m-1} + 2^{(m/2)-1}$	$\frac{1}{3}2^{m/2}(2^{m/2} - 1)(2^m - 1)$
$2^{m-1} + 2^{m/2}$	$\frac{1}{3}2^{(m/2)-2}(2^{(m-2)/2} - 1)(2^m - 1)$

Codeword Weight i	Number of Codewords of Weight i, B_i
0	1
12	310
16	527
20	186



The corresponding weight enumerator for the dual code is

$$B(x) = 1 + 310x^{12} + 527x^{16} + 186x^{20}.$$

Using the MacWilliams identity (3.13) we have

$$\begin{aligned}
A(x) &= 2^{-(n-k)}(1+x)^n B\left(\frac{1-x}{1+x}\right) \\
&= 2^{-10}(1+x)^{31}\left[1 + 310\left[\frac{1-x}{1+x}\right]^{12} + 527\left[\frac{1-x}{1+x}\right]^{16} + 186\left[\frac{1-x}{1+x}\right]^{20}\right] \\
&= 1 + 186x^5 + 806x^6 + 2635x^7 + 7905x^8 + 18910x^9 + 41602x^{10} + 85560x^{11} \\
&\quad + 142600x^{12} + 195300x^{13} + 251100x^{14} + 301971x^{15} + 301971x^{16} \\
&\quad + 251100x^{17} + 195300x^{18} + 142600x^{19} + 85560x^{20} + 41602x^{21} + 18910x^{22} \\
&\quad + 7905x^{23} + 2635x^{24} + 806x^{25} + 186x^{26} + x^{31}.
\end{aligned}$$

□

6.1.4 Asymptotic Results for BCH Codes

The Varshamov-Gilbert bound (see Exercise 3.26) indicates that if the rate R of a block code is fixed, then there exist binary (n, k) codes with distance $d_{\min}$ satisfying $k/n \geq R$

Table 6.3: Weight Distribution of the Dual of a Triple-Error Correcting Primitive Binary Narrow-Sense BCH Code, $n = 2^m - 1, m \geq 5, m$ Odd

Codeword Weight i	Number of Codewords of Weight i, B_i
0	1
$2^{m-1} - 2^{(m+1)/2}$	$\frac{1}{3}2^{(m-5)/2}(2^{(m-3)/2} + 1)(2^{m-1} - 1)(2^m - 1)$
$2^{m-1} - 2^{(m-1)/2}$	$\frac{1}{3}2^{(m-3)/2}(2^{(m-1)/2} + 1)(5 \cdot 2^{m-1} + 4)(2^m - 1)$
2^{m-1}	$(9 \cdot 2^{2m-4} + 3 \cdot 2^{m-3} + 1)(2^m - 1)$
$2^{m-1} + 2^{(m-1)/2}$	$\frac{1}{3}2^{(m-3)/2}(2^{(m-1)/2} - 1)(5 \cdot 2^{m-1} + 4)(2^m - 1)$
$2^{m-1} + 2^{(m+1)/2}$	$\frac{1}{3}2^{(m-5)/2}(2^{(m-3)/2} - 1)(2^{m-1} - 1)(2^m - 1)$

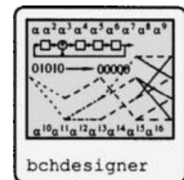
Table 6.4: Weight Distribution of the Dual of a Triple-Error Correcting Primitive Binary Narrow-Sense BCH Code, $n = 2^m - 1, m \geq 6, m$ Even

Codeword Weight i	Number of Codewords of Weight i, B_i
0	1
$2^{m-1} - 2^{(m/2)+1}$	$\frac{1}{960}(2^{m-1} + 2^{(m/2)+1})(2^m - 4)(2^m - 1)$
$2^{m-1} - 2^{m/2}$	$\frac{7}{48}2^m(2^{m-1} + 2^{m/2})(2^m - 1)$
$2^{m-1} - 2^{(m/2)-1}$	$\frac{2}{15}(2^{m-1} + 2^{(m/2)-1})(3 \cdot 2^m + 8)(2^m - 1)$
2^{m-1}	$\frac{1}{64}(29 \cdot 2^{2m} - 4 \cdot 2^m + 64)(2^m - 1)$
$2^{m-1} + 2^{(m/2)-1}$	$\frac{2}{15}(2^{m-1} - 2^{(m/2)-1})(3 \cdot 2^m + 8)(2^m - 1)$
$2^{m-1} + 2^{m/2}$	$\frac{7}{48}2^m(2^{m-1} - 2^{m/2})(2^m - 1)$
$2^{m-1} + 2^{(m/2)+1}$	$\frac{1}{960}(2^{m-1} - 2^{(m/2)+1})(2^m - 4)(2^m - 1)$

and $d_{\min}/n \geq H_2^{-1}(1 - R)$. It is interesting to examine whether a sequence of BCH codes can be constructed which meets this bound.

Definition 6.2 [220, p. 269] A family of codes over $GF(q)$ for a fixed q is said to be **good** if it contains an infinite sequence of codes $C_1, C_2, \ldots,$, where C_i is an (n_i, k_i, d_i) code such that both the rate $R_i = k_i/n_i$ and the relative distance d_i/n_i approach a nonzero limit as $i \to \infty$. □

The basic result about BCH codes (which we do not prove here; see [220, p. 269]) is this:



Theorem 6.3 *There does not exist a sequence of primitive BCH codes over $GF(q)$ with both k/n and d/n bounded away from zero.*

That is, as codes of a given rate become longer, the fraction of errors that can be corrected diminishes to 0.

Nevertheless, for codes of moderate length (up to n of a few thousand), the BCH codes are among the best codes known. The program `bchdesigner` can be used to design a binary BCH code for a given code length n, design correction capability t, and starting exponent b.

6.2 Reed-Solomon Codes

There are actually two distinct constructions for Reed-Solomon codes. While these initially appear to describe different codes, it is shown in Section 6.8.1 using Galois field Fourier transform techniques that the families of codes described are in fact equivalent. Most of the encoders and decoders in this chapter are concerned with the second code construction. However, there are important theoretical reasons to be familiar with the first code construction as well.

6.2.1 Reed-Solomon Construction 1

Definition 6.3 Let α be a primitive element in $GF(q^m)$ and let $n = q^m - 1$. Let $\mathbf{m} = (m_0, m_1, \ldots, m_{k-1}) \in GF(q^m)^k$ be a message vector and let $m(x) = m_0 + m_1 x + \cdots + m_{k-1}x^{k-1} \in GF(q^m)[x]$ be its associated polynomial. Then the encoding is defined by the mapping $\rho : m(x) \mapsto \mathbf{c}$ by

$$(c_0, c_1, \ldots, c_{n-1}) \triangleq \rho(m(x)) = (m(1), m(\alpha), m(\alpha^2), \ldots, m(\alpha^{n-1})).$$

That is, $\rho(m(x))$ evaluates $m(x)$ at all the non-zero elements of $GF(q^m)$.

The Reed-Solomon code of length $n = q^m - 1$ and dimension k over $GF(q^m)$ is the image under ρ of all polynomials in $GF(q^m)[x]$ of degree less than or equal to $k - 1$.

More generally, a Reed-Solomon code can be defined by taking $n \leq q$, choosing n distinct elements out of $GF(q^m)$, $\alpha_1, \alpha_2, \ldots, \alpha_n$ known as the **support set**, and defining the encoding operation as

$$\rho(m(x)) = (m(\alpha_1), m(\alpha_2), \ldots, m(\alpha_n)).$$

The code is the image of the support set under ρ of all polynomials in $GF(q^m)[x]$ of degree less than k.

□

Example 6.6 Let $GF(q^m) = GF(2^3) = GF(8)$. A (7, 3) Reed-Solomon code can be obtained by writing down all polynomials of degree 2 with coefficients in $GF(8)$, then evaluating them at the nonzero elements in the field. Such polynomials are, for example, $m(x) = \alpha + \alpha^3 x^2$ or $m(x) = \alpha^5 + \alpha^2 x + \alpha^4 x^2$ ($8^3 = 512$ polynomials in all). We will see that $d_{\min} = 5$, so this is a 2-error correcting code. □

Some properties of the Reed-Solomon codes are immediate from the definition.

Lemma 6.4 *The Reed-Solomon code is a linear code.*

The proof is immediate.

Lemma 6.5 *The minimum distance of an (n, k) Reed-Solomon code is $d_{min} = n - k + 1$.*

Proof Since $m(x)$ has at most $k - 1$ zeros in it, there are at most $k - 1$ zero positions in each nonzero codeword. Thus $d_{\min} \geq n - (k - 1)$. However, by the Singleton bound (see Section 3.3.1), we must have $d_{\min} \leq n - k + 1$. So $d_{\min} = n - k + 1$. □

Reed-Solomon codes achieve the Singleton bound and are thus maximum distance separable codes.

This construction of Reed-Solomon codes came first historically [286] and leads to important generalizations, some of which are introduced in Section 6.9.2. Recently a powerful decoding algorithm has been developed based on this viewpoint (see Section 7.6). However, the following construction has been important because of its relation with BCH codes and their associated decoding algorithms.

6.2.2 Reed-Solomon Construction 2

In constructing BCH codes, we looked for generator polynomials over $GF(q)$ (the small field) so we dealt with minimal polynomials. Since the minimal polynomial for an element β must have all the conjugates of β as roots, the product of the minimal polynomials usually exceeds the number $2t$ of roots specified.

The situation is somewhat different with RS codes. With RS codes, we can operate in the bigger field:

Definition 6.4 A Reed-Solomon code is a q^m-ary BCH code of length $q^m - 1$. □

In $GF(q^m)$, the minimal polynomial for any element β is simply $(x - \beta)$. The generator for a RS code is therefore

$$g(x) = (x - \alpha^b)(x - \alpha^{b+1}) \cdots (x - \alpha^{b+2t-1}),$$

where α is a primitive element. There are no extra roots of $g(x)$ included due to conjugates in the minimal polynomials, so the degree of g is exactly equal to $2t$. Thus $n - k = 2t$ for a RS code. The design distance is $\delta = n - k + 1$.

Example 6.7 Let $n = 7$. We want to design a narrow sense double-error correcting RS code. Let α be a root of the primitive polynomial $x^3 + x + 1$. The $2t = 4$ consecutive powers of α are $\alpha, \alpha^2, \alpha^3$ and α^4. The generator polynomial is

$$g(x) = (x - \alpha)(x - \alpha^2)(x - \alpha^3)(x - \alpha^4) = x^4 + \alpha^3 x^3 + x^2 + \alpha x + \alpha^3.$$

Note that the coefficients of $g(x)$ are in $GF(8)$, the extension ("big") field. We have designed a $(7, 3)$ code with 8^3 codewords. □

Example 6.8 Let $n = 2^4 - 1 = 15$ and consider a primitive, narrow sense, three-error correcting code over $GF(2^4)$, where the field is constructed modulo the primitive polynomial $x^4 + x + 1$. Let α be a primitive element in the field. The code generator has

$$\alpha, \alpha^2, \alpha^3, \alpha^4, \alpha^5, \alpha^6$$

as roots. The generator of the $(15, 9)$ code is

$$\begin{aligned} g(x) &= (x - \alpha)(x - \alpha^2)(x - \alpha^3)(x - \alpha^4)(x - \alpha^5)(x - \alpha^6) \\ &= \alpha^6 + \alpha^9 x + \alpha^6 x^2 + \alpha^4 x^3 + \alpha^{14} x^4 + \alpha^{10} x^5 + x^6 \end{aligned}$$

A (15,9) code is obtained. □

Example 6.9 Let $n = 2^8 - 1 = 255$ and consider a primitive, narrow sense, three-error correcting code over $GF(2^8)$, where the field is represented using the primitive polynomial $p(x) = 1 + x^2 + x^3 + x^4 + x^8$. Let α be a primitive element in the field. The generator is

$$\begin{aligned} g(x) &= (x - \alpha)(x - \alpha^2)(x - \alpha^3)(x - \alpha^4)(x - \alpha^5)(x - \alpha^6) \\ &= \alpha^{21} + \alpha^{181} x + \alpha^9 x^2 + \alpha^{137} x^3 + \alpha^2 x^4 + \alpha^{167} x^5 + x^6 \end{aligned}$$

This gives a (255,249) code. □

Codes defined over $GF(256)$, such as this, are frequently of interest in many computer oriented applications, because every field symbol requires eight bits — one byte — of data.

The remainder of this chapter deals with Reed-Solomon codes obtained via this second construction.

6.2.3 Encoding Reed-Solomon Codes

Reed-Solomon codes may be encoded just as any other cyclic code (provided that the arithmetic is done in the right field). Given a message vector $\mathbf{m} = (m_0, m_1, \ldots, m_{k-1})$ and its corresponding message polynomial $m(x) = m_0 + m_1 x + \cdots + m_{k-1}x^{k-1}$, where each $m_i \in GF(q)$, the systematic encoding process is

$$c(x) = m(x)x^{n-k} - R_{g(x)}[m(x)x^{n-k}]$$

where, as before, $R_{g(x)}[\cdot]$ denotes the operation of taking the remainder after division by $g(x)$.

Typically, the code is over $GF(2^m)$ for some m. The message symbols m_i can then be formed by grabbing m bits of data, then interpreting these as the *vector* representation of the $GF(2^m)$ elements.

Example 6.10 For the code from Example 6.8, the 4-bit data

$$5, 2, 1, 6, 8, 3, 10, 15, 4$$

are to be encoded. The corresponding message polynomial is

$$m(x) = 5 + 2x + x^2 + 6x^3 + 8x^4 + 3x^5 + 10x^6 + 15x^7 + 4x^8.$$

Using the vector to power conversion

$$5 = 0101_2 \leftrightarrow \alpha^8 \qquad 2 = 0010_2 \leftrightarrow \alpha \qquad 1 = 0001_2 \leftrightarrow 1$$

and so forth, the message polynomial (expressed in power form) is

$$m(x) = \alpha^8 + \alpha x + x^2 + \alpha^5 x^3 + \alpha^3 x^4 + \alpha^4 x^5 + \alpha^9 x^6 + \alpha^{12} x^7 + \alpha^2 x^8.$$

The systematically encoded code polynomial is

$$c(x) = \alpha^8 + \alpha^2 x + \alpha^{14} x^2 + \alpha^3 x^3 + \alpha^5 x^4 + \alpha x^5 + \alpha^8 x^6 + \alpha x^7 + x^8 + \alpha^5 x^9 + \alpha^3 x^{10} + \alpha^4 x^{11} + \alpha^9 x^{12} + \alpha^{12} x^{13} + \alpha^2 x^{14}$$

where the message is explicitly evident.

The following code fragment shows how to reproduce this example using the classes `GFNUM2m` and `polynomialT`:

```
int k=9;
int n=15;
GFNUM2m::initgf(4,0x13);  // 1 0011 = d^4+d+1
POLYC(GFNUM2m, m,{5,2,1,6,8,3,10,15,4});  // the message data
polynomialT<GFNUM2m> c;          // code polynomial
//
// ... build the generator g

c = (m<<(n-k)) + (m<<(n-k))% g;         // encode operation: pretty easy!

cout << "message=" << m << endl;
cout << "code=" << c << endl;
```

□

Using groups of m bits as the vector representation of the Galois field element is particularly useful when $m = 8$, since 8 bits of data correspond to one byte. Thus, a computer data file can be encoded by simply grabbing k bytes of data, then using those bytes as the coefficients of $m(x)$: no conversion of any sort is necessary (unless you want to display the answer in power form).

6.2.4 MDS Codes and Weight Distributions for RS Codes

We observed in Lemma 6.5 that Reed-Solomon codes are maximum distance separable (MDS), that is, $d_{\min} = n - k + 1$. Being MDS gives sufficient theoretical leverage that the weight distribution for RS codes can be explicitly described. We develop that result through a series of lemmas [220, p. 318].

Lemma 6.6 *A code $\mathcal{C}$ is MDS if and only if every set of $n - k$ columns of its parity check matrix H are linearly independent.*

Proof By Theorem 3.3, a code $\mathcal{C}$ contains a code of weight w if and only if w columns of H are linearly dependent. Therefore $\mathcal{C}$ has $d_{\min} = n-k+1$ if and only if no $n-k$ or fewer columns of H are linearly dependent. □

Lemma 6.7 *If an (n, k) code $\mathcal{C}$ is MDS, then so is the $(n, n-k)$ dual code $\mathcal{C}^\perp$. That is, $\mathcal{C}^\perp$ has $d_{min} = k + 1$.*

Proof Let $H = \begin{bmatrix}\mathbf{h}_1 & \dots & \mathbf{h}_n\end{bmatrix}$ be an $(n - k) \times n$ parity check matrix for $\mathcal{C}$. Then H is a generator matrix for $\mathcal{C}^\perp$. Suppose that for some message vector $\mathbf{m}$ there is a codeword $\mathbf{c} = \mathbf{m}H \in \mathcal{C}^\perp$ with weight $\leq k$. Then $\mathbf{c}$ has zero elements in $\geq n - k$ positions. Let the zero elements of $\mathbf{c}$ have indices $\{i_1, i_2, \dots, i_{n-k}\}$. Write

$$H = \begin{bmatrix}\mathbf{h}_1 & \mathbf{h}_2 & \cdots & \mathbf{h}_n\end{bmatrix}.$$

Then the zero elements of $\mathbf{c}$ are obtained from

$$\mathbf{0} = \mathbf{m}\begin{bmatrix}h_{i_1} & h_{i_2} & \cdots & h_{i_{n-i}}\end{bmatrix} \stackrel{\Delta}{=} \mathbf{m}\tilde{H}.$$

We thus have a $(n - k) \times (n - k)$ submatrix $\tilde{H}$ of H which is singular. Since the row-rank of a matrix is equal to the column-rank, there must therefore be $n - k$ columns of H which are linearly dependent. This contradicts the fact that $\mathcal{C}$ has minimum distance $n - k + 1$, so the minimal weight codeword of $\mathcal{C}^\perp$ must be $d_{\min} > k$. By the Singleton bound, we must have $d_{\min} = k + 1$. □

Lemma 6.8 *Every k columns of a generator matrix G of an MDS are linearly independent. (This means that any k symbols of the codeword may be taken as systematically encoded message symbols.)*

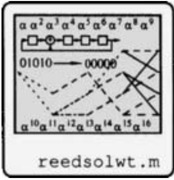
Proof Let G be the $k \times n$ generator matrix of an MDS code $\mathcal{C}$. Then G is the parity check matrix for $\mathcal{C}^\perp$. Since $\mathcal{C}^\perp$ has minimum distance $k + 1$, any combination of k rows of G must be linearly independent. Thus any $k \times k$ submatrix of G must be nonsingular. So by row reduction on G, any $k \times k$ submatrix can be reduced to the $k \times k$ identity, so that the corresponding k message symbols can appear explicitly. □

Lemma 6.9 *The number of codewords in a q-ary (n, k) MDS code C of weight $d_{min} = n - k + 1$ is $A_{n-k+1} = (q-1)\binom{n}{n-k+1}$.*

Proof By Lemma 6.8, select an arbitrary set of k coordinates as message positions for a message $\mathbf{m}$ of weight 1. The systematic encoding for these coordinates thus has $k-1$ zeros in it. Since the minimum distance of the code is $n-k+1$, all the $n-k$ parity symbols therefore must be nonzero. Since there are $\binom{n}{n-k+1} = \binom{n}{k-1}$ different ways of selecting the zero coordinates and $q-1$ ways of selecting the nonzero message symbol,

$$A_{n-k+1} = (q-1)\binom{n}{k-1}.$$

□



Lemma 6.10 *(Weight distribution for MDS codes) The number of codewords of weight j in a q-ary (n, k) MDS code is*

$$A_j = \binom{n}{j}(q-1)\sum_{i=0}^{j-d_{min}}(-1)^i\binom{j-1}{i}q^{j-d_{min}-i}. \tag{6.2}$$

Proof Using (3.25) from Exercise 3.16, generalized to q-ary codes we have

$$\sum_{i=0}^{n-j}\binom{n-i}{j}A_i = q^{k-j}\sum_{i=0}^{j}\binom{n-i}{j-i}B_i, \qquad j = 0, 1, \ldots, n.$$

Since $A_i = 0$ for $i = 1, \ldots, n-k$ and $B_i = 0$ for $i = 1, \ldots, k$, this becomes

$$\binom{n}{j} + \sum_{i=n-k+1}^{n-j}\binom{n-i}{j}A_i = q^{k-j}\binom{n}{j}, \qquad j = 0, 1, \ldots, k-1.$$

Setting $j = k-1$ we obtain

$$A_{n-k+1} = (q-1)\binom{n}{k-1},$$

as in Lemma 6.9. Setting $j = k-2$ we obtain

$$\binom{k-1}{k-2}A_{n-k+1} + A_{n-k+2} = \binom{n}{k-2}(q^2-1),$$

from which

$$A_{n-k+2} = \binom{n}{k-2}[(q^2-1) - (n-k+2)(q-1)].$$

Proceeding similarly, it may be verified that

$$A_{n-k+r} = \binom{n}{k-r}\sum_{j=0}^{r-1}(-1)^j\binom{n-k+r}{j}(q^{r-j}-1).$$

Letting $j = n-k+r$, it is straightforward from here to verify (6.2). □

Complete weight enumerators for Reed-Solomon codes are described in [35].

6.3 Decoding BCH and RS Codes: The General Outline

There are many algorithms which have been developed for decoding BCH or RS codes. In this chapter we introduce a general approach. In chapter 7 we present other approaches which follow a different outline.

The algebraic decoding BCH or RS codes has the following general steps:

1. Computation of the *syndrome*.
2. Determination of an *error locator polynomial*, whose roots provide an indication of *where* the errors are. There are several different ways of finding the locator polynomial. These methods include Peterson's algorithm for BCH codes, the Berlekamp-Massey algorithm for BCH codes; the Peterson-Gorenstein-Zierler algorithm for RS codes, the Berlekamp-Massey algorithm for RS codes, and the Euclidean algorithm. In addition, there are techniques based upon Galois-field Fourier transforms.
3. Finding the roots of the error locator polynomial. This is usually done using the *Chien search*, which is an exhaustive search over all the elements in the field.
4. For RS codes or nonbinary BCH codes, the error *values* must also be determined. This is typically accomplished using *Forney's algorithm.*

Throughout this chapter (unless otherwise noted) we assume narrow-sense BCH or RS codes, that is, $b = 1$.

6.3.1 Computation of the Syndrome

Since

$$g(\alpha) = g(\alpha^2) = \cdots = g(\alpha^{2t}) = 0$$

it follows that a codeword $\mathbf{c} = (c_0, \ldots, c_{n-1})$ with polynomial $c(x) = c_0 + \cdots + c_{n-1}x^{n-1}$ has

$$c(\alpha) = \cdots = c(\alpha^{2t}) = 0.$$

For a received polynomial $r(x) = c(x) + e(x)$ we have

$$S_j = r(\alpha^j) = e(\alpha^j) = \sum_{k=0}^{n-1} e_k \alpha^{jk}, \qquad j = 1, 2, \ldots, 2t.$$

The values $S_1, S_2, \ldots, S_{2t}$ are called the syndromes of the received data.

Suppose that $\mathbf{r}$ has ν errors in it which are at locations $i_1, i_2, \ldots, i_\nu$, with corresponding error values in these locations $e_{i_j} \neq 0$. Then

$$S_j = \sum_{l=1}^{\nu} e_{i_l} (\alpha^j)^{i_l} = \sum_{l=1}^{\nu} e_{i_l} (\alpha^{i_l})^j.$$

Let

$$X_l = \alpha^{i_l}.$$

Then we can write

$$S_j = \sum_{l=1}^{\nu} e_{i_l} X_l^j \qquad j = 1, 2, \ldots, 2t. \tag{6.3}$$

For *binary codes* we have $e_{i_l} = 1$ (if there is a non-zero error, it must be to 1). For the moment we restrict our attention to binary (BCH) codes. Then we have

$$S_j = \sum_{l=1}^{\nu} X_l^j. \tag{6.4}$$

If we know X_l, then we know the location of the error. For example, suppose we know that $X_1 = \alpha^4$. This means, by the definition of X_l that $i_1 = 4$; that is, the error is in the received digit r_4. We thus call the X_l the **error locators**.

The next stage in the decoding problem is to determine the error locators X_l given the syndromes S_j.

6.3.2 The Error Locator Polynomial

From (6.4) we obtain the following equations:

$$\begin{aligned} S_1 &= X_1 + X_2 + \cdots + X_\nu \\ S_2 &= X_1^2 + X_2^2 + \cdots + X_\nu^2 \\ &\vdots \\ S_{2t} &= X_1^{2t} + X_2^{2t} + \cdots + X_\nu^{2t}. \end{aligned} \tag{6.5}$$

The equations are said to be *power-sum symmetric functions*. This gives us $2t$ equations in the ν unknown error locators. In principle this set of nonlinear equations could be solved by an exhaustive search, but this would be computationally unattractive.

Rather than attempting to solve these nonlinear equations directly, a new polynomial is introduced, the *error locator polynomial*, which casts the problem in a different, and more tractable, setting. The error locator polynomial is defined as

$$\Lambda(x) = \prod_{l=1}^{\nu}(1 - X_l x) = \Lambda_\nu x^\nu + \Lambda_{\nu-1}x^{\nu-1} + \cdots + \Lambda_1 x + \Lambda_0, \tag{6.6}$$

where $\Lambda_0 = 1$. By this definition, if $x = X_l^{-1}$ then $\Lambda(x) = 0$; that is, the roots of the error locator polynomial are at the *reciprocals* (in the field arithmetic) of the error locators.

Example 6.11 Suppose in $GF(16)$ we find that $x = \alpha^4$ is a root of an error locator polynomial $\Lambda(x)$. Then the error locator is $(\alpha^4)^{-1} = \alpha^{11}$, indicating that there is an error in r_{11}. □

6.3.3 Chien Search

Assume for the moment that we actually have the error locator polynomial. (Finding the error locator polynomial is discussed below.) The next step is to find the roots of the error locator polynomial. The field of interest is $GF(q^m)$. Being a finite field, we can examine every element of the field to determine if it is a root. There exist other ways of factoring polynomials over finite fields (see, e.g., [25, 360]), but for the fields usually used for error correction codes and the number of roots involved, the Chien search may be the most efficient.

Suppose, for example, that $\nu = 3$ and the error locator polynomial is

$$\Lambda(x) = \Lambda_0 + \Lambda_1 x + \Lambda_2 x^2 + \Lambda_3 x^3 = 1 + \Lambda_1 x + \Lambda_2 x^2 + \Lambda_3 x^3.$$

We evaluate $\Lambda(x)$ at each nonzero element in the field in succession: $x = 1$, $x = \alpha$, $x = \alpha^2, \ldots, x = \alpha^{q^m-2}$. This gives us the following:

$$\begin{aligned}
\Lambda(1) &= 1 + \Lambda_1(1) + \Lambda_2(1)^2 + \Lambda_3(1)^3 \\
\Lambda(\alpha) &= 1 + \Lambda_1(\alpha) + \Lambda_2(\alpha)^2 + \Lambda_3(\alpha)^3 \\
\Lambda(\alpha^2) &= 1 + \Lambda_1(\alpha^2) + \Lambda_2(\alpha^2)^2 + \Lambda_3(\alpha^2)^3 \\
&\vdots \\
\Lambda(\alpha^{q^m-2}) &= 1 + \Lambda_1(\alpha^{q^m-2}) + \Lambda_2(\alpha^{q^m-2})^2 + \Lambda_3(\alpha^{q^m-2})^3.
\end{aligned}$$

The computations in this sequence can be efficiently embodied in the hardware depicted in Figure 6.1. A set of ν registers are loaded initially with the coefficients of the error locator polynomial, $\Lambda_1, \Lambda_2, \ldots, \Lambda_\nu$. The initial output is the sum

$$A = \sum_{j=1}^{\nu} \Lambda_j = \Lambda(x) - 1|_{x=1}.$$

If $A = 1$ then an error has been located (since then $\Lambda(x) = 0$). At the next stage, each register is multiplied by α^j, $j = 1, 2, \ldots, \nu$, so the register contents are $\Lambda_1\alpha, \Lambda_2\alpha^2, \ldots, \Lambda_\nu\alpha^\nu$. The output is the sum

$$A = \sum_{j=1}^{\nu} \Lambda_j\alpha^j = \Lambda(x) - 1|_{x=\alpha}.$$

The registers are multiplied again by successive powers of α, resulting in evaluation at α^2. This procedure continues until $\Lambda(x)$ has been evaluated at all nonzero elements of the field.

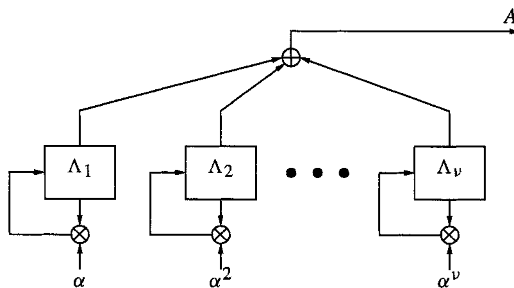


Figure 6.1: Chien search algorithm.

If the roots are distinct and all lie in the appropriate field, then we use these to determine the error locations. If they are not distinct or lie in the wrong field, then the received word is not within distance t of any codeword. (This condition can be observed if the error locator polynomial of degree ν does not have ν roots in the field that the operations take in; the remaining roots are either repeated or exist in an extension of this field.) The corresponding error pattern is said to be an *uncorrectable error pattern*. An uncorrectable error pattern results in a **decoder failure**.

6.4 Finding the Error Locator Polynomial

Let us return to the question of finding the error locator polynomial using the syndromes. Let us examine the structure of the error locator polynomial by expanding (6.6) for the case $\nu = 3$:

$$\begin{aligned}\Lambda(x) &= 1 - x(X_1 + X_2 + X_3) + x^2(X_1X_2 + X_1X_3 + X_2X_3) - x^3X_1X_2X_3 \\ &= \Lambda_0 + x\Lambda_1 + x^2\Lambda_2 + x^3\Lambda_3\end{aligned}$$

so that

$$\Lambda_0 = 1 \qquad \Lambda_1 = -(X_1 + X_2 + X_3) \qquad \Lambda_2 = X_1X_2 + X_1X_3 + X_2X_3$$

$$\Lambda_3 = -X_1X_2X_3.$$

In general, for an error locator polynomial of degree ν we find that

$$\begin{aligned}\Lambda_0 &= 1 \\ -\Lambda_1 &= \sum_{i=1}^{\nu} X_i = X_1 + X_2 + \cdots + X_\nu \\ \Lambda_2 &= \sum_{i<j} X_iX_j = X_1X_2 + X_1X_3 + \cdots + X_1X_\nu + \cdots + X_{\nu-1}X_\nu \\ -\Lambda_3 &= \sum_{i<j<k} X_iX_jX_k = X_1X_2X_3 + X_1X_2X_4 + \cdots + X_{\nu_2}X_{\nu-1}X_\nu \\ &\vdots \\ (-1)^\nu\Lambda_\nu &= X_1X_2\cdots X_\nu.\end{aligned} \tag{6.7}$$

That is, the coefficient of the error locator polynomial Λ_i is the sum of the product of all combinations of the error locators taken i at a time. Equations of the form (6.7) are referred to as the *elementary symmetric functions* of the error locators (so called because if the error locators $\{X_i\}$ are permuted, the same values are computed).

The power-sum symmetric functions of (6.5) provide a nonlinear relationship between the syndromes and the error locators. The elementary symmetric functions provide a nonlinear relationship between the coefficients of the error locator polynomial and the error locators. The key observation is that there is a *linear* relationship between the syndromes and the coefficients of the error locator polynomial. This relationship is described by the *Newton identities*, which apply over any field.

Theorem 6.11 *The syndromes (6.5) and the coefficients of the error locator polynomial are related by*

$$\begin{aligned}S_k + \Lambda_1S_{k-1} + \cdots + \Lambda_{k-1}S_1 + k\Lambda_k &= 0 \qquad 1 \le k \le \nu \\ S_k + \Lambda_1S_{k-1} + \cdots + \Lambda_{\nu-1}S_{k-\nu+1} + \Lambda_\nu S_{k-\nu} &= 0 \qquad k > \nu.\end{aligned} \tag{6.8}$$

That is,

$$
\begin{aligned}
k=1: &\quad S_1+\Lambda_1=0\\
k=2: &\quad S_2+\Lambda_1 S_1+2\Lambda_2=0\\
&\quad \vdots\\
k=\nu: &\quad S_\nu+\Lambda_1 S_{\nu-1}+\Lambda_2 S_{\nu-2}+\cdots+\Lambda_{\nu-1}S_1+\nu\Lambda_\nu=0\\
k=\nu+1: &\quad S_{\nu+1}+\Lambda_1 S_\nu+\Lambda_2 S_{\nu-1}+\cdots+\Lambda_\nu S_1=0\\
k=\nu+2: &\quad S_{\nu+2}+\Lambda_1 S_{\nu+1}+\Lambda_2 S_\nu+\cdots+\Lambda_\nu S_2=0\\
&\quad \vdots\\
k=2t: &\quad S_{2t}+\Lambda_1 S_{2t-1}+\Lambda_2 S_{2t-2}+\cdots+\Lambda_\nu S_{2t-\nu}=0.
\end{aligned}
\tag{6.9}
$$

For $k > \nu$, there is a linear feedback shift register relationship between the syndromes and the coefficients of the error locator polynomial,

$$S_j = -\sum_{i=1}^{\nu} \Lambda_i S_{j-i}. \tag{6.10}$$

The theorem is proved in Appendix 6.A.

Equation (6.10) can be expressed in a matrix form

$$
\begin{bmatrix}
S_1 & S_2 & \cdots & S_\nu\\
S_2 & S_3 & \cdots & S_{\nu+1}\\
S_3 & S_4 & \cdots & S_{\nu+2}\\
\vdots & & & \\
S_\nu & S_{\nu+1} & \cdots & S_{2\nu-1}
\end{bmatrix}
\begin{bmatrix}
\Lambda_\nu\\ \Lambda_{\nu-1}\\ \Lambda_{\nu-2}\\ \vdots\\ \Lambda_1
\end{bmatrix}
= -
\begin{bmatrix}
S_{\nu+1}\\ S_{\nu+2}\\ \vdots\\ S_{2\nu}
\end{bmatrix}.
$$

The $\nu \times \nu$ matrix, which we denote M_ν, is a Toeplitz matrix, constant on the diagonals. The number of errors ν is not known in advance, so it must be determined. The Peterson-Gorenstein-Zierler decoder operates as follows.

1. Set $\nu = t$.
2. Form M_ν and compute the determinant $\det(M_\nu)$ to determine if M_ν is invertible. If it is not invertible, set $\nu \leftarrow \nu - 1$ and repeat this step.
3. If M_ν is invertible, solve for the coefficients $\Lambda_1, \Lambda_2, \ldots, \Lambda_\nu$.

6.4.1 Simplifications for Binary Codes and Peterson's Algorithm

For binary codes, Newton's identities are subject to further simplifications. $nS_j = 0$ if n is even and $nS_j = S_j$ if n is odd. Furthermore, we have $S_{2j} = S_j^2$, since by (6.4) and Theorem 5.15

$$S_{2j} = \sum_{l=1}^{\nu} X_l^{2j} = \left(\sum_{l=1}^{\nu} X_l^j\right)^2 = S_j^2.$$

We can thus write Newton's identities (6.9) as

$$\begin{aligned} S_1 + \Lambda_1 &= 0 \\ S_3 + \Lambda_1 S_2 + \Lambda_2 S_1 + \Lambda_3 &= 0 \\ &\vdots \\ S_{2t-1} + \Lambda_1 S_{2t-2} + \cdots + \Lambda_t S_{t-1} &= 0, \end{aligned}$$

which can be expressed in the matrix equation

$$\begin{bmatrix} 1 & 0 & 0 & 0 & \cdots & 0 & 0 \\ S_2 & S_1 & 1 & 0 & \cdots & 0 & 0 \\ S_4 & S_3 & S_2 & S_1 & \cdots & 0 & 0 \\ \vdots & & & & & & \\ S_{2t-4} & S_{2t-5} & S_{2t-6} & S_{2t-7} & \cdots & S_{t-2} & S_{t-3} \\ S_{2t-2} & S_{2t-3} & S_{2t-4} & S_{2t-5} & \cdots & S_t & S_{t-1} \end{bmatrix} \begin{bmatrix} \Lambda_1 \\ \Lambda_2 \\ \vdots \\ \Lambda_t \end{bmatrix} = \begin{bmatrix} -S_1 \\ -S_3 \\ \vdots \\ -S_{2t-1} \end{bmatrix}, \tag{6.11}$$

or $A\mathbf{\Lambda} = -\mathbf{S}$. If there are in fact t errors, the matrix is invertible, as we can determine by computing the determinant of the matrix. If it is not invertible, remove two rows and columns, then try again. Once $\mathbf{\Lambda}$ is found, we find its roots. This matrix-based approach to solving for the error locator polynomial is called *Peterson's algorithm* for decoding binary BCH codes.

For small numbers of errors, we can provide explicit formulas for the coefficients of $\Lambda(x)$, which may be more efficient than the more generalized solutions suggested below [238].

1-error correction $\Lambda_1 = S_1$.

2-error correction $\Lambda_1 = S_1, \quad \Lambda_2 = (S_3 + S_1^3)/(S_1)$.

3-error correction $\Lambda_1 = S_1, \quad \Lambda_2 = (S_1^2 S_3 + S_5)/(S_1^3 + S_3), \quad \Lambda_3 = (S_1^3 + S_3) + S_1 \Lambda_2$.

4 error correction

$$\Lambda_1 = S_1, \quad \Lambda_2 = \frac{S_1(S_7 + S_1^7) + S_3(S_1^5 + S_5)}{S_3(S_1^3 + S_3) + S_1(S_1^5 + S_5)},$$

$$\Lambda_3 = S_1^3 + S_3 + S_1\Lambda_2, \quad \Lambda_4 = \frac{(S_5 + S_1^2 S_3) + (S_1^3 + S_3)\Lambda_2}{S_1}.$$

5-error correction $\Lambda_1 = S_1$,

$$\Lambda_2 = \frac{(S_1^3 + S_3)[(S_1^9 + S_9) + S_1^4(S_5 + S_1^2 S_3) + S_3^2(S_1^3 + S_3)] + (S_1^5 + S_5)(S_7 + S_1^7) + S_1(S_3^2 + S_1 S_5)}{(S_1^3 + S_3)[(S_7 + S_1^7) + S_1 S_3(S_1^3 + S_3)] + (S_5 + S_1^2 S_3)(S_1^5 + S_5)}$$

$$\Lambda_3 = (S_1^3 + S_3) + S_1\Lambda_2$$

$$\Lambda_4 = \frac{(S_1^9 + S_9) + S_3^2(S_1^3 + S_3) + S_1^4(S_5 + S_1^2 S_3) + \Lambda_2[(s_7 + S_1^7) + S_1 S_3(S_1^3 + S_3)]}{S_1^5 + S_5}$$

$$\Lambda_5 = S_5 + S_1^2 S_3 + S_1 S_4 + \Lambda_2(S_1^3 + S_3).$$

For large numbers of errors, Peterson's algorithm is quite complex. Computing the sequence of determinants to find the number of errors is costly. So is solving the system of equations once the number of errors is determined. We therefore look for more efficient techniques.

Example 6.12 Consider the (31,21) 2-error correcting code introduced in Example 6.2, with generator $g(x) = x^{10} + x^9 + x^8 + x^6 + x^5 + x^3 + 1$ having roots at $\alpha, \alpha^2, \alpha^3$ and α^4. Suppose the codeword

$$c(x) = 1 + x^3 + x^4 + x^5 + x^6 + x^8 + x^{10} + x^{14} + x^{16} + x^{17} + x^{18} + x^{20} + x^{21} + x^{23} + x^{24} + x^{25}$$

is transmitted and

$$r(x) = 1 + x^3 + x^5 + x^6 + x^8 + x^{10} + x^{14} + x^{16} + x^{17} + x^{20} + x^{21} + x^{23} + x^{24} + x^{25}$$

is received. The syndromes are

$$S_1 = r(\alpha) = \alpha^{17} \qquad S_2 = r(\alpha^2) = \alpha^3 \qquad S_3 = r(\alpha^3) = 1 \qquad S_4 = r(\alpha^4) = \alpha^6.$$

Using the results above we find

$$\Lambda_1 = S_1 = \alpha^{17} \qquad \Lambda_2 = \frac{S_3 + S_1^3}{S_1} = \alpha^{22},$$

so that $\Lambda(x) = 1 + \alpha^{17}x + \alpha^{22}x^2$. The roots of this polynomial (found, e.g., using the Chien search) are at $x = \alpha^{13}$ and $x = \alpha^{27}$. Specifically, we could write

$$\Lambda(x) = \alpha^{22}(x + \alpha^{13})(x + \alpha^{27}).$$

The *reciprocals* of the roots are at α^{18} and α^4, so that the errors in transmission occurred at locations 4 and 18,

$$e(x) = x^4 + x^{18}.$$

It can be seen that $r(x) + e(x)$ is in fact equal to the transmitted codeword. □

6.4.2 Berlekamp-Massey Algorithm

While Peterson's method involves straightforward linear algebra, it is computationally complex in general. Starting with the matrix A in (6.11), it is examined to see if it is singular. This involves either attempting to solve the equations (e.g., by Gaussian elimination or equivalent), or computing the determinant to see if the solution can be found. If A is singular, then the last two rows and columns are dropped to form a new A matrix. Then the attempted solution must be *recomputed* starting over with the new A matrix.

The Berlekamp-Massey algorithm takes a different approach. Starting with a *small* problem, it works up to increasingly longer problems until it obtains an overall solution. However, at each stage it is able to re-use information it has already learned. Whereas as the computational complexity of the Peterson method is $O(\nu^3)$, the computational complexity of the Berlekamp-Massey algorithm is $O(\nu^2)$.

We have observed from the Newton's identity (6.10) that

$$S_j = -\sum_{i=1}^{\nu} \Lambda_i S_{j-i}, \qquad j = \nu+1, \nu+2, \ldots, 2t. \tag{6.12}$$

This formula describes the output of a linear feedback shift register (LFSR) with coefficients $\Lambda_1, \Lambda_2, \ldots, \Lambda_\nu$. In order for this formula to work, we must find the Λ_j coefficients in such a way that the LFSR generates the known sequence of syndromes $S_1, S_2, \ldots, S_{2t}$. Furthermore, by the maximum likelihood principle, the number of errors ν determined must be the *smallest* that is consistent with the observed syndromes. We therefore want to determine the *shortest* such LFSR.

In the Berlekamp-Massey algorithm, we build the LFSR that produces the entire sequence $\{S_1, S_2, \ldots, S_{2t}\}$ by successively modifying an existing LFSR, if necessary, to produce increasingly longer sequences. We start with an LFSR that could produce S_1. We determine if that LFSR could also produce the sequence $\{S_1, S_2\}$; if it can, then no modifications are necessary. If the sequence cannot be produced using the current LFSR configuration, we determine a new LFSR that can produce the longer sequence. Proceeding inductively in this way, we start from an LFSR capable of producing the sequence $\{S_1, S_2, \ldots, S_{k-1}\}$ and modify it, if necessary, so that it can also produce the sequence $\{S_1, S_2, \ldots, S_k\}$. At each stage, the modifications to the LFSR are accomplished so that the LFSR is the shortest possible. By this means, after completion of the algorithm an LFSR has been found that is able to produce $\{S_1, S_2, \ldots, S_{2t}\}$ and its coefficients correspond to the error locator polynomial $\Lambda(x)$ of *smallest* degree.

Since we build up the LFSR using information from prior computations, we need a notation to represent the $\Lambda(x)$ used at different stages of the algorithm. Let L_k denote the length of the LFSR produced at stage k of the algorithm. Let

$$\Lambda^{[k]}(x) = 1 + \Lambda_1^{[k]} x + \cdots + \Lambda_{L_k}^{[k]} x^{L_k}$$

be the *connection polynomial* at stage k, indicating the connections for the LFSR capable of producing the output sequence $\{S_1, S_2, \ldots, S_k\}$. That is,

$$S_j = -\sum_{i=1}^{L_k} \Lambda_i^{[k]} S_{j-i} \qquad j = L_k + 1, \ldots, k. \tag{6.13}$$

Note: It is important to realize that some of the coefficients in $\Lambda^{[k]}(x)$ may be zero, so that L_k may be different from the degree of $\Lambda^{[k]}(x)$. In realizations which use polynomial arithmetic, it is important to keep in mind what the length is as well as the degree.

At some intermediate step, suppose we have a connection polynomial $\Lambda^{[k-1]}(x)$ of length L_{k-1} that produces $\{S_1, S_2, \ldots, S_{k-1}\}$ for some $k - 1 < 2t$. We check if this connection polynomial also produces S_k by computing the output

$$\hat{S}_k = -\sum_{i=1}^{L_{k-1}} \Lambda_i^{[k-1]} S_{k-i}.$$

If $\hat{S}_k$ is equal to S_k, then there is no need to update the LFSR, so $\Lambda^{[k]}(x) = \Lambda^{[k-1]}(x)$ and $L_k = L_{k-1}$. Otherwise, there is some nonzero *discrepancy* associated with $\Lambda^{[k-1]}(x)$,

$$d_k = S_k - \hat{S}_k = S_k + \sum_{i=1}^{L_{k-1}} \Lambda_i^{[k-1]} S_{k-i} = \sum_{i=0}^{L_{k-1}} \Lambda_i^{[k-1]} S_{k-i}. \tag{6.14}$$

In this case, we update the connection polynomial using the formula

$$\Lambda^{[k]}(x) = \Lambda^{[k-1]}(x) + A x^l \Lambda^{[m-1]}(x), \tag{6.15}$$

where A is some element in the field, l is an integer, and $\Lambda^{[m-1]}(x)$ is one of the prior connection polynomials produced by our process associated with nonzero discrepancy d_m. (Initialization of this inductive process is discussed in the proof of Theorem 6.13.) Using this new connection polynomial, we compute the new discrepancy, denoted by d'_k, as

$$d'_k = \sum_{i=0}^{L_k} \Lambda_i^{[k]} S_{k-i}$$
$$= \sum_{i=0}^{L_{k-1}} \Lambda_i^{[k-1]} S_{k-i} + A \sum_{i=0}^{L_{m-1}} \Lambda_i^{[m-1]} S_{k-i-l}. \tag{6.16}$$

Now, let $l = k - m$. Then, by comparison with the definition of the discrepancy in (6.14), the second summation gives

$$A \sum_{i=0}^{L_{m-1}} \Lambda_i^{[m-1]} S_{m-i} = A d_m.$$

Thus, if we choose $A = -d_m^{-1} d_k$, then the summation in (6.16) gives

$$d'_k = d_k - d_m^{-1} d_k d_m = 0.$$

So the new connection polynomial produces the sequence $\{S_1, S_2, \ldots, S_k\}$ with no discrepancy.

6.4.3 Characterization of LFSR Length in Massey's Algorithm

The update in (6.15) is, in fact, the heart of Massey's algorithm. If all we need is an algorithm to find a connection polynomial, no further analysis is necessary. However, the problem was to find the *shortest* LFSR producing a given sequence. We have produced a means of finding an LFSR, but have no indication yet that it is the shortest. Establishing this requires some additional effort in the form of two theorems.

Theorem 6.12 *Suppose that an LFSR with connection polynomial $\Lambda^{[k-1]}(x)$ of length L_{k-1} produces the sequence $\{S_1, S_2, \ldots, S_{k-1}\}$, but not the sequence $\{S_1, S_2, \ldots, S_k\}$. Then any connection polynomial that produces the latter sequence must have a length L_k satisfying*

$$L_k \geq k - L_{k-1}.$$

Proof The theorem is only of practical interest if $L_{k-1} < k - 1$; otherwise it is trivial to produce the sequence. Let us take, then, $L_{k-1} < k - 1$. Let

$$\Lambda^{[k-1]}(x) = 1 + \Lambda_1^{[k-1]} x + \cdots + \Lambda_{L_{k-1}}^{[k-1]} x^{L_{k-1}}$$

represent the connection polynomial which produces $\{S_1, \ldots, S_{k-1}\}$ and let

$$\Lambda^{[k]}(x) = 1 + \Lambda_1^{[k]} x + \cdots + \Lambda_{L_k}^{[k]} x^{L_k}$$

denote the connection polynomial which produces $\{S_1, S_2, \ldots, S_k\}$. Now we do a proof by contradiction.

Assume (contrary to the theorem) that

$$L_k \leq k - 1 - L_{k-1}. \tag{6.17}$$

From the definitions of the connection polynomials, we observe that

$$-\sum_{i=1}^{L_{k-1}} \Lambda_i^{[k-1]} S_{j-i} \begin{cases} = S_j & j = L_{k-1}+1, L_{k-1}+2, \ldots, k-1 \\ \neq S_k & j = k \end{cases} \tag{6.18}$$

and

$$-\sum_{i=1}^{L_k} \Lambda_i^{[k]} S_{j-i} = S_j \qquad j = L_k+1, L_k+2, \ldots, k. \tag{6.19}$$

In particular, from (6.19), we have

$$S_k = -\sum_{i=1}^{L_k} \Lambda_i^{[k]} S_{k-i}. \tag{6.20}$$

The values of S_i involved in this summation range from S_{k-1} to S_{k-L_k}. The indices of these values form a set $\{k - L_k, k - L_k + 1, \ldots, k - 1\}$. By the (contrary) assumption made in (6.17), we have $k - L_k \geq L_{k-1} + 1$, so that the set of indices $\{k - L_k, k - L_k + 1, \ldots, k - 1\}$ are a subset of the set of indices $\{L_{k-1} + 1, L_{k-1} + 2, \ldots, k - 1\}$ appearing in (6.18). Thus each S_{k-i} appearing on the right-hand side of (6.20) can be replaced by the summation expression from (6.18) and we can write

$$S_k = -\sum_{i=1}^{L_k} \Lambda_i^{[k]} S_{k-i} = \sum_{i=1}^{L_k} \Lambda_i^{[k]} \sum_{j=1}^{L_{k-1}} \Lambda_j^{[k-1]} S_{k-i-j}.$$

Interchanging the order of summation we have

$$S_k = \sum_{j=1}^{L_{k-1}} \Lambda_j^{[k-1]} \sum_{i=1}^{L_k} \Lambda_i^{[k]} S_{k-i-j}. \tag{6.21}$$

Now setting $j = k$ in (6.18), we obtain

$$S_k \neq -\sum_{i=1}^{L_{k-1}} \Lambda_i^{[k-1]} S_{k-i}. \tag{6.22}$$

In this summation the indices of S form the set $\{k - L_{k-1}, \ldots, k - 1\}$. By the (contrary) assumption (6.17), $L_k + 1 \leq k - L_{k-1}$, so the sequence of indices $\{k - L_{k-1}, \ldots, k - 1\}$ is a subset of the range $L_k + 1, \ldots, k$ of (6.19). Thus we can replace each S_{k-i} in the summation of (6.22) with the expression from (6.19) to obtain

$$S_k \neq \sum_{i=1}^{L_{k-1}} \Lambda_i^{[k-1]} \sum_{j=1}^{L_k} \Lambda_j^{[k]} S_{k-i-j}. \tag{6.23}$$

Comparing (6.21) with (6.23), the double summations are the same, but the equality in the first case and the inequality in the second case indicate a contradiction. Hence, the assumption on the length of the LFSRs must have been incorrect. By this contradiction, we must have

$$L_k \geq k - L_{k-1}.$$

If we take this to be the case, the index ranges which gave rise to the substitutions leading to the contradiction do not occur. □

Since the shortest LFSR that produces the sequence $\{S_1, S_2, \ldots, S_k\}$ must also produce the first part of that sequence, we must have $L_k \geq L_{k-1}$. Combining this with the result of the theorem, we obtain

$$L_k \geq \max(L_{k-1}, k - L_{k-1}). \tag{6.24}$$

We observe that the shift register cannot become shorter as more outputs are produced.

We have seen how to update the LFSR to produce a longer sequence using (6.15) and have also seen that there is a lower bound on the length of the LFSR. We now show that this lower bound can be achieved with equality, thus providing the *shortest* LFSR which produces the desired sequence.

Theorem 6.13 *In the update procedure, if $\Lambda^{[k]}(x) \neq \Lambda^{[k-1]}(x)$, then a new LFSR can be found whose length satisfies*

$$L_k = \max(L_{k-1}, k - L_{k-1}). \tag{6.25}$$

Proof We do a proof by induction. To check when $k = 1$ (which also indicates how to get the algorithm started), take $L_0 = 0$ and $\Lambda^{[0]}(x) = 1$. We find that

$$d_1 = S_1.$$

If $S_1 = 0$, then no update is necessary. If $S_1 \neq 0$, then we take $\Lambda^{[m]}(x) = \Lambda^{[0]}(x) = 1$, so that $l = 1 - 0 = 1$. Also, take $d_m = 1$. The updated polynomial is

$$\Lambda^{[1]}(x) = 1 + S_1 x,$$

which has degree L_1 satisfying

$$L_1 = \max(L_0, 1 - L_0) = 1.$$

In this case, (6.13) is vacuously true for the sequence consisting of the single point $\{S_1\}$.

Now let $\Lambda^{[m-1]}(x)$, $m < k-1$, denote the *last* connection polynomial before $\Lambda^{[k-1]}(x)$ with $L_{m-1} < L_{k-1}$ that can produce the sequence $\{S_1, S_2, \ldots, S_{m-1}\}$ but not the sequence $\{S_1, S_2, \ldots, S_m\}$. Then

$$L_m = L_{k-1};$$

hence, in light of the inductive hypothesis (6.25),

$$L_m = m - L_{m-1} = L_{k-1}, \qquad \text{or} \qquad L_{m-1} - m = -L_{k-1}. \tag{6.26}$$

By the update formula (6.15) with $l = k - m$, we note that

$$L_k = \max(L_{k-1}, k - m + L_{m-1}).$$

Using $L_{m-1} - m$ from (6.26) we find that

$$L_k = \max(L_{k-1}, k - L_{k-1}).$$

□

In the update step, we observe that the new length is the same as the old length if $L_{k-1} \geq k - L_{k-1}$, that is, if

$$2L_{k-1} \geq k.$$

In this case, the connection polynomial is updated, but there is no change in length.

The shift-register synthesis algorithm, known as Massey's algorithm, is presented first in pseudocode as Algorithm 6.1, where we use the notations

$$c(x) = \Lambda^{[k]}(x)$$

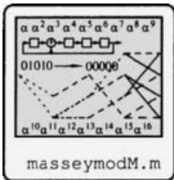
to indicate the "current" connection polynomial and

$$p(x) = \Lambda^{[m-1]}(x)$$

to indicate a "previous" connection polynomial. Also, N is the number of input symbols ($N = 2t$ for many decoding problems).

Algorithm 6.1 Massey's Algorithm

Input: $S_1, S_2, \ldots, S_N$
Initialize:
$L = 0$ (the current length of the LFSR)
$c(x) = 1$ (the current connection polynomial)
$p(x) = 1$ (the connection polynomial before last length change)
$l = 1$ (l is $k - m$, the amount of shift in update)
$d_m = 1$ (previous discrepancy)
for $k = 1$ to N
  $d = S_k + \sum_{i=1}^{L} c_i S_{k-i}$ (compute discrepancy)
  if ($d = 0$) (no change in polynomial)
    $l = l + 1$
  else
    if ($2L \geq k$) then (no-length change in update)
      $c(x) = c(x) - d d_m^{-1} x^l p(x)$
      $l = l + 1$
    else (update c with length change)
      $t(x) = c(x)$ (temporary storage)
      $c(x) = c(x) - d d_m^{-1} x^l p(x)$
      $L = k - L$
      $p(x) = t(x)$
      $d_m = d$
      $l = 1$
    end
  end
end



Example 6.13 For the sequence $S = \{1, 1, 1, 0, 1, 0, 0\}$ the feedback connection polynomial obtained by a call to `massey` is $\{1, 1, 0, 1\}$, which corresponds to the polynomial

$$C(x) = 1 + x + x^3.$$

Thus the elements of S are related by

$$S_j = S_{j-1} + S_{j-3},$$

for $j \geq 3$. Details of the operation of the algorithm are presented in Table 6.5. □

Table 6.5: Evolution of the Berlekamp-Massey Algorithm for the Input Sequence $\{1, 1, 1, 0, 1, 0, 0\}$.

k	S_k	d_k	$c(x)$	L	$p(x)$	l	d_m
1	1	1	$1+x$	1	1	1	1
2	1	0	$1+x$	1	1	2	1
3	1	0	$1+x$	1	1	3	1
4	0	1	$1+x+x^3$	3	$1+x$	1	1
5	1	0	$1+x+x^3$	3	$1+x$	2	1
6	0	0	$1+x+x^3$	3	$1+x$	3	1
7	0	0	$1+x+x^3$	3	$1+x$	4	1

Example 6.14 For the (31,21) binary double-error correcting code with decoding in Example 6.12, let us employ the Berlekamp-Massey algorithm to find the error locating polynomial. Recall from that example that the syndromes are $S_1 = \alpha^{17}$, $S_2 = \alpha^3$, $S_3 = 1$, and $S_4 = \alpha^6$. Running the Berlekamp-Massey algorithm over $GF(32)$ results in the computations shown in Table 6.6. The final connection polynomial $c(x) = 1 + \alpha^{17}x + \alpha^{22}x^2$ is the error location polynomial previously found using Peterson's algorithm. (In the current case, there are more computations using the Berlekamp-Massey algorithm, but for longer codes with more errors, the latter would be more efficient.)

Table 6.6: Berlekamp-Massey Algorithm for a Double-Error Correcting Code

k	S_k	d_k	$c(x)$	L	$p(x)$	l	d_m
1	α^{17}	α^{17}	$1+\alpha^{17}x$	1	1	1	α^{17}
2	α^3	0	$1+\alpha^{17}x$	1	1	2	α^{17}
3	1	α^8	$1+\alpha^{17}x+\alpha^{22}x^2$	2	$1+\alpha^{17}x$	1	α^8
4	α^6	0	$1+\alpha^{17}x+\alpha^{22}x^2$	2	$1+\alpha^{17}x$	2	α^8

□

6.4.4 Simplifications for Binary Codes

Consider again the Berlekamp-Massey algorithm computations for decoding a BCH code, as presented in Table 6.6. Note that d_k is 0 for every even k. This result holds in all cases for BCH codes:

Lemma 6.14 *When the sequence of input symbols to the Berlekamp-Massey algorithm are syndromes from a binary BCH code, then the discrepancy d_k is equal to 0 for all even k (when 1-based indexing is used).*

As a result, there is never an update for these steps of the algorithm, so they can be merged into the next step. This cuts the complexity of the algorithm approximately in half. A restatement of the algorithm for BCH decoding is presented below.

Algorithm 6.2 Massey's Algorithm for Binary BCH Decoding

Input: $S_1, S_2, \ldots, S_N$, where $N = 2t$
Initialize:
$L = 0$ (the current length of the LFSR)

$c(x) = 1$ (the current connection polynomial)
$p(x) = 1$ (the connection polynomial before last length change)
$l = 1$ (l is $k - m$, the amount of shift in update)
$d_m = 1$ (previous discrepancy)
for $k = 1$ to N in steps of 2
  $d = S_k + \sum_{i=1}^{L} c_i S_{k-i}$ (compute discrepancy)
  if ($d = 0$) (no change in polynomial)
    $l = l + 1$
  else
    if ($2L \geq k$) then (no-length change in update)
      $c(x) = c(x) - d d_m^{-1} x^l p(x)$
      $l = l + 1$
    else (update c with length change)
      $t(x) = c(x)$ (temporary storage)
      $c(x) = c(x) - d d_m^{-1} x^l p(x)$
      $L = k - L$
      $p(x) = t(x)$
      $d_m = d$
      $l = 1$
    end
  end
  $l = l + 1$; (accounts for the values of k skipped)
end

Example 6.15 Returning to the (31,21) code from the previous example, if we call the BCH-modified Berlekamp-Massey algorithm with the syndrome sequence $S_1 = \alpha^{17}$, $S_2 = \alpha^3$, $S_3 = 1$, and $S_4 = \alpha^6$, we obtain the results in Table 6.7. Only two steps of the algorithm are necessary and the same error locator polynomial is obtained as before. □

Table 6.7: Berlekamp-Massey Algorithm for a Double-Error Correcting code: Simplifications for the Binary Code

k	S_k	d_k	$c(x)$	L	$p(x)$	l	d_m
0	α^{17}	α^{17}	$1+\alpha^{17}x$	1	1	2	α^{17}
2	1	α^8	$1+\alpha^{17}x+\alpha^{22}x^2$	2	$1+\alpha^{17}x$	2	α^8

The odd-indexed discrepancies are zero due to the fact that for binary codes, the syndromes S_j have the property that

$$(S_j)^2 = S_{2j}. \tag{6.27}$$

We call this condition the syndrome conjugacy condition. Equation (6.27) follows from (6.4) and freshman exponentiation.

For the example we have been following,

$$S_1^2 = (\alpha^{17})^2 = \alpha^3 = S_2 \qquad S_2^2 = (\alpha^3)^2 = \alpha^6 = S_4.$$

Example 6.16 We now present an entire decoding process for the three-error correcting (15, 5) binary code generated by

$$g(x) = 1 + x + x^2 + x^4 + x^5 + x^8 + x^{10}.$$

Suppose the all-zero vector is transmitted and the received vector is

$$\mathbf{r} = (0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0).$$

Then $r(x) = x + x^3 + x^8$.

Step 1 Compute the syndromes. Evaluating $r(x)$ at $x = \alpha, \alpha^2, \ldots, \alpha^6$ we find the syndromes

$$S_1 = \alpha^{12} \quad S_2 = \alpha^9 \quad S_3 = \alpha^3 \quad S_4 = \alpha^3 \quad S_5 = 0 \quad S_6 = \alpha^6.$$

Step 2 Compute the error locator polynomial.

A call to the binary Berlekamp-Massey algorithm yields the following computations.

k	S_k	d_k	$c(x)$	L	$p(x)$	l	d_m
1	α^{12}	α^{12}	$1+\alpha^{12}x$	1	1	2	α^{12}
3	α^3	α^2	$1+\alpha^{12}x+\alpha^5x^2$	2	$1+\alpha^{12}x$	2	α^2
5	0	α^2	$1+\alpha^{12}x+\alpha^{10}x^2+\alpha^{12}x^3$	3	$1+\alpha^{12}x+\alpha^5x^2$	2	α^2

The error locator polynomial is thus

$$\Lambda(x) = 1 + \alpha^{12}x + \alpha^{10}x^2 + \alpha^{12}x^3.$$

Step 3 Find the roots of the error locator polynomial. Using the Chien search function, we find roots at α^7, α^{12} and α^{14}. Inverting these, the error locators are

$$X_1 = \alpha^8 \quad X_2 = \alpha^3 \quad X_3 = \alpha,$$

indicating that errors at positions 8, 3, and 1.

Step 4 Determine the error values: for a binary BCH code, any errors have value 1.

Step 5 Correct the errors: Add the error values (1) at the error locations, to obtain the decoded vector of all zeros.

□

6.5 Non-Binary BCH and RS Decoding

For nonbinary BCH or RS decoding, some additional work is necessary. Some extra care is needed to find the error locators, then the error values must be determined.

From (6.3) we can write

$$\begin{aligned}
S_1 &= e_{i_1}X_1 + e_{i_2}X_2 + \cdots + e_{i_\nu}X_\nu \\
S_2 &= e_{i_1}X_1^2 + e_{i_2}X_2^2 + \cdots + e_{i_\nu}X_\nu^2 \\
S_3 &= e_{i_1}X_1^3 + e_{i_2}X_2^3 + \cdots + e_{i_\nu}X_\nu^3 \\
&\vdots \\
S_{2t} &= e_{i_1}X_1^{2t} + e_{i_2}X_2^{2t} + \cdots + e_{i_\nu}X_\nu^{2t}.
\end{aligned}$$

Because of the e_{i_l} coefficients, these are not power-sum symmetric functions as was the case for binary codes. Nevertheless, in a similar manner it is possible to make use of an error locator polynomial.

Lemma 6.15 *The syndromes and the coefficients of the error locator polynomial* $\Lambda(x) = \Lambda_0 + \Lambda_1 x + \cdots + \Lambda_\nu x^\nu$ *are related by*

$$\Lambda_\nu S_{j-\nu} + \Lambda_{\nu-1}S_{j-\nu+1} + \cdots + \Lambda_1 S_{j-1} + S_j = 0. \tag{6.28}$$

Proof Evaluating the error locator polynomial $\Lambda(x) = \prod_{i=1}^{\nu}(1 - X_i x)$ at an error locator X_l,

$$\Lambda(X_l^{-1}) = 0 = \Lambda_\nu X_l^{-\nu} + \Lambda_{\nu-1} X_l^{1-\nu} + \cdots + \Lambda_1 X_l^{-1} + \Lambda_0.$$

Multiplying this equation by $e_{i_l} X_l^j$ we obtain

$$e_{i_l} X_l^j \Lambda(X_l^{-1}) = e_{i_l}(\Lambda_\nu X_l^{j-\nu} + \Lambda_{\nu-1} X_l^{j+1-\nu} + \cdots + \Lambda_1 X_l^{j-1} + \Lambda_0 X_l^j) = 0 \quad (6.29)$$

Summing (6.29) over l we obtain

$$\begin{aligned} 0 &= \sum_{l=1}^{\nu} e_{i_l}(\Lambda_\nu X_l^{j-\nu} + \Lambda_{\nu-1} X_l^{j+1-\nu} + \cdots + \Lambda_1 X_l^{j-1} + \Lambda_0 X_l^j) \\ &= \Lambda_\nu \sum_{l=1}^{\nu} e_{i_l} X_l^{j-\nu} + \Lambda_{\nu-1} \sum_{i=1}^{\nu} e_{i_l} X_l^{j+1-\nu} + \cdots + \Lambda_1 \sum_{l=1}^{\nu} e_{i_l} X_l^{j-1} + \Lambda_0 \sum_{l=1}^{\nu} e_{i_l} X_l^j. \end{aligned}$$

In light of (6.3), the latter equation can be written as

$$\Lambda_\nu S_{j-\nu} + \Lambda_{\nu-1} S_{j-\nu+1} + \cdots + \Lambda_1 S_{j-1} + \Lambda_0 S_j = 0.$$

□

Because (6.28) holds, the Berlekamp-Massey algorithm (in its non-binary formulation) can be used to find the coefficients of the error locator polynomial, just as for binary codes.

6.5.1 Forney's Algorithm

Having found the error-locator polynomial and its roots, there is still one more step for the non-binary BCH or RS codes: we have to find the error values. Let us return to the syndrome,

$$S_j = \sum_{l=1}^{\nu} e_{i_l} X_l^j, \qquad j = 1, 2, \ldots, 2t.$$

Knowing the error locators (obtained from the roots of the error locator polynomial) it is straightforward to set up and solve a set of linear equations:

$$\begin{bmatrix} X_1 & X_2 & X_3 & \cdots & X_\nu \\ X_1^2 & X_2^2 & X_3^2 & \cdots & X_\nu^2 \\ \vdots & & & & \\ X_1^{2t} & X_2^{2t} & X_3^{2t} & \cdots & X_\nu^{2t} \end{bmatrix} \begin{bmatrix} e_{i_1} \\ e_{i_2} \\ \vdots \\ e_{i_\nu} \end{bmatrix} = \begin{bmatrix} S_1 \\ S_2 \\ \vdots \\ S_{2t} \end{bmatrix}. \quad (6.30)$$

However, there is a method which is computationally easier and in addition provides us a key insight for another way of doing the decoding. It may be observed that the matrix in (6.30) is essentially a Vandermonde matrix. There exist fast algorithms for solving Vandermonde systems (see, e.g., [121]). One of these which applies specifically to this problem is known as *Forney's algorithm*.

Before presenting the formula, a few necessary definitions must be established. A *syndrome polynomial* is defined as

$$S(x) = S_1 + S_2 x + S_3 x^2 \cdots + S_{2t} x^{2t-1} = \sum_{j=0}^{2t-1} S_{j+1} x^j. \quad (6.31)$$

Also an *error-evaluator polynomial* $\Omega(x)$ is defined[1] by

$$\boxed{\Omega(x) = S(x)\Lambda(x) \pmod{x^{2t}}.} \tag{6.32}$$

This equation is called the **key equation**. Note that the effect of computing modulo x^{2t} is to discard all terms of degree $2t$ or higher.

Definition 6.5 Let $f(x) = f_0 + f_1 x + f_2 x^2 + \cdots + f_t x^t$ be a polynomial with coefficients in some field $\mathbb{F}$. The **formal derivative** $f'(x)$ of $f(x)$ is computed using the conventional rules of polynomial differentiation:

$$f'(x) = f_1 + 2f_2 x + 3f_3 x^2 + \cdots + t f_t x^{t-1}, \tag{6.33}$$

where, as usual, mf_i for $m \in \mathbb{Z}$ and $f_i \in \mathbb{F}$ denotes repeated addition:

$$mf_i = \underbrace{f_i + f_i + \cdots + f_i}_{m \text{ summands}}.$$

□

There is no implication of any kind of limiting process in formal differentiation: it simply corresponds to formal manipulation of symbols. Based on this definition, it can be shown that many of the conventional rules of differentiation apply. For example, the product rule holds:

$$[f(x)g(x)]' = f'(x)g(x) + f(x)g'(x).$$

If $f(x) \in \mathbb{F}[x]$, where $\mathbb{F}$ is a field of characteristic 2, then $f'(x)$ has no odd-powered terms.

Theorem 6.16 *(Forney's algorithm) The error values for a Reed-Solomon code are computed by*

$$e_{i_k} = -\frac{\Omega(X_k^{-1})}{\Lambda'(X_k^{-1})}, \tag{6.34}$$

where $\Lambda'(x)$ is the formal derivative of $\Lambda(x)$.

Proof First note that over any ring,

$$(1 - x^{2t}) = (1 - x)(1 + x + x^2 + \cdots + x^{2t-1}) = (1 - x)\sum_{j=0}^{2t-1} x^j. \tag{6.35}$$

Observe:

$$\begin{aligned}
\Omega(x) &= S(x)\Lambda(x) \pmod{x^{2t}} \\
&= \left(\sum_{j=0}^{2t-1}\sum_{l=1}^{\nu} e_{i_l} X_l^{j+1} x^j\right)\left(\prod_{i=1}^{\nu}(1 - X_i x)\right) \pmod{x^{2t}} \\
&= \sum_{l=1}^{\nu} e_{i_l} X_l \sum_{j=0}^{2t-1}(X_l x)^j \prod_{i=1}^{\nu}(1 - X_i x) \pmod{x^{2t}} \\
&= \sum_{l=1}^{\nu} e_{i_l} X_l \left[(1 - X_l x)\sum_{j=0}^{2t-1}(X_l x)^j\right]\prod_{i \neq l}(1 - X_i x) \pmod{x^{2t}}.
\end{aligned}$$

[1]Some authors define $S(x) = S_1 x + S_2 x^2 + \cdots + S_{2t} x^{2t}$, in which case they define $\Omega(x) = (1 + S(x))\Lambda(x) \pmod{x^{2t+1}}$ and obtain $e_{i_k} = -X_k \Omega(X_k^{-1})/\Lambda'(X_k^{-1})$.

From (6.35),

$$(1 - X_l x)\sum_{j=0}^{2t-1}(X_l x)^j = 1 - (X_l x)^{2t}.$$

Since $(X_l x)^{2t} \pmod{x^{2t}} = 0$ we have

$$S(x)\Lambda(x) \pmod{x^{2t}} = \sum_{l=1}^{\nu} e_{i_l} X_l \prod_{i \neq l}(1 - X_i x).$$

Thus

$$\Omega(x) = \sum_{l=1}^{\nu} e_{i_l} X_l \prod_{i \neq l}(1 - X_i x).$$

The trick now is to isolate a particular e_{i_k} on the right-hand side of this expression.

Evaluate $\Omega(x)$ at $x = X_k^{-1}$:

$$\Omega(X_k^{-1}) = \sum_{l=1}^{\nu} e_{i_l} X_l \prod_{i \neq l}(1 - X_i X_k^{-1}).$$

Every term in the sum results in a product that has a zero in it, except the term when $l = k$, since that term is skipped. We thus obtain

$$\Omega(X_k^{-1}) = e_{i_k} X_k \prod_{i \neq k}(1 - X_i X_k^{-1}).$$

We can thus write

$$e_{i_k} = \frac{\Omega(X_k^{-1})}{X_k \prod_{i \neq k}(1 - X_i X_k^{-1})}. \tag{6.36}$$

Once $\Omega(x)$ is known, the error values can thus be computed. However, there are some computational simplifications.

The formal derivative of $\Lambda(x)$ is

$$\Lambda'(x) = \frac{d}{dx}\prod_{i=1}^{\nu}(1 - X_i x) = -\sum_{l=1}^{\nu} X_l \prod_{i \neq l}(1 - X_i x).$$

Then

$$\Lambda'(X_k^{-1}) = -X_k \prod_{i \neq k}(1 - X_i X_k^{-1}).$$

Substitution of this result into (6.36) yields (6.34). □

Example 6.17 Working over $GF(8)$ in a code where $t = 2$, suppose $S(x) = \alpha^6 + \alpha^3 x + \alpha^4 x^2 + \alpha^3 x^3$. We find (say using the B-M algorithm and the Chien search) that the error locator polynomial is

$$\Lambda(x) = 1 + \alpha^2 x + \alpha x^2 = (1 + \alpha^3 x)(1 + \alpha^5 x).$$

That is, the error locators (reciprocals of the roots of $\Lambda(x)$) are $X_1 = \alpha^3$ and $X_2 = \alpha^5$. We have

$\Omega(x) = (\alpha^6 + \alpha^3 x + \alpha^4 x^2 + \alpha^3 x^3)(1 + \alpha^2 x + \alpha x^2) \pmod{x^4} = (\alpha^6 + x + \alpha^4 x^5) \pmod{x^4} = \alpha^6 + x$

and

$$\Lambda'(x) = \alpha^2 + 2\alpha x = \alpha^2.$$

So

$$e_{i_k} = -\frac{\alpha^6 + x}{\alpha^2}\bigg|_{x=X_k^{-1}} = \alpha^4 + \alpha^5 X_k^{-1}.$$

Using the error locator $X_1 = \alpha^3$ we find

$$e_3 = \alpha^4 + \alpha^5(\alpha^3)^{-1} = \alpha$$

and for the error locator $X_2 = \alpha^5$,

$$e_5 = \alpha^4 + \alpha^5(\alpha^5)^{-1} = \alpha^5.$$

The error polynomial is $e(x) = \alpha x^3 + \alpha^5 x^5$. □

Example 6.18 We consider the entire decoding process for (15,9) code of Example 6.8, using the message and code polynomials in Example 6.10. Suppose the received polynomial is

$$r(x) = \alpha^8 + \alpha^2 x + \underline{\alpha^{13}x^2} + \alpha^3 x^3 + \alpha^5 x^4 + \alpha x^5 + \alpha^8 x^6 + \alpha x^7 + \underline{\alpha x^8} + \alpha^5 x^9 + \alpha^3 x^{10}$$
$$+ \alpha^4 x^{11} + \alpha^9 x^{12} + \alpha^{12} x^{13} + \underline{\alpha^5 x^{14}}.$$

(Errors are in the underlined positions.)

The syndromes are

$$S_1 = r(\alpha) = \alpha^{13} \quad S_2 = r(\alpha^2) = \alpha^4 \quad S_3 = r(\alpha^3) = \alpha^8$$
$$S_4 = r(\alpha^4) = \alpha^2 \quad S_5 = r(\alpha^5) = \alpha^3 \quad S_6 = r(\alpha^6) = \alpha^8$$

so

$$S(x) = \alpha^{13} + \alpha^4 x + \alpha^8 x^2 + \alpha^2 x^3 + \alpha^3 x^4 + \alpha^8 x^5$$

and the error locator polynomial determined by the Berlekamp-Massey algorithm is

$$\Lambda(x) = 1 + \alpha^3 x + \alpha^{11} x^2 + \alpha^9 x^3.$$

The details of the Berlekamp-Massey computations are shown in Table 6.8.

Table 6.8: Berlekamp-Massey Algorithm for a Triple-Error Correcting Code

k	S_k	d_k	$c(x)$	L	$p(x)$	l	d_m
1	α^{13}	α^{13}	$1+\alpha^{13}x$	1	1	1	α^{13}
2	α^4	α^{13}	$1+\alpha^6 x$	1	1	2	α^{13}
3	α^8	α	$1+\alpha^6 x+\alpha^3 x^2$	2	$1+\alpha^6 x$	1	α
4	α^2	α^5	$1+\alpha^{12}x+\alpha^{12}x^2$	2	$1+\alpha^6 x$	2	α
5	α^3	α^{10}	$1+\alpha^{12}x+\alpha^8 x^2+x^3$	3	$1+\alpha^{12}x+\alpha^{12}x^2$	1	α^{10}
6	α^8	α^5	$1+\alpha^3 x+\alpha^{11}x^2+\alpha^9 x^3$	3	$1+\alpha^{12}x+\alpha^{12}x^2$	2	α^{10}

The roots of $\Lambda(x)$ are at α, α^7 and α^{13}, so the error locators (the reciprocal of the roots) are

$$X_1 = \alpha^{14} \quad X_2 = \alpha^8 \quad X_3 = \alpha^2,$$

corresponding to errors at positions 14, 8, and 2. The error evaluator polynomial is

$$\Omega(x) = \alpha^{13} + x + \alpha^2 x^2.$$

Then the computations to find the error values are:

$$\begin{array}{llll} X_1 = \alpha^{14}: & \Omega(X_1^{-1}) = \alpha^6 & \Lambda'(X_1^{-1}) = \alpha^5 & e_{14} = \alpha \\ X_2 = \alpha^8: & \Omega(X_2^{-1}) = \alpha^2 & \Lambda'(X_2^{-1}) = \alpha^{13} & e_8 = \alpha^4 \\ X_3 = \alpha^2: & \Omega(X_3^{-1}) = \alpha^{13} & \Lambda'(X_3^{-1}) = \alpha^{11} & e_2 = \alpha^2 \end{array}$$

The error polynomial is thus

$$e(x) = \alpha^2 x^2 + \alpha^4 x^8 + \alpha x^{14}$$

and the decoded polynomial is

$$\alpha^8 + \alpha^2 x + \alpha^{14} x^2 + \alpha^3 x^3 + \alpha^5 x^4 + \alpha x^5 + \alpha^8 x^6 + \alpha x^7 + x^8 + \alpha^5 x^9 + \alpha^3 x^{10} \\ + \alpha^4 x^{11} + \alpha^9 x^{12} + \alpha^{12} x^{13} + \alpha^2 x^{14}.$$

which is the same as the original codeword $c(x)$. □

6.6 Euclidean Algorithm for the Error Locator Polynomial

We have seen that the Berlekamp-Massey algorithm can be used to construct the error locator polynomial. In this section, we show that the Euclidean algorithm can also be used to construct error locator polynomials. This approach to decoding is often called the Sugiyama algorithm [324].

We return to the key equation:

$$\Omega(x) = S(x)\Lambda(x) \pmod{x^{2t}}. \tag{6.37}$$

Given only $S(x)$ and t, we desire to determine the error locator polynomial $\Lambda(x)$ and the error evaluator polynomial $\Omega(x)$. As stated, this problem seems hopelessly underconstrained. However, recall that (6.37) means that

$$\Theta(x)(x^{2t}) + \Lambda(x)S(x) = \Omega(x)$$

for some polynomial $\Theta(x)$. (See (5.16).) Also recall that the extended Euclidean algorithm returns, for a pair of elements (a, b) from a Euclidean domain, a pair of elements (s, t) such that

$$as + bt = c,$$

where c is the GCD of a and b. In our case, we run the extended Euclidean algorithm to obtain a sequence of polynomials $\Theta^{[k]}(x)$, $\Lambda^{[k]}(x)$ and $\Omega^{[k]}(x)$ satisfying

$$\Theta^{[k]}(x)x^{2t} + \Lambda^{[k]}(x)S(x) = \Omega^{[k]}(x).$$

This is exactly the circumstance described in Section 5.2.3. Recall that the stopping criterion there is based on the observation that the polynomial we are here calling $\Omega(x)$ must have degree $< t$.

The steps to decode using the Euclidean algorithm are summarized as follows:

1. Compute the syndromes and the syndrome polynomial $S(x) = S_1 + S_2 x + \cdots + S_{2t} x^{2t-1}$.
2. Run the Euclidean algorithm with $a(x) = x^{2t}$ and $b(x) = S(x)$, until $\deg(r_i(x)) < t$. Then $\Omega(x) = r_i(x)$ and $\Lambda(x) = t_i(x)$.
3. Find the roots of $\Lambda(x)$ and the error locators X_i.

4. Solve for the error values using (6.34).

Actually, since $\Lambda(x)$ has $\Lambda_0 = 1$, it may be necessary to normalize, $\Lambda(x) = t_i(x)/t_i(0)$.

Example 6.19 For the syndrome polynomial

$$S(x) = \alpha^{13} + \alpha^4 x + \alpha^8 x^2 + \alpha^2 x^3 + \alpha^3 x^4 + \alpha^8 x^5$$

of the triple-error correcting polynomial of Example 6.18, let

$$a(x) = x^6 \qquad b(x) = S(x).$$

Then calling the Euclidean algorithm to stop when the degree of $r_i(x)$ is less than 3 yields

$$s_i(x) = \alpha^{14} + \alpha^6 x + \alpha^2 x^2$$

$$t_i(x) = 1 + \alpha^3 x + \alpha^{11} x^2 + \alpha^9 x^3$$

$$r_i(x) = \alpha^{13} + x + \alpha^2 x^2.$$

The error locator polynomial is

$$\Lambda(x) = t_i(x) = 1 + \alpha^3 x + \alpha^{11} x^2 + \alpha^9 x^3,$$

as before. □

In terms of computational efficiency, it appears that the Berlekamp-Massey algorithm procedure may be slightly better than the Euclidean algorithm for binary codes, since the Berlekamp-Massey deals with polynomials no longer than the error locator polynomial, while the Euclidean algorithm may have intermediate polynomials of higher degree. However, the computational complexity is probably quite similar. Also, the error evaluator polynomial $\Omega(x)$ is automatically obtained as a useful byproduct of the Euclidean algorithm method.

6.7 Erasure Decoding for Nonbinary BCH or RS codes

Erasures and binary erasure decoding were introduced in Section 3.8. Here we describe erasure decoding for nonbinary BCH or RS codes.

Let the received word **r** have ν errors and f erasures, with the errors at $i_1, i_2, \ldots, i_\nu$ and the erasures at $j_1, j_2, \ldots, j_f$. We employ error locators as before, $X_1, X_2, \ldots, X_\nu$, with $X_k = \alpha^{i_k}$. Now introduce erasure locators

$$Y_1 = \alpha^{j_1} \qquad Y_2 = \alpha^{j_2} \qquad \cdots \qquad Y_f = \alpha^{j_f}.$$

The decoder must find: the errors locators X_k, the error values e_{i_k} at the error locations, and values at the erasures f_{j_k}.

We begin by creating an *erasure locator polynomial*,

$$\Gamma(x) = \prod_{l=1}^{f}(1 - Y_l x),$$

which is known, since the erasure locations are known.

Since received symbol values are necessary to compute the syndromes, it is convenient to (temporarily) fill in the erased symbol locations with zeros. Then (assuming for convenience a narrow sense code)

$$S_l = r(\alpha^l)\Big|_{\text{zeros at erasures}} = \sum_{k=1}^{\nu} e_{i_k} X_k^l + \sum_{k=1}^{f} f_{i_k} Y_k^l, \qquad l = 1, 2, \ldots, 2t.$$

As before, we create a syndrome polynomial,

$$S(x) = \sum_{l=0}^{2t-1} S_{l+1} x^l$$

and create the *key equation*

$$\Lambda(x)\,[\Gamma(x)S(x)] = \Omega(x) \pmod{x^{2t}}.$$

Letting

$$\Xi(x) = \Gamma(x)S(x) \pmod{x^{2t}}$$

be used to represent the data that are known once the syndromes are computed, we can write the key equation as

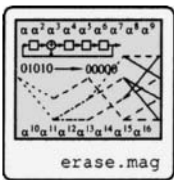
$$\Lambda(x)\Xi(x) = \Omega(x) \pmod{x^{2t}}.$$

This key equation has exactly the same form as that in (6.32). Thus, any of the decoding algorithms already introduced can be used to solve the key equation for $\Lambda(x)$ (e.g., the Berlekamp-Massey algorithm or the Euclidean algorithm), using $\Xi(x)$ in place of $S(x)$ in these algorithms. If the Berlekamp-Massey algorithm is used, then the input is the coefficients Ξ_0, Ξ_1, Ξ_{2t-1}, in place of $S_1, S_2, \ldots, S_{2t}$. If the Euclidean algorithm is used, set $a(x) = x^{2t}$ and $b(x) = \Xi(x)$ and stop when

$$\deg(r_i(x)) \leq \begin{cases} t + \frac{f}{2} & \text{if } f \text{ is even} \\ t + \frac{f-1}{2} & \text{if } f \text{ is odd.} \end{cases}$$

Once $\Lambda(x)$ is known, its roots are found (as usual). The error and erasure values can then be found using a modification of Forney's algorithm. The polynomial

$$\Phi(x) = \Lambda(x)\Gamma(x),$$



called the *combined error/erasure locator polynomial* is computed. Then

$$e_{i_k} = -\frac{\Omega(X_k^{-1})}{\Phi'(X_k^{-1})} \quad \text{and} \quad f_{j_k} = -\frac{\Omega(Y_k^{-1})}{\Phi'(Y_k^{-1})}.$$

Example 6.20 For a triple-error correcting ($t = 3$) Reed-Solomon code over $GF(16)$, suppose that

$$r(x) = \alpha^5 x^{11} + \alpha^6 x^9 + \mathsf{E}x^7 + \mathsf{E}x^6 + \alpha^{11}x^5 + x^4 + \alpha^{11}x^3 + \alpha^6 x^2 + \alpha^{12},$$

where E denotes that the position is erased. The erasure locations are thus at $j_1 = 7$ and $j_2 = 6$, the erasure locators are $Y_1 = \alpha^7$ and $Y_2 = \alpha^6$. The erasure locator polynomial is

$$\Gamma(x) = (1 - \alpha^6 x)(1 - \alpha^7 x) = 1 + \alpha^{10}x + \alpha^{13}x^2.$$

Let

$$\begin{aligned} \tilde{r}(x) &= r(x)|_{\text{erasures removed}} \\ &= \alpha^5 x^{11} + \alpha^6 x^9 + \alpha^{11}x^5 + x^4 + \alpha^{11}x^3 + \alpha^6 x^2 + \alpha^{12}. \end{aligned}$$

The syndromes are

$$S_1 = \tilde{r}(\alpha) = 1 \qquad S_2 = \tilde{r}(\alpha^2) = 0 \qquad S_3 = \tilde{r}(\alpha^3) = \alpha^9$$
$$S_4 = \tilde{r}(\alpha^4) = \alpha^{12} \qquad S_5 = \tilde{r}(\alpha^5) = \alpha^2 \qquad S_6 = \tilde{r}(\alpha^6) = \alpha^8,$$

so $S(x) = 1 + \alpha^9 x^2 + \alpha^{12} x^3 + \alpha^2 x^4 + \alpha^8 x^5$. Let

$$\Xi(x) = \Gamma(x) S(x) \bmod x^{2t} = \alpha^{13} x^5 + \alpha^2 x^4 + \alpha^6 x^3 + \alpha^{10} x^2 + \alpha^{10} x + 1$$

By Berlekamp-Massey or Euclid, we find that

$$\Lambda(x) = 1 + \alpha^{11} x,$$

which has a root at $x = \alpha^4$, so there is an error at $i_1 = 11$ and $X_1 = \alpha^{11}$. We find

$$\Omega(x) = \Lambda(x)\Xi(x) \bmod x^{2t} = \alpha^7 x^7 + \alpha^{14} x + 1$$

and

$$\Phi(x) = \Lambda(x)\Gamma(x) = \alpha^9 x^3 + x^2 + \alpha^{14} x + 1.$$

The error value is

$$e_1 = \frac{\Omega(X_1^{-1})}{\Phi'(X_1^{-1})} = \alpha^5$$

and the erasure values are

$$f_1 = \frac{\Omega(Y_1^{-1})}{\Phi'(Y_1^{-1})} = \alpha$$

$$f_2 = \frac{\Omega(Y_2^{-1})}{\Phi'(Y_2^{-1})} = \alpha^{14}.$$

The decoded polynomial is

$$\hat{c}(x) = \alpha^6 x^9 + \alpha x^7 + \alpha^{14} x^6 + \alpha^{11} x^5 + x^4 + \alpha^{11} x^3 + \alpha^6 x^2 + \alpha^{12}.$$

□

6.8 Galois Field Fourier Transform Methods

Just as a discrete Fourier transform can be defined over real or complex numbers, so it is possible to define a Fourier transform over a sequence of Galois field numbers. This transform yields valuable insight into the structure of the code and new decoding algorithms.

Recall (see, e.g., [253]) that the discrete Fourier transform (DFT) of a real (or complex) vector $\mathbf{x} = (x_0, x_1, \ldots, x_{n-1})$ is the vector $\mathbf{X} = (X_0, X_1, \ldots, X_{n-1})$ with components

$$X_k = \sum_{j=0}^{n-1} x_j e^{-2\pi i jk/n},$$

(where $i = \sqrt{-1}$) and that the inverse DFT computes the elements of $\mathbf{x}$ from $\mathbf{X}$ by

$$x_j = \frac{1}{n} \sum_{k=0}^{n-1} X_k e^{2\pi i jk/n}.$$

The quantity $e^{-2\pi i/n}$ is a primitive nth root of unity, that is, a complex number with order n. In a similar way, we can define a discrete Fourier transform of length n over a finite field having an element of order n.

Definition 6.6 Let $\mathbf{v} = (v_0, v_1, \ldots, v_{n-1})$ be a vector over $GF(q)$ of length n such that $n \mid q^m - 1$ for some positive integer m. Let $\alpha \in GF(q^m)$ have order n. The **Galois Field Fourier Transform** (GFFT) of $\mathbf{v}$ is the vector $\mathbf{V} = (V_0, V_1, \ldots, V_{n-1})$ with components

$$V_j = \sum_{i=0}^{n-1} \alpha^{ij} v_i \qquad j = 0, 1, \ldots, n-1. \tag{6.38}$$

We write $\mathbf{V} = \mathcal{F}[\mathbf{v}]$ and $\mathbf{v} \leftrightarrow \mathbf{V}$ to denote the Fourier transform relationship between $\mathbf{v}$ and $\mathbf{V}$, where the type of Fourier transform (a GFFT) is obtained from the context. □

Theorem 6.17 *In a field $GF(q)$ with characteristic p, the inverse GFFT of the vector $\mathbf{V} = (V_0, V_1, \ldots, V_{n-1})$ is the vector $\mathbf{v}$ with components*

$$v_i = n^{-1} \sum_{j=0}^{n-1} \alpha^{-ij} V_j, \tag{6.39}$$

where n^{-1} is the multiplicative inverse of n modulo p.

Proof [373, p. 194] Note that α is a root of $x^n - 1$. We can write

$$x^n - 1 = (x-1)(x^{n-1} + x^{n-2} + \cdots + x + 1).$$

Evaluating $x^n - 1$ at $x = \alpha^r$ for some integer r we have

$$(\alpha^r)^n - 1 = (\alpha^n)^r - 1 = 0.$$

If $r \not\equiv 0 \pmod{n}$, then α^r must be a zero of $(x^{n-1} + x^{n-2} + \cdots + x + 1)$. We therefore have

$$\sum_{j=0}^{n-1} \alpha^{rj} = 0 \qquad r \not\equiv 0 \pmod{n}.$$

When $r \equiv 0 \pmod{n}$ we get

$$\sum_{j=0}^{n-1} \alpha^{rj} = \sum_{j=0}^{n-1} 1 \equiv n \pmod{p}.$$

Substituting (6.38) into (6.39),

$$\begin{aligned}\sum_{j=0}^{n-1} \alpha^{-ij} V_j &= \sum_{j=0}^{n-1} \alpha^{-ij} \sum_{k=0}^{n-1} \alpha^{kj} v_k \\ &= \sum_{k=0}^{n-1} v_k \sum_{j=0}^{n-1} \alpha^{(k-i)j} \\ &= v_i n \pmod{p}.\end{aligned}$$

Multiplying both sides by $n^{-1} \pmod{p}$ we obtain the desired result. □

Cyclic convolution of the sequences $\mathbf{a} = (a_0, a_1, \ldots, a_{n-1})$ and $\mathbf{b} = (b_0, b_1, \ldots, b_{n-1})$ is denoted by

$$\mathbf{c} = \mathbf{a} \circledast \mathbf{b},$$

where $\circledast$ denotes cyclic convolution. The elements of $\mathbf{c}$ in the convolution are given by

$$c_i = \sum_{k=0}^{n-1} a_k b_{((i-k))},$$

where $((i-k))$ is used as a shorthand for $(i-k) \pmod{n}$. That is, the indices in $i-k$ "wrap around" in a cyclic manner. One of the most important results from digital signal processing is the convolution theorem; as applied to the DFT it says that the DFT of sequence obtained by cyclic convolution of $\mathbf{a}$ and $\mathbf{b}$ is the element-by-element product of the DFTs of $\mathbf{a}$ and $\mathbf{b}$. An identical result holds for the GFFT.

Theorem 6.18 *If*

$$\mathbf{a} \leftrightarrow \mathbf{A}$$
$$\mathbf{b} \leftrightarrow \mathbf{B}$$
$$\mathbf{c} \leftrightarrow \mathbf{C}$$

are all sequences of length n in a finite field $GF(q)$ such that $n \mid q^m - 1$ for some m, then

$$C_j = A_j B_j \qquad j = 0, 1, \ldots, n-1$$

if and only if

$$\mathbf{c} = \mathbf{a} \circledast \mathbf{b}$$

(cyclic convolution) — that is,

$$c_i = \sum_{k=0}^{n-1} a_k b_{((i-k))}.$$

Furthermore,

$$c_j = a_j b_j \qquad j = 0, 1, \ldots, n-1$$

if and only if

$$\mathbf{C} = n^{-1} \mathbf{A} \circledast \mathbf{B};$$

that is,

$$C_i = n^{-1} \sum_{k=0}^{n-1} A_k B_{((i-k))}.$$

Proof [373, p. 195] We prove the first part of the theorem. We compute the inverse GFFT of $\mathbf{C}$:

$$\begin{aligned}
c_i &= n^{-1} \sum_{j=0}^{n-1} \alpha^{-ij} C_j = n^{-1} \sum_{j=0}^{n-1} \alpha^{-ij} A_j B_j \\
&= n^{-1} \sum_{j=0}^{n-1} \alpha^{-ij} \left(\sum_{k=0}^{n-1} \alpha^{kj} a_k \right) B_j = n^{-1} \sum_{k=0}^{n-1} a_k \sum_{j=0}^{n-1} \alpha^{-(i-k)j} B_j \\
&= \sum_{k=0}^{n-1} a_k b_{((i-k))}.
\end{aligned}$$

□

Let us now turn our attention from vectors to polynomials.

Definition 6.7 The **spectrum** of the polynomial (codevector) $v(x) = v_0 + v_1 x + \cdots + v_{n-1}x^{n-1}$ is the GFFT of $\mathbf{v} = (v_0, v_1, \ldots, v_{n-1})$. □

We refer to the original vector $\mathbf{v} = (v_0, v_1, \ldots, v_{n-1})$ as a vector in the "time domain" (even though time has nothing to do with it) and its corresponding transform $\mathbf{V}$ as being in the "frequency domain."

Given a polynomial $v(x)$, note that

$$v(\alpha^j) = v_0 + v_1\alpha^j + v_2\alpha^{2j} + \cdots + v_{n-1}\alpha^{(n-1)j} = \sum_{i=0}^{n-1} v_i\alpha^{ij} = V_j. \qquad (6.40)$$

Thus, the jth component of the GFFT of $\mathbf{v}$ is obtained by evaluating $v(x)$ at $x = \alpha^j$. Let us also define a polynomial based on $\mathbf{V} = (V_0, V_1, \ldots, V_{n-1})$ by

$$V(x) = V_0 + V_1 x + V_2 x^2 + \cdots + V_{n-1}x^{n-1}.$$

Then

$$V(\alpha^{-i}) = V_0 + V_1\alpha^{-i} + V_2\alpha^{-2i} + \cdots + V_{n-1}\alpha^{-(n-1)i} = \sum_{j=0}^{n-1} V_i\alpha^{-ij} = nv_i. \qquad (6.41)$$

Based on (6.40) and (6.41), we can immediately prove the following theorem.

Theorem 6.19 *α^j is a zero of $v(x)$ if and only if the jth frequency component of the spectrum of $v(x)$ equals zero.*

α^{-i} is a zero of $V(x)$ if and only if the ith time component v_i of the inverse transform $\mathbf{v}$ of $\mathbf{V}$ equals zero.

Recall the basic idea of a minimal polynomial: a polynomial $p(x)$ has its coefficients in the base field $GF(q)$ if and only if its roots are conjugates of each other. We have a similar result for the GFFT:

Theorem 6.20 *[373, p. 196] Let $\mathbf{V}$ be a vector of length n over $GF(q^m)$, where $n \mid q^m - 1$ and $GF(q^m)$ has characteristic p. The inverse transform $\mathbf{v}$ of $\mathbf{V}$ contains elements exclusively from the subfield $GF(q)$ if and only if*

$$V_j^q \pmod{p} \equiv V_{qj (mod\, n)}, \qquad j = 0, 1, \ldots, n-1.$$

Proof Recall that in $GF(p^t)$,

$$(a+b)^{p^r} = a^{p^r} + b^{p^r}.$$

Also recall that an element $\beta \in GF(q^m)$ is in the subfield $GF(q)$ if and only if $\beta^q = \beta$.

Let $v_i \in GF(q)$. Then

$$V_j^q = \left(\sum_{i=0}^{n-1} \alpha^{ij} v_i\right)^q = \sum_{i=0}^{n-1} \alpha^{qij} v_i^q = \sum_{i=0}^{n-1} \alpha^{i(qj)} v_i = V_{qj (\text{mod}\, n)}.$$

Conversely, assume $V_j^q = V_{qj \pmod n}$. From the definition of the GFFT,

$$V_j^q = \left(\sum_{i=0}^{n-1} \alpha^{ij} v_i\right)^q = \sum_{i=0}^{n-1} \alpha^{iqj} v_i^q$$

and

$$V_{qj \pmod n} = \sum_{i=0}^{n-1} \alpha^{iqj} v_i,$$

hence

$$\sum_{i=0}^{n-1} \alpha^{iqj} v_i^q = \sum_{i=0}^{n-1} \alpha^{iqj} v_i.$$

Let $k = qj \pmod n$. Since $n = q^m - 1$, q and n must be relatively prime, so that as j ranges from 0 to $n-1$, k takes on all values in the same range, so we conclude the $v_i = v_i^q$. □

Example 6.21 Let us illustrate the idea of the spectrum of a polynomial by considering the spectra of the minimal polynomials in $GF(8)$. The conjugacy classes and their minimal polynomials are shown here:

Conjugacy Class	Minimal Polynomial
$\{0\}$	$M_-(x) = x$
$\{\alpha^0\}$	$M_0(x) = x + 1$
$\{\alpha, \alpha^2, \alpha^4\}$	$M_1(x) = (x-\alpha)(x-\alpha^2)(x-\alpha^4) = x^3 + x + 1$
$\{\alpha^3, \alpha^5, \alpha^6\}$	$M_3(x) = (x-\alpha^3)(x-\alpha^6)(x-\alpha^5) = x^3 + x^2 + 1$

Now let us find the GFFT of the sequences obtained from the coefficients of the polynomials:

$$\begin{aligned}
M_-(x): &\quad \mathcal{F}(0100000) = (\alpha^j)_{j=0}^6 = (1, \alpha, \alpha^2, \alpha^3, \alpha^4, \alpha^5, \alpha^6)\\
M_0(x): &\quad \mathcal{F}(1100000) = (1 + 1\alpha^j)_{j=0}^6 = (0, \alpha^3, \alpha^6, \alpha, \alpha^5, \alpha^4, \alpha^2)\\
M_1(x): &\quad \mathcal{F}(1101000) = (1 + \alpha^j + \alpha^{3j})_{j=0}^6 = (1, 0, 0, \alpha^4, 0, \alpha^2, \alpha)\\
M_3(x): &\quad \mathcal{F}(1011000) = (1 + \alpha^{2j} + \alpha^{3j})_{j=0}^6 = (1, \alpha^4, \alpha, 0, \alpha^2, 0, 0).
\end{aligned}$$

Note that the positions of the zeros in the spectra correspond to the roots of the minimal polynomials. □

We can now state the BCH bound in terms of spectra:

Theorem 6.21 *[373, p. 197] Let $n \mid q^m - 1$ for some m. A q-ary n-tuple with weight $\leq \delta - 1$ that also has $\delta - 1$ consecutive zeros in its spectrum must be the all-zero vector. That is, the minimum weight of the code is $\geq \delta$.*

Proof Let $\mathbf{c}$ have weight ν, having exactly nonzero coordinates at $i_1, i_2, \ldots, i_\nu$. Define the locator polynomial $\Lambda(x)$ whose zeros correspond to the nonzero coordinates of $\mathbf{c}$:

$$\Lambda(x) = (1 - x\alpha^{-i_1})(1 - x\alpha^{-i_2})\cdots(1 - x\alpha^{-i_\nu}) = \Lambda_0 + \Lambda_1 x + \cdots + \Lambda_\nu x^\nu.$$

We regard this polynomial as a polynomial in the frequency domain. The *inverse* transform of $\Lambda(x)$ (i.e., its coefficient sequence) is a time domain vector $\boldsymbol{\lambda}$ that has zero coordinates

in the exact positions where $\mathbf{c}$ has nonzero coordinates. Also, at the positions where $c_i = 0$, the λ_i are not zero. Thus $c_i \lambda_i = 0$ for all i. By the convolution theorem we must therefore have $\mathbf{C} \circledast \mathbf{\Lambda} = 0$.

Assume $\mathbf{c}$ has weight $\leq \delta - 1$, while $\mathbf{C}$ has $\delta - 1$ consecutive zeros (possibly consecutive by "wrapping around" the end of the vector $\mathbf{C}$ in a cyclic manner). From the definition, $\Lambda_0 = 1$. Cyclic convolution in the frequency domain gives us

$$\sum_{k=0}^{n-1} \Lambda_k C_{((i-k))} = 0$$

so

$$C_i = -\sum_{k=1}^{\delta-1} \Lambda_k C_{((i-k))}.$$

Substituting the sequence of $\delta - 1$ zeros into C_i gives $C_i = 0$; proceeding forward from that index shows that all the C_is are 0, so that $\mathbf{C} = \mathbf{0}$. □

Based on our transform interpretation, we have the following definition (construction) for a Reed-Solomon code: A Reed-Solomon code can be obtained by selecting as codewords all vectors whose transforms have $\delta - 1 = 2t$ consecutive zeros. That is, a vector $\mathbf{c}$ is a codeword in a code with minimum distance $2t + 1$ if its transform $\mathbf{C} = \mathcal{F}[\mathbf{c}]$ has a consecutive sequence of $2t$ zeros (where the sequence of zeros starts from some fixed index in the transform vector).

This definition of the code can be used to establish another encoding mechanism for Reed-Solomon codes. Given a message sequence $\mathbf{m} = (m_0, m_1, \ldots, m_{k-1})$, form the vector

$$\mathbf{C} = \begin{bmatrix} \mathbf{0}_{2t} & \mathbf{m} \end{bmatrix}.$$

Then the corresponding codeword is

$$\mathbf{c} = \mathcal{F}^{-1}[\mathbf{C}].$$

However, this encoding is not systematic.

6.8.1 Equivalence of the Two Reed-Solomon Code Constructions

In Section 6.2, two seemingly inequivalent constructions were presented for Reed-Solomon codes. Based on Theorem 6.21, a Reed-Solomon codeword has a consecutive sequence of $2t = d_{\min} - 1$ zeros in its GFFT. We furthermore know that the minimum distance of a Reed-Solomon code is $d_{\min} = n - k + 1$. We now show that the codewords constructed according to Construction 1 (Section 6.2.1) have a consecutive sequence of $n - k$ zeros in their spectrum, as required.

Let $m(x) = m_0 + m_1 x + \cdots + m_{k-1} x^{k-1}$ and let the codeword constructed according to Construction 1 be

$$\mathbf{c} = (m(1), m(\alpha), \ldots, m(\alpha^{n-1})),$$

so that

$$c_i = m(\alpha^i) = \sum_{l=0}^{k-1} m_l \alpha^{il}, \qquad i = 0, 1, \ldots, n-1. \tag{6.42}$$

Now compute the GFFT of $\mathbf{c}$ as

$$C_{-j} = \sum_{i=0}^{n-1} c_i \alpha^{-ij},$$

where the index $-j$ is to be interpreted cyclically (which is legitimate, since $\alpha^n = 1$). Substituting from (6.42) into the transform,

$$C_{-j} = \sum_{i=0}^{n-1}\sum_{l=0}^{k-1} m_l \alpha^{-ij}\alpha^{il} = \sum_{l=0}^{k-1} m_l \left[\sum_{i=0}^{n-1} \alpha^{i(l-j)}\right].$$

The inner summation is 0 if $l \neq j \pmod{n}$. This is the case for $-j = k, k+1, \ldots, n-1$, which is $n-k$ consecutive values of j. Thus, there are $n-k$ consecutive zeros in the GFFT of every codeword.

6.8.2 Frequency-Domain Decoding

We present in this section one way of using the GFFT to decode a BCH or Reed-Solomon code. Let $\mathbf{r} = \mathbf{c} + \mathbf{e}$ be a received vector and let $\mathbf{R}$, $\mathbf{C}$, and $\mathbf{E}$ denote the corresponding transformed vectors. By the linearity of the transform we have

$$\mathbf{R} = \mathbf{C} + \mathbf{E},$$

where

$$E_j = \sum_{i=0}^{n-1} \alpha^{ij} e_i.$$

Assume the code is a narrow sense code. Then the first $2t$ coordinates of $\mathbf{C}$ are equal to zero, so that $\mathbf{R}_j = \mathbf{E}_j$ for $j = 0, 1, \ldots, 2t-1$. (These are the syndromes for the decoder.) Completion of the decoding requires finding the the remaining $n-2t$ coordinates of $\mathbf{E}$, after which we can find $\mathbf{e}$ by inverse GFFT.

Let $\Lambda(x) = \prod_{l=1}^{\nu}(1 - X_l x)$, treating the coefficients as a spectrum $\boldsymbol{\Lambda}$. The inverse transform $\boldsymbol{\lambda} = \mathcal{F}^{-1}[\boldsymbol{\Lambda}]$ yields a vector which has zeros at the coordinates corresponding to the zeros of $\Lambda(x)$, so $\boldsymbol{\lambda}$ has a zero wherever $\mathbf{e}$ is nonzero. Thus

$$\lambda_i e_i = 0, \qquad i = 0, 1, \ldots, n-1.$$

Translating the product back to the frequency domain, we have by the convolution formula $\Lambda \circledast \mathbf{E} = \mathbf{0}$, or

$$\sum_{k=0}^{n-1} \Lambda_k E_{((j-k))} = 0, \qquad j = 0, 1, \ldots, n-1.$$

Now assume that ν errors have occurred, so that the degree of $\Lambda(x)$ is ν. Then $\Lambda_k = 0$ for $k > \nu$. We obtain the familiar LFSR relationship

$$\sum_{k=0}^{\nu} \Lambda_k E_{((j-k))} = 0, \qquad j = 0, 1, \ldots, n-1$$

or, since $\Lambda_0 = 1$,

$$E_j = -\sum_{k=0}^{\nu} \Lambda_k E_{j-k}. \tag{6.43}$$

This expresses an LFSR relationship between the transformed errors E_j and the coefficients of the error locator polynomial. Given the $2t$ known values of the transformed errors $\{E_0, E_1, \ldots, E_{2t-1}\}$, the error locator polynomial can be found using any of the methods described previously (such as the Berlekamp-Massey algorithm or the Euclidean algorithm). Knowing the Λ_i coefficients, the remainder of the E_j values can be found using (6.43). Then knowing $\mathbf{E}$, the error vector in the "time" domain can be found by an inverse Fourier transform: $\mathbf{e} = \mathcal{F}^{-1}[\mathbf{E}]$. Note that unless a fast inverse Fourier transform is available, this is essentially the same as a Chien search.

6.9 Variations and Extensions of Reed-Solomon Codes

In this section we briefly describe several variations on Reed-Solomon and BCH codes. More detail can be found in [220].

6.9.1 Simple Modifications

Several simple modifications are possible, of the sort described in Section 3.9.

An (n, k) Reed-Solomon code can be *punctured* by deleting any of its symbols, resulting in a $(n-1, k)$ code.

An (n, k) Reed-Solomon code C can be *extended* by adding additional parity check symbols. A code is *singly* extended by adding a single parity symbol. Interestingly enough, a single-extended Reed-Solomon code is still MDS. To see this, let $\mathbf{c} = (c_0, c_1, \ldots, c_{q-2})$ be a codewords from a $(q-1, k)$ q-ary narrow-sense t-error correcting code and let

$$c_{q-1} = -\sum_{j=0}^{q-2} c_j$$

be an overall parity check digit. Then an extended codeword is $(c_0, c_1, \ldots, c_{q-1})$. To see that this extended code is still MDS, we must show that the distance has, in fact, increased. To this end, suppose that $\mathbf{c}$ has, in fact, minimum weight $d_{\min}$ in C. Let $c(x)$ be corresponding code polynomial. The generator for the code is

$$g(x) = (x-\alpha)(x-\alpha^2)\cdots(x-\alpha^{2t}).$$

Now

$$c(1) = \sum_{i=0}^{q-2} c_i.$$

If $c(1) \neq 0$, then $c_{q-1} \neq 0$, so the new codeword in the extended code has minimum distance $d_{\min}+1$. If $c(1) = 0$, then $c(x)$ must be of the form $c(x) = u(x)(x-1)g(x)$ for some polynomial $u(x)$. That is, $c(x)$ is a code polynomial in the code having generator

$$g'(x) = (x-1)(x-\alpha)(x-\alpha^2)\cdots(x-\alpha^{2t}).$$

By the BCH bound, this code must have minimum distance $d_{\min}+1$. Since the new code is $(n+1, k)$ with minimum distance $d_{\min}+1$, it is MDS.

It is also possible to form a double-extended Reed-Solomon code which is MDS [220]. However, these extended codes are not, in general, cyclic.

6.9.2 Generalized Reed-Solomon Codes and Alternant Codes

Recall that according to Construction 1 of Reed-Solomon codes, codewords are obtained by

$$\mathbf{c} = (m(1), m(\alpha), \ldots, m(\alpha^{n-1})). \tag{6.44}$$

Now choose a vector $\mathbf{v} = (v_1, v_2, \ldots, v_n)$ whose elements are all nonzero. Then a generalization of (6.44) is

$$\mathbf{c} = (v_1 m(1), v_2 m(\alpha), \ldots, v_n m(\alpha^{n-1})).$$

Somewhat more generally, we have the following.

Definition 6.8 Let $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \ldots, \alpha_n)$ be n distinct elements of $GF(q^m)$ and let $\mathbf{v} = (v_1, v_2, \ldots, v_n)$ have nonzero (but not necessarily distinct) elements from $GF(q^m)$. Then the **generalized RS code**, denoted by $GRS_k(\boldsymbol{\alpha}, \mathbf{v})$, consists of all vectors

$$(v_1 m(\alpha_1), v_2 m(\alpha_2), \ldots, v_n m(\alpha_n))$$

as $m(x)$ ranges over all polynomials of degree $< k$. □

The $GRS_k(\boldsymbol{\alpha}, \mathbf{v})$ code is an (n, k) code and can be shown (using the same argument as for Construction 1) to be MDS.

The parity check matrix for the $GRS_k(\boldsymbol{\alpha}, \mathbf{v})$ code can be written as

$$H = \begin{bmatrix} y_1 & y_2 & \cdots & y_n \\ \alpha_1 y_1 & \alpha_2 y_2 & \cdots & \alpha_n y_n \\ \vdots & & & \\ \alpha_1^{r-1} y_1 & \alpha_2^{r-1} y_2 & \cdots & \alpha_n^{r-1} y_n \end{bmatrix} = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ \alpha_1 & \alpha_2 & \cdots & \alpha_n \\ \alpha_1^2 & \alpha_2^2 & \cdots & \alpha_n^2 \\ \vdots & & & \\ \alpha_1^{r-1} & \alpha_2^{r-1} & \cdots & \alpha_n^{r-1} \end{bmatrix} \begin{bmatrix} y_1 & & & \\ & y_2 & & \\ & & \ddots & \\ & & & y_n \end{bmatrix} \triangleq XY. \tag{6.45}$$

Here, $\mathbf{y} = (y_1, y_2, \ldots, y_n)$ with $y_i \in GF(q^m)$ and $y_i \neq 0$, is such that $GRS_k(\boldsymbol{\alpha}, \mathbf{v})^{\perp} = GRS_{n-k}(\boldsymbol{\alpha}, \mathbf{y})$.

If $H = XY$ is a parity check matrix, then for an invertible matrix C, $\tilde{H} = CXY$ is an equivalent parity check matrix.

While the elements of codewords of a $GRS_k(\boldsymbol{\alpha}, \mathbf{v})$ code are in general in $GF(q^m)$, it is possible to form a code from codewords whose elements lie in the base field $GF(q)$.

Definition 6.9 An **alternant** code $\mathcal{A}(\boldsymbol{\alpha}, \mathbf{y})$ consists of all codewords of $GRS_k(\boldsymbol{\alpha}, \mathbf{v})$ whose components all lie $GF(q)$. (We say that $\mathcal{A}(\boldsymbol{\alpha}, \mathbf{y})$ is the **restriction** of $GRS_k(\boldsymbol{\alpha}, \mathbf{v})$ to $GF(q)$.) That is, $\mathcal{A}(\boldsymbol{\alpha}, \mathbf{y})$ is the set of all vectors $\mathbf{c} \in GF(q)^n$ such that $H\mathbf{c} = \mathbf{0}$, for H in (6.45). Another way of saying this is that $\mathcal{A}$ is the **subfield subcode** of $GRS_k(\boldsymbol{\alpha}, \mathbf{v})$. □

Since we have an expression for a parity check matrix for the GRS code, it is of interest to find a parity check matrix for the alternant code. That is, we want to find a parity check matrix $\tilde{H}$ over $GF(q)$ corresponding to the parity check matrix H over $GF(q^m)$. This can be done as follows. Pick a basis $\alpha_1, \alpha_2, \ldots, \alpha_m$ for $GF(q^m)$ over $GF(q)$. (Recall that $GF(q^m)$ can be written as a vector space of elements of $GF(q)$.) A convenient basis set is

$\{1, \alpha, \alpha^2, \ldots, \alpha^{m-1}\}$, but any linearly independent set can do. Then, for each element H_{ij} of H, write

$$H_{ij} = \sum_{l=1}^{m} h_{ijl}\alpha_l,$$

where each $h_{ijl} \in GF(q)$. Now define $\tilde{H}$ to be the $(n-k)m \times n$ matrix obtained from H by replacing each entry H_{ij} by the column vector of components in $(h_{ij1}, h_{ij2}, \ldots, h_{ijm})$, so that

$$\tilde{H} = \begin{bmatrix} h_{111} & h_{121} & & h_{1n1} \\ h_{112} & h_{122} & \cdots & h_{1n2} \\ \vdots & \vdots & & \vdots \\ h_{11m} & h_{12m} & & h_{1nm} \\ \vdots & & & \\ h_{r11} & h_{r21} & & h_{rn1} \\ h_{r12} & h_{r22} & \cdots & h_{rn2} \\ \vdots & \vdots & & \vdots \\ h_{r1m} & h_{r2m} & & h_{rnm} \end{bmatrix}.$$

It can be argued that the dimension of the code must satisfy $k \geq n - mr$.

One of the important properties about alternant codes is the following:

Theorem 6.22 *[220, p. 334] $\mathcal{A}(\alpha, \mathbf{y})$ has minimum distance $d_{\min} \geq n - k + 1$.*

Proof Suppose $\mathbf{c}$ is a codeword having weight $\leq r = n - k$. Then $H\mathbf{c} = XY\mathbf{c} = \mathbf{0}$. Let $\mathbf{b} = Y\mathbf{c}$. Since Y is diagonal and invertible, $\text{wt}(\mathbf{b}) = \text{wt}(\mathbf{c})$. Then $X\mathbf{b} = \mathbf{0}$. However, X is a full-rank Vandermonde matrix, so this is impossible. □

In summary, we have a code of length n, dimension $k \geq n - mr$ and minimum distance $d_{\min} \geq n - r$.

The family of alternant codes encompasses a variety of interesting codes, depending on how the field and subfield are chosen. BCH and Reed-Solomon codes are alternant codes. So are Goppa codes, which are described next.

6.9.3 Goppa Codes

Goppa codes start off with a seemingly different definition but are, in fact, instances of alternant codes.

Definition 6.10 Let $L = \{\alpha_1, \alpha_2, \ldots, \alpha_n\}$ where each $\alpha_i \in GF(q^m)$. Let $G(x) \in GF(q^m)[x]$ be the **Goppa polynomial**, where each $\alpha_i \in L$ is *not* a root of G. That is, $G(\alpha_i) \neq 0$ for all $\alpha_i \in L$. For any vector $\mathbf{a} = (a_1, a_2, \ldots, a_n)$ with elements in $GF(q)$, associate the rational function

$$R_{\mathbf{a}}(x) = \sum_{i=1}^{n} \frac{a_i}{x - \alpha_i}.$$

Then the **Goppa code** $\Gamma(L, G)$ consists of all vectors $\mathbf{a} \in GF(q)^n$ such that

$$R_{\mathbf{a}}(x) \equiv 0 \pmod{G(x)}. \tag{6.46}$$

If $G(x)$ is irreducible, then $\Gamma(L, G)$ is an irreducible Goppa code. □

As we will see, Goppa codes have good distance properties: $d_{\min} \geq \deg(G) + 1$.

Goppa codes are linear codes. The parity check matrix can be found using (6.46), which can be re-expressed as: $R_{\mathbf{a}}(x) = 0$ in the ring $GF(q^m)[x]/G(x)$. Note that in this ring, $x - \alpha_i$ does have an inverse, since it does not divide $G(x)$. The inverse is

$$(x - \alpha_i)^{-1} = -\frac{G(x) - G(\alpha_i)}{x - \alpha_i} G(\alpha_i)^{-1}, \tag{6.47}$$

as can be shown by observing that

$$-(x - \alpha_i)\frac{G(x) - G(\alpha_i)}{x - \alpha_i} G(\alpha_i)^{-1} \equiv 1 \pmod{G(x)}$$

by applying the definition of $\equiv$. Let $G(x) = \sum_{i=0}^{r} g_i x^i$ with $g_r \neq 0$. It can be verified by long division and collection of terms that

$$\begin{aligned}\frac{G(x) - G(\alpha_i)}{x - \alpha_i} &= g_r(x^{r-1} + x^{r-2}\alpha_i + \cdots + \alpha_i^{r-1}) + g_{r-1}(x^{r-2} + \cdots + \alpha_i^{r-2}) + \cdots \\ &\quad + g_2(x + \alpha_i) + g_1.\end{aligned} \tag{6.48}$$

Substituting (6.47) into (6.46), we have that $\mathbf{a}$ is in $\Gamma(L, G)$ if and only if

$$\sum_{i=1}^{n} a_i \frac{G(x) - G(\alpha_i)}{x - \alpha_i} G(\alpha_i)^{-1} = 0 \tag{6.49}$$

as a polynomial (and not just modulo $G(x)$). Since the polynomial must be 0, the coefficients of each x^i must each be zero individually. Substituting (6.48) into (6.49) and equating each of the coefficients of $x^{r-1}, x^{r-2}, \ldots, 1$ to 0, we see that $\mathbf{a}$ is in $\Gamma(l, G)$ if and only if $H\mathbf{a} = \mathbf{0}$, where

$$H = \begin{bmatrix} g_r G(\alpha_1)^{-1} & \cdots & g_1 G(\alpha_n)^{-1} \\ (g_{r-1} + \alpha_1 g_r) G(\alpha_1)^{-1} & \cdots & (g_{r-1} + \alpha_n g_r) G(\alpha_n)^{-1} \\ \vdots & & \\ (g_1 + \alpha_1 g_2 + \cdots + \alpha_1^{r-1} g_r) G(\alpha_1)^{-1} & \cdots & (g_1 + \alpha_n g_2 + \cdots + \alpha_n^{r-1} g_r) G(\alpha_n)^{-1} \end{bmatrix}.$$

Note that the matrix H can be written as

$$\begin{aligned} H = &\begin{bmatrix} g_r & 0 & 0 & \cdots & 0 \\ g_{r-1} & g_r & 0 & \cdots & 0 \\ g_{r-2} & g_{r-1} & g_r & \cdots & 0 \\ \vdots & & & & \\ g_1 & g_2 & g_3 & \cdots & g_r \end{bmatrix} \begin{bmatrix} 1 & 1 & \cdots & 1 \\ \alpha_1 & \alpha_2 & \cdots & \alpha_n \\ \alpha_1^2 & \alpha_2^2 & \cdots & \alpha_n^2 \\ \vdots & & & \\ \alpha_1^{r-1} & \alpha_2^{r-1} & \cdots & \alpha_n^{r-1} \end{bmatrix} \\ &\times \begin{bmatrix} G(\alpha_1)^{-1} & & & \\ & G(\alpha_2)^{-1} & & \\ & & \ddots & \\ & & & G(\alpha_n)^{-1} \end{bmatrix} \end{aligned}$$

or $H = CXY$. Since C is lower triangular with a nonzero along the diagonal, C is invertible. It follows that an equivalent parity check matrix is

$$\tilde{H} = XY = \begin{bmatrix} G(\alpha_1)^{-1} & \cdots & G(\alpha_n)^{-1} \\ \alpha_1 G(\alpha_1)^{-1} & \cdots & \alpha_n G(\alpha_n)^{-1} \\ \vdots & & \\ \alpha_1^{r-1} G(\alpha_1)^{-1} & \cdots & \alpha_n^{r-1} G(\alpha_n)^{-1} \end{bmatrix}.$$

We observe from the structure of the parity check matrix (compare with (6.45)) that the Goppa code is an alternant code, with $\mathbf{y} = (G(\alpha_1)^{-1}, \ldots, G(\alpha_n)^{-1})$. In fact, it can be shown that the $\Gamma(L, G)$ code can be obtained as the restriction to $GF(q)$ of the $GRS_{n-r}(\boldsymbol{\alpha}, \mathbf{v})$ code, where

$$v_i = \frac{G(\alpha_i)}{\prod_{j \neq i}(\alpha_i - \alpha_j)}.$$

6.9.4 Decoding Alternant Codes

Efficient algorithms exist for decoding alternant codes [220, Section 12.9]. These exactly parallel the steps used for decoding Reed-Solomon codes: (1) A syndrome is computed (the details are somewhat different than for RS codes); (2) An error locator polynomial is found, say, using the Berlekamp-Massey or the Euclidean algorithm; (3) The roots are found; and (4) Error values are computed if necessary. A decoding algorithm for Goppa codes also appears in [256].

6.9.5 Cryptographic Connections: The McEliece Public Key Cryptosystem

In this section we present another connection between error correction coding and cryptography. In this case, we show how an error correction code can be used to make a public key encryption system. The original system was based on Goppa codes (hence its inclusion in this context), but other codes might also be used.

The person A wishing to communicate picks an irreducible polynomial $G(x)$ of degree t over $GF(2^m)$ "at random" and constructs the generator matrix G for the (n, k) Goppa code using $G(x)$. This code is capable of correcting any pattern of up to t errors. Note that there are efficient decoding algorithms for this code.

Now A scrambles the generator G by selecting a random dense invertible $k \times k$ matrix S and a random $n \times n$ permutation matrix P. He computes $\tilde{G} = SGP$. A message $\mathbf{m}$ would be encoded using this generator as

$$\tilde{\mathbf{c}} = (\mathbf{m}S)GP = (\tilde{\mathbf{m}}G)P.$$

Since P simply reorders the elements of the codeword corresponding to the message $\tilde{\mathbf{m}}$, the code with generator $\tilde{G}$ has the same minimum distance as the code with generator G. The **public key** for this system is the scrambled generator $\tilde{G}$. The **private key** is the set (S, G, P).

Encryption of a message $\mathbf{m}$ is accomplished using the public key by computing

$$\mathbf{e} = \mathbf{m}\tilde{G} + \mathbf{z},$$

where $\mathbf{z}$ is a random "noise" vector of length n and weight t. $\mathbf{e}$ is transmitted as the encrypted information.

Because the encoding is not systematic and there is noise added, the message is not explicitly evident in **e**. The encrypted message **m** could be discovered if **e** could be decoded (in the error correction coding sense). However, the scrambled matrix $\tilde{G}$ no longer has the algebraic structure that provides an efficient decoding algorithm. Optimal decoding without some structure to exploit can have NP-complete complexity [24]. Hence, a recipient of **e** can recover **m** only with extreme effort.

Decryption of **e** knowing (S, G, P), however, is straightforward. Knowing P, first compute $\tilde{\mathbf{e}} = \mathbf{e}P^{-1}$. Note that while the noise is permuted, no additional noise terms are added. Now decode using a fast Goppa decoder, effectively getting rid of the noise **z**, to obtain the scrambled message $\tilde{\mathbf{m}} = \mathbf{m}S$. Finally, invert to obtain $\mathbf{m} = \tilde{\mathbf{m}}S^{-1}$.

Programming Laboratory 6: Programming the Berlekamp-Massey Algorithm

Background

Reading: Sections 6.4.2, 6.4.3, 6.4.4.

The Berlekamp-Massey algorithm provides one of the key steps in the decoding of BCH or Reed-Solomon codes. Specifically, it provides a means to determine the error-locating polynomial given the syndromes.

We have encountered LFSRs in previous labs: in binary form in Lab 2 and in the context of the Sugiyama algorithm in Lab 4. The problem addressed by the Berlekamp-Massey algorithm is to find the coefficients $\{c_1, c_2, \ldots, c_\nu\}$ satisfying (6.12) with the *smallest* ν. (The Sugiyama algorithm introduced in Lab 4 provides another solution to this same problem.) The LFSR coefficients are represented in a polynomial, the *connection polynomial*

$$c(x) = 1 + c_1x + c_2x^2 + \cdots + c_\nu x^\nu.$$

The Berlekamp-Massey algorithm is described in Algorithm 6.1 on page 258. Simplifications for binary BCH codes are presented in Algorithm 6.2 on page 259.

Assignment

Preliminary Exercises

1) For operations in $\mathbb{Z}_5$, work through the Berlekamp-Massey algorithm for the sequence $\{2, 3, 4, 2, 2, 3\}$. Verify that the sequence of connection polynomials is

Initial:	$c = 1$	$L = 0$
$k = 1$	$c = 1 + 3x$	$L = 1$
$k = 2$	$c = 1 + x$	$L = 1$
$k = 3$	$c = 1 + x + 4x^2$	$L = 2$
$k = 4$	$c = 1 + 2x$	$L = 2$
$k = 5$	$c = 1 + 2x + 2x^2 + 2x^3$	$L = 3$
$k = 6$	$c = 1 + 3x + 4x^2 + 2x^3$	$L = 3$

2) For operations in $GF(2)$, work through the Berlekamp-Massey algorithm for the sequence $\{1, 1, 1, 0, 1, 0, 0\}$. Verify that the sequence of connection polynomials is

Initial:	$c = 1$	$L = 0$
$k = 1$	$c = 1 + x$	$L = 1$
$k = 2$	$c = 1 + x$	$L = 1$
$k = 3$	$c = 1 + x$	$L = 1$
$k = 4$	$c = 1 + x + x^3$	$L = 3$
$k = 5$	$c = 1 + x + x^3$	$L = 3$
$k = 6$	$c = 1 + x + x^3$	$L = 3$
$k = 7$	$c = 1 + x + x^3$	$L = 3$

3) For operations in $GF(2^4)$, work through the Berlekamp-Massey algorithm for the sequence $\{0, \alpha^3, \alpha^4, \alpha^7\}$. Verify that the sequence of connection polynomials is

Initial:	$c = 1$	$L = 0$
$k = 1$	$c = 1$	$L = 0$
$k = 2$	$c = 1 + \alpha^3x^2$	$L = 2$
$k = 3$	$c = 1 + \alpha x + \alpha^3x^2$	$L = 2$
$k = 4$	$c = 1 + \alpha x + \alpha^{10}x^2$	$L = 2$

Programming Part

1) Write a function `berlmass` which: **Either**

- Accepts a sequence of numbers of arbitrary type and returns a connection polynomial for an LFSR generating that sequence. The function should have the following declaration:

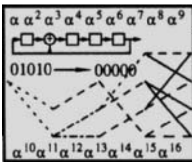
```
template <class T> polynomialT<T>
berlmass(const T* s, int n);
// Accept a sequence s of type T and length n
// (s[0] ... s[n-1])
// and return a connection polynomial c of
// shortest length generating that sequence.
```

- **Or,** accepts an array of numbers of arbitrary type and an argument into which the coefficients of the connection polynomial are written. The function should have the following declaration:

```
template <class T> void
berlmass2(const T* s, int n, T* c, int& L)
// s = input coefficients s[0],s[1],... s[n-1]
// c = connection polynomial coefficients.
// (Must be allocated prior to calling)
// L = degree of connection polynomial
```

You may want to create a class BCH which encapsulates one of these functions. (See the discussion below.) The difference between these is that the first form deals explicitly with the polynomial, while the second deals only with the coefficients. Tradeoffs between these are discussed below.

2) Test your function using the examples from the preliminary exercises.



Algorithm 6.3 Test BM Algorithm
File: testBM.cc

3) Over $GF(16)$ verify that the sequence $\{\alpha^8, \alpha, \alpha^{13}, \alpha^2, \alpha^5, \alpha^{11}\}$ is generated by the LFSR with connection polynomial $1 + \alpha^8 x + \alpha^2 x^2$.

4) Over $GF(16)$ verify that the sequence $\{0, 0, \alpha^5, 0, 1, \alpha^{10}\}$ is produced by $1 + \alpha^{10}x^2 + \alpha^5 x^3$.

5) Write a function berlmassBCH which accepts data satisfying the syndrome conjugacy condition (6.27) and computes the connection polynomial using the reduced complexity algorithm. Test your algorithm by verifying that:

a) For the sequence $\{1, 1, \alpha^{10}, 1, \alpha^{10}, \alpha^5\}$ the connection polynomial is $1 + x + \alpha^5 x^3$, with computations over $GF(2^4)$.

b) For the sequence $\{\alpha^{14}, \alpha^{13}, 1, \alpha^{11}, \alpha^5, 1\}$ the connection polynomial is $1 + \alpha^{14}x + \alpha^{11}x^2 + \alpha^{14}x^3$, with computations over $GF(2^4)$.

Resources and Implementation Suggestions

Two implementations are suggested for the algorithms, one which employs polynomials and the other which employs arrays. The polynomial implementation is somewhat easier to write than the array implementation, since the single statement

```
c = c - (p << shift)*(d/dm);
```

suffices to provide the update in (6.15). The algorithm outlined in Algorithm 6.1 can thus be almost literally translated into C++.

There are a couple of precautions, however. First, in the update, there may be cancellations in the higher order terms of the polynomial, so the actual degree of the polynomial $c(x)$ is less than the expected degree L. However, L should not be modified. The impact of this is that when the discrepancy is computed, the actual degree of $c(x)$ should be used, not L. The discrepancy can be computed as in the following piece of code:

```
// compute the discrepancy
// in the polynomial implementation
d = s[k];
for(j=1; j <= c.getdegree(); j++) {   // sum
   d += c[j]*s[k-j];
}
```

The other thing to be aware of is that there is internal memory allocation and deallocation that takes place whenever a polynomialT is assigned to a polynomialT of different degree. This introduces an operational overhead to the algorithm.

Which brings up the array form: By implementing the operations explicitly using arrays, the overhead of memory management can be (almost) eliminated and only a little more work is necessary. For example, the update formula (6.15) can be represented using a simple loop:

```
// update the polynomial
for(j = shift; j <= L; j++) {
  // Compute: c = c - (p<< shift)*(d/dm);
   c[j] -= p[j-shift]*ddm;
}
```

The function must have passed to it an array of sufficient size to hold the polynomial. Also, it must allocate arrays of sufficient size to represent the largest possible t and p polynomials, then de-allocate them on exit. However, there is a caution associated with this implementation: if the function is called repeatedly, the prior data in c does mess up the computations. Additional care is necessary to ensure that terms that should be 0 actually are — the simple loop above does not suffice for this purpose.

This brings up the final implementation suggestion: the function would be most cleanly represented using a class with internal storage allocation. Then the necessary internal space could be allocated once upon instantiation of an object then used repeatedly. You may therefore want to write it this way (and use arrays internally).

Programming Laboratory 7: Programming the BCH Decoder

Objective

In this lab you implement a BCH decoder and thoroughly test it.

Preliminary Exercises

Reading: Sections 6.3, 6.4.

1) For the field $GF(2^4)$ generated by $1 + x + x^4$, show that the minimal polynomials of α, α^3, and α^5 are

$$M_1(x) = 1 + x + x^4$$

$$M_3(x) = 1 + x + x^2 + x^3 + x^4$$

$$M_5(x) = 1 + x + x^2.$$

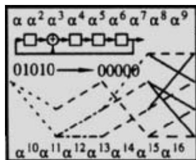
2) Show that the generator for a (15,5) three-error correcting binary BCH code is

$$g(x) = 1 + x + x^2 + x^4 + x^5 + x^8 + x^{10}.$$

3) Determine the actual minimum weight of this code.

Programming Part

1) Write a class `ChienSearch` which implements the Chien search algorithm over $GF(2^m)$. In the interest of speed, the class constructor should allocate space necessary for the registers as well as space for the computed roots. A member function should accept an error locator polynomial (or an array of its coefficients) and compute all the roots of the polynomial.



Algorithm 6.4 Chien Search
File: `ChienSearch.h`
`ChienSearch.cc`
`testChien.cc`

Test your algorithm by verifying that over $GF(2^4)$ the roots of the error locator polynomial

$$\Lambda(x) = 1 + x + \alpha^5 x^3$$

are at α^3, α^{10} and α^{12}.

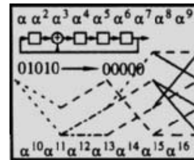
2) Build a BCH decoder class which decodes a vector of n binary elements.

3) Thoroughly test your decoder on the (15,5) BCH code with generator

$$g(x) = 1 + x + x^2 + x^4 + x^5 + x^8 + x^{10}$$

over the field $GF(2^4)$. You should correct all patterns of up to three errors. The test program in Algorithm 6.5 provides an exhaustive test of *all* patterns of three errors (with some duplication).

Your finished decoder should finish without any uncorrected errors. It is important to make sure that you are able to successfully decode all error patterns. This test is likely to shake out several minor problems with the functions you have written. After all errors are corrected, you can be quite confident in the functions you have written.



Algorithm 6.5 BCH Decoder
File: `BCHdec.h`
`BCHdec.cc`
`testBCH.cc`

4) Modify the BCH decoder to use the reduced-complexity Berlekamp-Massey algorithm for BCH syndromes (the one described in Section 6.4.4) to find the error locator polynomial from the syndromes. Ensure that your decoder is still able to decode all patterns of up to three errors.

5) Modify the BCH decoder to use the Sugiyama algorithm from lab 4 to find the error locator polynomial from the syndromes. Ensure that your decoder is still able to decode all patterns of up to three errors.

Resources and Implementation Suggestions

1) The syndromes are numbered $s_1, s_2, \ldots, s_{2t}$, whereas the discussion surrounding the Berlekamp-Massey algorithm used zero-based indexing, $y_0, y_1, \ldots, y_{2t-1}$. There is no difficulty here: simply interpret $y_j = s_{j+1}$ and call the Berlekamp-Massey algorithm with an array such that the first element contains the first syndrome.

2) If you represent the received vector as a polynomial `polynomialT<GFNUM2m>`, then evaluating it to compute the syndromes is very straightforward using the `( )` operator in class `polynomialT`.

If you choose to represent it as an array (to avoid memory management overhead), for efficiency the polynomial evaluation should be done using *Horner's rule*. As an example, to evaluate a cubic polynomial, you would not want to use

```
p = c[0] + c[1]*x + c[2]*x*x + c[3]*x*x*x;
```

since multiplications are wasted in repeatedly computing products of x. Instead, it is better to write in nested form as

```
p = ((c[3]*x+c[2])*x+c[1])*x+c[0];
```

This nesting can be efficiently coded as

```
p = c[j=n];
while(j>0)
    p = p*x + c[--j];
```

3) If you use an array implementation of the Berlekamp-Massey algorithm (in contrast to a `polynomialT` implementation) you may need to take special care that the coefficients of the connection polynomial are zeroed out properly.

Follow-On Ideas and Problems

The BCH codes described in this lab are narrow sense, in that the roots of the polynomial $g(x)$ contain the list $\beta, \beta^2, \ldots, \beta^{2t}$ for a primitive element β. A non-narrow sense BCH code uses the roots

$$\beta^b, \beta^{b+1}, \ldots, \beta^{b+2t-1}$$

for an arbitrary b. Describe how the decoding algorithm is modified for non-narrow sense BCH codes.

Programming Laboratory 8: Reed-Solomon Encoding and Decoding

Objective

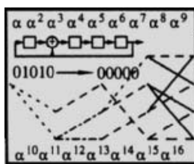
In this lab you are to extend the binary BCH decoder implemented in lab 7 to nonbinary Reed-Solomon codes. In addition, instead of simply decoding random errors, you will create a systematic encoder.

Background

Reading: Sections 6.3, 6.4, 6.5, 6.6.

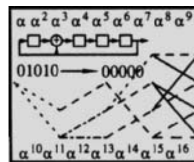
Programming Part

1) Create a class `RSenc` which implements a Reed-Solomon encoder for primitive, narrow-sense codes. Verify that the function works by encoding the data as in Example 6.10. Your class declaration might be as in `RSenc.h`.



Algorithm 6.6 Reed-Solomon Encoder Declaration
File: `RSenc.h`
`RSenc.cc`

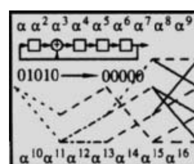
2) Create a class `RSdec` which implements a Reed-Solomon decoder for primitive, narrow-sense codes. Use the Berlekamp-Massey algorithm to find $\Lambda(x)$, followed by the Chien search to find the error locators and the Forney algorithm to find the error values. You should be able to use much of the code that you have written previously (the Berlekamp-Massey algorithm, the Chien search from the BCH lab) as well as create new code for the Forney algorithm. A declaration for the class might be as in `RSdec.h`.



Algorithm 6.7 Reed-Solomon Decoder Declaration
File: `RSdec.h`
`RSdec.cc`

After creating an `RSdec` object, a call to the `decode` member function converts the array `r` to the decoded array `dec`.

3) Test your decoder by decoding 10000 patterns of up to three errors for the (255,249) code over the field $GF(2^8)$ using $p(x) = x^8 + x^4 + x^3 + x^2 + 1$. A program which does this testing is `testRS.cc`.

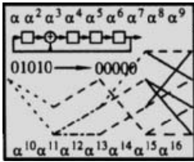


Algorithm 6.8 Reed-Solomon Decoder Testing
File: `testRS.cc`

4) After you have tested and debugged your decoder, replace the Berlekamp-Massey algorithm with the Euclidean algorithm to determine $\Lambda(x)$ and $\Omega(x)$. Test the resulting

algorithm as before, ensuring that many random patterns of up to three errors are decoded.

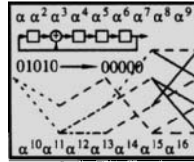
5) After you have tested and debugged your encoder and decoder objects, you are ready to use them to protect data files on the computer. `rsencode` is program to encode data using a $(255, 255-2t)$ Reed-Solomon code, where t defaults to 3, but can be set using a command-line argument. The corresponding decoder program is `rsdecode`.



Algorithm 6.9 Reed-Solomon File Encoder and Decoder
File: `rsencode.cc` `rsdecode.cc`

In the program, special care is taken to handle the last block of data, writing out the length of the block if it is less than 255.

Starting with this source code, use your encoder and decoder objects to build complete encoder and decoder programs, `rsencode` and `rsdecode`. Test your encoders and decoders on short files (< 249 bytes) and longer files. Test the program when the encoded data is corrupted (say, using the `bsc` program).



Algorithm 6.10 Binary Symmetric Channel Simulator
File: `bsc.c`

Appendix 6.A Proof of Newton's Identities

Newton's identities relate the coefficients of a polynomial to the power sum identities obtained from the roots of the polynomial. We derive them here in the general case, then make application to the error locator polynomial.

Let

$$\begin{aligned} f(x) &= (x-x_1)(x-x_2)\cdots(x-x_n) \\ &= x^n - \sigma_1 x^{n-1} + \sigma_2 x^{n-2} + \cdots + (-1)^{n-1}\sigma_{n-1}x + (-1)^n\sigma_n. \end{aligned}$$

The power sums are $s_k = x_1^k + x_2^k + \cdots + x_n^k, \quad k = 1, 2, \ldots, n$, and the elementary symmetric functions are

$$\begin{aligned} \sigma_1 &= x_1 + \cdots x_n \\ \sigma_2 &= x_1x_2 + x_1x_3 + \cdots x_1x_n + \cdots x_{n-1}x_n \\ &\vdots \\ \sigma_n &= x_1x_2\cdots x_n. \end{aligned}$$

Theorem 6.23 (Newton's Identities) *The elementary symmetric functions σ_k and the power sum symmetric functions s_k are related by*

$$\begin{aligned} s_k - \sigma_1 s_{k-1} + \cdots + (-1)^{k-1}\sigma_{k-1}s_1 + (-1)^k k\sigma_k = 0 \quad & 1 \le k \le n \\ s_k - \sigma_1 s_{k-1} + \cdots + (-1)^{n-1}\sigma_{n-1}s_{k-n+1} + (-1)^n s_{k-n}\sigma_n = 0 \quad & k > n. \end{aligned} \tag{6.50}$$

Proof Let σ_i^n be the ith elementary symmetric function in n variables and let s_i^n be the symmetric power sum function in n variables. Also, let $\sigma_0^n = 1$ and $\sigma_i^n = 0$ if $i < 0$ or $i > n$. Then the two Newton's identities (6.8) are subsumed into the single relationship

$$s_k^n - \sigma_1^n s_{k-1}^n + \cdots + (-1)^{k-1}\sigma_{k-1}^n s_1^n + (-1)^k k\sigma_k^n = 0 \qquad \text{for all } k \ge 1, \tag{6.51}$$

or, more concisely,

$$\sum_{j=0}^{k-1}\left[(-1)^j s_{k-j}^n \sigma_j^n\right] + (-1)^k k\sigma_k^n = 0 \qquad \text{for all } k \geq 1. \tag{6.52}$$

The proof of (6.51) relies on the observations that

$$s_k^m = s_k^{m-1} + x_m^k, \qquad m = 1, 2, \ldots, n \tag{6.53}$$

and

$$\sigma_i^m = \sigma_i^{m-1} + x_m\sigma_{i-1}^{m-1}, \qquad i = 1, \ldots, n; \quad m = 0, \ldots, n. \tag{6.54}$$

The former equation is by definition and the latter is by the multiplication operation. We do induction on the number of variables. When $m = n = 1$, (6.52) implies $s_1^1 = \sigma_1^1$, which is true by direct computation. Assume (6.52) is true for $n - 1$; we obtain the inductive hypothesis

$$\sum_{j=0}^{k-1}\left[(-1)^j s_{k-j}^{n-1}\sigma_j^{n-1}\right] + (-1)^k k\sigma_k^{n-1} = 0. \tag{6.55}$$

Then for n, using (6.53) and (6.54),

$$\sum_{j=0}^{k-1}\left[(-1)^j s_{k-j}^n\sigma_j^n\right] + (-1)^k k\sigma_k^n =$$

$$\sum_{j=0}^{k-1}(\underbrace{s_{k-j}^{n-1}}_{a} + \underbrace{x_n^{k-j}}_{b})(\underbrace{\sigma_j^{n-1}}_{c} + \underbrace{x_n\sigma_{j-1}^{n-1}}_{d}) + (-1)^k k(\underbrace{\sigma_k^{n-1}}_{e} + \underbrace{x_n\sigma_{k-1}^{n-1}}_{f})$$

$$= \underbrace{\sum_{j=0}^{k-1}(-1)^j s_{k-j}^{n-1}\sigma_j^{n-1} + (-1)^k k\sigma_k^{n-1}}_{a\times c+e=A} + \underbrace{x^n\left[\sum_{j=0}^{k-1}(-1)^j s_{k-j}^{n-1}\sigma_{j-1}^{n-1} + (-1)^k k\sigma_{k-1}^{n-1}\right]}_{a\times d+f=B}$$

$$\underbrace{+\underbrace{\sum_{j=0}^{k-1}(-1)^j x_n^{k-j}\sigma_j^{n-1}}_{b\times c} + \underbrace{\sum_{j=0}^{k-1}(-1)^j x_n^{k-j+1}\sigma_{j-1}^{n-1}}_{b\times d}}_{C}.$$

The terms in A are equal to zero by (6.55). The terms in B are

$$B = x^n\left[\left(-\sum_{l=0}^{k-2}(-1)^l s_{(k-1)-l}^{n-1}\sigma_l^{n-1} - (-1)^{k-1}(k-1)\sigma_{k-1}^{n-1}\right) - (-1)^{k-1}\sigma_{k-1}^{n-1}\right]$$
$$= -x^n(-1)^{k-1}\sigma_{k-1}^{n-1}$$

using (6.55) again. The terms in C cancel each other except for one term, so that

$$C = (-1)^{k-1}x_n\sigma_{k-1}^{n-1}.$$

Thus $B + C = 0$ and $\sum_{j=0}^{k-1} s_{k-j}^n\sigma_j^n + (-1)^k k\sigma_k^n = 0$. □

Since $f(x)$ is of the form $f(x) = \prod_{i=1}^n(x - X_i)$ and $\Lambda(x)$ is of the form $\Lambda(x) = \prod_{i=1}^n(1 - X_i x)$, it follows that $\Lambda(x) = x^n f(1/x)$, so that $\Lambda_i = (-1)^i\sigma_i$. This gives the form of the Newton identities shown in (6.8).

6.10 Exercises

6.1 For a binary, narrow-sense, triple error-correcting BCH code of length 15:

(a) Compute a generator polynomial for this code.
(b) Determine the rate of the code.
(c) Construct the parity check matrix and generator matrix for this code.

6.2 Find the generator $g(x)$ for a narrow-sense, binary double-error correcting code of blocklength $n = 63$.

6.3 Find a generator polynomial for a narrow-sense, double-error correcting binary BCH code of length 21.

6.4 Find a generator for a narrow-sense, double-error correcting quaternary BCH code of length 21.

6.5 Compute the weight distribution of a double-error-correcting binary BCH code of length $n = 15$.

6.6 Construct a narrow-sense Reed-Solomon code of length 15 and design distance 3. Find the generator polynomial and the parity-check and generator matrices. How does the rate of this code compare with the rate of the code found in Exercise 1.1?

6.7 Compute the weight distribution of an $(8, 3)$ 8-ary MDS code.

6.8 Show that when a MDS code is punctured, it is still MDS. *Hint*: Puncture, then use the Singleton bound.

6.9 Show that when a MDS code is shortened, it is still MDS. *Hint*: Let C be MDS and let C_i be the subset of codewords in C which are 0 in the ith position. Shorten on coordinate i, and use the Singleton bound.

6.10 [204] Show for a binary BCH t-error correcting code of length n that, if $2t + 1 \mid n$, then the minimum distance of the code is exactly $2t + 1$. *Hint*: Write $n = q(2t + 1)$ and show that $(x^n + 1)/(x^q + 1)$ is a code polynomial of weight exactly $2t + 1$. See Exercise 5.55.

6.11 Find $g(x)$ for a narrow-sense, double-error correcting RS code using $\alpha \in GF(2^4)$ as the primitive element. For this code, suppose the received data produces the syndromes $S_1 = \alpha^4$, $S_2 = 0$, $S_3 = \alpha^8$ and $S_4 = \alpha^2$. Find the error locator polynomial and the error locations using the Peterson-Gorenstein-Zierler decoder.

6.12 For a triple-error correcting, primitive, narrow-sense, binary BCH code of length 15, suppose that

$$r(x) = x^{12} + x^{11} + x^9 + x^8 + x^7 + x^6 + x^3 + x^2 + 1.$$

(a) Determine the syndromes S_1, S_2, S_3, S_4, S_5, and S_6.
(b) Check that $S_2 = S_1^2$, $S_4 = S_2^2$, and $S_6 = S_3^2$.
(c) Using the Peterson-Gorenstein-Zierler algorithm, determine the error locator polynomial and the decoded codeword.
(d) Find the error locator polynomial using the (nonbinary) Berlekamp-Massey algorithm. Provide the table illustrating the operation of the Berlekamp-Massey algorithm.
(e) Find the error locator polynomial using the binary Berlekamp-Massey algorithm. Provide the table illustrating the operation of the Berlekamp-Massey algorithm.
(f) Find the error locator polynomial using the Euclidean algorithm (the Sugiyama algorithm). Show the steps in the operations of the algorithm.

6.13 For a triple-error correcting, primitive, narrow-sense, binary BCH code of length 15, suppose that

$$r(x) = x^{13} + x^9 + x^4 + x^3 + 1.$$

(a) Determine the syndromes S_1, S_2, S_3, S_4, S_5, and S_6.
(b) Check that $S_2 = S_1^2$, $S_4 = S_2^2$, and $S_6 = S_3^2$.

(c) Find the error locator polynomial $\Lambda(x)$ using the (nonbinary) Berlekamp-Massey algorithm. Provide the table illustrating the operation of the Berlekamp-Massey algorithm. Also, find the factorization of $\Lambda(x)$, and determine the error locations and the decoded codeword.

(d) Find the error locator polynomial using the binary Berlekamp-Massey algorithm. Provide the table illustrating the operation of the Berlekamp-Massey algorithm. Compare the error locator polynomial with that found using the Berlekamp-Massey algorithm.

(e) Find the error locator polynomial using the Euclidean algorithm (the Sugiyama algorithm). Show the steps in the operations of the algorithm. Compare with the error locator polynomial found using the Berlekamp-Massey algorithm.

6.14 For a triple-error correcting, narrow-sense, Reed-Solomon code of length 15, suppose that

$$r(x) = \alpha^4 x^{12} + \alpha^7 x^9 + \alpha^2 x^7 + x^6 + \alpha^{10} x^5 + \alpha x^4 + \alpha^{12} x^3 + \alpha^4 x^2 + \alpha^{13}$$

(a) Determine the syndromes S_1, S_2, S_3, S_4, S_5, and S_6.

(b) Find the error locator polynomial $\Lambda(x)$ using the Berlekamp-Massey algorithm. Provide the table illustrating the operation of the Berlekamp-Massey algorithm. Also, find the factorization of $\Lambda(x)$ and determine the error locations.

(c) Determine the error values using Forney's algorithm and determine the decoded codeword.

(d) Find the error locator polynomial using the Euclidean algorithm (the Sugiyama algorithm). Show the steps in the operations of the algorithm. Compare with the error locator polynomial found using the Berlekamp-Massey algorithm.

6.15 For a triple-error correcting, narrow-sense, Reed-Solomon code of length 15, suppose that

$$r(x) = \alpha^{11} x^9 + \alpha^3 x^7 + \alpha x^6 + \alpha^7 x^5 + \mathsf{E} x^4 + \mathsf{E} x^3 + \alpha^4 x^2 + \alpha^9 x + \alpha^9,$$

where E denotes that the position is erased.

(a) Determine the erasure locator polynomial $\Gamma(x)$.

(b) Determine $\tilde{r}(x)$ and the syndromes S_1, S_2, S_3, S_4, S_5, and S_6.

(c) Find $\Xi(x)$.

(d) Using the Berlekamp-Massey or Euclidean algorithm determine $\Lambda(x)$ and find the error locations.

(e) Determine the error and erasure values using Forney's algorithm, and determine the decoded codeword.

6.16 The decoding algorithms described in this chapter assume that narrow-sense codes are used. Carefully describe what changes would have to be made in the decoder if a *non*-narrow sense code is used. In particular:

(a) How do computations change for finding the error locating polynomial $\Lambda(x)$?

(b) How does the Forney algorithm change for finding the error values?

6.17 The Berlekamp-Massey algorithm (Algorithm 6.1) requires division in the field, that is, finding a multiplicative inverse. This can be more complex than multiplication or addition. Is it possible to modify the algorithm so that it still produces an error locator polynomial, but does not require any divisions?

6.18 Let C be the $(2^m - 1, k)$ Reed-Solomon code with minimum distance d. Show that C contains the primitive $(2^m - 1, k)$ binary BCH code C' of length $2^m - 1$ with design distance d. This is an example of a subfield subcode.

6.19 [204] Is there a binary t-error-correcting BCH code of length $n = 2^m + 1$ for $m \geq 3$ and $t < 2^{m-1}$. If so, determine its generator polynomial.

6.20 [204] Let $b = -t$. A BCH code with designed distance $d = 2t + 2$ whose generator polynomial has $\beta^{-t}, \ldots, \beta^{-1}, 1, \beta, \ldots, \beta^{t}$ and their conjugates as roots. Show that this is a reversible cyclic code. (See Exercise 4.14a.) Also, show that if t is odd, then the minimum distance of this code is at least $2t + 4$.

6.21 [204] A t-error correcting Reed-Solomon code of length $n = 2^m - 1$ over $GF(2^m)$ has the following parity check matrix.

$$H = \begin{bmatrix} 1 & \alpha & \alpha^2 & \cdots & \alpha^{n-1} \\ 1 & \alpha^2 & \alpha^4 & \cdots & \alpha^{2(n-1)} \\ \vdots & & & & \\ 1 & \alpha^{2t} & \alpha^{4t} & \cdots & \alpha^{2t(n-1)} \end{bmatrix},$$

where α is primitive in $GF(2^m)$. Now form the parity check matrix

$$H' = \begin{bmatrix} 0 & 1 & \\ 0 & 0 & \\ \vdots & \vdots & H \\ 0 & 0 & \\ 1 & 0 & \end{bmatrix}.$$

Show that the code with parity check matrix H' also has minimum distance $2t+1$. *Hint*: Consider $2t \times 2t$ submatrices. For submatrices including the first columns, think of the cofactor expansion.

6.22 Let $m(x) = m_0 + m_1 x + \cdots + m_{k-1}x^{k-1} \in GF(2^m)[x]$. Form the polynomial

$$c(x) = m(1) + m(\alpha)x + m(\alpha^2)x^2 + \cdots + m(\alpha^{2^m-2})x^{2^m-2}.$$

Show that $c(x)$ has $\alpha, \alpha^2, \ldots, \alpha^{2^m-k-1}$ as roots. What can you conclude about the set of all $\{c(x)\}$ as $m(x)$ varies?

6.23 Let $\mathbf{v} = [v_0, v_1, \ldots, v_{n-1}]$, $v_i \in GF(q)$ be a sequence and let $v(x) = v_0 + v_1 x + \cdots + v_{n-1}x^{n-1}$. Let $V_j = v(\alpha^j) = \sum_{i=0}^{n-1} v_i \alpha^{ij}$, $j = 0, 1, \ldots, n-1$, where α is a primitive nth root of unity. The **Mattson-Solomon polynomial** (see [220, p. 239] is defined as

$$A(z) = \sum_{j=0}^{n-1} A_{n-j} z^j = \sum_{j=1}^{n} A_j z^{n-j}.$$

(a) Show that $A_0 = A_n$.

(b) Show that the a_i can be recovered from $A(z)$ by $a_i = n^{-1}A(\alpha^i)$.

(c) Show that if the a_i are binary valued and operations take place in a field $GF(2^m)$, then $R_{z^n+1}[A(z)^2] = A(z)$.

(d) Show that if $c(x) = R_{x^n+1}[a(x)b(x)]$ (that is, the product modulo $x^n + 1$), then $C(z) = A(z) \odot B(z)$, where $\odot$ denotes the element-by-element product, $A(z) \odot B(z) = \prod_{i=0}^{n-1} A_i B_i z^i$, and conversely.

6.24 Using the definition of the formal derivative in (6.33) for operations over the commutative ring with identity $R[x]$, where R is a ring:

(a) Show that $[f(x)g(x)]' = f'(x)g(x) + f(x)g'(x)$.

(b) Show that if $f^2(x) \mid g(x)$ then $f(x) \mid g'(x)$.

(c) Show that $(f_1 f_2)^{(n)} = \sum_{i=0}^{n} \binom{n}{i} f_1^{(i)} f_2^{(n-i)}$, where $()^{(n)}$ denotes the nth formal derivative.

(d) Show that $(f_1 f_2 \cdots f_r)' = \sum_{i=1}^{r} f_i' \prod_{j \neq i} f_j$.

(e) From the latter conclude that with $f = f_1 f_2 \cdots f_r$,

$$\frac{f'}{f} = \frac{f_1'}{f_1} + \cdots + \frac{f_r'}{f_r}.$$

This is a partial fraction decomposition of f'/f.

(f) Let u be a simple root of $f(x)$ and let $g(x) = f(x)/(x-u)$. Show that $f'(u) = g(u)$.

(g) Show that if f has a repeated root at u, then $(x-u)$ is a factor of (f, f').

6.25 Prove Lemma 6.14. *Hint*: Use the fact that $S_{2v} = S_v^2$ for binary codes.

6.26 Express Massey's algorithm in a form appropriate for using 0-based indexing; that is, when the syndromes are numbered $S_0, S_1, \ldots, S_{N-1}$, where $N = 2t$.

6.27 Compute the GFFT in $GF(8)$ of the vector represented by the polynomial $v(x) = 1 + \alpha^2 x + \alpha^3 x^4 + \alpha^6 x^5$. Also, compute the inverse GFFT of $V(x)$ and ensure that $v(x)$ is obtained.

6.28 Let $\{a_i\}$ be a sequence of length n, $\{a_i\} = \{a_0, a_1, \ldots, a_{n-1}\}$. Let $\{a_{((i-1))}\}$ denote the cyclic shift of the sequence $\{a_i\}$. Let $\{a_i\} \leftrightarrow \{A_j\}$ denote that there is a GFFT relationship between the sequences. Prove the following properties of the GFFT.

$$\text{Cyclic shift property: } \{a_{((i-1))}\} \leftrightarrow \{\alpha^j A_j\}$$

$$\text{Modulation property: } \{\alpha^i a_i\} \leftrightarrow \{A_{((j+1))}\}$$

6.29 Determine the GFFT of the vector $v_i = \alpha^{ri}$. Determine the GFFT of the vector $v_i = \beta \delta_{i-l}$; that is, it has value $v_i = \beta$ when $i = l$ and value $v_i = 0$ when $i \neq l$. Assume $0 \le l < n$.

6.30 List the minimal polynomials over $GF(16)$. Compute the GFFT of the corresponding vectors and note the positions of the zeros of the spectra compared to the roots of the minimal polynomials.

6.31 Computing the GFFT. Let $v_i \in GF(2^m)$, $i = 0, 1, \ldots, n-1$. Let

$$V_j = \beta^{j^2} \sum_{i=0}^{n-1} \beta^{-(j-i)^2} (\beta^{i^2} v_i). \tag{6.56}$$

Show when β is a square root of α that V_j of (6.56) equal to V_j of (6.38). That is, (6.56) can be used to compute the GFFT. This is called the Bluestein chirp algorithm. The chirp transform can be computed as a pointwise product of v_i by β^{i^2} followed by a cyclic convolution with β^{-i^2}.

6.32 Describe how to obtain a (mn, mk) binary code from a (n, k) code over $GF(2^m)$. Let d be the minimum distance of the (n, k) code and let d_m be the minimum distance of the (mn, mk) code. How does $d_m/(mn)$ compare with d/n in general?

6.33 Let $M_i(x)$ be the minimal polynomial of α^i, where α^i is a primitive element in $GF(2^n)$. Let

$$h^*(x) = \frac{x^{2^n-1} + 1}{\text{LCM}(M_1(x), M_2(x), \ldots, M_{2k}(x))}$$

and let $V(h^*)$ be the set of sequences of length $2^n - 1$ which are annihilated by $h^*(x)$. Show that for any two sequences $a, b \in V(g^*)$, $\text{wt}(a+b) > 2k$.

6.34 (Justeson codes) Let $\mathcal{C}$ be an (n, k) Reed-Solomon code over $GF(q^m)$. Let $\mathbf{c} = (c_0, c_1, \ldots, c_{n-1})$ be a codeword in $\mathcal{C}$. Form the $2 \times n$ matrix by

$$M = \begin{bmatrix} c_0 & c_1 & c_2 & \cdots & c_{n-1} \\ \alpha^0 c_0 & \alpha^1 c_1 & \alpha^2 c_2 & \cdots & \alpha^{n-1} c_{n-1} \end{bmatrix}.$$

Replace each $GF(q^m)$ element of this matrix by a $GF(q)$ m-tuple, resulting in a $2m \times N$ matrix. The set of such matrices produces the *Justeson codes*, a q-ary code whose codewords have length $2mN$ obtained by stacking up the elements of the matrix into a vector.

(a) Show that the Justeson code is linear.

(b) Explain why at least $n - k + 1$ of the columns of a matrix M are nonzero.

(c) Explain why no two nonzero columns can be the same.

(d) Let I be an integer satisfying

$$\sum_{i=1}^{I}(q-1)^i\binom{2m}{i} \leq n-k+1.$$

Show that the minimum distance of the Justeson code is lower-bounded by

$$d_{\min} \geq \sum_{i=1}^{I} i(q-1)^i\binom{2m}{i}.$$

Hint: In a $2m$-tuple there are $\binom{2m}{i}$ ways of picking i nonzero places and $q-1$ different nonzero values. The minimum distance greater than or equal to the sum of the weights of the columns.

It can be shown that [33] asymptotically $d_{\min}/n \geq 0.11(1-2R)$, where $R = k/2n$ is the rate of the code. Thus Justeson codes have a fractional distance bounded away from 0 if $R < 1/2$.

6.35 Newton's identities: Show that $\sigma_i^m = \sigma_i^{m-1} + x_n \sigma_{i-1}^{m-1}$ is true.

6.36 Newton's identities: Let

$$f(x) = (x-x_1)(x-x_2)\cdots(x-x_n).$$

Show that $f(x)$ can be written in terms of the elementary functions by

$$f(x) = x^n - \sigma_1 x^{n-1} + \sigma_2 x^{n-2} + \cdots + (-1)^{n-1}\sigma_{n-1}x + (-1)^n \sigma_n.$$

Hint: Give an inductive proof using the identity (6.54).

6.11 References

Reed-Solomon codes were presented first in [286]; these are, of course, structurally similar to the BCH codes announced in [151] and [36]. The first decoding algorithms were based on solving Toeplitz systems; this is the essence of the Peterson decoding algorithm [261]. The generalization to nonbinary codes appears in [125]. The Berlekamp-Massey algorithm was presented in [25] and [222]. The Forney algorithm appears in [86] and the Chien search appears in [49]. More information about weight distributions of BCH codes appears in [186]. The theoretical aspects of these codes are considerably richer than this chapter can contain and could cover several chapters. Interested readers are referred to [220]

There are many other decoding algorithms to explore for these codes. It is possible to scale the Berlekamp-Massey algorithm so that no divisions are required to produce an error-locator polynomial [38]; this may be of interest in hardware implementations. However, computing the error value still requires divisions. Decoding algorithms based on finite-field Fourier transforms have been developed; for a thorough survey see [33]. See also [372] for a systolic decoding algorithm and [285] for an algorithm based on Fermat transforms. A work which shows the underlying similarity of the Euclidean algorithm-based methods and the Berlekamp-Massey based methods is [52].

Blahut [33] has made many of the contributions to Galois field Fourier transform methods for coding theory; the presentation here closely follows [373]. Berlekamp-Massey algorithm

is presented in [222]; our presentation closely follows that of [246]. A very theoretical description of BCH and RS codes appears in [220], with considerable material devoted to MDS codes.

The discussion of Newton's identities is suggested by an exercise in [60]. The discussion of alternant and Goppa codes is summarized from [220]. A decoding algorithm for Goppa codes appears in [256]. McEliece public key cryptosystem was presented in [229].

Chapter 7

Alternate Decoding Algorithms for Reed-Solomon Codes

7.1 Introduction: Workload for Reed-Solomon Decoding

In this chapter we present two alternatives to the decoding algorithms presented in chapter 6. The first is based upon a new key equation and is called **remainder decoding**. The second method is a list decoder capable of decoding beyond the design distance of the code.

A primary motivation behind the remainder decoder is that implementations of it may have lower decode complexity. The decode complexity for a conventional decoding algorithm for an (n, k) code having redundancy $\rho = n - k$ is summarized by the following steps:

1. Compute the syndromes. ρ syndromes must be computed, each with a computational cost of $O(n)$, for a total cost of $O(\rho n)$. Furthermore, all syndromes must be computed, regardless of the number of errors.
2. Find the error locator polynomial and the error evaluator. This has a computation cost $O(\rho^2)$ (depending on the approach).
3. Find the roots of the error locator polynomial. This has a computation cost of $O(\rho n)$ using the Chien search.
4. Compute the error values, with a cost of $O(\rho^2)$.

Thus, if $\rho < n/2$, the most expensive steps are computing the syndromes and finding the roots. In remainder decoding, decoding takes place by computing remainders instead of syndromes; the remaining steps retain similar complexity. This results in potentially faster decoding. Furthermore, as we demonstrate, it is possible to find the error locator polynomial using a highly-parallelizable algorithm. The general outline for the new decoding algorithm is as follows:

1. Compute the remainder polynomial $r(x) = R(x) \mod g(x)$, with complexity $O(n)$ (using very simple hardware).
2. Compute an error locator polynomial $W(x)$ and an associated polynomial $N(x)$. The complexity is $O(\rho^2)$. Architectures exist for parallel computation.
3. Find the roots of the error locator polynomial, complexity $O(\rho n)$.
4. Compute the error values, complexity $O(\rho^2)$.

7.2 Derivations of Welch-Berlekamp Key Equation

We present in this section two derivations of a new key equation called the Welch-Berlekamp (WB) key equation. The first derivation uses the definition of the remainder polynomial. The

second derivation shows that the WB key equation can be obtained from the conventional Reed-Solomon key equation.

7.2.1 The Welch-Berlekamp Derivation of the WB Key Equation

The generator polynomial for an (n, k) RS code can be written as

$$g(x) = \prod_{i=b}^{b+d-2} (x - \alpha^i),$$

which is a polynomial of degree $d - 1$, where $d = d_{\min} = 2t + 1 = n - k + 1$. We denote the received polynomial as $R(x) = c(x) + E(x)$. We designate the first $d - 1$ symbols of $R(x)$ as *check symbols*, and the remaining k symbols as *message symbols*. This designation applies naturally to systematic encoding of codewords, but we use it even in the case that systematic encoding is not employed. Let $L_c = \{0, 1, \ldots, d-2\}$ be the index set of the check locations, with corresponding check locators $L_{\alpha^c} = \{\alpha^k, 0 \leq k \leq d-2\}$. Also, let $L_m = \{d-1, d, \ldots, n-1\}$ denote the index set of the message locations, with corresponding message locators $L_{\alpha^m} = \{\alpha^k, d-1 \leq k \leq n-1\}$.

We define the *remainder polynomial* as

$$r(x) = R(x) \mod g(x)$$

and write $r(x)$ in terms of its coefficients as

$$r(x) = \sum_{i=0}^{d-2} r_i x^i.$$

The degree of $r(x)$ is $\leq d - 2$. This remainder can be computed using conventional LFSR hardware that might be used for the encoding operation, with computational complexity $O(n)$.

Lemma 7.1 $r(x) \equiv E(x) \mod g(x)$ *and* $r(\alpha^k) = E(\alpha^k)$ *for* $k \in \{b, b+1, \ldots, b+d-2\}$.

Proof Since $R(x) = m(x)g(x) + E(x)$ for some message polynomial $m(x)$, the remainder polynomial does not depend upon the transmitted codeword. Thus

$$r(x) \equiv E(x) \mod g(x).$$

We can write $E(x) = q(x)g(x) + e(x)$ for some divisor polynomial $q(x)$. Thus $E(x) \equiv e(x) \mod g(x)$. Then $E(\alpha^k) = q(\alpha^k)g(\alpha^k) + e(\alpha^k) = e(\alpha^k) = r(\alpha^k)$ for $k \in \{b, b+1, \ldots, b+d-2\}$. □

Notation: At some points in the development, it is convenient to use the notation $r_k = r[\alpha^k]$ to indicate the remainder at index k (with locator α^k). Similarly, we will use $Y[\alpha^k]$ to indicate an error value at index k.

Single error in a message location. To derive the WB key equation, we assume initially that a single error occurs. We need to make a distinction between whether the error location e is a message location or a check location. Initially we assume that $e \in L_m$ with error value Y. We thus take $E(x) = Yx^e$, or the (error position, error value) = $(\alpha^e, Y) = (X, Y)$. The notation $Y = Y[X]$ is also used to indicate the error value at the error locator X.

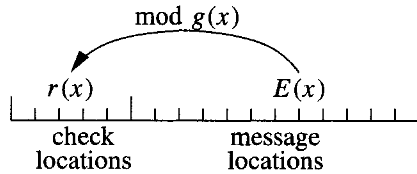


Figure 7.1: Remainder computation when errors are in message locations.

When $e \in L_m$, then the modulo operation $Yx^e \mod g(x)$ "folds" the polynomial back into the lower order terms, as pictured in Figure 7.1. Evaluating $r(x)$ at generator root locations we have by Lemma 7.1,

$$r(\alpha^k) = E(\alpha^k) = Y(\alpha^k)^e = YX^k, \qquad k \in \{b, b+1, \ldots, b+d-2\}, \tag{7.1}$$

where $X = \alpha^e$ is the error locator. It follows that

$$r(\alpha^k) - Xr(\alpha^{k-1}) = YX^k - XYX^{k-1} = 0, \qquad k \in \{b+1, b+2, \ldots, b+d-2\}.$$

Define the polynomial $u(x) = r(x) - Xr(\alpha^{-1}x)$, which has degree less than $d-1$. Then $u(x)$ has roots at $\alpha^{b+1}, \alpha^{b+2}, \ldots, \alpha^{b+d-2}$, so that $u(x)$ is divisible by the polynomial

$$p(x) = \prod_{i=b+1}^{b+d-2} (x - \alpha^k) = \sum_{i=0}^{d-2} p_i x^i,$$

which has degree $d-2$. Thus $u(x)$ must be a *scalar* multiple of $p(x)$,

$$u(x) = ap(x) \tag{7.2}$$

for some $a \in GF(q)$. Equating coefficients between $u(x)$ and $p(x)$ we obtain

$$r_i(1 - X\alpha^{-i}) = ap_i, \qquad i = 0, 1, \ldots, d-2.$$

That is,

$$r_i(\alpha^i - X) = a\alpha^i p_i, \qquad i = 0, 1, \ldots, d-2. \tag{7.3}$$

We define the error locator polynomial as $W_m(x) = x - X = x - \alpha^e$. (This definition is different from the error locator we defined for the conventional decoding algorithm, since the roots of $W_m(x)$ are the message locators, not the reciprocals of message locators.) Using $W_m(x)$, we see from (7.3) that

$$r_i W_m(\alpha^i) = a\alpha^i p_i, \qquad i = 0, 1, \ldots, d-2. \tag{7.4}$$

Since the error is in a message location, $e \in L_m$, $W_m(\alpha^i)$ is not zero for $i = 0, 1, \ldots, d-2$. We can solve for r_i as

$$r_i = a\alpha^i p_i / W_m(\alpha^i). \tag{7.5}$$

We now eliminate the coefficient a from (7.5). The error value Y can be computed using (7.1), choosing $k = b$:

$$Y = Y[X] = X^{-b} r(\alpha^b) = X^{-b} \sum_{i=0}^{d-2} r_i \alpha^{ib} = X^{-b} \sum_{i=0}^{d-2} \frac{a\alpha^i p_i}{W_m(\alpha^i)} \alpha^{ib} = aX^{-b} \sum_{i=0}^{d-2} \frac{\alpha^{i(b+1)} p_i}{(\alpha^i - X)}.$$

Define

$$f(x) = x^{-b} \sum_{i=0}^{d-2} \frac{\alpha^{i(b+1)} p_i}{(\alpha^i - x)}, \qquad x \in L_{\alpha^m},$$

which can be pre-computed for all values of $x \in L_{\alpha^m}$. Then

$$Y = af(X)$$

or $a = Y/f(X)$. We thus write (7.4) as

$$r_i = \frac{Y\alpha^i p_i}{f(X) W_m(\alpha^i)}. \tag{7.6}$$

Multiple errors in message locations. Now assume that there are $\nu \geq 1$ errors, with error locators $X_i \in L_{\alpha^m}$ and corresponding error values $Y_i = Y[X_i]$ for $i = 1, 2, \ldots, \nu$. Corresponding to each error there is a "mode" yielding a relationship $r(\alpha^k) = Y_i X_i^k$, each of which has a solution of the form (7.6). Thus by linearity we can write

$$r_k = r[\alpha^k] = p_k \alpha^k \sum_{i=1}^{\nu} \frac{Y_i}{f(X_i)(\alpha^k - X_i)}, \qquad k = 0, 1, \ldots, d-2. \tag{7.7}$$

Now define the function

$$F(x) = \sum_{i=1}^{\nu} \frac{Y_i}{f(X_i)(x - X_i)} \tag{7.8}$$

having poles at the error locations. This function can be written as

$$F(x) = \sum_{i=1}^{\nu} \frac{Y_i}{f(X_i)(x - X_i)} = \frac{N_m(x)}{W_m(x)},$$

where

$$W_m(x) = \prod_{i=1}^{\nu} (x - X_i)$$

is the error locator polynomial for the errors among the symbol locations and where $N_m(x)$ is the numerator obtained by adding together the terms in $F(x)$. It is clear that $\deg(N_m(x)) < \deg(W_m(x))$. Note that the representation in (7.8) corresponds to a partial fraction expansion of $N_m(x)/W_m(x)$. Using this notation, (7.7) can be written

$$r_k = p_k \alpha^k F(\alpha^k) = p_k \alpha^k N_m(\alpha^k)/W_m(\alpha^k)$$

or

$$N_m(\alpha^k) = \frac{r_k}{p_k \alpha^k} W_m(\alpha^k), \qquad k \in L_c = \{0, 1, \ldots, d-2\}. \tag{7.9}$$

$N_m(x)$ and $W_m(x)$ have the degree constraints $\deg(N_m(x)) < \deg(W_m(x))$ and $\deg(W_m(x)) \leq \lfloor (d-1)/2 \rfloor = t$, since no more than t errors can be corrected. Equation (7.9) has the form of the **key equation** we seek.

Errors in check locations For a single error occurring in a check location $e \in L_c$, then $r(x) = E(x)$ — there is no "folding" by the modulo operation. Then $u(x) = r(x) - Xr(\alpha^{-1}x)$ must be identically 0, so the coefficient a in (7.2) is equal to 0. We can write

$$r_k = \begin{cases} Y & k = e \\ 0 & \text{otherwise.} \end{cases}$$

If there are errors in both check locations and message locations, let $E_m = \{i_1, i_2, \ldots, i_{\nu_1}\} \subset L_m$ denote the error locations among the message locations and let $E_c = \{i_{\nu_1+1}, \ldots, i_\nu\} \subset L_c$ denote the error locations among the check locations. Let $E_{\alpha^m} = \{\alpha^{i_1}, \alpha^{i_2}, \ldots, \alpha^{i_{\nu_1}}\}$ and $E_{\alpha^c} = \{\alpha^{i_{\nu_1+1}}, \ldots, \alpha^{\nu}\}$ denote the corresponding error locators. The (error location, error value) pairs for the errors in message locations are $(X_i, Y_i), i = 1, 2, \ldots, \nu_1$. The pairs for errors in check locations are $(X_i, Y_i), i = \nu_1 + 1, \ldots, \nu$. Then by linearity,

$$r_k = p_k \alpha^k \sum_{i=1}^{\nu_1} \frac{Y_i}{f(X_i)(\alpha^k - X_i)} + \begin{cases} Y_j & \text{if error locator } X_j = \alpha^k \text{ is in a check location} \\ 0 & \text{otherwise.} \end{cases} \tag{7.10}$$

Because of the extra terms added on in (7.10), equation (7.9) does not apply when $k \in E_c$, so we have

$$N_m(\alpha^k) = \frac{r_k}{p_k \alpha^k} W_m(\alpha^k), \qquad k \in L_c \setminus E_c. \tag{7.11}$$

To account for the errors among the check symbols, let $W_c(x) = \prod_{i \in L_c}(x - \alpha^i)$ be the error locator polynomial for errors in check locations. Let

$$N(x) = N_m(x) W_c(x) \qquad \text{and} \qquad W(x) = W_m(x) W_c(x).$$

Since $N(\alpha^k) = W(\alpha^k) = 0$ for $k \in E_c$, we can write

$$\boxed{N(\alpha^k) = \frac{r_k}{p_k \alpha^k} W(\alpha^k), \qquad k \in L_c = \{0, 1, \ldots, d-2\}.} \tag{7.12}$$

That is, the equation is now satisfied for *all* values of k in L_c. Equation (7.12) is the Welch-Berlekamp (WB) **key equation**, to be solved subject to the conditions

$$\deg(N(x)) < \deg(W(x)) \qquad \deg(W(x)) \le (d-1)/2.$$

The polynomial $W(x)$ is the error locator polynomial, having roots at *all* the error locators. We write (7.12) as

$$\boxed{N(x_i) = W(x_i) y_i \qquad i = 1, 2, \ldots, m = 2t = d-1} \tag{7.13}$$

for "points" $(x_i, y_i) = (\alpha^{i-1}, r_{i-1}/(p_{i-1}\alpha^{i-1})), i = 1, 2, \ldots, m = 2t$.

Example 7.1 Consider a (15, 9) triple-error correcting RS code over $GF(16)$ generated by $1+x+x^4$ with $b = 2$. The generator for the code is

$$g(x) = \alpha^{12} + \alpha^{14}x + \alpha^{10}x^2 + \alpha^7 x^3 + \alpha x^4 + \alpha^{11}x^5 + x^6.$$

The function $p(x)$ is

$$p(x) = \alpha^{10} + \alpha^9 x + \alpha^{11}x^2 + \alpha^6 x^3 + \alpha^9 x^4 + x^5.$$

The function $f(x)$ evaluated at message locators is tabulated as

x:	α^6	α^7	α^8	α^9	α^{10}	α^{11}	α^{12}	α^{13}	α^{14}
$f(x)$:	1	α^8	α^{12}	α^2	α^{11}	α^7	α^7	α^8	α^5

Suppose the received data is $R(x) = \alpha^5 x^4 + \alpha^7 x^9 + \alpha^8 x^{12}$ (three errors). Then

$$r(x) = R(x) \pmod{g(x)} = \alpha^2 + \alpha x + \alpha^6 x^3 + \alpha^3 x^5.$$

The (x_i, y_i) data appearing in the key equation are

i	1	2	3	4	5	6
x_i	1	α	α^2	α^3	α^4	α^5
y_i	α^7	α^6	0	α^{12}	0	α^{13}

□

Hereafter we will refer to the $N(x)$ and $W(x)$ as $N_1(x)$ and $W_2(x)$, referring to the first (WB) derivation.

7.2.2 Derivation From the Conventional Key Equation

A WB-type key equation may also be obtained starting from the conventional key equation and syndromes. Let us denote the syndromes as

$$S_i = R(\alpha^{b+i}) = r(\alpha^{b+i}) = \sum_{j=0}^{d-2} r_j (\alpha^{b+i})^j, \qquad i = 0, 1, \ldots, d-2.$$

The conventional error locator polynomial is $\Lambda(x) = \prod_{i=1}^{\nu}(1 - X_i x) = \Lambda_0 + \Lambda_1 x + \cdots + \Lambda_\nu x^\nu$ where $\Lambda_0 = 1$; the Welch-Berlekamp error locator polynomial is $W(x) = \prod_{i=1}^{\nu}(x - X_i) = W_0 + W_1 x + \cdots + x^\nu$. These are related by $\Lambda_i = W_{\nu-i}$. The conventional key equation can be written as (see 6.10)

$$\sum_{i=0}^{\nu} \Lambda_i S_{k-i} = 0, \qquad k = \nu, \nu+1, \ldots, d-2.$$

Writing this in terms of coefficients of W we have

$$\sum_{i=0}^{\nu} W_i S_{k+i} = 0 \qquad k = 0, 1, \ldots, d-2-\nu,$$

or

$$\sum_{i=0}^{\nu} W_i \sum_{j=0}^{d-2} r_j \alpha^{j(b+k+i)} = 0.$$

Rearranging,

$$\sum_{j=0}^{d-2} r_j \left(\sum_{i=0}^{\nu} W_i \alpha^{ji} \right) \alpha^{j(k+b)} = 0, \qquad k = 0, 1, \ldots, d-2-\nu. \tag{7.14}$$

Letting

$$f_j = r_j W(\alpha^j) \alpha^{jb}, \tag{7.15}$$

equation (7.14) can be written as

$$\sum_{j=0}^{d-2} f_j \alpha^{jk} = 0, \qquad k = 0, 1, \ldots, d-2-\nu,$$

which corresponds to the Vandermonde set of equations

$$\begin{bmatrix} 1 & 1 & \cdots & 1 & 1 \\ 1 & \alpha & \cdots & \alpha^{d-3} & \alpha^{d-2} \\ 1 & \alpha^2 & \cdots & \alpha^{2(d-3)} & \alpha^{2(d-2)} \\ \vdots & & & & \\ 1 & \alpha^{d-2-\nu} & \cdots & \alpha^{(d-2-\nu)(d-3)} & \alpha^{(d-2-\nu)(d-2)} \end{bmatrix} \begin{bmatrix} f_0 \\ f_1 \\ \vdots \\ f_{d-2} \end{bmatrix} = \mathbf{0}$$

with $(d-1-\nu) \times (d-1)$ matrix V. The bridge to the WB key equation is provided by the following lemma.

Lemma 7.2 *[63] Let V be a $m \times r$ matrix with $r > m$ having Vandermonde structure*

$$V = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ u_1 & u_2 & \cdots & u_r \\ u_1^2 & u_2^2 & \cdots & u_r^2 \\ \vdots & & & \\ u_1^{m-1} & u_2^{m-1} & \cdots & u_r^{m-1} \end{bmatrix}$$

with the $\{u_i\}$ all distinct. For any vector $\mathbf{z}$ in the nullspace of V (satisfying $V\mathbf{z} = \mathbf{0}$), there exists a unique *polynomial $N(x)$ of degree less than $r - m$ such that*

$$z_i = \frac{N(u_i)}{F'(u_i)}, \qquad i = 1, 2, \ldots, r,$$

where $F(x) = \prod_{i=1}^{r}(x - u_i)$.

Proof A vector $\mathbf{z}$ in the nullspace must satisfy

$$\sum_{i=1}^{r} u_i^j z_i = 0, \qquad j = 0, 1, \ldots, m-1. \tag{7.16}$$

Let $N(x)$ be a polynomial of degree $< r - m$. Then the highest degree of polynomials of the form

$$x^j N(x), \qquad j = 0, 1, \ldots, m-1$$

is less than $r - 1$. Now let us construct an interpolating function $\phi_j(x)$ such that for $u_1, u_2, \ldots, u_r$, $\phi_j(u_i) = u_i^j N(u_i)$, for each $j = 0, 1, \ldots, m-1$. Using the Lagrange interpolating function (5.27), we can write

$$\phi_j(x) = \sum_{i=1}^{r} \frac{F(x)}{x - u_i} \frac{u_i^j N(u_i)}{F'(u_i)}, \qquad j = 0, 1, \ldots, m-1. \tag{7.17}$$

Since it is the case that $\phi_j(x) = x^j N(x)$ has degree less than $r - 1$, the coefficient of the monomial of degree $r - 1$ on the right-hand side of (7.17) must be equal to 0. Since the leading coefficient of $F(x)$ is 1, the leading coefficient of $F(x)/(x - u_i)$ is 1. Thus

$$\sum_{i=1}^{r} \frac{u_i^j N(u_i)}{F'(u_i)} = 0, \qquad j = 0, 1, \ldots, m-1.$$

Thus if $z_i = N(u_i)/F'(u_i)$, the nullspace relationship (7.16) is satisfied.

The dimension of the nullspace of V is $r-m$. The dimension of the space of polynomials of degree $< r-m$ is $r-m$. There must therefore be a one-to-one correspondence between vectors in the nullspace of V and the polynomials $N(x)$. Thus $N(x)$ is unique. □

Returning to the key equation problem, by this lemma, there exists a polynomial $N(x)$ of degree less than ν such that $f_j = N(\alpha^j)/g_0'(\alpha^j)$, where

$$g_0(x) = \prod_{i=0}^{d-2}(x - \alpha^i).$$

Thus from (7.15),

$$r_j W(\alpha^j)\alpha^{jb} = \frac{N(\alpha^j)}{g_0'(\alpha^j)}, \quad j = 0, 1, \ldots, d-2.$$

This gives rise to a form of the key equation,

$$\boxed{N(\alpha^k) = r_k g_0'(\alpha^k)\alpha^{kb} W(\alpha^k), \qquad k = 0, 1, \ldots, d-2} \tag{7.18}$$

with $\deg(N(x)) < \deg(W(x)) \leq \lfloor (d-1)/2 \rfloor$. We call this the Dabiri-Blake (DB) form of the WB key equation. We can also write this in terms of the original generator polynomial $g(x)$:

$$\boxed{N(\alpha^k) = r_k g'(\alpha^{b+k})\alpha^{b(2-d+k)} W(\alpha^k), \qquad k = 0, 1, \ldots, d-2.} \tag{7.19}$$

With $x_{i+1} = \alpha^i$ and $y_{i+1} = r_k g'(\alpha^{b+k})\alpha^{b(2-d+k)}$, this can be expressed in the form (7.13).

Example 7.2 For the same code and $R(x)$ as in Example 7.1, the (x_i, y_i) data are

i	1	2	3	4	5	6
x_i	1	α	α^2	α^3	α^4	α^5
y_i	α^9	α^8	0	α^{14}	0	1

□

We will refer to the $N(x)$ and $W(x)$ derived using the DB method as $N_2(x)$ and $W_2(x)$.

7.3 Finding the Error Values

We begin with the key equation in the WB form, (7.12). Assuming that the error locator $W(x)$ has been found — as discussed in Section 7.4 — we consider here how to compute the error values Y_i corresponding to an error locator X_i; we denote this as $Y[X_i]$. For an error location in a message location, we have from (7.7) and (7.8)

$$r_k = p_k\alpha^k \sum_{i=1}^{\nu_1} \frac{Y[X_i]}{f(X_i)(\alpha^k - X_i)} = p_k\alpha^k \frac{N_1(\alpha^k)}{W_1(\alpha^k)}.$$

By definition,

$$\sum_{i=1}^{\nu} \frac{Y[X_i]}{f(X_i)(x - X_i)} = \frac{N_1(x)}{W_1(x)} = \frac{N_1(x)}{\prod_{i \in E_{cm}}(x - X_i)}, \tag{7.20}$$

where we use $E_{cm} = E_c \cup E_m$ to denote the set of all error locations. Multiplying both sides of (7.20) by $W(x)$ and evaluating at $x = X_k$, we obtain

$$\frac{Y[X_k]\prod_{i\neq k}(X_k - X_i)}{f(X_k)} = N_1(X_k),$$

since the factor $(x - X_k)$ in the denominator of (7.20) cancels the corresponding factor in $W_1(x)$, but all other terms in the sum are zero since they have factors of zero in the product.

Now taking the formal derivative, we observe that

$$W_1'(x) = \sum_{i\in E_{cm}} \prod_{j\neq i}(x - X_i)$$

so that $W_1'(X_k) = \prod_{j\neq k}(X_k - X_i)$. Thus

$$Y[X_k] = f(X_k)\frac{N_1(X_k)}{W_1'(X_k)}. \tag{7.21}$$

When the error is in a check location, $X_j = \alpha^k$ for $k \in E_c$, we must revert to (7.10),

$$r_k = Y[X_j] + p_k\alpha^k \sum_{i=1}^{\nu_1} \frac{Y[X_i]}{f(X_i)(\alpha^k - X_i)} = Y[X_j] + p_k X_j \frac{N_1(X_j)}{W_1(X_j)}.$$

Thus for $X_j = \alpha^k$,

$$Y[X_j] = r_k - p_k X_j \frac{N_1(X_j)}{W_1(X_j)}.$$

Both the numerator and the denominator are 0, so a "L'Hopital's rule" must be used. Using $N_1(x) = N_m(x)W_c(x)$ and $W_2(x) = W_m(x)W_c(x)$,

$$N_1'(X_j) = N_m(X_j)W_c'(X_j) + N_m'(X_j)W_c(X_j) = N_m(X_j)W_c'(X_j)$$
$$W_1'(X_j) = W_m(X_j)W_c'(X_j) + W_m'(X_j)W_c(X_j) = W_m(X_j)W_c'(X_j)$$

so $N_1'(X_j)/W_1'(X_j) = N_m(X_j)/W_m(X_j)$. The error value is thus

$$Y[X_j] = r_k - p_k X_j \frac{N_1'(X_j)}{W_1'(X_j)}. \tag{7.22}$$

Now consider the error values for the DB form of the WB equation, (7.18). It is shown in Exercise 7.6 that

$$g'(\alpha^{b+k})\alpha^{b(k+2-d)} p_k \alpha^k = -\tilde{C} = -\alpha^{b(d-2)} \prod_{i=0}^{d-3} (\alpha^{r+1} - \alpha^{i+1})$$

so that

$$\frac{N_2(\alpha^k)/W_2(\alpha^k)}{N_1(\alpha^k)/W_1(\alpha^k)} = -\tilde{C}.$$

It is shown in Exercise 7 that $f(\alpha^k)g(\alpha^{b+k}) = -\tilde{C}\alpha^{b(d-1-k)}$. From these two facts we can express the error locators for the DB form as

$$Y[X_k] = -\frac{N_2(\alpha^k)\alpha^{b(d-1-k)}}{W_2'(\alpha^k)g(\alpha^{b+k})} = -\frac{N_2(X_k)X_k^{-b}\alpha^{b(d-1)}}{W_2'(X_k)g(X_k\alpha^b)} \quad \text{(message location)} \tag{7.23}$$

$$Y_k[X_k] = r_k - \frac{N_2'(\alpha^k)\alpha^{b(d-2-k)}}{W_2'(\alpha^k)g'(\alpha^{b+k})} = r_k - \frac{N_2'(X_k)X_k^{-b}\alpha^{b(d-2)}}{W_2'(X_k)g'(X_k\alpha^b)} \quad \text{(check location).} \tag{7.24}$$

7.4 Methods of Solving the WB Key Equation

The key equation problem can be expressed as follows: Given a set of *points* $(x_i, y_i), i = 1, 2, \ldots, m$ over some field $\mathbb{F}$, the problem of finding polynomials $N(x)$ and $W(x)$ with $\deg(N(x)) < \deg(W(x))$ satisfying

$$N(x_i) = W(x_i)y_i, \qquad i = 1, 2, \ldots, m \tag{7.25}$$

is called a *rational interpolation problem*[1], since in the case that $W(x_i) \neq 0$ we have

$$y_i = \frac{N(x_i)}{W(x_i)}.$$

A solution to the rational interpolation problem provides a pair $[N(x), W(x)]$ satisfying (7.25).

We present two different algorithms for solving the rational interpolation problem. Either of these algorithms can be used with the data from either of the two forms of the key equations derived in Section 7.2.

7.4.1 Background: Modules

An additional algebraic structure is used in the algorithm below. In preparation for what follows, we pause to introduce this structure, which is a **module**. Modules are to rings what vector spaces are to fields. That is, they act like vector spaces, but they are built out of rings instead of fields. More formally, we have the following:

Definition 7.1 [61] A **module over a ring** R (or R-module) is a set M together with a binary operation (usually denoted as addition) and an operation of R on M called scalar multiplication with the following properties:

M1 M is an Abelian group under addition.

M2 For all $a \in R$ and $f, g \in M$, $a(f + g) = af + ag \in M$.

M3 For all $a, b \in R$ and all $f \in M$, $(a + b)f = af + bf \in M$.

M4 For all $a, b \in R$ and $f \in M$, $(ab)f = a(bf) \in M$.

M5 If 1 is the multiplicative identity in R, then $1f = f$ for all $f \in M$.

For $f, g \in M$ and $a, b \in R$, we say that $af + bg$ is an R**-linear combination** of f and g.

A **submodule** of a module M is a subset of M which is closed under addition and scalar multiplication by elements of R. □

Thus, the structure appears to be exactly that of a vector space (see Section 2.4). However, there are a few important distinctions, as the following example shows.

Example 7.3 Let $R = \mathbb{F}[x, y, z]$, where $\mathbb{F}$ is some field and R is the ring of polynomials in the three variables x, y, and z. Let

$$\mathbf{f}_1 = \begin{bmatrix} y \\ -x \\ 0 \end{bmatrix} \quad \mathbf{f}_2 = \begin{bmatrix} z \\ 0 \\ -x \end{bmatrix} \quad \mathbf{f}_3 = \begin{bmatrix} 0 & z & -y \end{bmatrix}.$$

[1] Strictly speaking, this is a weak rational interpolation problem, since in the form it is written it does not have to address concerns when $W(x_i) = 0$.

Let M be the module generated by R-linear combinations of $\mathbf{f}_1$, $\mathbf{f}_2$, and $\mathbf{f}_3$. We could denote this as $M = \langle \mathbf{f}_1, \mathbf{f}_2, \mathbf{f}_3 \rangle$. This set is minimal, in the sense that $\langle \mathbf{f}_i, \mathbf{f}_j \rangle$ does not generate M. However, they are linearly dependent, since

$$z\mathbf{f}_1 - y\mathbf{f}_2 + x\mathbf{f}_3 = \mathbf{0}.$$

We thus have a minimal generating set which is not linearly independent. This phenomenon cannot occur in any vector space. □

Definition 7.2 If a module M over a ring R has a generating set which is R-linearly independent, then M is said to be a **free module**. The number of generating elements is the **rank** of the module. □

Example 7.4 Let $R = \mathbb{F}[x]$ be a ring of polynomials and let $M = R^m$ be the module formed by columns of m polynomials. This is a free module. The standard basis for this module is

$$\mathbf{e}_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \quad \mathbf{e}_2 = \begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \quad \cdots \quad \mathbf{e}_m = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix}.$$

□

7.4.2 The Welch-Berlekamp Algorithm

In this section we describe a method of solving the rational interpolation problem which is structurally similar to the Berlekamp-Massey algorithm, in that it provides a sequence of solution pairs which are updated in the event that there is a discrepancy when a new point is considered. We are interested in a solution satisfying $\deg(N(x)) < \deg(W(x))$ and $\deg(W(x)) \leq m/2$.

Definition 7.3 The **rank** of a solution $[N(x), W(x)]$ is defined as

$$\text{rank}[N(x), W(x)] = \max\{2\deg(W(x)), 1 + 2\deg(N(x))\}.$$

□

We construct a solution to the rational interpolation problem of rank $\leq m$ and show that it is unique. By the definition of the rank, the degree of $N(x)$ is less than the degree of $W(x)$.

A polynomial expression for the interpolation problem (7.25) is useful. Let $P(x)$ be an interpolating polynomial such that $P(x_i) = y_i$, $i = 1, 2, \ldots, m$. For example, $P(x)$ could be the Lagrange interpolating polynomial,

$$P(x) = \sum_{i=1}^{m} y_i \frac{\prod_{k=1,k\neq i}^{m}(x - x_k)}{\prod_{k=1,k\neq i}^{m}(x_i - x_k)}.$$

By the evaluation homomorphism (see Section 5.3.1), the equation $N(x_i) = W(x_i)y_i$ is equivalent to

$$N(x) = W(x)P(x) \pmod{(x - x_i)}.$$

Since this is true for each point (x_i, y_i) and since the polynomials $(x - x_i)$, $i = 1, 2, \ldots, m$ are pairwise relatively prime, by the ring isomorphism introduced in conjunction with the Chinese remainder theorem we can write

$$N(x) = W(x)P(x) \pmod{\Pi(x)}, \tag{7.26}$$

where

$$\Pi(x) = \prod_{i=1}^{m}(x - x_i).$$

Definition 7.4 Suppose $[N(x), W(x)]$ is a solution to (7.25) and that $N(x)$ and $W(x)$ share a common factor $f(x)$, such that $N(x) = n(x)f(x)$ and $W(x) = w(x)f(x)$. If $[n(x), w(x)]$ is also a solution to (7.25), the solution $[N(x), W(x)]$ is said to be **reducible**. A solution which has no common factors of degree > 0 which may be factored out leaving a solution is said to be **irreducible**. □

It may be that an irreducible solution does have common factors of the form $(x - y_i)$, but which cannot be factored out while satisfying (7.26).

We begin with an statement regarding the existence of the solution.

Lemma 7.3 *There exists at least one irreducible solution to (7.26) with rank* $\leq m$.

Interestingly, this proof makes no use of any particular algorithm to construct a solution — it is purely an existence proof.

Proof Let $S = \{[N(x), W(x)] : \text{rank}(N, W) \leq m\}$ be the set of polynomials meeting the rank specification. For $[N(x), W(x)] \in S$ and $[M(x), V(x)] \in S$ and f a scalar value, define

$$\begin{aligned} [N(x), W(x)] + [M(x), V(x)] &= [N(x) + M(x), W(x) + V(x)] \\ f[N(x), W(x)] &= [fN(x), fW(x)]. \end{aligned} \tag{7.27}$$

We thus make S into a module over $\mathbb{F}[x]$. The dimension of S is $m + 1$, since a basis for the $N(x)$ component is

$$\{1, x, \ldots, x^{\lfloor (m-1)/2 \rfloor}\} \qquad (1 + \lfloor (m-1)/2 \rfloor \text{ dimensions})$$

and a basis for the $W(x)$ component is

$$\{1, x, \ldots, x^{\lfloor m/2 \rfloor}\} \qquad (1 + \lfloor m/2 \rfloor \text{ dimensions})$$

so the dimension of the Cartesian product is $1 + \lfloor (m-1)/2 \rfloor + 1 + \lfloor m/2 \rfloor = m + 1$.

By (7.26) and by the division algorithm for every $[N(x), W(x)] \in S$ there exists a quotient $Q(x)$ and remainder $R(x)$ with $\deg(R(x)) < m$ such that

$$N(x) - W(x)P(x) = Q(x)\Pi(x) + R(x).$$

Now define the mapping $E : S \to \{h \in \mathbb{F}[x] | \deg(h) < m)$ by

$$E([N(x), W(x)]) = R(x) \tag{7.28}$$

(the remainder polynomial). The dimension of the range of E is m. Thus, E is a linear mapping from a space of dimension $m + 1$ to a space of dimension m, so the dimension of its kernel is > 0. But the kernel is exactly the set of solutions to (7.26). There must therefore exist at least one solution to (7.26) with rank $\leq m$. □

The Welch-Berlekamp algorithm finds a rational interpolant of minimal rank by building successive interpolants for increasingly larger sets of points. First a minimal rank rational interpolant is found for the single point (x_1, y_1). This is used to construct a rational interpolant of minimal rank for the pair of points $\{(x_1, y_1), (x_2, y_2)\}$, and so on, until a minimal rank interpolant for the entire set of points $\{(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)\}$ is found.

Definition 7.5 We say that $[N(x), W(x)]$ satisfy the interpolation(k) problem if

$$N(x_i) = W(x_i)y_i \qquad i = 1, 2, \ldots, k. \tag{7.29}$$

□

The Welch-Berlekamp algorithm finds a sequence of solutions $[N^{[k]}, W^{[k]}]$ of minimum rank satisfying the interpolation(k) problem, for $k = 1, 2, \ldots, m$. We can also express the interpolation(k) problem as

$$N(x) = W(x)P_k(x) \quad (\text{mod } \Pi_k(x)),$$

where $\Pi_k(x) = \prod_{i=1}^{k}(x - x_i)$ and $P_k(x)$ is a polynomial that interpolates (at least) the first k points, $P(x_i) = y_i, i = 1, 2, \ldots, k$.

As with the Berlekamp-Massey algorithm, the Welch-Berlekamp algorithm propagates two solutions, using one of them in the update of the other. For the Welch-Berlekamp algorithm, the two sets of solution maintain the property that they are *complements* of each other, as defined here.

Definition 7.6 Let $[N(x), W(x)]$ and $[M(x), V(x)]$ be two solutions of interpolation(k) such that

$$\text{rank}[N(x), W(x)] + \text{rank}[M(x), V(x)] = 2k + 1$$

and

$$N(x)V(x) - M(x)W(x) = f\Pi(x)$$

for some scalar f. Then $[N(x), W(x)]$ and $[M(x), V(x)]$ are **complementary.** □

The key results to construct the algorithm are presented in Lemmas 7.4 and 7.6.

Lemma 7.4 *Let* $[N(x), W(x)]$ *be an irreducible solution to the interpolation*(k) *problem with rank* $\leq k$. *Then there exists at least one solution to the interpolation*(k) *problem which is a complement of* $[N(x), W(x)]$.

Proof Define the set similar to that in Lemma 7.3,

$$S = \{[M(x), V(x)] | \, \text{rank}[M(x), V(x)] \leq 2k + 1 - \text{rank}(N(x), W(x))\}.$$

It may be verified that, under the operations defined in (7.27), S is a module of dimension $2k+2-\text{rank}(N(x), W(x))$. Let K be the kernel of the mapping E defined in (7.28). Since $\dim(\mathcal{R}(E)) = k$ and $\dim(S) = 2k + 1 - \text{rank}(N(x), W(x))$, we must have $\dim(K) = \dim(S) - \dim(\mathcal{R}(E)) = k + 1 - \text{rank}(N(x), W(x))$. We now show that there is an element $[M(x), V(x)] \in K$ which is not of the form $[g(x)N(x), g(x)W(x)]$. Let

$$\begin{aligned} T = \{[g(x)N(x), g(x)W(x)] &: g \text{ is a polynomial such that} \\ &\text{rank}[g(x)N(x), g(x)W(x)] \leq 2k + 1 - \text{rank}[N(x), W(x)]\}. \end{aligned}$$

Then $T \subset S$. By the definitions we have

$$\text{rank}[g(x)N(x), g(x)W(x)] \leq 2k + 1 - \text{rank}[N(x), W(x)]$$

and

$$\text{rank}[g(x)N(x), g(x)W(x)] = 2\deg(g(x)) + \text{rank}[N(x), W(x)]$$

so that

$$\deg(g(x)) \leq k - \text{rank}[N(x), W(x)],$$

which therefore bounds the dimension of the subspace T. Since K has dimension $\geq k + 1 - \text{rank}[N(x), W(x)]$, there must be a point $[M(x), V(x)] \in K \setminus T$ such that

$$\text{rank}[N(x), W(x)] + \text{rank}[M(x), V(x)] \leq 2k + 1. \tag{7.30}$$

Since $[M(x), V(x)] \notin T$, $[M(x), V(x)]$ is not reducible to $[N(x), W(x)]$. We now need another lemma.

Lemma 7.5 *If* $[N(x), W(x)]$ *is an irreducible solution to the interpolation*(k) *problem and* $[M(x), V(x)]$ *is another solution such that* $\text{rank}[N(x), W(x)] + \text{rank}[M(x), V(x)] \leq 2k$, *then* $[M(x), V(x)]$ *can be reduced to* $[N(x), W(x)]$.

The proof of this lemma is developed in the exercises. Since we have argued that $[M(x), V(x)]$ is not reducible to $[N(x), W(x)]$, by this lemma we must have that the inequality in (7.30) must be satisfied with equality:

$$\text{rank}[N(x), W(x)] + \text{rank}[M(x), V(x)] = 2k + 1.$$

Therefore, one of $\text{rank}[N(x), W(x)]$ and $\text{rank}[M(x), V(x)]$ is even and the other is odd. So it must be that either

$$\begin{aligned}2k + 1 = \text{rank}[N(x), W(x)] + \text{rank}[M(x), V(x)] &= (1 + 2\deg(N(x)) + 2\deg(V(x))) \\ &> 2\deg(W(x)) + (1 + 2\deg(M(x)))\end{aligned}$$

(in which case $\deg(N(x)V(x)) = k$ and $\deg(W(x)M(x)) < k$) or

$$\begin{aligned}2k + 1 = \text{rank}[N(x), W(x)] + \text{rank}[M(x), V(x)] &= 2\deg(W(x)) + 1 + 2\deg(M(x)) \\ &> 1 + 2\deg(N(x)) + 2\deg(V(x))\end{aligned}$$

(in which case $\deg(M(x)W(x)) = k$ and $\deg(N(x)V(x)) < k$) so that, in either case,

$$\deg(N(x)V(x) - M(x)W(x)) = k.$$

Since $\Pi_k(x)$ for the interpolation(k) problem has degree k, it must be the case that $N(x)V(x) - M(x)W(x) = f\Pi_k(x)$ for some scalar f. □

Lemma 7.6 *If* $[N(x), W(x)]$ *is an irreducible solution to the interpolation*(k) *problem and* $[M(x), V(x)]$ *is one of its complements, then for any* $a, b \in \mathbb{F}$ *with* $n \neq 0$, $[bM(x) - aN(x), bV(x) - aW(x)]$ *is also one of its complements.*

Proof Since $[N(x), W(x)]$ and $[M(x), V(x)]$ are solutions, it must be that

$$N(x) \equiv W(x)P(x) \pmod{\Pi(x)} \qquad M(x) \equiv V(x)P(x) \pmod{\Pi(x)}.$$

Multiplying the first equation by a and the second equation by b and subtracting the first from the second yields

$$bM(x) - aN(x) \equiv (bV(x) - aW(x))P(x) \pmod{\Pi(x)}$$

so that $[bM(x) - aN(x), bV(x) - aW(x)]$ is a solution. We now show that it is complementary.

It is straightforward to show that $[bM(x) - aN(x), bV(x) - aW(x)]$ cannot be reduced to $[N(x), W(x)]$ (since $[M(x), V(x)]$ cannot be reduced to $[N(x), W(x)]$ by complementarity and Lemma 7.5). By lemma 7.5 we must therefore have

$$\text{rank}[N(x), W(x)] + \text{rank}[bM(x) - aN(x), bV(x) - aW(x)] = 2k + 1.$$

□

Lemma 7.5 also implies that there exists only one irreducible solution to the interpolation(k) problem with rank $\leq k$, and that this solution must have at least one complement.

We are now ready to state and prove the theorem describing the Welch-Berlekamp algorithm.

Theorem 7.7 *Suppose that* $[N^{[k]}, W^{[k]}]$ *and* $[M^{[k]}, V^{[k]}]$ *are two complementary solutions of the interpolation*(k) *problem. Suppose also that* $[N^{[k]}, W^{[k]}]$ *is the solution of lower rank. Let*

$$\begin{aligned} b_k &= N^{[k]}(x_{k+1}) - y_{k+1}W^{[k]}(x_{k+1}) \\ a_k &= M^{[k]}(x_{k+1}) - y_{k+1}V^{[k]}(x_{k+1}). \end{aligned} \tag{7.31}$$

(These are analogous to the discrepancies *of the Berlekamp-Massey algorithm.) If* $b_k = 0$ *(the discrepancy is 0, so no update is necessary) then*

$$[N^{[k]}(x), W^{[k]}(x)] \text{ and } [(x - x_{k+1})M^{[k]}(x), (x - x_{k+1})V^{[k]}(x)]$$

are two complementary solutions of the interpolation($k+1$) *problem, and* $[N^{[k]}(x), W^{[k]}(x)]$ *is the solution of lower rank.*

If $b_k \neq 0$ *(the discrepancy is not 0, so an update is necessary), then*

$$[(x-x_{k+1})N^{[k]}(x), (x-x_{k+1})W^{[k]}(x)] \text{ and } [b_k M^{[k]}(x) - a_k N^{[k]}(x), b_k N^{[k]} - a_k W^{[k]}(x)]$$

are two complementary solutions. The solution with lower rank is the solution to the interpolation($k+1$) *problem.*

Proof Since $[N^{[k]}(x), W^{[k]}(x)]$ and $[M^{[k]}(x), V^{[k]}(x)]$ are complementary,

$$\text{rank}[N^{[k]}(x), W^{[k]}(x)] + \text{rank}[M^{[k]}(x), V^{[k]}(x)] = 2k + 1$$

and

$$N^{[k]}(x)V^{[k]}(x) - M^{[k]}(x)W^{[k]}(x) = f\prod_{i=1}^{k}(x - x_i)$$

for some scalar f.

If $b_k = 0$: It is clear that $[N^{[k]}(x), W^{[k]}(x)]$ is a solution to the interpolation($k+1$) problem. For $[(x - x_{k+1})M^{[k]}(x), (x - x_{k+1})V^{[k]}(x)]$ we must have

$$(x - x_{k+1})M^{[k]}(x) \equiv (x - x_{k+1})V^{[k]}(x)P_{k+1}(x) \pmod{\Pi_{k+1}(x)},$$

which is clearly true since $M^{[k]}(x) \equiv V^{[k]}(x)P_k(x) \pmod{\Pi_k(x)}$. When $x = x_{k+1}$, then both sides are 0. Since $\text{rank}[(x-x_{k+1})M^{[k]}(x), (x-x_{k+1})V^{[k]}(x)] = \text{rank}[M^{[k]}(x), V^{[k]}(x)] +$ 2 we have

$$\begin{aligned} \text{rank}[N^{[k]}(x), W^{[k]}(x)] + \text{rank}[(x - x_{k+1})M^{[k]}(x), (x - x_{k+1})V^{[k]}(x)] &= 2k + 1 + 2 \\ &= 2(k+1) + 1. \end{aligned}$$

Furthermore,

$$(x - x_{k+1})N^{[k]}(x)V^{[k]}(x) - (x - x_{k+1})M^{[k]}(x)W^{[k]}(x) = f\prod_{i=1}^{k+1}(x - x_i)$$

so that $[N^{[k]}(x), W^{[k]}(x)]$ and $[(x-x_{k+1})M^{[k]}(x), (x-x_{k+1})V^{[k]}(x)]$ are complementary.

If $b_k \neq 0$: Since $[N^{[k]}(x), W^{[k]}(x)]$ satisfies

$$N^{[k]}(x) \equiv W^{[k]}(x)P_{k+1} \pmod{\Pi_k(x)} \tag{7.32}$$

it follows that

$$(x-x_{k+1})N^{[k]}(x) \equiv (x-x_{k+1})W^{[k]}(x)P_{k+1} \pmod{\Pi_{k+1}(x)}, \tag{7.33}$$

since it holds by (7.32) for the first k points and for the point (x_{k+1}, y_{k+1}), both sides of (7.33) are 0. Thus $[(x-x_{k+1})N^{[k]}(x), (x-x_{k+1})W^{[k]}(x)]$ is a solution to the interpolation$(k+1)$ problem.

That $[b_k M^{[k]}(x) - a_k N^{[k]}(x), b_k V^{[k]}(x) - a_k W^{[k]}(x)]$ is a solution to the interpolation(k) problem follows since

$$M^{[k]}(x) \equiv V^{[k]}(x)P_{k+1}(x) \pmod{\Pi_k(x)} \text{ and } N^{[k]}(x) \equiv W^{[k]}(x)P_{k+1}(x) \pmod{\Pi_k(x)}.$$

Multiplying the first of these equivalences by b_k and the second by a_k and subtracting gives the solution for the first k points.

To show that $[b_k M^{[k]}(x) - a_k N^{[k]}(x), b_k V^{[k]}(x) - a_k W^{[k]}(x)]$ is also a solution at the point (x_{k+1}, y_{k+1}), substitute a_k and b_k into the following to show that equality holds:

$$b_k M^{[k]}(x_{k+1}) - a_k N^{[k]}(x_{k+1}) = (b_k V^{[k]}(x_{k+1}) - a_k W^{[k]}(x_{k+1}))y_{k+1}.$$

It is clear from the inductive hypothesis that

$$\begin{aligned} &\deg[(x-x_{k+1})N^{[k]}(x), (x-x_{k+1})W^{[k]}(x)] \\ &\quad + \deg[b_k M^{[k]}(x) - a_k N^{[k]}(x), b_k V^{[k]}(x) - a_k W^{[k]}(x)] = 2(k+1)+1 \end{aligned}$$

and that

$$\begin{aligned} &(x-x_{k+1})N^{[k]}(x)(b_k V^{[k]}(x) - a_k W^{[k]}(x)) \\ &\quad - (b_k M^{[k]}(x) - a_k N^{[k]}(x))(x-x_{k+1})W^{[k]}(x) = f\prod_{i=1}^{k+1}(x-x_i) \end{aligned}$$

for some scalar f. Hence these two solutions are complementary. □

Based on this theorem, the Welch-Berlekamp rational interpolation function is shown in Algorithm 7.1.

Algorithm 7.1 Welch-Berlekamp Interpolation

Input: $(x_i, y_i), i = 1, \ldots, m$
Returns: $[N^{[m]}(x), W^{[m]}(x)]$ of minimal rank satisfying the interpolation problem.
Initialize:
$N^{[0]}(x) = 0$; $V^{[0]}(x) = 0$; $W^{[0]}(x) = 1$; $M^{[0]}(x) = 1$;
for $i = 0$ to $m-1$
  $b_i = N^{[i]}(x_{i+1}) - y_{i+1}W^{[i]}(x_{i+1})$ (compute discrepancy)
  if($b_i = 0$) then (no change in $[N, W]$ solution)
    $N^{[i+1]}(x) = N^{[i]}(x)$; $W^{[i+1]}(x) = W^{[i]}(x)$;
    $M^{[i+1]}(x) = (x-x_{i+1})M^{[i]}(x)$; $V^{[i+1]}(x) = (x-x_{i+1})V^{[i]}(x)$
  else (update to account for discrepancy)
    $a_i = M^{[i]}(x_{i+1}) - y_{i+1}V^{[i]}(x_{i+1})$; (compute other discrepancy)

$M^{[i+1]}(x) = (x - x_{i+1})N^{[i]}(x)$; $V^{[i+1]}(x) = (x - x_{i+1})W^{[i]}(x)$;
$N^{[i+1]}(x) = b_i M^{[i]}(x) - a_i N^{[i]}(x)$; $W^{[i+1]}(x) = b_i V^{[i]}(x) - a_i W^{[i+1]}(x)$;
if(rank[$N^{[i+1]}(x)$, $W^{[i+1]}(x)$] > rank[$M^{[i+1]}(x)$, $V^{[i+1]}(x)$]) (swap for minimal rank)
swap [$N^{[i+1]}(x)$, $W^{[i+1]}(x)$] $\leftrightarrow$ [$M^{[i+1]}(x)$, $V^{[i+1]}(x)$]
end (if)
end (else)
end (for)

Example 7.5 Using the code and data from Example 7.1, the Welch-Berlekamp algorithm operates as follows. Initially, $N^{[0]}(x) = 0$, $W^{[0]}(x) = 1$, $M^{[0]}(x) = 1$, and $V^{[0]}(x) = 0$.

i	b_i	a_i	$N^{[i]}(x)$	$W^{[i]}(x)$	$M^{[i]}(x)$	$V^{[i]}(x)$	swap?
0	α^7	1	α^7	1	0	$1+x$	no
1	α^{10}	α^{10}	α^2	$\alpha^{10}x$	$\alpha^8+\alpha^7x$	$\alpha+x$	no
2	α^2	α^{12}	$\alpha^{11}+\alpha^9x$	$\alpha^3+\alpha^{12}x$	$\alpha^4+\alpha^2x$	$\alpha^{12}+\alpha^{10}x^2$	no
3	α^{12}	α^{10}	$\alpha^{11}+\alpha^9x$	$\alpha^{13}+x+\alpha^7x^2$	$\alpha^{14}+x+\alpha^9x^2$	$\alpha^6+\alpha^{14}x+\alpha^{12}x^2$	no
4	α^4	α^{11}	$\alpha^4+\alpha^8x+\alpha^{13}x^2$	$\alpha^{13}+\alpha^5x+\alpha^9x^2$	$1+\alpha^4x+\alpha^9x^2$	$\alpha^2+\alpha^{11}x+\alpha^{12}x^2+\alpha^7x^3$	no
5	α^2	0	$\alpha^2+\alpha^6x+\alpha^{11}x^2$	$\alpha^4+\alpha^{13}x+\alpha^{14}x^2+\alpha^9x^3$	$\alpha^9+\alpha^{11}x+\alpha^{13}x^2+\alpha^{13}x^3$	$\alpha^3+\alpha^9x+\alpha^{12}x^2+\alpha^9x^3$	no

Using the Chien search, it can be determined that the error locator $W(x) = \alpha^4+\alpha^{13}x+\alpha^{14}x^2+\alpha^9x^3$ has roots at α^4, α^9 and α^{12}. For the error location $X = \alpha^4$ (a check location), using (7.22) the error value is found to be

$$Y[\alpha^4] = r_4 - p_4 X\frac{N'(X)}{W'(X)} = 0 - \alpha^9\alpha^4\frac{W'(\alpha^4)}{N'(\alpha^4)} = \alpha^5.$$

For the error location $X = \alpha^9$ (a message location), using (7.21) the error value is found to be

$$Y[X] = f(X)\frac{N(X)}{W'(X)} = \alpha^2\frac{N(\alpha^9)}{W'(\alpha^9)} = \alpha^7.$$

Similarly, for the error location $X = \alpha^{12}$, the error value is $Y[X] = \alpha^8$. The error polynomial is thus

$$E(x) = \alpha^5x^4 + \alpha^7x^9 + \alpha^8x^{12}$$

and all errors are corrected. □

Example 7.6 Using the same code and the data from Example 7.2, the Welch-Berlekamp algorithm operates as follows.

i	b_i	a_i	$N^{[i]}(x)$	$W^{[i]}(x)$	$M^{[i]}(x)$	$V^{[i]}(x)$	swap?
0	α^9	1	α^9	1	0	$1+x$	no
1	α^{12}	α^{12}	α^6	$\alpha^{12}x$	$\alpha^{10}+\alpha^9x$	$\alpha+x$	no
2	α^6	α^{14}	α^2+x	$\alpha^7+\alpha x$	$\alpha^8+\alpha^6x$	$\alpha^{14}x+\alpha^{12}x^2$	no
3	α^3	α^{14}	$\alpha^6+\alpha^4x$	$\alpha^6+\alpha^8x+x^2$	$\alpha^5+\alpha^6x+x^2$	$\alpha^{10}+\alpha^3x+\alpha x^2$	no
4	α^{14}	α^2	$\alpha^5+\alpha^9x+\alpha^{14}x^2$	$\alpha^{12}+\alpha^4x+\alpha^8x^2$	$\alpha^{10}+\alpha^{14}x+\alpha^4x^2$	$\alpha^{10}+\alpha^4x+\alpha^5x^2+x^3$	no
5	α^3	0	$\alpha^{13}+\alpha^2x+\alpha^7x^2$	$\alpha^{13}+\alpha^7x+\alpha^8x^2+\alpha^3x^3$	$\alpha^{10}+\alpha^{12}x+\alpha^{14}x^2+\alpha^{14}x^3$	$\alpha^2+\alpha^8x+\alpha^{11}x^2+\alpha^8x^3$	no

Using the Chien search, it can be determined that the error locator $W(x) = \alpha^{13} + \alpha^7 x + \alpha^8 x^2 + \alpha^3 x^3$ has roots at α^4, α^9 and α^{12}. For the error location $X = \alpha^4$ (a check location), the error value is found using (7.24) to be

$$Y[\alpha^4] = r_4 - \frac{N'(X)X^{-b}\alpha^{b(d-1)}}{W'(X)g(X\alpha^b)} = 0 - \frac{N'(X)X^{-b}\alpha^{2(6)}}{W'(X)g(X\alpha^2)} = \alpha^5$$

For the error location $X = \alpha^9$ (a message location), the error value is found using (7.23) to be

$$Y[X] = \frac{N(X)X^{-b}\alpha^{b(d-1)}}{W'(X)g(X\alpha^b)} = \frac{N(\alpha^9)\alpha^{-18}\alpha^{2(6)}}{W'(\alpha^9)g(\alpha^9\alpha^2)} = \alpha^7.$$

Similarly, for the error location $X = \alpha^{12}$, the error value is $Y[X] = \alpha^8$. The error polynomial is thus

$$E(x) = \alpha^5 x^4 + \alpha^7 x^9 + \alpha^8 x^{12}$$

and all errors are corrected. □

7.4.3 A Modular Approach to the Solution of the Welch-Berlekamp Key Equation

In this section we present an alternative approach to the solution of the Welch-Berlekamp key equation, due to [63], which makes use of modules and the concept of exact sequences. This solution is interesting theoretically because it introduces several important and powerful algebraic concepts. In addition, as shown in [63], it is suitable for representation in a parallel pipelined form for fast decoding.

The problem, again, is to find polynomials $N(x)$ and $W(x)$ satisfying the rational interpolation problem

$$N(x_i) = W(x_i)y_i, \qquad i = 1, 2, \ldots, m \tag{7.34}$$

with $\deg(N(x)) < \deg(W(x))$ and $\deg(W(x))$ minimal. We observe that the set of solutions to (7.34), without regard to the degree requirements, can be expressed more abstractly as the kernel of the homomorphism

$$\phi_i : \mathbb{F}[x] \rightarrow \mathbb{F} \text{ defined by } \phi_i(w(x), n(x)) = n(x_i) - w(x_i)y_i. \tag{7.35}$$

Any pair of polynomials $(w(x), n(x))$ in the kernel of ϕ_i yields a solution to (7.34) at x_i.

By the Chinese remainder theorem, the equations in (7.34) can be collectively expressed as a congruence

$$N(x) \equiv W(x)P(x) \pmod{\Pi(x)} \tag{7.36}$$

where $P(x)$ is any interpolating polynomial, $P(x_i) = y_i, i = 1, 2, \ldots, m$, and $\Pi(x) = \prod_{i=1}^{m}(x - x_i)$. Our approach is to develop a linear space of solutions $[w(x), n(x)]$ without regard to the minimality of degree, then to establish a means of selecting a point out of that space with minimal degree. The approach is made somewhat more general by defining a module as follows.

Definition 7.7 For fixed $D(x)$ and $G(x)$, let $\mathcal{M}$ be the module consisting of all pairs $[w(x), n(x)]$ satisfying

$$G(x)n(x) + D(x)w(x) \equiv 0 \pmod{\Pi(x)}. \tag{7.37}$$

□

The module $\mathcal{M}$ corresponds to the space of solutions of (7.36) when $G(x) = 1$ and $D(x) = -P(x)$.

Lemma 7.8 *$\mathcal{M}$ is a free $\mathbb{F}[x]$-module of rank 2 having a basis vectors*

$$[\Pi(x)\delta(x), \Pi(x)\gamma(x)] \quad [-G(x)/\lambda(x), D(x)/\lambda(x)],$$

where

$$\lambda(x) = \mathrm{GCD}(G(x), D(x)), \qquad (\lambda(x), \Pi(x)) = 1 \tag{7.38}$$

and

$$\delta(x)(D(x)/\lambda(x)) + \gamma(x)(G(x)/\lambda(x)) = 1. \tag{7.39}$$

Proof It is straightforward to verify that (7.37) holds for

$$[w(x), n(x)] = [-G(x)/\lambda(x), D(x)/\lambda(x)]$$

and for

$$[w(x), n(x)] = [\Pi(x)\delta(x), \Pi(x)\gamma(x)],$$

so these bases are in $\mathcal{M}$.

We must show that an arbitrary element $[w(x), n(x)] \in \mathcal{M}$ can be expressed as a $\mathbb{F}[x]$-linear combination of these basis vectors. Since $[w(x), n(x)] \in \mathcal{M}$, $G(x)n(x) + D(x)w(x) \equiv 0 \pmod{\Pi(x)}$, or

$$[w(x), n(x)] \begin{bmatrix} D(x) \\ G(x) \end{bmatrix} \equiv 0 \pmod{\Pi(x)}. \tag{7.40}$$

Consider the matrix

$$A = \begin{bmatrix} \delta(x) & \gamma(x) \\ -G(x)/\lambda(x) & D(x)/\lambda(x) \end{bmatrix}.$$

By (7.39), $\det(A) = 1$, so that A^{-1} is also a polynomial matrix. There therefore exist polynomials $n^*(x)$ and $w^*(x)$ such that

$$[w(x), n(x)] = [w^*(x), n^*(x)]A. \tag{7.41}$$

Substituting this into (7.40),

$$[w^*(x), n^*(x)] \begin{bmatrix} \delta(x) & \gamma(x) \\ -G(x)/\lambda(x) & D(x)/\lambda(x) \end{bmatrix} \begin{bmatrix} D(x) \\ G(x) \end{bmatrix} \equiv 0 \pmod{\Pi(x)},$$

or, using (7.39) again,

$$[w^*(x), n^*(x)] \begin{bmatrix} \lambda(x) \\ 0 \end{bmatrix} \equiv 0 \pmod{\Pi(x)}.$$

Thus $w^*(x)\lambda(x) \equiv 0 \pmod{\Pi(x)}$, so that by (7.38) $\Pi(x) \mid w^*(x)$. Thus there is a polynomial $\tilde{w}(x)$ such that $w^*(x) = \Pi(x)\tilde{w}(x)$. Equation (7.41) can thus be written as

$$[w(x), n(x)] = [\tilde{w}(x), n^*(x)] \begin{bmatrix} \delta(x)\Pi(x) & \gamma(x)\Pi(x) \\ -G(x)/\lambda(x) & D(x)/\lambda(x) \end{bmatrix} \triangleq [\tilde{w}(x), n^*(x)]\Psi, \tag{7.42}$$

indicating that an arbitrary element $[n(x), w(x)]$ can be expressed as a $\mathbb{F}[x]$-linear combination of the basis vectors.

□

It is convenient to represent the set of basis vectors for $\mathcal{M}$ as *rows* of a matrix. We use Ψ to represent a basis matrix. Then any point $[w(x), n(x)]$ can be represented as

$$[w(x), n(x)] = [a(x), b(x)]\Psi$$

for some $[a(x), b(x)] \in \mathbb{F}[x]^2$.

Lemma 7.9 *For any matrix Ψ whose rows form a basis of $\mathcal{M}$,* $\det(\Psi) = \alpha\Pi(x)$*, where* $\alpha \in \mathbb{F}$ *is not zero.*

Conversely, if the rows of $\Phi \in \mathbb{F}[x]^{2\times 2}$ are in $\mathcal{M}$ and $\det \Phi = \alpha\Pi(x)$ *for some nonzero* $\alpha \in \mathbb{F}$*, then Φ is a basis matrix for $\mathcal{M}$.*

Proof For the matrix Ψ in (7.42),

$$\det(\Psi) = \frac{\Pi}{\lambda}[D\delta + G\gamma] = \Pi(x)$$

by (7.39). Let Ψ' be any other basis matrix. Then there must be a matrix T such that

$$\Psi' = T\Psi \qquad \Psi = T^{-1}\Psi'.$$

Since T is invertible, it must be that $\det(T)$ is a unit in $\mathbb{F}[x]$, that is, $\det(T) \in \mathbb{F}$. Thus $\det(\Psi') = \det(T\Psi) = \det(T)\det(\Psi) = \alpha\Pi(x)$.

To prove the converse statement, for a matrix Φ whose rows are in $\mathcal{M}$, there must be a matrix T such that $\Phi = T\Psi$. Then

$$\alpha\Pi(x) = \det(\Phi) = \det(T)\det(\Psi) = \det(T)\Pi(x)$$

so that $\det(T) = \alpha$, which is a unit. Thus T is invertible and $\Psi = T^{-1}\Phi$. By this we observe that Φ must be a basis matrix. □

Let us return to the question of finding the intersection of the modules which are the kernels of ϕ_i defined in (7.35). To this end, we introduce the notion of an *exact* sequence.

Definition 7.8 Let R be a ring [such as $\mathbb{F}[x]$] and let $\mathfrak{N}$, $\mathfrak{B}$ and $\mathfrak{R}$ be R-modules. Let f and g be module homomorphisms, $f : \mathfrak{N} \to \mathfrak{B}$ and $g : \mathfrak{B} \to \mathfrak{R}$. The sequence

$$\mathfrak{N} \xrightarrow{f} \mathfrak{B} \xrightarrow{g} \mathfrak{R}$$

is said to be **exact** if $\text{im}(f) = \ker(g)$. □

As an example, let $\mathcal{M}_i$ be the module of rank two that is the kernel of ϕ_i and let $\Psi_i(x)$ be a 2×2 basis matrix for the $\mathcal{M}_i$. Define $\psi_i(w(x), n(x)) = [w(x), n(x)]\Psi_i(x)$. Then

$$\mathbb{F}[x]^2 \xrightarrow{\psi_i} \mathbb{F}[x]^2 \xrightarrow{\phi_i} \mathbb{F}$$

is an exact sequence.

The main idea in the intersection-finding algorithm is embodied in the following lemma.

Lemma 7.10 *Let*

$$\mathfrak{R} \xrightarrow{\psi} \mathfrak{B} \xrightarrow{\phi_1} \mathfrak{R}$$

be exact and let $\phi_2 : \mathfrak{B} \to \mathfrak{R}'$ be another module homomorphism. Then

$$\ker(\phi_1) \cap \ker(\phi_2) = \psi(\ker(\phi_2 \circ \psi)).$$

Proof Consider the function $\phi_2 \circ \psi : \mathfrak{N} \to \mathfrak{R}'$. Since $\ker(\phi_2 \circ \psi) \subset \mathfrak{N}$, it follows that $\psi(\ker(\phi_2 \circ \psi)) \subset \text{im}(\psi)$, which by exactness is equal to $\ker(\phi_1)$. Thus $\psi(\ker(\phi_2 \circ \psi) \subset \ker(\phi_1)$.

Furthermore, by definition, $\phi_2(\psi(\ker(\phi_2 \circ \psi))) = \{0\}$, that is, $\psi(\ker(\phi_2 \circ \psi)) \in \ker(\phi_2)$. Combining these we see that

$$\psi(\ker(\phi_2 \circ \psi)) \subset \ker(\phi_1) \cap \ker(\phi_2). \tag{7.43}$$

By definition,

$$\phi_2 \circ \psi(\psi^{-1}(\ker(\phi_1) \cap \ker(\phi_2))) = \{0\}$$

so that

$$\psi^{-1}(\ker(\phi_1) \cap \ker(\phi_2)) \subset \ker(\phi_2 \circ \psi)$$

or, applying ψ to both sides,

$$\ker(\phi_1) \cap \ker(\phi_2) \subset \psi(\ker(\phi_2 \circ \psi)). \tag{7.44}$$

Combining (7.43) and (7.44) we see that

$$\ker(\phi_1) \cap \ker(\phi_2) = \psi(\ker(\phi_2 \circ \psi)).$$

□

This lemma extends immediately: If $\mathfrak{N} \xrightarrow{\psi} \mathfrak{B} \xrightarrow{\phi_1} \mathfrak{R}$ is exact and $\phi_i : \mathfrak{B} \to \mathfrak{R}', i = 1, 2, \ldots, m$ are module homomorphisms, then

$$\ker(\phi_1) \cap \ker(\phi_2) \cap \cdots \cap \ker(\phi_m) = \psi(\ker(\phi_2 \circ \psi) \cap \ker(\phi_3 \circ \psi) \cap \cdots \cap \ker(\phi_m \circ \psi)).$$

Consider the solution of the congruence

$$G_i n(x) + D_i w(x) \equiv 0 \pmod{x - x_i}, i = 1, 2, \ldots, m \tag{7.45}$$

for given G_i and D_i. Define the homomorphisms for this problem as

$$\phi_i(w(x), n(x)) = G_i n(x_i) + D_i w(x_i) = [w(x_i), n(x_i)] \begin{bmatrix} D_i \\ G_i \end{bmatrix}, \tag{7.46}$$

and

$$\psi_i(w(x), n(x)) = \begin{cases} [w(x), n(x)] \begin{bmatrix} -G_i & D_i \\ (x - x_i) & 0 \end{bmatrix} & \text{if } D_i \neq 0 \\ [w(x), n(x)] \begin{bmatrix} -G_i & 0 \\ 0 & (x - x_i) \end{bmatrix} & \text{if } D_i = 0 \text{ and } G_i \neq 0. \end{cases} \tag{7.47}$$

The module of solutions of (7.45) for a particular value of i is denoted as $\mathcal{M}_i$.

Lemma 7.11 *For ϕ_i and ψ_i as just defined, the sequence*

$$\mathbb{F}[x]^2 \xrightarrow{\psi_i} \mathbb{F}[x]^2 \xrightarrow{\phi_i} \mathbb{F}$$

is exact.

Proof We consider the case that $D_i \neq 0$. By substitution of the elements $(w(x), n(x))$ from each row of the matrix defined in (7.47), $\Psi_i(x) = \begin{bmatrix} -G_i & D_i \\ (x - x_i) & 0 \end{bmatrix}$, into (7.46) it is clear that the rows are two elements of $\ker(\phi_i)$. Also, the determinant of the matrix is equal to $-D_i(x - x_i)$. Thus by Lemma 7.9, $\Psi_i(x)$ is a basis matrix for the module $\mathcal{M}_i = \ker(\phi_i)$.

The case when $D_i = 0$ follows similarly, making use of the fact that $D_i = 0$. □

Each homomorphism ϕ_i can be written as

$$\phi_i(w(x), n(x)) = [w(x_i), n(x_i)] \begin{bmatrix} D_i \\ G_i \end{bmatrix}$$

so that (D_i, G_i) characterizes the homomorphism. Let $\phi_i^{[0]} = \phi_i, i = 1, 2, \ldots, m$ represent the initial set of homomorphisms, with initial parameters $(D_i^{[0]}, G_i^{[0]}) = (D_i, G_i)$. The superscript indicates the iteration number.

In the first step of the algorithm, a homomorphism ϕ_{j_1} is chosen from the set

$$\{\phi_1^{[0]}, \phi_2^{[0]}, \ldots, \phi_m^{[0]}\}$$

such that $D_i^{[0]} \neq 0$. (The second subscript in ϕ_{j_1} also denotes the iteration number.) The homomorphism ψ_{j_1} of (7.47), described by the matrix $\Psi_{j_1}^{[0]}(y)$, is formed,

$$\Psi_{j_1}^{[1]}(y) = \begin{bmatrix} -G_{j_1}^{[0]} & D_{j_1}^{[0]} \\ (x - x_{j_1}) & 0 \end{bmatrix}.$$

By Lemma 7.11, the sequence $\mathbb{F}[x]^2 \xrightarrow{\psi_{j_1}} \mathbb{F}[x]^2 \xrightarrow{\phi_{j_1}} \mathbb{F}$ is exact. Now define

$$\phi_i^{[1]} = \phi_i^{[0]} \circ \psi_{j_1}. \tag{7.48}$$

Then from Lemma 7.10

$$\mathcal{M} = \psi_{j_1}(\ker(\phi_1^{[1]}) \cap \ker(\phi_2^{[1]}) \cap \cdots \cap \ker(\phi_m^{[1]})).$$

The interpolation problem is therefore equivalent to finding a basis for

$$\ker(\phi_1^{[1]}) \cap \ker(\phi_2^{[1]}) \cap \cdots \cap \ker(\phi_m^{[1]}). \tag{7.49}$$

We can write

$$\phi_i^{[1]}(w(x), n(x)) = [w(x_i), n(x_i)] \begin{bmatrix} -G_{j_1}^{[0]} & D_{j_1}^{[0]} \\ (x_i - x_{j_1}) & 0 \end{bmatrix} \begin{bmatrix} D_i^{[0]} \\ G_i^{[0]} \end{bmatrix} \tag{7.50}$$

$$= [w(x_i), n(x_i)] \Psi_{j_1}^{[1]}(x_i) \begin{bmatrix} D_i^{[0]} \\ G_i^{[0]} \end{bmatrix}$$

$$= [w(x_i), n(x_i)] \begin{bmatrix} D_{j_1}^{[0]} G_i^{[0]} - D_i^{[0]} G_{j_1}^{[0]} \\ D_i^{[0]}(x_i - x_{j_1}) \end{bmatrix} \triangleq [w(x_i), n(x_i)] \begin{bmatrix} D_i^{[1]} \\ G_i^{[1]} \end{bmatrix} \tag{7.51}$$

Thus the homomorphisms $\phi_i^{[1]}$ are defined by $(D_i^{[1]}, G_i^{[1]})$, where

$$\begin{bmatrix} D_i^{[1]} \\ G_i^{[1]} \end{bmatrix} = \Psi_{j_1}^{[1]}(x_i) \begin{bmatrix} D_i^{[0]} \\ G_i^{[0]} \end{bmatrix}. \tag{7.52}$$

When $i = j_1$ we have

$$\begin{bmatrix} D_{j_1}^{[1]} \\ G_{j_1}^{[1]} \end{bmatrix} = \begin{bmatrix} -G_{j_1}^{[0]} & D_{j_1}^{[0]} \\ (x_{j_1} - x_{j_1}) & 0 \end{bmatrix} \begin{bmatrix} D_{j_1}^{[0]} \\ G_{j_1}^{[0]} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \tag{7.53}$$

so that $\ker(\phi_{j_1}^{[1]}) = \mathbb{F}[x]^2$. (That is, $\phi_{j_1}^{[1]}$ maps all pairs of polynomials to 0.) This means that among the modules listed in (7.49) there is a trivial module, reducing the number of nontrivial modules to deal with by one. It also means that the equation is satisfied for index $i = j_1$.

At the next step, a single homomorphism $\phi_{j_2}^{[1]}$ is chosen from the set $\{\phi_1^{[1]}, \phi_2^{[1]}, \ldots, \phi_m^{[1]}\}$ such that $D_{j_2}^{[1]}$, defined by (7.52), is nonzero. The corresponding homomorphism ψ_{j_2} is also formed via its matrix representation,

$$\Psi_{j_2}^{[2]}(y) = \begin{bmatrix} -G_{j_2}^{[1]} & D_{j_2}^{[1]} \\ (x - x_{j_2}) & 0 \end{bmatrix}.$$

This choice of j_2 gives rise to a new set of homomorphisms $\{\phi_i^{[2]}\}$ by

$$\phi_i^{[2]} = \phi_i^{[1]} \circ \psi_{j_2}, \qquad i = 1, 2, \ldots, m.$$

Then from (7.48),

$$\phi_i^{[2]} = \phi_i^{[0]} \circ \psi_{j_1} \circ \psi_{j_2}, \qquad i = 1, 2, \ldots, m$$

so that

$$\phi_i^{[2]}(w(x), n(x)) = [w(x_i), n(x_i)]\Psi_{j_2}^{[2]}(x_i)\Psi_{j_1}^{[1]}(x_i)\begin{bmatrix} D_i^{[0]} \\ G_i^{[0]} \end{bmatrix} = [w(x_i), n(x_i)]\Psi^{[2]}\begin{bmatrix} D_i^{[1]} \\ G_i^{[1]} \end{bmatrix}$$

$$\triangleq [w(x_i), n(x_i)]\begin{bmatrix} D_i^{[2]} \\ G_i^{[2]} \end{bmatrix}.$$

From (7.53) it follows immediately that $G_{j_1}^{[2]} = D_{j_1}^{[2]} = 0$. It is also straightforward to show that

$$\begin{bmatrix} D_{j_2}^{[2]} \\ G_{j_2}^{[2]} \end{bmatrix} = \Psi_{j_2}^{[2]}(x_{j_2})\Psi_{j_1}^{[1]}(x_{j_2})\begin{bmatrix} D_{j_2}^{[0]} \\ G_{j_2}^{[0]} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix},$$

Thus $\ker(\phi_{j_1}^{[2]}) = \ker(\phi_{j_2}^{[2]}) = \mathbb{F}[x]^2$, and the equations corresponding to the indices j_1 and j_2 are satisfied. The number of nontrivial modules under consideration is again reduced by one. Furthermore, by Lemma 7.10,

$$\mathcal{M} = \psi_{j_1} \circ \psi_{j_2}(\ker(\phi_1^{[2]}) \cap \ker(\phi_2^{[2]}) \cap \cdots \cap \ker(\phi_m^{[2]})).$$

We continue iterating in this manner until iteration number $l \leq m$ at which the set of $D_i^{[l]} = 0$ for all $i = 1, 2, \ldots, m$. Now consider the set of homomorphisms $\{\phi_1^{[l]}, \phi_2^{[l]}, \ldots, \phi_m^{[l]}\}$. Define

$$\mathcal{M}^{[l]} = \ker(\phi_1^{[l]}) \cap \ker(\phi_2^{[l]}) \cap \cdots \cap \ker(\phi_m^{[l]}) \subset \mathbb{F}[x]^2$$

and let

$$\psi = \psi_{j_1} \circ \psi_{j_2} \circ \cdots \circ \psi_{j_l}.$$

By Lemma 7.10,

$$\mathcal{M} = \psi(\mathcal{M}^{[l]}).$$

By construction, $D_i^{[l]} = 0$ for $i = 1, 2, \ldots, m$. Since

$$\phi_i^{[l]}(w(x), n(x)) = G_i^{[l]}n(x_i) - D_i^{[l]}w(x_i), \quad i = 1, 2, \ldots, m,$$

the pair $(1, 0) \in \mathcal{M}^{[l]}$, which implies that $\psi(1, 0) \in \mathcal{M}$. We have thus found a solution to the interpolation problem! It remains, however, to establish that it satisfies the degree requirements. This is done by keeping track of how the homomorphism ψ is built.

We can write $\psi(w(x), n(x))$ as

$$\psi(w(x), n(x)) = (w(x), n(x)) \begin{bmatrix} -G_{j_l}^{[l-1]} & D_{j_l}^{[l-1]} \\ (x - x_{j_l}) & 0 \end{bmatrix} \begin{bmatrix} -G_{j_{l-1}}^{[l-2]} & D_{j_{l-1}}^{[l-2]} \\ (x - x_{j_{l-1}}) & 0 \end{bmatrix} \cdots \begin{bmatrix} -G_{j_1}^{[0]} & D_{j_1}^{[0]} \\ (x - x_{j_1}) & 0 \end{bmatrix}.$$

Let

$$\Psi^1(x) = \Psi_{j_1}^{[1]}(x) = \begin{bmatrix} -G_{j_1} & D_{j_1} \\ (x - x_{j_1}) & 0 \end{bmatrix} \tag{7.54}$$

and

$$\Psi^i(x) = \prod_{p=1}^{i} \Psi_{j_p}^{[p]}(x) = \Psi_{j_i}^{[i]}(x)\Psi^{i-1}(x), \qquad i = 2, 3, \ldots, l. \tag{7.55}$$

Use $\Psi(x) = \Psi^{[l]}(x)$. Then $\psi(w(x), n(x)) = [w(x), n(x)]\Psi(x)$. Let us write this as

$$\Psi(x) = \begin{bmatrix} \Psi_{1,1}(x) & \Psi_{1,2}(x) \\ \Psi_{2,1}(x) & \Psi_{2,2}(x) \end{bmatrix} = \begin{bmatrix} \psi(1,0) \\ \psi(0,1) \end{bmatrix}.$$

Our culminating lemma indicates that this construction produces the desired answer.

Lemma 7.12 *Let* $[\Psi_{1,1}(x), \Psi_{2,2}(x)]$ *be the image of* $(1, 0)$ *under the mapping* ψ. *That is,*

$$[\Psi_{1,1}(x), \Psi_{1,2}(x)] = \psi(1, 0) = \psi_{j_1} \circ \psi_{j,2} \circ \cdots \circ \psi_{j_l}(1, 0).$$

Then $[\Psi_{1,1}(x), \Psi_{1,2}(x)]$ *is a solution to the interpolation problem (7.34) and, furthermore,*

$$\deg(\Psi_{1,1}) < m/2 \qquad \deg(\Psi_{1,2}) \leq m/2.$$

Proof The fact that $(\Psi_{1,1}, \Psi_{1,2})$ satisfies the interpolation equations has been shown by their construction. It remains to be shown that the degree requirements are met. The following conditions are established:

$$\deg(\Psi_{1,1}^{[i]}) \leq \left\lfloor \frac{i}{2} \right\rfloor \qquad \deg(\Psi_{1,2}^{[i]}) \leq \left\lfloor \frac{i-1}{2} \right\rfloor$$
$$\deg(\Psi_{2,1}^{[i]}) \leq \left\lceil \frac{i}{2} \right\rceil \qquad \deg(\Psi_{2,2}^{[i]}) \leq \left\lceil \frac{i-1}{2} \right\rceil$$

This is immediate when $i = 1$. The remainder are proved by induction. For example,

$$\deg(\Psi_{2,2}^{[i+1]}) = 1 + \deg(\Psi_{1,2}^{[i]}) \leq 1 + \lfloor (i-1)/2 \rfloor = \lceil i/2 \rceil.$$

This and the other inductive steps make use of the facts that

$$1 + \lfloor i/2 \rfloor = \lceil (i+1)/2 \rceil \qquad \lfloor (i+1)/2 \rfloor = \lceil i/2 \rceil.$$

□

The solution to the WB equation can thus be computed using the following algorithm.

Algorithm 7.2 Welch-Berlekamp Interpolation, Modular Method, v. 1

1 **Input:** Points $(x_i, y_i), i = 1, 2, \ldots, m$.
Returns: $N(x)$ and $W(x)$ satisfying $N(x_i) = W(x_i)y_i, i = 1, 2, \ldots, m$.
2 **Initialization:** Set $G_i^{[0]} = 1$, $D_i^{[0]} = -y_i, i = 1, 2, \ldots, m$, $\Psi^{[0]} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$.
3 for $s = 1$ to m
4 Choose j_s such that $D_{j_s}^{[s-1]} \neq 0$. If no such j_s, **break**.
5 for $k = 1$ to m (may be done in parallel)
6 $\begin{bmatrix} D_k^{[s]} \\ G_k^{[s]} \end{bmatrix} = \begin{bmatrix} -G_{j_s}^{[s-1]} & D_{j_s}^{[s-1]} \\ (x_k - x_{j_s}) & 0 \end{bmatrix} \begin{bmatrix} D_k^{[s-1]} \\ G_k^{[s-1]} \end{bmatrix}$
7 end (for)
8 $\begin{bmatrix} \Psi_{1,1}^{[s]} & \Psi_{1,2}^{[s]} \\ \Psi_{2,1}^{[s]} & \Psi_{2,2}^{[s]} \end{bmatrix} = \begin{bmatrix} -G_{j_s}^{[s-1]} & D_{j_s}^{[s-1]} \\ (x - x_{j_s}) & 0 \end{bmatrix} \begin{bmatrix} \Psi_{1,1}^{[s-1]} & \Psi_{1,2}^{[s-1]} \\ \Psi_{2,1}^{[s-1]} & \Psi_{2,2}^{[s-1]} \end{bmatrix}$
9 end (for)
10 $W(x) = \Psi_{1,1}^{[s]}$, $N(x) = \Psi_{1,2}^{[s]}$.

As pointed out in the comment on line 5, computation of the new parameters G and D in line 6 may be done in parallel.

Example 7.7 For the (x_i, y_i) data of Example 7.1, the execution of the algorithm is as follows (using 1-based indexing):

$s = 1$: $\{D_i^{[0]}\} = \{\alpha^7, \alpha^6, 0, \alpha^{12}, 0, \alpha^{13}\}$. Choose: $j_1 = 1$. $\Psi^{[1]}(x) = \begin{bmatrix} 1 & \alpha^7 \\ 1+x & 0 \end{bmatrix}$.

$s = 2$: $\{D_i^{[1]}\} = \{0, \alpha^{10}, \alpha^7, \alpha^2, \alpha^7, \alpha^5\}$. Choose: $j_2 = 2$. $\Psi^{[2]}(x) = \begin{bmatrix} \alpha^{10}x & \alpha^2 \\ \alpha + x & \alpha^8 + \alpha^7 x \end{bmatrix}$

$s = 3$: $\{D_i^{[2]}\} = \{0, 0, \alpha^2, \alpha^4, \alpha^2, \alpha^{14}\}$. Choose: $j_3 = 3$. $\Psi^{[3]}(x) = \begin{bmatrix} \alpha^3 + \alpha^{12}x & \alpha^{11} + \alpha^9 x \\ \alpha^{12}x + \alpha^{10}x^2 & \alpha^4 + \alpha^2 x \end{bmatrix}$

$s = 4$: $\{D_i^{[3]}\} = \{0, 0, 0, \alpha^{12}, \alpha^4, \alpha^2\}$. Choose: $j_4 = 4$.

$$\Psi^{[4]}(x) = \begin{bmatrix} \alpha^{13} + x + \alpha^7 x^2 & \alpha^{11} + \alpha^9 x \\ \alpha^6 + \alpha^{14}x + \alpha^{12}x^2 & \alpha^{14} + x + \alpha^9 x^2 \end{bmatrix}$$

$s = 5$: $\{D_i^{[4]}\} = \{0, 0, 0, 0, \alpha^4, 0\}$. Choose: $j_5 = 5$.

$$\Psi^{[5]}(x) = \begin{bmatrix} \alpha^{13} + \alpha^5 x + \alpha^9 x^2 & \alpha^4 + \alpha^8 x + \alpha^{13}x^2 \\ \alpha^2 + \alpha^{11}x + \alpha^{12}x^2 + \alpha^7 x^3 & 1 + \alpha^4 x + \alpha^9 x^2 \end{bmatrix}$$

$s = 6$: $\{D_i^{[5]}\} = \{0, 0, 0, 0, 0, \alpha^2\}$. Choose: $j_6 = 6$.

$$\Psi^{[6]}(x) = \begin{bmatrix} \alpha^4 + \alpha^{13}x + \alpha^{14}x^2 + \alpha^9 x^3 & \alpha^2 + \alpha^6 x + \alpha^{11}x^2 \\ \alpha^3 + \alpha^9 x + \alpha^{12}x^2 + \alpha^9 x^3 & \alpha^9 + \alpha^{11}x + \alpha^{13}x^2 + \alpha^{13}x^3 \end{bmatrix}$$

At the end of the iteration we take

$$W(x) = \Psi_{1,1}(x) = \alpha^4 + \alpha^{13}x + \alpha^{14}x^2 + \alpha^9 x^3 \qquad N(x) = \Psi_{1,2}(x) = \alpha^2 + \alpha^6 x + \alpha^{11}x^2$$

These are the same $(N(x), W(x))$ as were found in Example 7.5. The roots of $W(x)$ and the error values can be determined as before. □

At the end of the algorithm, we have

$$\Psi(x) = \Psi^{[l]}(x)$$

This algorithm produces a $(D_i^{[s]}, G_i^{[s]})$ pairs satisfying

$$\begin{bmatrix} D_{j_u}^{[s]} \\ G_{j_u}^{[s]} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \qquad \text{for } u \le s \le l.$$

Thus $\ker(\phi_{j_s}^{[l]}) = \mathbb{F}[x]^2$ for $s = 1, 2, \ldots, l$. By Lemma 7.10,

$$\ker(\phi_{j_1}) \cap \ker(\phi_{j_2}) \cap \cdots \cap \ker(\phi_{j_l}) = \psi(\mathbb{F}[x]^2). \tag{7.56}$$

Now consider $\psi(0, 1) = (\Psi_{2,1}(x), \Psi_{2,2}(x))$ (that is, it is the second row of the matrix $\Psi(x)$. By (7.56), $\psi(0, 1) \in \ker(\phi_{j_1}) \cap \ker(\phi_{j_2}) \cap \cdots \cap \ker(\phi_{j_l})$; that is, it solves all of the equations

$$D_{j_s}\Psi_{2,1}(x_{j_s}) + G_{j_s}\Psi_{2,2}(x_{j_s}) = 0 \qquad s = 1, 2, \ldots, l. \tag{7.57}$$

Furthermore, we know by construction that the first row of the $\Psi(x)$ matrix satisfies the equations

$$D_{j_s}\Psi_{1,1}(x_{j_s}) + G_{j_s}\Psi_{1,2}(x_{j_s}) = 0 \qquad s = 1, 2, \ldots, l. \tag{7.58}$$

(Note that these are the *original* (D_i, G_i) pairs which are not zero, not the modified $(D_i^{[l]}, G_i^{[l]})$ pairs.)

Like the Welch-Berlekamp algorithm, Algorithm 7.2 requires the propagation of four polynomials. However, we now show that it is possible to generate a solution using only two polynomials. This is done by showing how to compute the error values using only two elements in the $\Psi(x)$ matrix. We show how to compute the error values for the DB form of the key equation. (Extension to the WB form of the key equation is straightforward.)

Lemma 7.13 *[63, p. 879] Define*

$$h(x) = \frac{g(\alpha^b x)}{\prod_{s=1}^{l}(x - x_{j_s})}.$$

Then the error value corresponding to the location $x_i = \alpha^k$ is

$$Y[x_i] = \frac{\alpha^{b(d-1)} x_i^{-b} (-1)^l \prod_{s=1}^{l-1} D_{j_s}^{[s-1]}}{\Psi_{1,1}'(x_i)\Psi_{2,1}(x_i)h(x_i)}. \tag{7.59}$$

Proof We consider separately the cases that the error is in a message location or a check location. Recall that the x_k, $k = 1, 2, \ldots, m$ defined in the interpolation problem represent check locations. Suppose first the error is in a message location, with locator x_i. From (7.54) and (7.55) we have

$$\det(\Psi(x)) = \Psi_{1,1}(x)\Psi_{2,2}(x) - \Psi_{1,2}(x)\Psi_{2,1}(x) = (-1)^l \prod_{s=1}^{l} D_{j_s}^{[s-1]}(x - x_{j_s}). \tag{7.60}$$

By definition, the x_{j_s} are all in check locations, so the right-hand side of the equation is not zero at $x = x_i$. But since x_i is an error location, we must have $\Psi_{1,1}(x_i) = 0$. We thus obtain

$$\Psi_{1,2}(x_i) = -\frac{(-1)^l \prod_{s=1}^{l} D_{j_s}^{[s-1]}(x_i - x_{j_s})}{\Psi_{2,1}(x_i)}. \tag{7.61}$$

Recalling that the error value for a message location is (see (7.23))

$$Y[x_i] = -\frac{N_2(x_i)x_i^{-b}\alpha^{b(d-1)}}{W_2'(x_i)g(x_i\alpha^b)} = \frac{\Psi_{1,2}(x_i)x_i^{-b}\alpha^{b(d-1)}}{\Psi_{1,1}'(x_i)g(x_i\alpha^b)}$$

and substituting from (7.61), the result follows.

Now consider the case that x_i is a check location. From (7.58), when $\Psi_{1,1}(x_i) = 0$, we must have $\Psi_{1,2}(x_i) = 0$. As observed in (7.57), $(\Psi_{2,1}(x), \Psi_{2,2}(x))$ satisfies the interpolation problem (7.18) for $\alpha^k \in \{x_{j_s}, s = 1, 2, \ldots, l\}$. The function $(h(x), h(x))$ satisfies it (trivially) for other values of α^k that are check locations, since it is zero on both sides of the interpolation equation. Thus $(h(x)\Psi_{2,1}(x), h(x)\Psi_{2,2})(x))$ satisfies (7.18). Thus

$$h(x_i)\Psi_{2,2}(x_i) = h(x_i)\Psi_{2,1}(x_i)r[x_i]g'(x_i\alpha^b)x_i^b\alpha^{b(2-d)}.$$

It can be shown that $\Psi_{2,1}(x_i) \neq 0$ and that $h(x_i) \neq 0$ (see Exercise 7.11), so that

$$r[x_i] = \frac{\Psi_{2,2}(x_i)x_i^{-b}\alpha^{b(d-2)}}{\Psi_{2,1}(x_i)g'(\alpha^b x_i)}.$$

Now substitute into (7.24):

$$\begin{aligned} Y[x_i] &= \frac{\Psi_{2,2}(x_i)x_i^{-b}\alpha^{b(d-2)}}{\Psi_{2,1}(x_i)g'(\alpha^b x_i)} - \frac{\Psi_{1,2}'(x_i)x_i^{-b}\alpha^{b(d-2)}}{\Psi_{1,1}'(x_i)g'(x_i\alpha^b)} \\ &= \frac{(\Psi_{1,1}'(x_i)\Psi_{2,2}(x_i) - \Psi_{2,1}(x_i)\Psi_{1,2}'(x_i))x_i^{-b}\alpha^{b(d-2)}}{\Psi_{1,1}'(x_i)\Psi_{2,1}(x_i)g'(\alpha^b x_i)}. \end{aligned}$$

Since $\Psi_{1,1}(x_i) = 0$ and $\Psi_{1,2}(x_i) = 0$, we have

$$(\Psi_{1,1}(x)\Psi_{2,2}(x) - \Psi_{1,2}(x)\Psi_{2,1}(x))'\big|_{x=x_i} = \Psi_{1,1}'(x_i)\Psi_{2,2}(x_i) - \Psi_{1,2}'(x_i)\Psi_{2,1}(x_i)$$

which, in turn, using (7.60) is equal to

$$(\det(\Psi(x)))'\big|_{x=x_i} = (-1)^l \prod_{\substack{s=1 \\ i\neq j_s}}^{l} D_{j_s}^{[s-1]}(x_i - x_{j_s}).$$

We thus have

$$Y[x_i] = \frac{x_i^{-b}\alpha^{b(d-2)}(-1)^l \prod_{\substack{s=1 \\ j_s\neq i}}^{l} D_{j_s}^{[s-1]}(x_i - x_{j_s})}{\Psi_{1,1}'(x_i)\Psi_{2,1}(x_i)g'(\alpha^b x_i)}.$$

Since $g'(\alpha^b x_i) = \alpha^b \prod_{\substack{k=0 \\ k\neq i}}^{d-2} (\alpha^b x_i - \alpha^{b+k})$, we can write

$$Y[x_i] = \frac{x_i^{-b}\alpha^{b(d-2)}(-1)^l \prod_{s=1}^{l} D_{j_s}^{[s-1]}}{\Psi_{1,1}'(x_i)\Psi_{2,1}(x_i)h(x_i)}.$$

□

Since by this lemma the only polynomials needed to compute the error values are $\Psi_{1,1}(x)$ and $\Psi_{2,1}(x)$ (the first column of $\Psi(z)$), we only need to propagate these. This gives rise to the following decoding algorithm.

Algorithm 7.3 Welch-Berlekamp Interpolation, Modular Method, v. 2

1 **Input:** Points $(x_i, y_i), i = 1, 2, \ldots, m$ from DB form of key equation.
Returns: $\Psi_{2,1}(x)$ and $\Psi_{1,1}(x)$ which can be used to find error values.
2 **Initialization:** Set $G_i^{[0]} = 1$, $D_i^{[0]} = -y_i, i = 1, 2, \ldots, m$, $\Psi_{1,1}^{[0]} = 1$; $\Psi_{2,1}^{[0]} = 0$.
3 for $s = 1$ to m
4 Choose j_s such that $D_{j_s}^{[s-1]} \neq 0$. If no such j_s, **break**.
5 for $k = 1$ to m (may be done in parallel)
6 $\begin{bmatrix} D_k^{[s]} \\ G_k^{[s]} \end{bmatrix} = \begin{bmatrix} -G_{j_s}^{[s-1]} & D_{j_s}^{[s-1]} \\ (x_k - x_{j_s}) & 0 \end{bmatrix} \begin{bmatrix} D_k^{[s-1]} \\ G_k^{[s-1]} \end{bmatrix}$
7 end (for)
8 $\begin{bmatrix} \Psi_{1,1}^{[s]} \\ \Psi_{2,1}^{[s]} \end{bmatrix} = \begin{bmatrix} -G_{j_s}^{[s-1]} & D_{j_s}^{[s-1]} \\ (x - x_{j_s}) & 0 \end{bmatrix} \begin{bmatrix} \Psi_{1,1}^{[s-1]} \\ \Psi_{2,1}^{[s-1]} \end{bmatrix}$
9 end (for)

Example 7.8 For the (x_i, y_i) data in Example 7.2 (which form is necessary, since the method is based on the error values computed for the DB form), the algorithm operates as follows (using 1-based indexing).

$s = 1$: $\{D_i^{[0]}\} = \{\alpha^9, \alpha^8, 0, \alpha^14, 0, 1\}$. Choose: $j_1 = 1$. $(\Psi_{1,1}(x), \Psi_{2,1}(x)) = (1, 1 + x)$
$s = 2$. $\{D_i^{[1]}\} = \{0, \alpha^{12}, \alpha^9, \alpha^4, \alpha^9, \alpha^7\}$. Choose: $j_2 = 2$. $(\Psi_{1,1}(x), \Psi_{2,1}(x)) = (\alpha^{12}x, \alpha + x)$
$s = 3$. $\{D_i^{[2]}\} = \{0, 0, \alpha^6, \alpha^8, \alpha^6, \alpha^3\}$. Choose: $j_3 = 3$. $(\Psi_{1,1}(x), \Psi_{2,1}(x)) = (\alpha^7 + \alpha x, \alpha^{14}x + \alpha^{12}x^2,)$
$s = 4$. $\{D_i^{[3]}\} = \{0, 0, 0, \alpha^3, \alpha^{10}, \alpha^8\}$. Choose: $j_4 = 4$. $(\Psi_{1,1}(x), \Psi_{2,1}(x)) = (\alpha^6 + \alpha^8 x + x^2, \alpha^10 + \alpha^3 x + \alpha x^2,)$
$s = 5$. $\{D_i^{[4]}\} = \{0, 0, 0, 0, \alpha^{14}, 0\}$. Choose: $j_5 = 5$. $(\Psi_{1,1}(x), \Psi_{2,1}(x)) = (\alpha^{12} + \alpha^4 x + \alpha^8 x^2, \alpha^{10} + \alpha^4 x + \alpha^5 x^2 + x^3)$
$s = 6$. $\{D_i^{[4]}\} = \{0, 0, 0, 0, 0, \alpha^3\}$. Choose: $j_6 = 6$. $(\Psi_{1,1}(x), \Psi_{2,1}(x)) = (\alpha^{13} + \alpha^7 x + \alpha^8 x^2 + \alpha^3 x^3, \alpha^2 + \alpha^8 x + \alpha^{11} x^2 + \alpha^8 x^3)$

The roots of $\Psi_{1,1}(x) = \alpha^{13} + \alpha^7 x + \alpha^8 x^2 + \alpha^3 x^3$ are at α^4, α^9 and α^{12}. Using (7.59) the error value are found to be α^5, α^7 and α^8, respectively. □

One last simplification is now developed for this algorithm. As we shall show, there are some computations which Algorithm 7.3 performs which turn out to be unnecessary. If we can postpone some of them until it is certain that they are needed, unnecessary computations can be avoided. This idea was originally suggested as using *queues* in [23].

Let

$$\Omega^s = \{k \in \{1, 2, \ldots, 2t\} : D_k^s = 0 \text{ and } G_k^s = 0\}$$

and let

$$\Sigma^s = \{k \in \{1, 2, \ldots, 2t\} : D_k^s = 0 \text{ and } G_k^s \neq 0\}.$$

For a $k \in \Sigma^2$, we can (by normalization if necessary) assume without loss of generality that $G_k^s = 1$.

Let j_s be such that $D_{s_j}^{[s-1]} \neq 0$ (that is, $j_s \notin \Sigma^{s-1}$ and $j_s \notin \Omega^{s-1}$) and let $k \in \Sigma^{s-1}$ such that $D_k^{[s-1]} = 0$ and $G_k^{[s-1]} = 1$. Then

$$\begin{bmatrix} D_k^{[s]} \\ G_k^{[s]} \end{bmatrix} = \begin{bmatrix} -G_{j_s}^{[s-1]} & D_{j_s}^{[s-1]} \\ (x_k - x_{j_s}) & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix},$$

so that for all $k \in \Sigma^{s-1}$, $G_k^s = 0$ and $D_k^{[s]} = D_{j_s}^{[s-1]} \neq 0$. Hence $k \notin \Sigma^s$. Therefore, at the $(s+1)$st step of the algorithm, if $\Sigma^{s-1} \neq \emptyset$, j_{s+1} can be chosen from Σ^{s-1}. So let $j_{s+1} \in \Sigma^{s-1}$. For $k \in \Sigma^{s-1}$ with $k \neq j_{s+1}$,

$$\begin{bmatrix} D_k^{[s+1]} \\ G_k^{[s+1]} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ (x_k - x_{j_{s+1}}) & 0 \end{bmatrix} \begin{bmatrix} D_k^{[s]} \\ 0 \end{bmatrix}$$

so that $D_k^{[s+1]} = 0$ and $G_k^{[s+1]} \neq 0$. From this, $k \in \Sigma^{s+1}$ and $j_{s+1} \in \Omega^{s+1}$.

Thus, the for loop in lines 5–7 of Algorithm 7.3 can exclude those values of k that are in $\Sigma^{s-1} \cup \Sigma^s \cup \Omega^s$, since over the next two iterations computations involving them are predictable (and eventually they may not need to be done at all). This leads to the following statement of the algorithm.

Algorithm 7.4 Welch-Berlekamp Interpolation, Modular Method, v. 3

1 **Input:** Points $(x_i, y_i), i = 1, 2, \ldots, m$.
Returns: $\Psi_{2,1}(x)$ and $\Psi_{1,1}(x)$ which can be used to find error values.
2 **Initialization:** Set $\Sigma^0 = \Sigma^{-1} = \Omega^0 = \emptyset$, $G_i^{[0]} = 1$, $D_i^{[0]} = -y_i$, $i = 1, 2, \ldots, m$, $\Psi_{1,1}^{[0]} = 1$; $\Psi_{2,1}^{[0]} = 0$.
3 for $s = 1$ to m
4 If $\Sigma^{s-2} \neq \emptyset$ choose $j_s \in \Sigma^{s-2}$. Otherwise, choose j_s such that $D_{j_s}^{[s-1]} \neq 0$.
If no such j_s, **break**.
5 for $k = 1$ to m such that $k \notin \Sigma^{s-2} \cup \Sigma^{s-1} \cup \Omega^{s-1}$ (may be done in parallel)
6 $$\begin{bmatrix} D_k^{[s]} \\ G_k^{[s]} \end{bmatrix} = \begin{bmatrix} -G_{j_s}^{[s-1]} & D_{j_s}^{[s-1]} \\ (x_k - x_{j_s}) & 0 \end{bmatrix} \begin{bmatrix} D_k^{[s-1]} \\ G_k^{[s-1]} \end{bmatrix}$$
7 end (for)
8 $$\begin{bmatrix} \Psi_{1,1}^{[s]} \\ \Psi_{2,1}^{[s]} \end{bmatrix} = \begin{bmatrix} -G_{j_s}^{[s-1]} & D_{j_s}^{[s-1]} \\ (x - x_{j_s}) & 0 \end{bmatrix} \begin{bmatrix} \Psi_{1,1}^{[s-1]} \\ \Psi_{2,1}^{[s-1]} \end{bmatrix}$$
9 $\Sigma^{s-2} \leftarrow \Sigma^{s-1}$; Update Σ^s and Ω^s.
10 end (for)

7.5 Erasure Decoding with the Welch-Berlekamp Key Equation

In the event that some of the positions in $R(x)$ are erased, the algorithms can be modified as follows. Let the erasure locator polynomial be $\Gamma(x) = \prod_i (x - \alpha_i^e)$, where the e_i are in this case the erasure locations.

For the Welch-Berlekamp algorithm of Section 7.4.2, erasure/error decoding proceeds exactly as in the case of errors-only decoding, except that initial W polynomial is set equal to the erasure locator polynomial, $W^{[0]}(x) = \Gamma(x)$. The formulas for computing the error and erasure values are exactly the same as for errors-only decoding.

For the Dabiri-Blake decoding, Algorithm 7.2, think of $W(x)$ as consisting of two factors, one factor due to errors and the other due to erasures,

$$W(x) = W_1(x)\Gamma(x).$$

Then the interpolation problem is written as

$$N(x_i) = W(x_i)y_i = W_1(x_i)\Gamma(x_i)y_i.$$

Now let $\tilde{y}_i = \Gamma(x_i)y_i$ and run Algorithm 7.2 on the data $(x_i, \tilde{y}_i)$, except that the initialization $\Psi_{1,1}^{[0]}(x) = \Gamma(x)$ is performed. The error value computations are unchanged.

7.6 The Guruswami-Sudan Decoding Algorithm and Soft RS Decoding

The algebraic decoders presented to this point in the book are bounded distance decoders, meaning they are capable of decoding up to $t_0 = \lfloor(d_{\min} - 1)/2\rfloor$ errors. In the remainder of this chapter we discuss a list decoding approach to Reed-Solomon (and related) codes which is capable of decoding beyond t_0 errors. A *list decoder* generally returns several possible decoded messages. However, for many codes the size of the list is usually small, so that only one decoded message is usually returned. An extension of this decoding algorithm provides a means for *algebraic soft decision* decoding of Reed-Solomon codes.

7.6.1 Bounded Distance, ML, and List Decoding

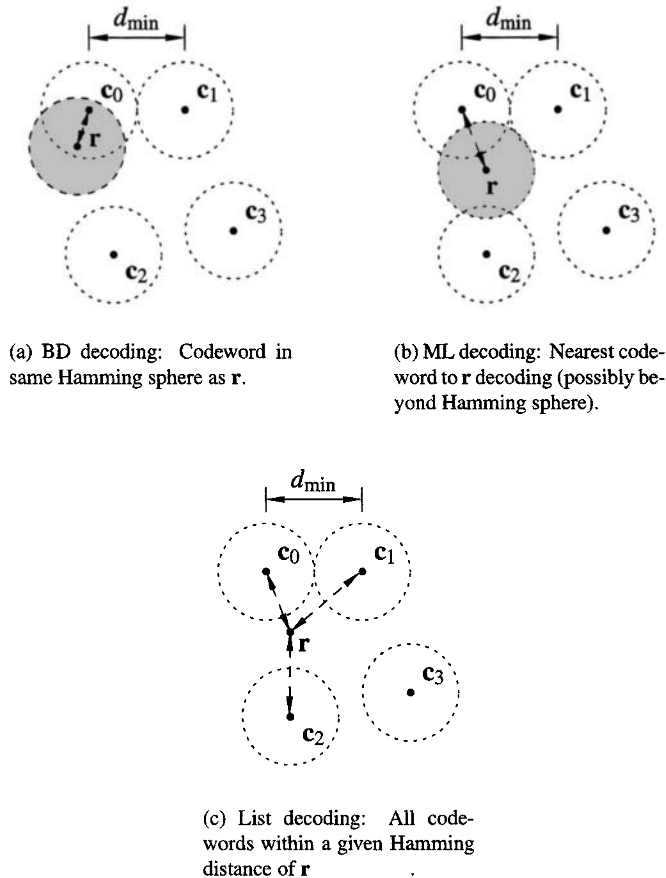
Consider an (n, k, d) Reed-Solomon code $\mathcal{C}$. Three different decoding paradigms can be employed in decoding such codes.

In **bounded distance (BD) decoding**, the following problem is solved: For a distance e such that $2e + 1 \leq d_{\min}$, given a received vector $\mathbf{r}$, find a codeword $\mathbf{c} \in \mathcal{C}$ which is within a Hamming distance e of $\mathbf{r}$. There exist many efficient algorithms exist for solving this (all of the algorithms in chapter 6 are BD decoders), and for a t-error correcting code the answer is unique when $t \geq e$. However, if $\mathbf{r}$ lies at a distance greater than $\lfloor(d_{\min} - 1)/2\rfloor$ from any codeword, a decoding failure will result.

In **maximum likelihood (ML) decoding** (also known as **nearest codeword problem**), the codeword $\mathbf{c}$ which is closest to $\mathbf{r}$ is selected. Provided that the number of errors e satisfies $2e + 1 \leq d_{\min}$, the ML and BD algorithms decode identically. However, ML decoding may be able to decode beyond $\lfloor(d_{\min} - 1)/2\rfloor$ errors. The ML decoding problem, however, is computationally difficult in general [24].

In **list decoding** the problem is to find *all* codewords $\mathbf{c} \in \mathcal{C}$ which are within a given distance e of the received word $\mathbf{r}$.

The Guruswami-Sudan (GS) algorithm is essentially a list-decoding algorithm, providing lists of all codewords within a distance t_m of the received word $\mathbf{r}$. Whereas the BD decoder is able to correct a fraction $\tau = \frac{1}{n}\frac{n-k}{2} = \frac{1-R}{2}$ of the errors, the Guruswami-Sudan algorithm is able to correct a up to $t_{GS} = \lceil n - \sqrt{nk} - 1\rceil$ errors, so that the fraction is (asymptotically) $\tau = 1 - \sqrt{R}$. Thus the GS algorithm has better error correction capability for every code rate R.



(a) BD decoding: Codeword in same Hamming sphere as **r**.

(b) ML decoding: Nearest codeword to **r** decoding (possibly beyond Hamming sphere).

(c) List decoding: All codewords within a given Hamming distance of **r**

Figure 7.2: Comparing BD, ML, and list decoding.

7.6.2 Error Correction by Interpolation

The motivating idea behind the GS decoding algorithm can be expressed as follows. Under Construction 1 of Reed-Solomon codes, a set of data points $(x_i, c_i), i = 1, \ldots, n$ are generated with a polynomial relationship $c_i = m(x_i)$ for some polynomial $m(x)$ which has at most degree $k-1$. A set of points (x_i, c_i) are produced. The c_i are corrupted by some noise process, producing points (x_i, y_i) in which as many as e of the r_i are in error. The problem now is to fit a polynomial $p(x)$, of degree $< k$ through the data points, such that $p(x_i) = y_i$. However, since some of the points are in error, we seek an interpolating match only for $n - e$ of the data points, so $|\{i : p(x_i) = y_i\}| \geq n - e$. Then, based on this interpolating polynomial, the points in error are recovered. That is, if i is the index of a point in error, then we say that the recovered value is $\hat{r}_i = p(x_i)$.

The Guruswami-Sudan decoding is based generally on this idea of interpolation. The interpolating polynomial is constructed as a polynomial in *two* variables, $Q(x, y)$ which satisfies the interpolating condition $Q(x_i, y_i) = 0$. In addition to simply interpolating, an

interpolation multiplicity m_i is introduced which defines the order of the interpolation at each point. This is roughly equivalent to specifying the value of the function and its $m_i - 1$ derivatives in the interpolating polynomial. This interpolation multiplicity improves the correction capability. Furthermore, as we will see in Section 7.6.8, we will see that the interpolating multiplicity can be used for soft decision decoding of RS codes. From the bivariate polynomial $Q(x, y)$ the polynomials $p(x)$ are extracted by factorization, which will satisfy the property $p(x_i) = y_i$ for a sufficiently large number of locations (x_i, y_i). Then each polynomial $p(x)$ represents a possible transmitted codeword and the set of polynomials is the list of possible decoded codewords.

There are thus two main steps to the decoding algorithm:

The interpolation step The decoder constructs a two-variable polynomial

$$Q(x, y) = \sum_{i=0}^{d_x} \sum_{j=0}^{d_y} a_{i,j} x^i y^j \tag{7.62}$$

such that $Q(x_i, y_i) = 0$ for $i = 1, 2, \ldots, n$ (with a certain multiplicity of the zero, to be discussed below), and for which the "degree" (actually, the weighted degree) of $Q(x, y)$ is as small as possible.

Explaining and justifying this step will require a discussion of the concept of the degree of multivariable polynomials, which is presented in Section 7.6.3.

The problem can be set up and solved using straightforward linear algebra. However, a potentially more efficient (and interesting) algorithm due to Kötter [193] is presented in Section 7.6.5. An algorithm accomplishing the solution, which is an extension of the Berlekamp-Massey algorithm to vectors, is presented in Section 7.6.5.

The factorization step The decoder then finds all factors of $Q(x, y)$ of the form $y - p(x)$, where $p(x)$ is a polynomial of degree v or less. This step produces the list of polynomials

$$\mathcal{L} = \{p_1(x), p_2(x), \ldots, p_L(x)\}$$

that agree with (x_i, y_i) in at least t_m places. That is, $|\{i : p_j(x_i) = y_i\}| \geq t_m$ for every $p_j \in \mathcal{L}$.

An algorithm due to Roth and Ruckenstein [297] which performs the factorization by reducing it to single-variable factorization (amenable, e.g., to root-finding via the Chien search) is presented in Section 7.6.7.

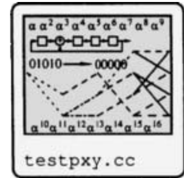
The quantity t_m is the *designed decoding radius.* The larger t_m is, the more potential errors can be corrected. The quantity t_m depends in a nondecreasing way on m_i; that is $t_0 \leq t_1 \leq t_2 \cdots$, and there is a multiplicity m_0 such that $t_{m_0} = t_{m_0+1} = \cdots \triangleq t_{GS}$ that describes the maximum error correction capability of the decoding algorithm.

7.6.3 Polynomials in Two Variables

In this section, we describe the concepts associated with polynomials in two variables which are necessary to understand the algorithm. We will see that for a polynomial $p(x)$ of degree v such that $(y - p(x)) \mid Q(x, y)$ it is natural to consider the (univariate) polynomial

$Q(x, p(x))$. For $Q(x, y)$ as in (7.62), this gives

$$Q(x, p(x)) = \sum_{i=0}^{d_x} \sum_{j=0}^{d_y} a_{i,j} x^i p(x)^j,$$



which is a polynomial in x of degree $d_x + vd_y$. It is thus natural to define the degree of $Q(x, y)$ as $d_x + vd_y$. This is called the $(1, v)$ **weighted degree** of $Q(x, y)$.

Degree and Monomial Order

For a polynomial in a single variable, the notion of degree is straightforward. For polynomials defined over multiple variables, however, there is some degree of flexibility available in defining the order of the polynomials. Various algorithms (and their complexity) depend on the particular order employed. (For a full discussion of the degree of multinomials, the reader is referred to [60, section 2.2].)

Let $\mathbb{F}$ be a field and let $\mathbb{F}[x, y]$ denote the commutative ring of polynomials in the variables x and y with coefficients from F. A polynomial $Q(x, y) \in \mathbb{F}[x, y]$ can be written as

$$Q(x, y) = \sum_{i,j \geq 0} a_{i,j} x^i y^j,$$

in which only a finite number of coefficients are nonzero.

Example 7.9 Let $\mathbb{F} = \mathbb{R}$ and let

$$Q(x, y) = 3x^3 y + 4xy^3 + 5x^4.$$

Looking forward to the upcoming definitions, we ask the questions: What is the degree of $Q(x, y)$? What is the leading term of $Q(x, y)$? How is $Q(x, y)$ to be written with the terms ordered in increasing "degree"? □

To address the questions raised in this example, it will be convenient to impose an *ordering* on the set of monomials

$$\mathbb{M}[x, y] = \{x^i y^j : i, j \geq 0\} \subset \mathbb{F}[x, y].$$

That is, we want to be able to say when a monomial $x^{i_1} y^{j_1}$ is "less than" $x^{i_2} y^{j_2}$. Let $\mathbb{N}^2$ denote the set of pairs of natural numbers (pairs of nonnegative integers).

Definition 7.9 A **monomial ordering** is a relation "$<$" on $\mathbb{M}[x, y]$ with the following properties.

MO1 For $(a_1, a_2) \in \mathbb{N}^2$ and $(b_1, b_2) \in \mathbb{N}^2$, if $a_1 \leq b_1$ and $a_2 \leq b_2$ then $x^{a_1} y^{a_2} \leq x^{b_1} y^{b_2}$. (That is, the monomial $x^{a_1} y^{a_2}$ "comes before" the monomial $x^{b_1} y^{b_2}$.)

MO2 The relation "$<$" is a total ordering. That is, if $\mathbf{a} = (a_1, a_2) \in \mathbb{N}^2$ and $\mathbf{b} = (b_1, b_2) \in \mathbb{N}^2$ are distinct, then either $x^{a_1} y^{a_2} < x^{b_1} y^{b_2}$ or $x^{b_1} y^{b_2} < x^{a_1} y^{a_2}$.

MO3 For any (a_1, a_2), (b_1, b_2) and $(c_1, c_2) \in \mathbb{N}^2$, if $x^{a_1} y^{a_2} \leq x^{b_1} y^{b_2}$, then $x^{a_1} y^{a_2} x^{c_1} y^{c_2} \leq x^{b_1} y^{b_2} x^{c_1} y^{c_2}$.

□

Of the many possible monomial orderings one might consider, those which will be most important in this development are the *weighted degree* (WD) monomial orderings. Each

Table 7.1: Monomials Ordered Under (1, 3)-revlex Order

Monomial: $\phi_j(x, y)$	1	x	x^2	x^3	y	x^4	xy	x^5	x^2y	x^6	x^3y	y^2	x^7	x^4y	xy^2
Weight: (w-revlex):	0	1	2	3	3	4	4	5	5	6	6	6	7	7	7
j:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Monomial: $\phi_j(x, y)$	x^8	x^5y	x^2y^2	x^9	x^6y	x^3y^2	y^3	x^{10}	x^7y	x^4y^2	xy^3	x^{11}	x^8y	x^5y^2	x^2y^3
Weight: (w-revlex):	8	8	8	9	9	9	9	10	10	10	10	11	11	11	11
j:	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29

Table 7.2: Monomials Ordered Under (1, 3)-lex Order

Monomial: $\phi_j(x, y)$	1	x	x^2	y	x^3	xy	x^4	x^2y	x^5	y^2	x^3y	x^6	xy^2	x^4y	x^7
Weight: (w-lex):	0	1	2	3	3	4	4	5	5	6	6	6	7	7	7
j:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Monomial: $\phi_j(x, y)$	x^2y^2	x^5y	x^8	y^3	x^3y^2	x^6y	x^9	xy^3	x^4y^2	x^7y	x^{10}	x^2y^3	x^5y^2	x^8y	x^{11}
Weight: (w-lex):	8	8	8	9	9	9	9	10	10	10	10	11	11	11	11
j:	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29

WD monomial order is characterized by a fixed pair $\mathbf{w} = (u, v)$ of non-negative integers where not both are 0. Then the $\mathbf{w}$-degree of the monomial $x^i y^j$ is defined as

$$\deg_{\mathbf{w}} x^i y^j = ui + vj.$$

The WD monomial order is sufficient to define a *partial order* on $\mathbb{M}[x, y]$. However there may be ties under this order, since there are monomials $\phi_1(x, y)$ and $\phi_2(x, y)$ with $\phi_1(x, y) \neq \phi_2(x, y)$ with $\deg_{\mathbf{w}} \phi_1(x, y) = \deg_{\mathbf{w}} \phi_2(x, y)$, so that it is not yet an order. There are two common ways to break such ties.

Definition 7.10 In **w-lexicographic** (or **w-lex**) **order**, if $ui_1 + vj_1 = ui_2 + vj_2$, we say that $x^{i_1}y^{j_1} < x^{i_2}y^{j_2}$ if $i_1 < i_2$. In **w-reverse lexicographic** (or **w-revlex**) **order**, if $ui_1 + vj_1 = ui_2 + vj_2$, we say that $x^{i_1}y^{j_1} < x^{i_2}y^{j_2}$ if $i_1 > i_2$. These orderings are denoted by $<_{\text{wlex}}$ and $<_{\text{wrevlex}}$. □

Example 7.10 Let $\mathbf{w} = (1, 3)$. Let $\phi_1(x, y) = x^2y^3$; then $\deg_{\mathbf{w}} \phi_1 = 11$. Let $\phi_2(x, y) = x^8y$; then $\deg_{\mathbf{w}} \phi_2 = 11$, so there is a tie in the degree. Under $\mathbf{w}$-lex order, $\phi_1(x, y) <_{\text{wlex}} \phi_2(x, y)$. Under $\mathbf{w}$-revlex order, $\phi_2(x, y) <_{\text{wrevlex}} \phi_1(x, y)$. □

By a fixed monomial order $<$ a set of monomials $\{\phi_i\}$ can be uniquely ordered:

$$1 = \phi_0(x, y) < \phi_1(x, y) < \phi_2(x, y) < \cdots.$$

Example 7.11 Let $\mathbf{w} = (1, 3)$. Table 7.1 shows the first 30 monomials $\phi_j(x, y)$ ordered in $\mathbf{w}$-revlex order, along with the $\mathbf{w}$-degree of the monomial and the order index j. The first 30 monomials ordered in $\mathbf{w}$-lex order are shown in Table 7.2. □

For purposes of characterizing the performance of the decoder, it is useful to know how many monomials there are up to a given weighted degree.

Lemma 7.14 *Let $C(v, l)$ be the number of monomials of weighted $(1, v)$-degree less than or equal to l. Then*

$$C(v,l) = \left(\left\lfloor \frac{l}{v} \right\rfloor + 1\right)\left(l + 1 - \frac{v}{2}\left\lfloor \frac{l}{v} \right\rfloor\right). \tag{7.63}$$

Furthermore,

$$C(v,l) \geq \frac{l}{v}\frac{l+2}{2} > \frac{l^2}{2v}. \tag{7.64}$$

Example 7.12 Some values of $C(3, l)$ computed using (7.63) are

l	1	2	3	4	5	6	7	8	9	10	11
$C(3,l)$	2	3	5	7	9	12	15	18	22	26	30
bound	0.5	1.67	2.5	4	5.83	8	10.5	13.33	16.5	20	23.83

These can be compared with the data in Table 7.1 or Table 7.2. □

Proof For a fixed j_2, the monomials $x^{j_1}y^{j_2}$ of $(1, v)$-degree less than or equal to l is $l + 1 - vj_2$. The largest y-degree of a monomial of $(1, v)$-degree less than l is $\lfloor l/v \rfloor$. The total number of monomials of $(1, v)$-degree $\leq l$ is thus

$$\begin{aligned} C(v,l) &= \sum_{j_2=0}^{\lfloor l/v \rfloor}(l + 1 - vj_2) = (l+1)\left(\left\lfloor \frac{l}{v} \right\rfloor + 1\right) - \frac{v}{2}\left\lfloor \frac{l}{v} \right\rfloor\left(\left\lfloor \frac{l}{v} \right\rfloor + 1\right) \\ &= \left(\left\lfloor \frac{l}{v} \right\rfloor + 1\right)\left(l + 1 - \frac{v}{2}\left\lfloor \frac{l}{v} \right\rfloor\right). \end{aligned}$$

The bound follows since

$$\left(\left\lfloor \frac{l}{v} \right\rfloor + 1\right)\left(l + 1 - \frac{v}{2}\left\lfloor \frac{l}{v} \right\rfloor\right) \geq \left(\left\lfloor \frac{l}{v} \right\rfloor + 1\right)\left(l + 1 - \frac{l}{2}\right) \geq \frac{l}{v}\frac{l+2}{2}.$$

□

A polynomial $Q(x, y) = \sum_{i,j \geq 0} a_{i,j}x^i y^j$ with the monomials ordered by a fixed monomial ordering can be written uniquely as

$$Q(x,y) = \sum_{j=0}^{J} a_j \phi_j(x,y)$$

for some set of coefficients $\{a_j\}$, with $a_J \neq 0$. The integer J is called the **rank** of $Q(x, y)$, denoted $\text{rank}(Q(x, y))$. The monomial $\phi_J(x, y)$ is called the **leading monomial** of $Q(x, y)$, denoted $\text{LM}(Q(x, y))$. The coefficient a_J is called the **leading coefficient** of $Q(x, y)$. The weighted degree of the leading monomial of $Q(x, y)$ is called the **weighted degree** of $Q(x, y)$, or **w**-degree, denoted $\deg_{\mathbf{w}}(Q(x, y))$:

$$\deg_{\mathbf{w}}(Q(x,y)) = \deg_{\mathbf{w}} \text{LM}(Q(x,y)) = \max\{\deg_{\mathbf{w}} \phi_j(x,y) : a_j \neq 0\}$$

We also say that the y-degree of $Q(x, y)$ is the degree of $Q(1, y)$ as a polynomial in y.

Example 7.13 Let $\mathbf{w} = (1, 3)$ and let $<$ be the **w**-revlex ordering. When

$$Q(x,y) = 1 + xy + x^4y + x^2y^3 + x + y + x^8y$$

is written with monomials ordered in increasing degree under $<$, we have

$$Q(x, y) = 1 + x + y + xy + x^4y + x^8y + x^2y^3.$$

The y-degree of $Q(x, y)$ is 3. The $\mathrm{LM}(Q) = x^2y^3$, $\mathrm{rank}(Q) = 29$ (refer to Table 7.1) and $\deg_{\mathbf{w}}(Q) = 11$.

When $Q(x, y)$ is written under **w**-lex ordering,

$$Q(x, y) = 1 + x + y + xy + x^4y + x^2y^3 + x^8y,$$

and $\mathrm{LM}(Q) = x^8y$ and $\mathrm{rank}(Q) = 28$ (refer to Table 7.2). □

Having defined an order on monomials, this can be extended to a partial order on polynomials.

Definition 7.11 For two polynomials $P(x, y)$, $Q(x, y) \in \mathbb{F}[x, y]$, we say that $P(x, y) < Q(x, y)$ if $\mathrm{LM}(P(x, y)) < \mathrm{LM}(Q(x, y))$. (This is a partial order on $\mathbb{F}[x, y]$, since distinct polynomials may have the same leading monomial.) □

Zeros and Multiple Zeros

In the GS decoder, we are interested in fitting an interpolating polynomial with a multiplicity of zeros. We define in this section what we mean by this.

We first make an observation about zeros at 0 of polynomials in one variable.

Definition 7.12 For $m \leq n$, the polynomial

$$Q(x) = a_m x^m + a_{m+1}x^{m+1} + \cdots + a_n x^n = \sum_{r=m}^{n} a_r x^r,$$

where $a_0 = a_1 = \cdots = a_{m-1} = 0$, is said to a zero of **order** or **multiplicity** m at 0. We write $\mathrm{ord}(Q; 0) = m$. □

Let $\mathcal{D}_r$ denote the rth formal derivative operator (see Section 6.5.1). Then we observe that

$$Q(0) = \mathcal{D}_1 Q(0) = \cdots = \mathcal{D}_{m-1} Q(0) = 0.$$

So the order of a zero can be expressed in terms of derivative conditions in this case.

Let us generalize this result to zeros of order m at other locations. We say that $Q(x)$ has a zero of order m at α if $Q(x + \alpha)$ has a zero of order m at 0. This can be expressed using a kind of Taylor series, which applies over any field, known as **Hasse's theorem**.

Lemma 7.15 *[145] If $Q(x) = \sum_{i=0}^{n} a_i x^i \in \mathbb{F}[x]$, then for any $\alpha \in \mathbb{F}$,*

$$Q(x + \alpha) = \sum_{r=0}^{n} Q_r(\alpha) x^r,$$

where

$$Q_r(x) = \sum_{i=0}^{n} \binom{i}{r} a_i x^{i-r},$$

and where we take $\binom{i}{r} = 0$ if $r > i$.

The proof follows by straightforward application of the binomial theorem. $Q_r(x)$ is called the rth **Hasse derivative** of $Q(x)$. We will denote $Q_r(x)$ by $D_r Q(x)$. In the case that $\mathbb{F}$ is a field of characteristic 0, then

$$D_r Q(x) = Q_r(x) = \frac{1}{r!}\frac{d^r}{dx^r}Q(x), \tag{7.65}$$

so that D_r does, in fact, act like a differentiating operator, but with a scaling factor of $1/r!$. We can write

$$Q(x+\alpha) = \sum_{r=0}^{n} D_r Q(\alpha)x^r.$$

Thus, if $Q(x)$ has a zero of order m at α we must have the first m coefficients of this series equal to 0:

$$Q(\alpha) = D_1 Q(\alpha) = \cdots = D_{m-1}Q(\alpha) = 0.$$

Extending Definition 7.12 we have the following:

Definition 7.13 $Q(x)$ has a zero of **order** (or **multiplicity**) m at α if

$$Q(\alpha) = D_1 Q(\alpha) = \cdots = D_{m-1}(\alpha) = 0.$$

This is denoted as $\text{ord}(Q;\alpha) = m$. □

These concepts extend to polynomials in two variables.

Definition 7.14 Let $Q(x, y) \in \mathbb{F}[x, y]$ and let α and β be such that $Q(\alpha, \beta) = 0$. Then we say that Q has a **zero** at (α, β).

Let $Q(x, y) = \sum_{i,j\geq 0} a_{i,j}x^i y^j$. We say that Q has a **zero of multiplicity** m (or **order** m) at $(0, 0)$ if the coefficients $a_{i,j} = 0$ for all $i + j < m$. When Q has a zero of order m at $(0, 0)$ we write $\text{ord}(Q : 0, 0) = m$.

Similarly, we say that $Q(x, y)$ has a zero of order m at (α, β), denoted as $\text{ord}(Q;\alpha,\beta) = m$, if $Q(x+\alpha, y+\beta)$ has a zero of order m at $(0, 0)$. □

Example 7.14 $Q(x, y) = x^4y + x^3y^2 + x^4y^4$ has a zero of order 5 at $(0, 0)$.

$Q(x, y) = x + y$ has a zero of order 1 at $(0, 0)$.

$Q(x, y) = (x-\alpha)^4(y-\beta) + (x-\alpha)^3(y-\beta)^2 + (x-\alpha)^4(y-\beta)^4$ has a zero of order 5 at (α, β). □

We observe that a zero of order m requires that $\binom{m+1}{2} = m(m+1)/2$ coefficients are 0. For example, for a zero of order 3 at (0,0), the $\binom{3+1}{2} = 6$ coefficients

$$a_{0,0}, a_{0,1}, a_{0,2}, a_{1,0}, a_{1,1}, a_{2,0}$$

are all zero.

Lemma 7.15 is extended in a straightforward way to two variables.

Lemma 7.16 *If* $Q(x, y) = \sum_{i,j} a_{i,j}x^i y^j$, *then*

$$Q(x+\alpha, y+\beta) = \sum_{r,s} Q_{r,s}(\alpha, \beta)x^r y^s$$

where

$$Q_{r,s}(x, y) = \sum_i \sum_j \binom{i}{r}\binom{j}{s} a_{i,j}x^{i-r}y^{j-s}. \tag{7.66}$$

Again, the proof is by straightforward application of the binomial theorem. We will denote

$$Q_{r,s}(x, y) = D_{r,s}Q(x, y); \tag{7.67}$$

this is sometimes called the (r, s)th **Hasse (mixed partial) derivative** of $Q(x, y)$.

Based on this notation, we observe that if $\text{ord}(Q : \alpha, \beta) = m$, then

$$D_{r,s}Q(\alpha, \beta) = 0 \text{ for all } r, s \text{ such that } r + s < m, \tag{7.68}$$

which is a total of $\binom{m+1}{2}$ constraints.

Example 7.15 We demonstrate some Hasse derivatives over $\mathbb{F} = GF(5)$.

$$D_{1,0}x = 1 \qquad D_{1,0}y = 0$$
$$D_{0,1}x = 0 \qquad D_{0,1}y = 1$$
$$D_{1,0}x^2 = \binom{2}{1}\binom{0}{0}x^{2-1}y^{0-0} = 2x$$
$$D_{2,0}x^5 = \binom{5}{2}\binom{0}{0}x^{5-2} = 10x^3 = 0 \text{ (over } GF(5))$$
$$D_{2,1}x^2y^3 = \binom{2}{2}\binom{3}{1}x^{2-2}y^{3-1} = 3y^2.$$

We note that $D_{r,s}$ acts very much like a partial differentiation operator, except for the division $1/r!s!$ suggested by (7.65). □

7.6.4 The GS Decoder: The Main Theorems

With the notation of the previous section, we can now describe the GS decoder in greater detail. For an (n, k) RS code over the field $\mathbb{F}$ with support set (see Section 6.2.1) $(x_1, x_2, \ldots, x_n)$ and a positive integer m, the GS(m) decoder accepts a vector $\mathbf{r} = (y_1, y_2, \ldots, y_n) \in \mathbb{F}^n$ as an input and produces a list of polynomials $\{p_1, p_2, \ldots, p_L\}$ as the output by the following two steps:

Interpolation step: The decoder constructs a nonzero two-variable polynomial of the form

$$Q(x, y) = \sum_{j=0}^{C} a_j \phi_j(x, y)$$

of minimal $(1, v)$-degree which has a zero of order m at each of the points (x_i, y_i), $i = 1, 2, \ldots, n$. Here, the $\phi_j(x, y)$ are monomials of the form $x^p y^q$, ordered according to the $(1, v)$-revlex monomial order such that $\phi_0 < \phi_1 < \cdots$.

Related to this step are two fundamental questions: Does such a polynomial exist? How can it be constructed?

Factorization step: The polynomial $Q(x, y)$ is factored by finding a set of polynomials $p(x)$ such that $y - p(x) \mid Q(x, y)$. We form the set of such polynomials, called the y-roots of $Q(x, y)$,

$$\mathcal{L} = \{p(x) \in \mathbb{F}[x] : (y - p(x)) \mid Q(x, y)\}.$$

Questions related to this step are: How does this relate to the error correction capability of the code? How is the factorization computed? How many polynomials are in $\mathcal{L}$?

As we will see, it is also possible to employ a different interpolation order m_i at each point (x_i, y_i). This is developed further in Section 7.6.8.

The Interpolation Theorem

We address the existence of the interpolating polynomial with the following theorem.

Theorem 7.17 (**The Interpolation Theorem**) *Let*

$$Q(x, y) = \sum_{i,j} a_{i,j} x^i y^j = \sum_{i=0}^{C} a_i \phi(x, y), \tag{7.69}$$

where the monomials are ordered according to an arbitrary monomial order. Then a nonzero $Q(x, y)$ polynomial exists that interpolates the points $(x_i, y_i), i = 1, 2, \ldots, n$ with multiplicity m at each point if

$$C = n\binom{m+1}{2}. \tag{7.70}$$

Proof There is a zero of multiplicity m at (x_i, y_i) if

$$D_{r,s} Q(x_i, y_i) = 0 \text{ for all } (r, s) \text{ such that } 0 \le r + s < m. \tag{7.71}$$

Using the Hasse partial derivatives defined in (7.66) and (7.67), equation (7.71) can be written as

$$\sum_k \sum_j \binom{k}{r}\binom{j}{s} a_{k,j} x_i^{k-r} y_i^{j-s} = 0 \qquad i = 1, 2, \ldots, n, \text{ and } r + s < m. \tag{7.72}$$

There are $\binom{m+1}{2}$ linear homogeneous equations (constraints) for each value of i, for a total of $n\binom{m+1}{2}$ equations. If $C = n\binom{m+1}{2}$, then there are $C+1$ variables $a_0, a_1, \ldots, a_C$ in (7.69). There must be at least one nonzero solution to the set of linear equations (7.71). □

The solution to (7.72) can be computed using straightforward linear algebra, with complexity $O(C^3)$. Other algorithms are discussed in Section 7.6.5.

The Factorization Theorem

The main results regarding the factorization step are provided by the following lemma and theorem.

Lemma 7.18 *Let $Q(x_i, y_i)$ have zeros of multiplicity m at the points $(x_i, y_i), i = 1, 2, \ldots, n$. If $p(x)$ is a polynomial such that $y_i = p(x_i)$, then $(x - x_i)^m \mid Q(x, p(x))$.*

Proof To gain insight, suppose initially that $(x_i, y_i) = (0, 0)$, so $0 = p(0)$. Thus we can write $p(x) = x\tilde{p}(x)$ for some polynomial $\tilde{p}(x)$. Then for

$$Q(x, y) = \sum_{i+j \ge m} a_{i,j} x^i y^j,$$

(where the sum is over $i + j \ge m$ since $Q(x, y)$ has zeros of multiplicity m) it follows that

$$Q(x, p(x)) = Q(x, x\tilde{p}(x)) = \sum_{i+j \ge m} a_{i,j} x^i (x\tilde{p}(x))^j,$$

which is divisible by x^m. This establishes the result for the point $(0, 0)$.

Now let (x_i, y_i) be a general input point with $y_i = p(x_i)$. Let $p^{(i)}(x) = p(x+x_i)-y_i$, so that $p^{(i)}(0) = 0$. Thus $p^{(i)}(x) = x\tilde{p}^{(i)}(x)$ for some polynomial $\tilde{p}^{(i)}(x)$. Let $Q^{(i)}(x, y) = Q(x + x_i, y + y_i)$, so $Q^{(i)}(0, 0) = 0$. The problem has been shifted so that $Q^{(i)}(0, 0)$ and $p^{(i)}$ behave like the results above at $(0, 0)$. Thus $x_i^m \mid Q^{(i)}(x, p^{(i)(x)})$. Shifting back gives the desired conclusion. □

Theorem 7.19 **(The Factorization Theorem)** *Let $Q(x, y)$ be an interpolating polynomial of $(1, v)$-weighted degree $\leq l$ such that $D_{r,s}Q(x_i, y_i) = 0$ for $i = 1, 2, \ldots, n$ and for all $r + s < m$. (That is, each (x_i, y_i) is interpolated up to order m.) Let $p(x)$ be a polynomial of degree at most v such that $y_i = p(x_i)$ for at least K_m values of i in $\{1, 2, \ldots, n\}$. If $mK_m > l$, then $(y - p(x)) \mid Q(x, y)$.*

Before proving this theorem, let us put it in context. If $p(x)$ is a polynomial of degree less than k, then $p(x)$ produces a codeword $\mathbf{c} \in \mathcal{C}$ by the mapping

$$p(x) \rightarrow (p(x_1), p(x_2), \ldots, p(x_n)) \in \mathcal{C}.$$

For this codeword, $y_i = p(x_i)$ for at least K_m places. Let $t_m = n - K_m$. Then $\mathbf{c}$ differs from the received vector $\mathbf{r} = (y_1, y_2, \ldots, y_m)$ in as many as t_m places. Thus, $p(x)$ identifies a codeword $\mathbf{c}$ at a distance no greater than t_m from $\mathbf{r}$. This codeword is a candidate to decode $\mathbf{r}$. So, if $p(x)$ agrees in at least K_m places, then by the factorization theorem, $p(x)$ is a y-root of $Q(x, y)$, and is therefore placed on the list of candidate decodings.

Proof Let $g(x) = Q(x, p(x))$. By the definition of the weighted degree, and by the fact that $Q(x, y)$ has $(1, v)$-weighted degree $\leq l$, $g(x)$ is a polynomial of degree at most l. By Lemma 7.18, $(x - x_i)^m \mid g(x)$ for each point such that $y_i = p(x_i)$. Let S be the set of points where there is agreement such that $y_i = p(x_i)$, that is, $S = \{i \in \{1, \ldots, n\} : y_i = p(x_i)\}$ and let

$$s(x) = \prod_{i \in S}(x - x_i)^m.$$

Then $s(x) \mid g(x)$. Since $|S| \geq K_m$ (by hypothesis), we have $\deg s(x) \geq mK_m$. We thus have a polynomial of degree $\geq mK_m$ dividing $g(x)$ which is of degree $< mK_m$. It must therefore be the case that $g(x)$ is identically 0, or $Q(x, p(x)) = 0$. Now think of $Q(x, y)$ as a polynomial in y with coefficients in $\mathbb{F}[x]$. Employ the division algorithm to divide by $(y - p(x))$:

$$Q(x, y) = (y - p(x))q(x, y) + r(x).$$

Evaluating at $y = p(x)$ we have

$$0 = Q(x, p(x)) = r(x)$$

so that $(y - p(x)) \mid Q(x, p(x))$. □

The degree of $p(x)$ in Theorem 7.19 is at most v. Since $p(x)$ is to interpolate points as $y_i = p(x_i)$ and there is (by the Reed-Solomon encoding process) a polynomial relationship of degree $< k$ between the support set and the codewords, we must have $\deg p(x) < k$. We thus set

$$\boxed{v = k - 1.}$$

This establishes the weighted order to be used in the algorithm.

The Correction Distance

Let us now establish a connection between the correction distance t_m, the multiplicity m, and the maximum $(1, v)$-weighted degree l of $Q(x, y)$. The point of the interpolation theorem is that the number of variables in the interpolating polynomial must exceed the number of equations (constraints), which is $n\binom{m+1}{2}$. Recall from Lemma 7.14 that the number of monomials of weighted $(1, v)$-degree l is $C(v, l)$. So by the interpolation theorem (Theorem 7.17) we must have

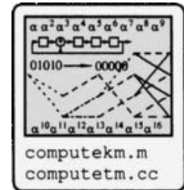
$$n\binom{m+1}{2} < C(v, l). \tag{7.73}$$

By the Factorization Theorem (Theorem 7.19) we must also have

$$mK_m > l \text{ or } mK_m \geq l + 1 \text{ or } mK_m - 1 \geq l. \tag{7.74}$$

Since $C(v, l)$ is increasing its second argument, replacing the second argument with a larger value makes it larger. Thus, from (7.73) and (7.74) we have

$$C(v, mK_m - 1) > n\binom{m+1}{2}. \tag{7.75}$$



For $m \geq 1$ we will *define* K_m to be the smallest value for which (7.75) is true:

$$K_m = \min\left\{K : C(v, mK - 1) > n\binom{m+1}{2}\right\}. \tag{7.76}$$

From the factorization theorem, K_m is the number of *agreements* between $\mathbf{y}$ and a codeword, so $t_m = n - K_m$ is the distance between $\mathbf{y}$ and a codeword; it is the error correction distance. For $m = 0$, we define K_m to be $n - t_0 = n - \lfloor(n - k)/2\rfloor = \lceil(n + v + 1)/2\rceil$. As the following example shows, K_m is non-increasing with m.

Example 7.16 Figure 7.3 shows K_m as a function of m for a (32, 8) Reed-Solomon code. There is an immediate decrease with m for small values of m, followed by a long plateau. At $m = 120$, K_m decreases to its final value — there are no more decreases beyond that. □

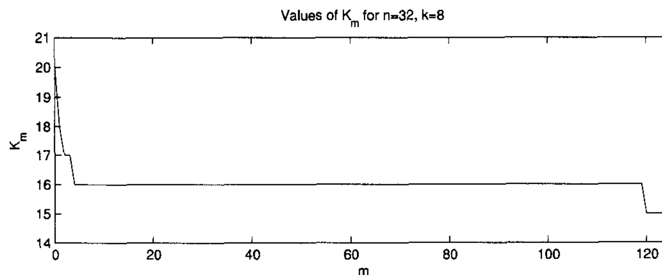


Figure 7.3: K_m as a function of m for a (32, 8) Reed-Solomon code.

There is a multiplicity m_0 beyond which $K_{m_0} = K_{m_0+1} = \cdots$ — no further decreases are possible. We denote this as K_∞. Since K_m is nonincreasing with m, $t_m = n - K_m$ is nondecreasing with m. That is, increasing the multiplicity m can increase the error correction distance, up till the point $t_\infty = n - K_\infty$ is reached, which is the asymptotic decoding capability of the GS decoder.

We will denote

$$K_\infty = \lfloor\sqrt{vn}\rfloor + 1$$

so that

$$t_\infty = n - K_\infty = n - 1 - \lfloor\sqrt{vn}\rfloor = n - 1 - \lfloor\sqrt{(k-1)n}\rfloor.$$

The following theorem indicates that the decoding distance of the GS decoder improves (or at least does not decrease) with increasing m. (As becomes evident below, the decoder algorithmic complexity increases with m, so this improved performance is obtained with higher complexity.)

Theorem 7.20 *[230] K_m is nonincreasing with m:*

$$K_0 \geq K_\infty \geq v + 1 \tag{7.77}$$

$$K_0 \geq K_1 \tag{7.78}$$

$$K_m \geq K_\infty \tag{7.79}$$

$$K_m \geq K_{m+1} \tag{7.80}$$

$$K_m = K_\infty \text{ for all sufficiently large } m. \tag{7.81}$$

Proof We will give only a partial proof. (The remaining results require bounds on $C(v, l)$ which are more fully developed in [230].) Proof of (7.77)[2]:

$$\begin{aligned} K_0 &= \lceil(n + v + 1)/2\rceil \geq \lfloor(n + v + 1)/2\rfloor + 1 \\ &\geq \lfloor\sqrt{n(v+1)}\rfloor \qquad \text{(arithmetic-geometric inequality)} \\ &\geq \lfloor\sqrt{nv}\rfloor + 1 = K_\infty. \end{aligned}$$

Proof of (7.81): It must be shown that for all sufficiently large m

$$C(v, mK_\infty - 1) > n\binom{m+1}{2}. \tag{7.82}$$

Using the bound (7.64) we have

$$C(v, mK_\infty - 1) \geq \frac{(mK_\infty - 1)(mK_\infty + 1)}{2v} > \frac{m^2K_m^2}{2v} = n\frac{m(m+1)}{2}\left(\frac{m}{m+1}\frac{K_\infty^2}{vn}\right).$$

So (7.82) holds if $\frac{m}{m+1}\frac{K_\infty^2}{vn} > 1$, or when

$$m > \left(\frac{K_\infty^2}{vn} - 1\right)^{-1}. \tag{7.83}$$

In order for the bound in (7.83) to make sense, the term on the right-hand side must be positive, which establishes the lower bound $K_\infty = \lfloor\sqrt{vn}\rfloor + 1$. Suppose it were the case that K_∞ were smaller, say, $K_\infty = \lfloor\sqrt{vn}\rfloor$. Then $(\lfloor\sqrt{vn}\rfloor/(vn) - 1)$ would be negative. Thus $K_\infty = \lfloor\sqrt{vn}\rfloor + 1$ is the smallest possible value. □

For an (n, k) decoder capable of correcting t errors, let $\tau = t/n$ denote the fraction of errors corrected and let $R = k/n$ denote the rate. For the conventional $t = t_0$ decoder, the

[2]The arithmetic-geometric inequality states that for positive numbers $z_1, z_2, \ldots, z_m$, $(z_1z_2\cdots z_m)^{1/m} \leq \frac{1}{m}(z_1 + z_2 + \cdots z_m)$; that is, the geometric mean is less than the arithmetic mean. Equality holds only in the case that all the z_i are equal.

fraction of errors corrected is (asymptotically) $\tau_0 = (1 - R)/2$. For the Guruswami-Sudan algorithm, $\tau_\infty = 1 - \sqrt{R}$ (asymptotically). Figure 7.4 shows the improvement in fraction of errors corrected as a function of rate. The increase in the decoding capability is substantial, particularly for low rate codes.

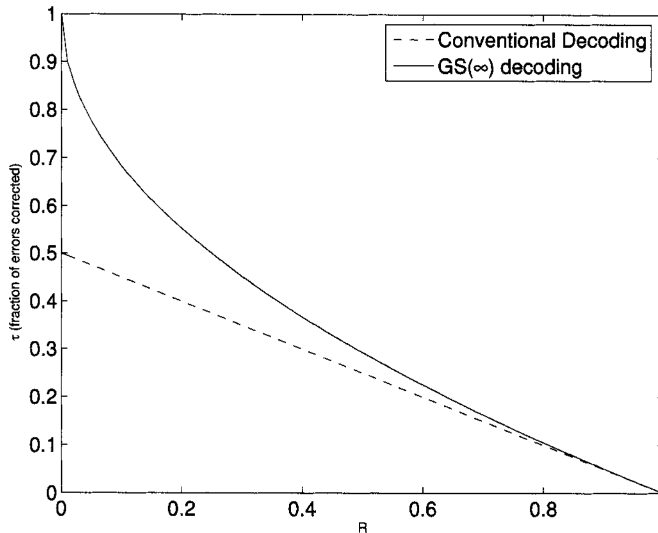


Figure 7.4: Fraction of errors corrected as a function of rate.

The Number of Polynomials in the Decoding List

The GS algorithm returns a list of polynomials $\mathcal{L} = \{p_1(x), p_2(x), \ldots, p_L(x)\}$. The transmitted codeword will be in $\mathcal{L}$ if the number of channel errors is $\leq t_m$. There may be other codewords as well, since this is a list decoding algorithm. How many polynomials can be in $\mathcal{L}$? (This material is drawn from [230].)

Recall that $Q(x, y)$ is a polynomial of $(1, v)$-weighted degree $\leq l$ and that the polynomials $p(x)$ are those such that $(y - p(x)) \mid Q(x, y)$. The maximum number of such polynomials is thus the y-degree of $Q(x, y)$. We denote this number as L_m.

Let $B(v, L)$ be the rank of the polynomial y^L with respect to the $(1, v)$-weighted revlex order.

Example 7.17 Using Table 7.1, we have

L	0	1	2	3	4	5
$B(3, L)$	0	4	11	21	34	50

□

Then

$$L_m = \max\{L : B(v, L) \leq n\binom{m+1}{2}\}.$$

(Because if there is a y^{L_m} in a monomial, then $x^i y^{L_m}$ has rank $> m\binom{m+1}{2}$ for $i > 0$.) We will develop an analytical expression for L_m and a bound for it.

Lemma 7.21

$$B(v, L) = \frac{vL^2}{2} + \frac{(v+2)L}{2}. \tag{7.84}$$

Proof Note (e.g., from Table 7.1) that y^L is the last monomial of $(1, v)$-degree L in revlex order, so that

$$B(v, L) = |\{(i, j) : i + vj \le Lv\}| - 1. \tag{7.85}$$

Then we have the recursive expression

$$\begin{aligned} B(v, L) &= (|\{(i, j) : i + vj \le (L-1)v\}| - 1) + |\{(i, j) : (L-1)v + 1 \le i + vj \le Lv\}| \\ &= B(v, L-1) + vL + 1. \end{aligned}$$

Then by induction,

$$\begin{aligned} B(v, L-1) + vL + 1 &= \frac{v(L-1)^2}{2} + \frac{(v+2)(L-1)}{2} + vL + 1 \\ &= \frac{vL^2}{2} + \frac{(v+2)L}{2} = B(v, L). \end{aligned}$$

□

Define the function $r_B(x)$ as that value of L such that $B(v, L) \le x \le B(v, L+1)$. That is,

$$r_B(x) = \arg\max\{L \in \mathbb{N} : B(v, L) \le x\}.$$

Then $L_m = r_B(n\binom{m+1}{2})$. Now we have a lemma relating r_B to $B(v, L)$.

Lemma 7.22 *If $B(v, x) = f(x)$ is a continuous increasing function of $x > 0$, taking integer values when x is integer, then*

$$r_B(x) = \lfloor f^{-1}(x) \rfloor.$$

More generally, if $g(x) \le B(v, x) \le f(x)$, where both f and g are continuous increasing functions of $x > 0$, then

$$\lfloor f^{-1}(x) \rfloor \le r_B(x) \le \lfloor g^{-1}(x) \rfloor. \tag{7.86}$$

Proof Let $L = r_B(x)$. By definition of $r_B(x)$ we have $B(v, L) \le x$. Invoking the inequalities associated with these quantities we have

$$g(L) \le B(v, L) \le x < B(v, L) + 1 \le f(L+1).$$

Thus $L \le g^{-1}(x)$ and $f^{-1}(x) < L + 1$. That is,

$$r_B(x) \le g^{-1}(x) \qquad \text{and} \qquad f^{-1}(x) < r_B(x) + 1,$$

or

$$f^{-1}(x) - 1 < r_B(x) \le g^{-1}(x).$$

Since $r_B(x)$ is integer valued, taking $\lfloor \cdot \rfloor$ throughout we obtain (7.86). □

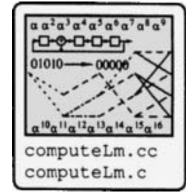
Using (7.84), we have $B(v, L) = f(L) = vL^2/2 + (v+2)L/2$. If $f(L) = x$, then (using the quadratic formula)

$$f^{-1}(x) = \sqrt{\left(\frac{v+2}{2v}\right)^2 + \left(\frac{2x}{v}\right)} - \left(\frac{v+2}{2v}\right).$$

Using Lemma 7.22 we reach the conclusion:

Theorem 7.23

$$L_m = r_B(n\binom{m+1}{2}) = \left\lfloor (f^{-1}(n\binom{m+1}{2}))\right\rfloor$$
$$= \left\lfloor \sqrt{\left(\frac{v+2}{2v}\right)^2 + \left(\frac{nm(m+1)}{v}\right)} - \left(\frac{v+2}{2v}\right)\right\rfloor. \quad (7.87)$$



A convenient upper bound is (see Exercise 7.22)

$$L_m < (m + \frac{1}{2})\sqrt{n/v}.$$

7.6.5 Algorithms for Computing the Interpolation Step

As observed in the proof of the Interpolation Theorem (Theorem 7.17), the interpolating polynomial can be found by solving the set of

$$C = n\binom{m+1}{2}$$

linear interpolation constraint equations with $> C$ unknowns, represented by (7.72). However, brute force numerical solutions (e.g., Gaussian elimination) would have complexity cubic in the size of the problem. In this section we develop two other solutions which have, in principle, lower complexity.

The interpolation constraint operations in (7.68) act as *linear functionals.*

Definition 7.15 A mapping $D : \mathbb{F}[x, y] \to \mathbb{F}$ is said to be a **linear functional** if for any polynomials $Q(x, y)$ and $P(x, y) \in \mathbb{F}[x, y]$ and any constants $u, v \in \mathbb{F}$,

$$D(uQ(x, y) + vP(x, y)) = uDQ(x, y) + vDP(x, y).$$

□

The operation

$$Q(x, y) \mapsto D_{r,s}Q(\alpha, \beta)$$

is an instance of a linear functional. We will recast the interpolation problem and solve it as a more general problem involving linear functionals: Find $Q(x, y)$ satisfying a set of constraints of the form

$$D_i Q(x, y) = 0, \qquad i = 1, 2, \ldots, C,$$

where each D_i is a linear functional. For our problem, each D_i corresponds to some $D_{r,s}$, according to a particular order relating i to (r, s). (But other linear functionals could also be used, making this a more general interpolation algorithm.)

Let us write $Q(x, y) = \sum_{j=0}^{J} a_j \phi_j(x, y)$, where the $\phi_j(x, y)$ are ordered with respect to some monomial order, and where $a_C \neq 0$. The upper limit J is bounded by $J \leq C$, where C is given by (7.70). The operation of any linear functional D_i on Q is

$$D_i Q = \sum_{j=0}^{J} a_j D_i \phi_j(x, y) \triangleq \sum_{j=0}^{J} a_j d_{i,j}, \quad (7.88)$$

with coefficients $d_{i,j} = D_i \phi_j(x, y)$.

Finding Linearly Dependent Columns: The Feng-Tzeng Algorithm

The set of functionals $D_1, D_2, \ldots, D_C$ can be represented as the columns of a matrix $\mathcal{D}$,

$$\mathcal{D} = \begin{bmatrix} d_{1,0} & d_{1,1} & \cdots & d_{1,J} \\ d_{2,0} & d_{2,1} & \cdots & d_{2,J} \\ \vdots & & & \\ d_{C,0} & d_{C,1} & \cdots & d_{C,J} \end{bmatrix} \triangleq \begin{bmatrix} \mathbf{d}^{(0)} & \mathbf{d}^{(1)} & \cdots & \mathbf{d}^{(J)} \end{bmatrix} \in \mathbb{F}^{C\times(J+1)}. \qquad (7.89)$$

Example 7.18 Over the field $GF(5)$, we desire to create a polynomial of minimal (1,3)-revlex rank through the following points, with the indicated multiplicities:

Point	Multiplicity
(1,1)	1
(2,3)	2
(4,2)	2

There are $1 + \binom{3}{2} + \binom{3}{2} = 1 + 3 + 3 = 7$ constraints. These constraints are (using the notation $Q_{r,s}$ introduced in (7.66))

$$\begin{array}{ccc} & Q_{0,0}(1,1) = 0 & \\ Q_{0,0}(2,3) = 0 & Q_{0,1}(2,3) = 0 & Q_{1,0}(2,3) = 0 \\ Q_{0,0}(4,2) = 0 & Q_{0,1}(4,2) = 0 & Q_{1,0}(4,2) = 0. \end{array}$$

With seven constraints, some linear combination of the first eight monomials listed in Table 7.1 suffices. These monomials are $1, x, x^2, x^3, y, x^4, xy$, and x^5. The polynomial we seek is

$$Q(x,y) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + a_4 y + a_5 x^4 + a_6 xy + a_7 x^5.$$

This polynomial should satisfy the constraints (using (7.67))

$$\begin{array}{lllll} D_{0,0}1 = 1 & D_{0,0}x = x & D_{0,0}y = y & \cdots & D_{0,0}x^5 = x^5 \\ D_{1,0}1 = 0 & D_{1,0}x = 1 & D_{1,0}y = 0 & \cdots & D_{1,0}x^5 = 5x^4 = 0 \\ D_{0,1}1 = 0 & D_{0,1}y = 1 & D_{0,1}y = 1 & \cdots & D_{0,1}x^5 = 0. \end{array}$$

Now form a matrix $\mathcal{D}$ whose columns correspond to the eight monomials and whose rows correspond to the seven constraints.

$$\mathcal{D} = \begin{array}{l} \\ Q_{0,0}(1,1): \\ Q_{0,0}(2,3): \\ Q_{0,1}(2,3): \\ Q_{1,0}(2,3): \\ Q_{0,0}(4,2): \\ Q_{0,1}(4,2): \\ Q_{1,0}(4,2): \end{array} \begin{array}{c} \begin{array}{cccccccc} 1 & x & x^2 & x^3 & y & x^4 & xy & x^5 \end{array} \\ \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 2 & 2^2 & 2^3 & 3 & 2^4 & 2\cdot 3 & 2^5 \\ 0 & 0 & 0 & 0 & 1 & 0 & 2 & 0 \\ 0 & 1 & 2\cdot 2 & 3\cdot 2^2 & 0 & 4\cdot 2^3 & 3 & 5\cdot 2^4 \\ 1 & 4 & 4^2 & 4^3 & 2 & 4^4 & 4\cdot 2 & 4^5 \\ 0 & 0 & 0 & 0 & 1 & 0 & 4 & 0 \\ 0 & 1 & 2\cdot 4 & 3\cdot 4^2 & 0 & 4\cdot 4^3 & 2 & 5\cdot 4^4 \end{bmatrix} \end{array}. \qquad (7.90)$$

□

The condition that all the constraints are simultaneously satisfied,

$$D_i Q(x,y) = 0, \qquad i = 1, 2, \ldots, C$$

can be expressed using the matrix $\mathcal{D}$ as

$$\mathcal{D}\begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_J \end{bmatrix} = \mathbf{0} \quad \text{or} \quad \mathbf{d}^{(0)}a_0 + \mathbf{d}^{(1)}a_1 + \cdots + \mathbf{d}^{(J)}a_J = 0,$$

so the columns of $\mathcal{D}$ are *linearly dependent*. The decoding problem can be expressed as follows:

> **Interpolation problem 1**: Determine the smallest J such that the first J columns of $\mathcal{D}$ are linearly dependent.

This is a problem in computational linear algebra, which may be solved by an extension of the Berlekamp-Massey (BM) algorithm known as the Feng-Tzeng algorithm. Recall that the BM algorithm determines the shortest linear-feedback shift register (LFSR) which annihilates a sequence of scalars. The Feng-Tzeng algorithm produces the shortest "LFSR" which annihilates a sequence of vectors.

We will express the problem solved by the Feng-Tzeng algorithm this way. Let

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1N} \\ a_{21} & a_{22} & \cdots & a_{2N} \\ \vdots & & & \\ a_{M1} & a_{M2} & \cdots & a_{MN} \end{bmatrix} = \begin{bmatrix} \mathbf{a}_1 & \mathbf{a}_2 & \cdots & \mathbf{a}_N \end{bmatrix}.$$

The first $l+1$ columns of A are linearly dependent if there exist coefficients $c_1, c_2, \ldots, c_l$, not all zero, such that

$$\mathbf{a}_{l+1} + c_1\mathbf{a}_l + \cdots c_l\mathbf{a}_1 = \mathbf{0}. \tag{7.91}$$

The problem is to determine the *minimum* l and the coefficients $c_1, c_2, \ldots, c_l$ such that the linear dependency (7.91) holds.

Let $C(x) = c_0 + c_1x + \cdots + c_lx^l$, where $c_0 = 1$, denote the set of coefficients in the linear combination. Let $\mathbf{a}(x) = \mathbf{a}_0 + \mathbf{a}_1x + \cdots + \mathbf{a}_Nx^N$ be a representation of the matrix A, with $\mathbf{a}_0 = 1$ (the vector of all ones) and let $a^{(i)}(x) = a_{i,0} + a_{i,1}x + \cdots + a_{i,N}x^N$ be the ith row of $\mathbf{a}(x)$. We will interpret $C(x)\mathbf{a}(x)$ element by element; that is,

$$C(x)\mathbf{a}(x) = \begin{bmatrix} C(x)a^{(1)}(x) \\ C(x)a^{(2)}(x) \\ \vdots \\ C(x)a^{(M)}(x) \end{bmatrix}.$$

For $n = l+1, l+2, \ldots, N$, let $[C(x)\mathbf{a}(x)]_n$ denote the coefficient (vector) of x^n in $C(x)\mathbf{a}(x)$. That is,

$$[C(x)\mathbf{a}(x)]_n = c_0\mathbf{a}_n + c_1\mathbf{a}_{n-1} + \cdots + c_l\mathbf{a}_{n-l} = \sum_{j=0}^{l} c_j\mathbf{a}_{n-j}.$$

The problem to be solved can be stated as follows: Determine the minimum l and a polynomial $C(x)$ with $\deg C(x) \leq l$ such that $[C(x)\mathbf{a}(x)]_{l+1} = 0$.

The general flavor of the algorithm is like that of the BM algorithm, with polynomials being updated if they result in a discrepancy. The algorithm proceeds element-by-element

through the matrix down the columns of the matrix. At each element, the discrepancy is computed. If the discrepancy is nonzero, previous columns on this row are examined to see if they had a nonzero discrepancy. If so, then the polynomial is updated using the previous polynomial that had the discrepancy. If there is no previous nonzero discrepancy on that row, then the discrepancy at that location is saved and the column is considered "blocked" — that is, no further work is done on that column and the algorithm moves to the next column.

Let $C^{(i-1,j)}(x) = c_0^{(i-1,j)} + c_1^{(i-1,j)}x + \cdots c_{j-1}^{(i-1,j)}x^{j-1}$, with $c_0^{(i-1,j)} = 1$, be defined for each column j, where $j = 1, 2, \ldots, l+1$, and for $i = 1, 2, \ldots, M$, where each polynomial $C^{(i-1,j)}$ has the property that

$$[C^{(i-1,j)}(x)a^{(h)}(x)]_j = a_{h,j} + c_1^{(i-1,j)}a_{h,j-1} + \cdots + c_{j-1}^{(i-1,j)}a_{h,1} = 0 \qquad \text{for } h \le i-1.$$

That is, in column j at position $(i-1, j)$ of the matrix and all previous rows there is *no* discrepancy. The initial polynomial for the first column is defined as $C^{(0,1)}(x) = 1$. The discrepancy at position (i, j) is computed as

$$d_{i,j} = [C^{(i-1,j)}(x)a^{(i)}(x)]_j = a_{i,j} + c_1^{(i-1,j)}a_{i,j-1} + \cdots + c_{j-1}^{(i-1,j)}a_{i,1}.$$

If $d_{i,j} = 0$, then no update to the polynomial is necessary and $C^{(i,j)}(x) = C^{(i-1,j)}(x)$. If, on the other hand, $d_{i,j} \neq 0$, then an update is necessary. If there is on row i a previous column u that had a nonzero discrepancy (that was not able to be resolved by updating the polynomial), then the polynomial is updated according to

$$C^{(i,j)}(x) = C^{(i-1,j)}(x) - \frac{d_{i,j}}{d_{i,u}}C^{(u)}(x). \tag{7.92}$$

where u is the column where the previous nonzero discrepancy occurred and $C^{(u)}(x)$ is the polynomial which had the nonzero discrepancy in column u.

If there is a nonzero discrepancy $d_{i,j}$, but there is no previous nonzero discrepancy on that row, then that discrepancy is saved, the row at which the discrepancy occurred is saved, $\rho(j) = i$, and the polynomial is saved, $C^j(x) = C^{(i-1,j)}(x)$. The column is considered "blocked," and processing continues on the next column with $C^{(0,j+1)}(x) = C^{(i-1,j)}(x)$.

The following lemma indicates that the update (7.92) zeros the discrepancy.

Lemma 7.24 *If $d_{i,j} \neq 0$ and there is a previous polynomial $C^{(u)}(x)$ at column u, so that $C^{(u)}(x) = C^{(i-1,u)}(x)$ and $d_{i,u} \neq 0$, then the update (7.92) is such that*

$$[C^{(i,j)}(x)a^{(r)}(x)]_j = 0 \qquad \text{for } r = 1, 2, \ldots, i.$$

Proof We have

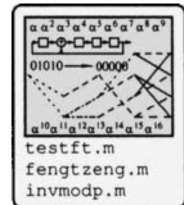
$$\begin{aligned}[C^{(i,j)}(x)a^{(r)}(x)]_j &= [C^{(i-1,j)}(x), a^{(r)}(x)]_j - \frac{d_{i,j}}{d_{i,u}}[C^{(u)}(x)a^{(r)}(x)x^{j-u}]_j \\ &= [C^{(i-1,j)}(x), a^{(r)}(x)]_j - \frac{d_{i,j}}{d_{i,u}}[C^{(u)}(x)a^{(r)}(x)]_u \\ &= \begin{cases} 0 - \frac{d_{i,j}}{d_{i,u}}0 & \text{for } r = 0, 1, \ldots, i-1 \\ d_{i,j} - \frac{d_{i,j}}{d_{i,u}}d_{i,u} & \text{for } r = i, \end{cases}\end{aligned}$$

where the second equality follows from a result in Exercise 7.18. □

Algorithm 7.5 The Feng-Tzeng Algorithm

This representation is due to McEliece [230]

1 **Input:** A matrix A of size $M \times N$
2 **Output:** A polynomial C of minimal degree annihilating columns of A
3 **Initialize:** $s = 0$ (column counter)
4 $C = 1$ (C holds the current polynomial)
5 `dsave` = zeros(1,N) (holds nonzero discrepancies)
6 ρ = zeros(1,N) (holds row where nonzero discrepancy is)
7 **Begin:**
8 while(1) (loop over columns)
9 $s = s + 1$; $r = 0$; (move to beginning of next column)
10 `columnblocked` = 0;
11 while(1) (loop over rows in this column)
12 $r = r + 1$; (move to next row)
13 $d_{rs} = [C(x)a^{(r)}(x)]_s$ (compute discrepancy here using current poly.)
14 if($d_{rs} \neq 0$) (if nonzero discrepancy)
15 if(there is a u such that $\rho(u) = r$) (if a previous nonzero disc. on this row)
16 $C(x) = C(x) - \frac{d_{rs}}{\texttt{dsave}(u)} C_u(x) x^{s-u}$; (update polynomial)
17 else (no previous nonzero discrepancy on this row)
18 $\rho(s) = r$; (save row location of nonzero discrepancy)
19 $C_s(x) = C(x)$; (save polynomial for this column)
20 `dsave`$(s) = d_{rs}$; (save nonzero discrepancy for this column)
21 `columnblocked` = 1; (do no more work on this column)
22 end (else)
23 end (if d_{rs})
24 if($r \geq M$ or `columnblocked=1`) break; end; (end of loop over row)
25 end (while (1))
26 if(`columnblocked=0`) break; end; (end loop over columns)
27 end (while(1))
28 End



It can be shown that the polynomial $C(x)$ produced by this algorithm results in the *minimal* number of first columns of A which are linearly dependent [83].

Example 7.19 We apply the algorithm to the matrix $\mathcal{D}$ of (7.90) in Example 7.18. The following matrix outlines the steps of the algorithm.

$$
\begin{bmatrix}
1_a & \boxed{1}_b & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1_c & \boxed{2}_d & 0 & \boxed{2}_h & 0 & \boxed{1}_m & 0 \\
0 & 0 & 0 & 0 & 1_i & 0 & \boxed{3}_n & 0 \\
0 & 0 & 1_e & \boxed{2}_f & 0 & \boxed{1}_j & \boxed{2}_o & 0 \\
0 & 0 & 0 & 2_g & 0 & \boxed{4}_k & \boxed{1}_p & \boxed{1}_r \\
0 & 0 & 0 & 0 & 0 & 0 & 2_q & 0 \\
0 & 0 & 0 & 0 & 0 & 2_l & 0 & \boxed{2}_s
\end{bmatrix}
$$

The initially nonzero discrepancies are shown in this matrix; those which are in squares resulted in polynomials updated by (7.92) and the discrepancy was zeroed.

a: Starting with $C(x) = 1$, a nonzero discrepancy is found. There are no previous nonzero discrepancies on this row, so $C^{(1)}(x) = 1$ is saved and we jump to the next column.

b: Nonzero discrepancy found, but the polynomial was updated using previous nonzero discrepancy on this row (at a): $C(x) = 4x + 1$.

c: Nonzero discrepancy found and no update is possible. Save $C^{(2)}(x) = 4x + 1$ and the discrepancy and jump to next column.

d: Nonzero discrepancy found, but updated polynomial computed (using polynomial at c) $C(x) = 2x^2 + 2x + 1$.

e: Nonzero discrepancy found; no update possible. Save $C^{(3)}(x) = 2x^2 + 2x + 1$ and the discrepancy and and jump to next column.

f: Update polynomial: $C(x) = x^3 + 3x^2 + 1$

g: Save $C^{(4)}(x) = x^3 + 3x^2 + 1$

h: Update polynomial: $C(x) = 2x^4 + 4x^3 + 3x^2 + 1$

i: Save $C^{(5)}(x) = 2x^4 + 4x^3 + 3x^2 + 1$

j: Update polynomial: $C(x) = 3x^5 + 3x^3 + 3x^2 + 1$

k: Update polynomial: $C(x) = x^5 + 4x^4 + 3x^3 + 1x^2 + 1$

l: Save $C^{(6)}(x) = x^5 + 4x^4 + 3x^3 + 1x^2 + 1$

m: Update polynomial $C(x) = x^6 + 4x^4 + 3x^3 + x^2 + 1$

n: Update polynomial $C(x) = 3x^5 + 3x^3 + 3x^2 + 1$

o: Update polynomial $C(x) = x^6 + 4x^5 + 3x^4 + 3x^3 + 3x^2 + 1$

p: Update polynomial $C(x) = 3x^6 + 3x^4 + 3x^2 + 1$

q: Save $C^{(7)}(x) = 3x^6 + 3x^4 + 3x^2 + 1$

r: Update polynomial $C(x) = 2x^7 + 4x^6 + 3x^2 + 1$

s: Update polynomial $C(x) = x^7 + 2x^5 + 4x^4 + 2x^2 + 1$

Returning to the interpolation problem of Example 7.18, we obtain from the coefficients of $C(x)$ the coefficients of the polynomial

$$Q(x, y) = 1 + 0x + 2x^2 + 4x^3 + 0y + 2x^4 + 0xy + x^5.$$

It can be easily verified that this polynomial satisfies the interpolation and multiplicity constraints specified in Example 7.18. □

The computational complexity goes as the cube of the size of the matrix. One view of this algorithm is that it is simply a restatement of conventional Gauss-Jordan reduction and has similar computational complexity.

Finding the Intersection of Kernels: The Kötter Algorithm

Let $\mathbb{F}_L[x, y] \subset \mathbb{F}[x, y]$ denote the set of polynomials whose y-degree is $\leq L$. (The variable y is distinguished here because eventually we will be looking for y-roots of $Q(x, y)$.) Then any $Q(x, y) \in \mathbb{F}_L[x, y]$ can be written in the form

$$Q(x, y) = \sum_{k=0}^{L} q_k(x) y^k$$

for polynomials $q_k(x) \in \mathbb{F}[x]$. $\mathbb{F}_L[x, y]$ is an $\mathbb{F}[x]$-module (see Section 7.4.1): for any polynomials $a(x)$ and $b(x)$ in $\mathbb{F}[x]$ and polynomials $Q(x, y)$ and $P(x, y)$ in $\mathbb{F}_L[x, y]$,

$$(a(x)P(x, y) + b(x)Q(x, y)) \in \mathbb{F}_L[x, y]$$

since the y-degree of the linear combination does not change.

For a linear functional D, we will generically write K_D as the kernel of D:

$$K_D = \ker D = \{Q(x, y) \in \mathbb{F}[x, y] : DQ(x, y) = 0\}.$$

For a set of linear functionals $D_1, D_2, \ldots, D_C$ defined on $\mathbb{F}_L[x, y]$, let $K_1, K_2, \ldots, K_C$ be their corresponding kernels, so that

$$K_i = \ker D_i = \{Q(x, y) \in \mathbb{F}[x, y] : D_i Q(x, y) = 0\}.$$

Then a solution to the problem

$$D_i Q(x, y) = 0 \text{ for all } i = 1, 2, \ldots, C \tag{7.93}$$

lies in the intersection of the kernels $K = K_1 \cap K_2 \cap \cdots K_C$. We see that the interpolation problem can be expressed as follows:

Interpolation Problem 2: Determine the polynomial of minimal rank in K.

To find the intersection constructively, we will employ *cumulative kernels*, defined as follows: $\overline{K}_0 = \mathbb{F}_L[x, y]$ and

$$\overline{K}_i = \overline{K}_{i-1} \cap K_i = K_1 \cap \cdots \cap K_i.$$

That is, $\overline{K}_i$ is the space of solutions of the first i problems in (7.93). The solution of the interpolation is a polynomial of minimum $(1, v)$-degree in $\overline{K}_C$.

We will partition the polynomials in $\mathbb{F}_L[x, y]$ according to the exponent of y. Let

$$S_j = \{Q(x, y) \in \mathbb{F}_L[x, y] : \mathrm{LM}(Q) = x^i y^j \text{ for some } i\}$$

be the set of polynomials whose leading monomial has y-degree j. Let $g_{i,j}$ be the minimal element of $\overline{K}_i \cap S_j$, where here and throughout the development "minimal" or "min" means *minimal rank*, with respect to a given monomial order. Then $\{g_{C,j}\}_{j=0}^{L}$ is a set of polynomials that satisfy all of the constraints (7.93).

The Kötter algorithm generates a sequence of sets of polynomials $(G_0, G_1, \ldots, G_C)$, where

$$G_i = (g_{i,0}, g_{i,1}, \ldots, g_{i,L}),$$

and where $g_{i,j}$ is a minimal element of $\overline{K}_i \cap S_j$. (That is, it satisfies the first i constraints and the y-degree is j.) Then the output of the algorithm is the element of G_C of minimal order in the set G_C which has polynomials satisfying all C constraints:

$$Q(x, y) = \min_{0 \le j \le L} g_{C,j}(x, y).$$

This satisfies all the constraints (since it is in $\overline{K}_C$) and is of minimal order.

We introduce a linear functional and some important properties associated it. For a given linear functional D, define the mapping $[\cdot, \cdot]_D : \mathbb{F}[x, y] \times \mathbb{F}[x, y] \to \mathbb{F}[x, y]$ by

$$[P(x, y), Q(x, y)]_D = (DQ(x, y))P(x, y) - (DP(x, y))Q(x, y).$$

Lemma 7.25 *For all* $P(x, y), Q(x, y) \in \mathbb{F}[x, y]$, $[P(x, y), Q(x, y)]_D \in \ker D$. *Furthermore, if* $P(x, y) > Q(x, y)$ *(with respect to some fixed monomial order) and* $Q(x, y) \notin K_D$, *then* $\operatorname{rank}[P(x, y), Q(x, y)]_D = \operatorname{rank} P(x, y)$.

Proof We will prove the latter statement of the lemma (the first part is in Exercise 7.19). For notational convenience, let $a = DQ(x, y)$ and $b = DP(x, y)$. (Recall that $a, b \in \mathbb{F}$.) If $a \neq 0$ and $P(x, y) > Q(x, y)$, then $[P(x, y), Q(x, y)]_D = aP(x, y) - bQ(x, y)$, which does not change the leading monomial, so that $\text{LM}[P(x, y), Q(x, y)]_D = \text{LM}P(x, y)$ and furthermore, $\text{rank}[P(x, y), Q(x, y)]_D = \text{rank}\, P(x, y)$. □

The algorithm is initialized with

$$G_0 = (g_{0,0}, g_{0,1}, \ldots, g_{0,L}) = (1, y, y^2, \ldots, y^L).$$

To form G_{i+1} given the set G_i, we form the set

$$J_i = \{j : D_{i+1}(g_{i,j}) \neq 0\}$$

as the set of polynomials in G_i which do not satisfy the $i + 1$st constraint. If J is not empty, then an update is necessary (i.e., there is a discrepancy). In this case, let j^* index the polynomial of minimal rank,

$$j^* = \arg\min_{j \in J_i} g_{i,j}$$

and let f denote the polynomial g_{i,j^*}:

$$f = \min_{j \in J_i} g_{i,j}. \tag{7.94}$$

The update rule is as follows:

$$g_{i+1,j} = \begin{cases} g_{i,j} & \text{if } j \notin J_i \\ [g_{i,j}, f]_{D_{i+1}} & \text{if } j \in J_i \text{ but } j \neq j^* \\ [xf, f]_{D_{i+1}} & \text{if } j = j^*. \end{cases} \tag{7.95}$$

The key theorem governing this algorithm is the following.

Theorem 7.26 *For* $i = 0, \ldots, C$,

$$g_{i,j} = \min\{g : g \in \overline{K}_i \cap S_j\} \quad \textit{for } j = 0, 1, \ldots, L. \tag{7.96}$$

Proof The proof is by induction on i. The result is trivial when $i = 0$. It is to be shown that

$$g_{i+1,j} = \min\{g : g \in \overline{K}_{i+1} \cap S_j\} \quad \text{for } j = 0, 1, \ldots, L$$

is true, given that (7.96) is true. We consider separately the three cases in (7.95).

Case 1: $j \notin J_i$, so that $g_{i+1,j} = g_{i,j}$. The constraint $D_{i+1}g_{i,j} = 0$ is satisfied (since $j \notin J_i$), so $g_{i+1,j} = g_{i,j} \in K_{i+1}$. By the inductive hypothesis, $g_{i+1,j} \in \overline{K}_i \cap S_j$. Combining these, we have $g_{i+1,j} \in \overline{K}_{i+1} \cap S_j$. Since $g_{i,j}$ is minimal in $\overline{K}_i \cap S_j$, it must also be minimal in the set $\overline{K}_{i+1} \cap S_j$, since the latter is contained in the former.

Case 2: In this case,

$$g_{i+1,j} = [g_{i,j}, f]_{D_{i+1}} = (D_{i+1}g_{i,j})f - (D_{i+1}f)g_{i,j},$$

which is a linear combination of f and $g_{i,j}$. Since both f and $g_{i,j}$ are in $\overline{K}_i$, $g_{i+1,j}$ is also in $\overline{K}_i$. By Lemma 7.25, $g_{i+1,j} \in K_{i+1}$. Combining these inclusions, $g_{i+1,j} \in \overline{K}_i \cap K_{i+1} = \overline{K}_{i+1}$.

By (7.94), $\text{rank}\, g_{i,j} > \text{rank}\, f$, so that by Lemma 7.25, $\text{rank}\, g_{i+1,j} = \text{rank}\, g_{i,j}$. Since $g_{i,j} \in S_j$, it follows that $g_{i+1,j} \in S_j$ also. And, since $g_{i+1,j}$ has the same rank as $g_{i,j}$, which is minimal in $\overline{K}_i \cap S_j$, it must also be minimal in the smaller set $\overline{K}_{i+1} \cap S_j$.

Case 3: In this case the update is

$$g_{i+1,j} = [xf, f]_{D_{i+1}} = (D_{i+1}xf)f - (D_{i+1}f)xf$$

which is a linear combination of f and xf. But $f \in \overline{K}_i$ by the induction hypothesis.

We must show that $xf \in \overline{K}_i$. Let $\tilde{f}(x, y) = xf(x, y)$. In Exercise 7.21 it is shown that, in terms of the Hasse partial derivatives of f and $\tilde{f}$,

$$\tilde{f}_{r,s}(x, y) = f_{r-1,s}(x, y) + xf_{r,s}(x, y). \tag{7.97}$$

If $f_{r-1,s}(x_j, y_j) = 0$ and $f_{r,s}(x_j, y_j) = 0$ for $j = 0, 1, \ldots, i$, then $\tilde{f}_{r,s}(x_j, y_j) = 0$, so $xf \in \overline{K}_i$. This will hold provided that the sequence of linear functionals $(D_0, D_1, \ldots, D_C)$ are ordered such that $D_{r-1,s}$ always precedes $D_{r,s}$.

Assuming this to be the case, we conclude that $f \in \overline{K}_i$ and $xf \in \overline{K}_i$, so that $g_{i+1,j} \in \overline{K}_i$. By Lemma 7.25, $g_{i+1,j} \in K_{i+1}$, so $g_{i+1,j} \in \overline{K}_i \cap K_{i+1} = \overline{K}_{i+1}$.

Since $f \in S_j$ (by the induction hypothesis), then $xf \in S_j$ (since multiplication by x does not change the y-degree). By Lemma 7.25, rank $g_{i+1,j} =$ rank xf, which means that $g_{i+1,j} \in S_j$ also.

Showing that $g_{i+1,j}$ is minimal is by contradiction. Suppose there exists a polynomial $h \in \overline{K}_{i+1} \cap S_j$ such that $h < g_{i+1,j}$. Since $h \in \overline{K}_i \cap S_j$, we must have $f \leq h$ and rank $g_{i+1,j} =$ rank xf. There can be no polynomial $f' \in S_j$ with rank $f <$ rank $f' <$ rank xf, so that it follows that $\text{LM}(h) = \text{LM}(f)$. By suitable normalization, the leading coefficients of h and f can be equated. Now let

$$\tilde{f} = h - f.$$

By linearity, $\tilde{f} \in \overline{K}_i$. By cancellation of the leading terms, $\tilde{f} < f$. Now $D_{i+1}h = 0$, since $h \in \overline{K}_{i+1}$, but $D_{i+1}f \neq 0$, since $j \in J_i$ in (7.94). Thus we have a polynomial $\tilde{f}$ such that $\tilde{f} \in \overline{K}_i \setminus \overline{K}_{i+1}$ and $\tilde{f} < f$. But f was supposed to be the minimal element of $\overline{K}_i \setminus \overline{K}_{i+1}$, by its selection in (7.94). This leads to a contradiction: $g_{i+1,j}$ must be minimal. □

Let us return to the ordering condition raised in Case 3. We must have an order in which $(r-1, s)$ precedes (r, s). This is accomplished when the (r, s) data are ordered according to $(m-1, 1)$ lex order:

$$(0, 0), (0, 1), \ldots, (0, m-1), (1, 0), (1, 1), \ldots, (1, m-2), \ldots, (m-1, 0).$$

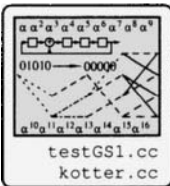
At the end of the algorithm, we select the minimal element out of G_C as $Q_0(x, y)$.

Kötter's algorithm for polynomial interpolation is shown in Algorithm 7.6. This algorithm is slightly more general than just described: the point (x_i, y_i) is interpolated up to order m_i, where m_i can vary with i, rather than having a fixed order m at each point. There is one more explanation necessary, regarding line 19. In line 19, the update is computed as

$$\begin{aligned}
g_{i,j} &= \Delta\, xf - (D_i xf)f = (D_{r,s}f(x_i, y_i))(xf(x, y) - (D_{r,s}xf(x, y)))\Big|_{x=x_i, y=y_i} f(x, y) \\
&= D_{r,s}f(x_i, y_i) - \left[x_i D_{r,s}f(x_i, y_i) + D_{r-1,s}f(x_i, y_i)\right] f(x, y) \qquad \text{(using (7.97))} \\
&= D_{r,s}f(x_i, y_i)f(f, x)(x - x_i) \qquad \text{(since } D_{r-1,s}f(x_i, y_i) = 0\text{)} \\
&= (\text{const})(x - x_i)f(x, y).
\end{aligned}$$

Algorithm 7.6 Kötter's Interpolation for Guruswami-Sudan Decoder

1 **Input:** Points: $(x_i, y_i), i = 1, \ldots, n$; Interpolation order m_i; a $(1, v)$ monomial order; $L = L_m$
2 **Returns:** $Q_0(x, y)$ satisfying the interpolation problem.
3 **Initialize:** $g_j = y^j$ for $j = 0$ to L.
4 for $i = 1$ to n (go from $i - 1$st stage to ith stage)
5 $C = (m_i + 1)m_i/2$ (compute number of derivatives involved)
6 for $(r, s) = (0, 0)$ to $(m_i - 1, 0)$ by $(m_i - 1, 1)$ lex order (from 0 to C)
7 for $j = 0$ to L
8 $\Delta_j = D_{r,s} g_j(x_i, y_i)$ (compute "discrepancy")
9 end (for j)
10 $J = \{j : \Delta_j \neq 0\}$ (set of nonzero discrepancies)
11 if($J \neq \emptyset$)
12 $j^* = \arg\min\{g_j : j \in J\}$ (polynomial of least weighted degree)
13 $f = g_{j^*}$
14 $\Delta = \Delta_{j^*}$
15 for($j \in J$)
16 if($j \neq j^*$)
17 $g_j = \Delta g_j - \Delta_j f$ (update without change in rank)
18 else if($j = j^*$)
19 $g_j = (x - x_i) f$ (update with change in rank)
20 end (if)
21 end (if j)
22 end (for J)
23 end (for (r, s))
24 end (for i)
25 $Q_0(x, y) = \min_j \{g_j(x, y)\}$ (least weighted degree)



Example 7.20 The points $(1, \alpha^3)$, (α, α^4), (α^2, α^5), (α^3, α^7) and (α^4, α^8) are to be interpolated by a polynomial $Q(x, y)$ using the Kötter algorithm, where operations are over $GF(2^4)$ with $1+\alpha+\alpha^4 = 0$. Use multiplicity $m = 1$ interpolation at each point and the $(1, 2)$-revlex order.

At each point, there is one constraint, since $m = 1$.

Initial: G_0: $g_0(x, y) = 1$, $g_1(x, y) = y$, $g_2(x, y) = y^2$, $g_3(x, y) = y^3$, $g_4(x, y) = y^4$.

$i = 0$: $(r, s) = (0, 0)$, $(x_i, y_i) = (1, \alpha^3)$.

Discrepancies: $\Delta_0 = g_0(1, \alpha^3) = 1$, $\Delta_1 = g_1(1, \alpha^3) = \alpha^3$, $\Delta_2 = g_2(1, \alpha^3) = \alpha^6$, $\Delta_3 = g_3(1, \alpha^3) = \alpha^9$, $\Delta_4 = g_4(1, \alpha^3) = \alpha^{12}$. $J = \{0, 1, 2, 3, 4\}$, $j^* = 0$, $f = 1$, $\Delta = 1$

G_1: $g_0 = 1 + x$, $g_1 = \alpha^3 + y$, $g_2 = \alpha^6 + y^2$, $g_3 = \alpha^9 + y^3$, $g_4 = \alpha^{12} + y^4$.

$i = 1$: $(r, s) = (0, 0)$, $(x_i, y_i) = (\alpha, \alpha^4)$.

Discrepancies: $\Delta_0 = \alpha^4$, $\Delta_1 = \alpha^7$, $\Delta_2 = \alpha^{14}$, $\Delta_3 = \alpha^8$, $\Delta_4 = \alpha^{13}$. $J = \{0, 1, 2, 3, 4\}$, $j^* = 0$, $f = 1 + x$, $\Delta = \alpha^4$.

G_2: $g_0 = \alpha + \alpha^4 x + x^2$, $g_1 = \alpha^7 x + \alpha^4 y$, $g_2 = (\alpha^{11} + \alpha^{14} x) + \alpha^4 y^2$, $g_3 = (\alpha^3 + \alpha^8 x) + \alpha^4 y^3$, $g_4 = (\alpha^{12} + \alpha^{13} x) + \alpha^4 y^4$.

$i = 2$: $(r, s) = (0, 0)$, $(x_i, y_i) = (\alpha^2, \alpha^5)$.

Discrepancies: $\Delta_0 = \alpha^{13}$, $\Delta_1 = 0$, $\Delta_2 = \alpha^8$, $\Delta_3 = \alpha^6$, $\Delta_4 = \alpha^2$. $J = \{0, 2, 3, 4\}$, $j^* = 0$, $f = \alpha + \alpha^4 x + x^2$, $\Delta = \alpha^{13}$.

G_3: $g_0 = \alpha^3 + \alpha^{11} x + \alpha^{10} x^2 + x^3$, $g_1 = \alpha^7 x + \alpha^4 y$, $g_2 = \alpha^8 x^2 + \alpha^2 y^2$, $g_3 = (\alpha^{14} + \alpha^7 x + \alpha^6 x^2) + \alpha^2 y^3$, $g_4 = (\alpha^{12} + \alpha x + \alpha^2 x^2) + \alpha^2 y^4$.

$i = 3$: $(r, s) = (0, 0)$, $(x_i, y_i) = (\alpha^3, \alpha^7)$.

Discrepancies: $\Delta_0 = \alpha^{14}$, $\Delta_1 = \alpha^{14}$, $\Delta_2 = \alpha^7$, $\Delta_3 = \alpha^2$, $\Delta_4 = \alpha^3$. $J = \{0, 1, 2, 3, 4\}$, $j^* = 1$, $f = \alpha^7 x + \alpha^4 y$.

G_4: $g_0 = (\alpha^2 + \alpha^7 x + \alpha^9 x^2 + \alpha^{14} x^3) + (\alpha^3) y$, $g_1 = (\alpha^{10} x + \alpha^7 x^2) + (\alpha^7 + \alpha^4 x) y$, $g_2 = (\alpha^{14} x + \alpha^7 x^2) + \alpha^{11} y + \alpha y^2$, $g_3 = (\alpha^{13} + \alpha^5 x + \alpha^5 x^2) + \alpha^6 y + \alpha y^3$, $g_4 = (\alpha^{11} + \alpha^5 x + \alpha x^2) + \alpha^7 y + \alpha y^4$.

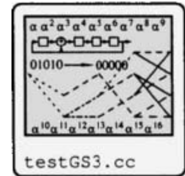
$i = 4$: $(r, s) = (0, 0)$, $(x_i, y_i) = (\alpha^4, \alpha^8)$.

Discrepancies: $\Delta_0 = \alpha^{11}$, $\Delta_1 = \alpha^8$, $\Delta_2 = \alpha^{11}$, $\Delta_3 = \alpha^2$, $\Delta_4 = \alpha^{10}$. $J = \{0, 1, 2, 3, 4\}$, $j^* = 0$, $f = (\alpha^2 + \alpha^7 x + \alpha^9 x^2 + \alpha^{14} x^3) + (\alpha^3) y$.

G_5: $g_0 = (\alpha^6 + \alpha^9 x + \alpha^5 x^2 + \alpha x^3 + \alpha^{14} x^4) + (\alpha^7 + \alpha^3 x) y$, $g_1 = (\alpha^9 + \alpha^8 x + \alpha^9 x^2 + \alpha^6 x^3) + (\alpha^{12} + x) y$, $g_2 = (\alpha^{13} + \alpha^{12} x + \alpha^{11} x^2 + \alpha^{10} x^3) + \alpha y + \alpha^{12} y^2$, $g_3 = (\alpha^{14} + \alpha^3 x + \alpha^6 x^2 + \alpha x^3) + \alpha y + \alpha^{12} y^3$, $g_4 = (\alpha^2 + \alpha^5 x + \alpha^6 x^2 + \alpha^9 x^3) + \alpha^8 y + \alpha^{12} y^4$.

Final: $Q_0(x, y) = g_1(x, y) = (\alpha^9 + \alpha^8 x + \alpha^9 x^2 + \alpha^6 x^3) + (\alpha^{12} + x) y$.

□


testGS3.cc

Example 7.21 Let C be a $(15, 7)$ code over $GF(2^4)$ and let $m(x) = \alpha + \alpha^2 x + \alpha^3 x^2 + \alpha^4 x^3 + \alpha^5 x^4 + \alpha^6 x^5 + \alpha^7 x^6$ be encoded over the support set $(1, \alpha, \ldots, \alpha^{14})$. That is, the codeword is

$$(m(1), m(\alpha), m(\alpha^2), \ldots, m(\alpha^{14})).$$

The corresponding code polynomial is

$$c(x) = \alpha^6 + \alpha^{11} x + x^2 + \alpha^6 x^3 + \alpha x^4 + \alpha^{14} x^5 + \alpha^8 x^6 + \alpha^{11} x^7 + \alpha^8 x^8 + \alpha x^9 + \alpha^{12} x^{10} + \alpha^{12} x^{11} + \alpha^{14} x^{12} + x^{13} + \alpha x^{14}.$$

Suppose the received polynomial is

$$r(x) = \alpha^6 + \alpha^9 x + x^2 + \alpha^2 x^3 + \alpha x^4 + \alpha^9 x^5 + \alpha^8 x^6 + \alpha^3 x^7 + \alpha^8 x^8 + \alpha x^9 + \alpha^{12} x^{10} + \alpha^{12} x^{11} + \alpha^{14} x^{12} + x^{13} + \alpha x^{14}.$$

That is, the error polynomial is

$$e(x) = \alpha^2 x + \alpha^3 x^3 + \alpha^4 x^5 + \alpha^5 x^7.$$

For this code and interpolation multiplicity, let

$$t_m = 4 \qquad L_m = 3.$$

The first decoding step is to determine an interpolating polynomial for the points

$$(1, \alpha^6), (\alpha, \alpha^9), (\alpha^2, 1), (\alpha^3, \alpha^2), (\alpha^4, \alpha), (\alpha^5, \alpha^9), (\alpha^6, \alpha^8), (\alpha^7, \alpha^3),$$
$$(\alpha^8, \alpha^8), (\alpha^9, \alpha), (\alpha^{10}, \alpha^{12}), (\alpha^{11}, \alpha^{12}), (\alpha^{12}, \alpha^{14}), (\alpha^{13}, 1), (\alpha^{14}, \alpha)$$

with interpolation multiplicity $m = 2$ at each point using the $(1, 6)$-revlex degree.

Using `testGS3`, the final set of interpolating polynomials $G_C = \{g_0, g_1, g_2, g_3\}$ can be shown to be

$$\begin{aligned}
g_0(x,y) = {} & (\alpha^{13} + \alpha x + \alpha^4 x^2 + \alpha^6 x^3 + \alpha^{13} x^4 + \alpha^{11} x^5 + \alpha^{10} x^6 + \alpha^8 x^8 + \alpha^8 x^9 \\
& + \alpha^5 x^{10} + \alpha^9 x^{11} + \alpha^3 x^{12} + \alpha^6 x^{13} + \alpha^3 x^{14} + \alpha^2 x^{15} + \alpha^{10} x^{16} + \alpha^7 x^{17} \\
& + \alpha^{11} x^{18} + \alpha^8 x^{19} + \alpha^3 x^{20} + \alpha^{14} x^{21} + \alpha^{14} x^{22}) + (\alpha^6 + \alpha^5 x + \alpha^3 x^2 + \alpha^2 x^3 \\
& + \alpha^{13} x^4 + \alpha^7 x^5 + \alpha^7 x^6 + x^7 + \alpha^7 x^8 + \alpha^{11} x^9 + \alpha^{14} x^{10} + \alpha^{12} x^{11} + \alpha^5 x^{12} \\
& + \alpha^8 x^{13} + \alpha^{13} x^{14} + \alpha^9 x^{15}) y + (\alpha^{13} + \alpha^9 x + \alpha^{11} x^2 + \alpha^7 x^3 + x^4 + \alpha^{10} x^5 \\
& + \alpha^{11} x^6 + \alpha^5 x^7 + \alpha^5 x^8 + \alpha^7 x^9) y^2 + (1 + \alpha^2 x + \alpha^2 x^2 + \alpha^4 x^3) y^3 \\
g_1(x,y) = {} & (\alpha^{13} + \alpha^{13} x + \alpha^{14} x^2 + x^3 + \alpha^3 x^4 + \alpha^6 x^5 + \alpha^{12} x^6 + \alpha^{14} x^7 + \alpha^8 x^8 + \alpha^6 x^9 \\
& + \alpha^8 x^{10} + \alpha^7 x^{11} + \alpha^{13} x^{12} + \alpha x^{13} + \alpha^{11} x^{14} + \alpha^{10} x^{15} + x^{16} + \alpha^9 x^{17} + \alpha^{14} x^{18} \\
& + \alpha^3 x^{19} + \alpha^3 x^{21}) + (\alpha^2 + \alpha^9 x + \alpha^{11} x^2 + \alpha x^3 + \alpha^{11} x^4 + \alpha^{10} x^5 + \alpha^5 x^6 + \alpha x^7 \\
& + \alpha^6 x^8 + \alpha^{10} x^{10} + \alpha^7 x^{11} + \alpha^{10} x^{12} + \alpha^{13} x^{13} + \alpha^4 x^{14}) y + (\alpha^4 + \alpha^5 x + \alpha^{12} x^2 \\
& + \alpha^{12} x^3 + \alpha^{11} x^4 + \alpha^5 x^5 + \alpha^7 x^6 + \alpha x^7) y^2 + (\alpha^{11} + \alpha x + \alpha^{14} x^2) y^3 \\
g_2(x,y) = {} & (\alpha^{13} + \alpha^{11} x + \alpha^5 x^2 + \alpha^{13} x^3 + \alpha^7 x^4 + \alpha^4 x^5 + \alpha^{11} x^6 + \alpha^{12} x^7 + \alpha^9 x^8 \\
& + \alpha^4 x^9 + \alpha^{10} x^{10} + \alpha^5 x^{11} + \alpha^2 x^{12} + \alpha^{14} x^{13} + \alpha^6 x^{14} + \alpha^8 x^{15} + \alpha^{13} x^{16} + \alpha^7 x^{17} + \\
& + \alpha^4 x^{18} + \alpha x^{19} + \alpha^7 x^{21}) + (1 + \alpha^7 x + \alpha^9 x^2 + \alpha^{14} x^3 + \alpha^9 x^4 + \alpha^8 x^5 + \alpha^3 x^6 \\
& + \alpha^{14} x^7 + \alpha^4 x^8 + \alpha^8 x^{10} + \alpha^5 x^{11} + \alpha^8 x^{12} + \alpha^{11} x^{13} + \alpha^2 x^{14}) y + (\alpha^3 x + \alpha^7 x^2 \\
& + \alpha^{10} x^3 + \alpha^6 x^4 + \alpha^3 x^5 + \alpha^{14} x^7 + x^8) y^2 + (\alpha^9 + \alpha^{14} x + \alpha^{12} x^2) y^3 \\
g_3(x,y) = {} & (\alpha^5 + \alpha^9 x + \alpha^{13} x^2 + \alpha^2 x^3 + \alpha x^4 + \alpha^{14} x^5 + \alpha^2 x^6 + \alpha x^7 + \alpha^{12} x^8 \\
& + \alpha x^9 + \alpha^{14} x^{10} + \alpha^7 x^{11} + \alpha^9 x^{13} + \alpha^5 x^{14} + \alpha^5 x^{15} + \alpha^9 x^{16} + \alpha^{11} x^{17} + \alpha^3 x^{18} \\
& + \alpha^{13} x^{19}) + (\alpha^7 + \alpha^7 x + \alpha^8 x^2 + \alpha^{14} x^3 + x^4 + \alpha^{11} x^5 + \alpha^7 x^6 + \alpha^2 x^7 + \alpha^5 x^8 \\
& + \alpha^{14} x^9 + \alpha^{12} x^{10} + \alpha^8 x^{11} + \alpha^5 x^{12} + \alpha^5 x^{13}) y + (\alpha^5 + \alpha x + \alpha^{10} x^2 + \alpha^{11} x^3 \\
& + \alpha^{13} x^4 + \alpha^6 x^5 + \alpha^4 x^6 + \alpha^8 x^7) y^2 + (1 + \alpha^8 x) y^3
\end{aligned}$$

of weighted degrees 22, 20, 20, and 19, respectively. The one of minimal order selected is $Q(x, y) = g_3(x, y)$. □

The computational complexity of this algorithm, like the Feng-Tzeng algorithm, goes as the cube of the size of the problem, $O(m^3)$, with a rather large multiplicative factor. The cubic complexity of these problems makes decoding impractical for large values of m.

7.6.6 A Special Case: $m = 1$ and $L = 1$

We will see below how to take the interpolating polynomial $Q(x, y)$ and find its y-roots. This will handle the general case of arbitrary m. However, we treat here an important special case which occurs when $m = 1$ and $L = 1$. In this case, the y-degree of $Q(x, y)$ is equal to 1 and it is not necessary to employ a sophisticated factorization algorithm. This special case allows for *conventional* decoding of Reed-Solomon codes without computation of syndromes and without the error evaluation step.

When $L = 1$ let us write

$$Q(x, y) = P_1(x)y - P_0(x).$$

If it is the case that $P_1(x) \,|\, P_0(x)$, let

$$p(x) = \frac{P_0(x)}{P_1(x)}.$$

Then it is clear that $y - p(x) \,|\, Q(x, y)$, since

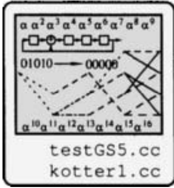
$$P_1(x)y - P_0(x) = P_1(x)(y - p(x)).$$

Furthermore, it can be shown that the $p(x)$ returned will produce a codeword within a distance $\leq \lfloor (n-k)/2 \rfloor$ of the transmitted codeword. Hence, it decodes up to the design distance.

In light of these observations, the following decoding algorithm is an alternative to the more conventional approaches (e.g.: find syndromes; find error locator; find roots; find error values; or: find remainder; find rational interpolator; find roots; find error values).

Algorithm 7.7 Guruswami-Sudan Interpolation Decoder with $m = 1$ and $L = 1$

1 **Input:** Points: $(x_i, y_i), i = 1, \ldots, n$; a $(1, k-1)$ monomial order
2 **Returns:** $p(x, y)$ as a decoded codeword if it exists
3 **Initialize:** $g_0(x, y) = 1$, $g_1(x, y) = y$
4 for $i = 1$ to n
5 $\quad \Delta_0 = g_0(x_i, y_i)$ (compute discrepancies)
6 $\quad \Delta_1 = g_1(x_i, y_i)$
7 $\quad J = \{j : \Delta_j \neq 0\}$ (set of nonzero discrepancies)
8 $\quad$ if($J \neq \emptyset$)
9 $\quad\quad j^* = \arg\min\{g_j : j \in J\}$ (polynomial of min. weighted degree)
10 $\quad\quad f = g_{j^*}$
11 $\quad\quad \Delta = \Delta_{j^*}$
12 $\quad\quad$ for($j \in J$)
13 $\quad\quad\quad$ if($j \neq j^*$)
14 $\quad\quad\quad\quad g_j = \Delta g_j - \Delta_j f$ (update without change in rank)
15 $\quad\quad\quad$ else if($j = j^*$)
16 $\quad\quad\quad\quad g_j = (x - x_i) f$ (update with change in rank)
17 $\quad\quad\quad$ end(if)
18 $\quad\quad$ end(for j)
19 $\quad$ end(if J)
20 end(for i)
21 $Q(x, y) = \min_j \{g_j(x, y)\}$ (least weighted degree)
22 Write $Q(x, y) = P_1(x)y - P_0(x)$
23 $r(x) = P_0(x) \mod P_1(x)$.
24 if($r(x) = 0$)
25 $\quad p(x) = P_0(x)/P_1(x)$.
26 $\quad$ if($\deg p(x) \leq k - 1$) then
27 $\quad\quad p(x)$ is decoded message
28 $\quad$ end(if)
29 else
30 $\quad$ Uncorrectable error pattern
31 end(if)



Example 7.22 Consider a (5, 2) RS code over $GF(5)$, using support set (0, 1, 2, 3, 4). Let $m(x) = 1 + 4x$. Then the codeword

$$(m(0), m(1), m(2), m(3), m(4)) = (1, 0, 4, 3, 2) \rightarrow c(x) = 1 + 4x^2 + 3x^3 + 2x^4.$$

Let the received polynomial be $r(x) = 1 + 2x + 4x^2 + 3x^3 + 2x^4$. The following table indicates the steps of the algorithm.

i	(x_i, y_i)	$g_0(x, y)$	$g_1(x, y)$
–	–	1	y
0	(0,1)	x	$4 + y$
1	(1,0)	$4x + x^2$	$(4 + 4x) + y$
2	(2,4)	$(2 + x + x^2) + 3y$	$(2 + 4 + 4x^2) + 3y$
3	(3,3)	$(4 + 4x + 3x^2 + x^3) + (1 + 3x)y$	$(3 + 4x + 3x^2) + (2 + 3x)y$
4	(4,2)	$(4 + 3x + 2x^2 + 4x^3 + x^4) + (1 + 4x + 3x^2)y$	$(3 + 4x + 3x^2) + (2 + 3x)y$

(An interesting thing happens at $i = 2$: It appears initially that $g_0(x, y) = (2 + x + x^2) + 3y$ is no longer in the set S_0, the set of polynomials whose leading monomial has y-degree 0, because a term with y appears in $g_0(x, y)$. However, the leading term is actually x^2. Similar behavior is seen at other steps.) At the end of the algorithm we take

$$Q(x, y) = (2 + 3x)y + (3 + 4x + 3x^2) = (2 + 3x)y - (2 + x + 2x^2),$$

so that

$$p(x) = \frac{2 + x + 2x^2}{2 + 3x} = 1 + 4x,$$

which was the original message. □

7.6.7 An Algorithm for the Factorization Step: The Roth-Ruckenstein Algorithm

We now consider the problem of factorization when the y-degree of $Q(x, y) \geq 1$. Having obtained the polynomial $Q(x, y)$ interpolating a set of data points (x_i, y_i), $i = 1, 2, \ldots, n$, the next step in the Guruswami-Sudan decoding algorithm is to determine all factors of the form $y - p(x)$, where $p(x)$ is a polynomial of degree $\leq v$, such that $(y - p(x)) \mid Q(x, y)$. We have the following observation:

Lemma 7.27 $(y - p(x)) \mid Q(x)$ *if and only if* $Q(x, p(x)) = 0$.

(This is analogous to the result in univariate polynomials that $(x - a) \mid g(x)$ if and only if $g(a) = 0$.)

Proof Think of $Q(x, y)$ as a polynomial in the variable y with coefficients over $\mathbb{F}(x)$. The division algorithm applies, so that upon division by $y - p(x)$ we can write

$$Q(x, y) = q(x, y)(y - p(x)) + r(x),$$

where the y-degree of $r(x)$ must be 0, since the divisor $y - p(x)$ has y-degree 1. Evaluating at $y = p(x)$ we see that $Q(x, p(x)) = r(x)$. Then $Q(x, p(x)) = 0$ if and only if $r(x) = 0$ (identically). □

Definition 7.16 A function $p(x)$ such that $Q(x, p(x)) = 0$ is called a y-**root** of $Q(x, y)$. □

The algorithm described in this section, the **Roth-Ruckenstein** algorithm [297], finds y-roots. (Another algorithm due to Gao and Shokrollahi [114] is also know to be effective for this factorization.)

The notation $\langle\langle Q(x,y)\rangle\rangle$ denotes the coefficient (polynomial) of the highest power of x that divides $Q(x,y)$. That is, if $x^m \mid Q(x,y)$ but $x^{m+1} \nmid Q(x,y)$, then

$$\langle\langle Q(x,y)\rangle\rangle = \frac{Q(x,y)}{x^m}.$$

Then

$$Q(x,y) = \langle\langle Q(x,y)\rangle\rangle x^m$$

for some $m \geq 0$.

Example 7.23 If $Q(x,y) = xy$, then $\langle\langle Q(x,y)\rangle\rangle = y$. If $Q(x,y) = x^2y^4+x^3y^5$, then $\langle\langle Q(x,y)\rangle\rangle = y^4 + xy^5$. If $Q(x,y) = 1 + x^2y^4 + x^3y^5$, then $\langle\langle Q(x,y)\rangle\rangle = Q(x,y)$. □

Let $p(x) = a_0 + a_1x + a_2x^2 + \cdots + a_vx^v$ be a y-root of $Q(x,y)$. The Roth-Ruckenstein algorithm will determine the coefficients of $p(x)$ one at a time. The coefficient a_0 is found using the following lemma.

Lemma 7.28 *Let $Q_0(x,y) = \langle\langle Q(x,y)\rangle\rangle$. If $(y - p(x)) \mid Q(x,y)$, then $y = p(0) = a_0$ is a root of the equation $Q_0(0,y) = 0$.*

Proof If $(y - p(x)) \mid Q(x,y)$, then $(y - p(x)) \mid x^m Q_0(x,y)$ for some $m \geq 0$. But since $y - p(x)$ and x^m must be relatively prime, it must be the case that $(y - p(x)) \mid Q_0(x,y)$, so that $Q_0(x,y) = T_0(x,y)(y - p(x))$ for some quotient polynomial $T_0(x,y)$. Setting $y = p(0)$ we have

$$Q_0(0,y) = Q_0(0,p(0)) = T_0(0,p(0))(p(0) - p(0)) = T_0(0,p(0))0 = 0.$$

□

From this lemma, the set of possible values of the coefficient a_0 of $p(x)$ are the roots of the polynomial $Q_0(0,y)$. The algorithm now works by inductively "peeling off" layers, leaving a structure by which a_1 can similarly be found, then a_2, and so forth. It is based on the following theorem, which defines the peeling off process and extends Lemma 7.28.

Theorem 7.29 *Let $Q_0(x,y) = \langle\langle Q(x,y)\rangle\rangle$. Let $p_0(x) = p(x) = a_0 + a_1x + \cdots a_vx^v \in \mathbb{F}_v[x]$. For $j \geq 1$ define*

$$p_j(x) = \frac{(p_{j-1}(x) - p_{j-1}(0))}{x} = a_j + \cdots + a_vx^{v-j} \tag{7.98}$$

$$T_j(x,y) = Q_{j-1}(x, xy + a_{j-1}) \tag{7.99}$$

$$Q_j(x,y) = \langle\langle T_j(x,y)\rangle\rangle. \tag{7.100}$$

Then for any $j \geq 1$, $(y - p(x)) \mid Q(x,y)$ if and only if $(y - p_j(x)) \mid Q_j(x,y)$.

Proof The proof is by induction: We will show that $(y - p_{j-1}(x)) \mid Q_{j-1}(x,y)$ if and only if $(y - p_j(x)) \mid Q_j(x,y)$.

$\Rightarrow$: Assuming $(y - p_{j-1}(x)) \big| Q_{j-1}(x, y)$, we write

$$Q_{j-1}(x, y) = (y - p_{j-1}(x))u(x, y)$$

for some quotient polynomial $u(x, y)$. Then from (7.99),

$$T_j(x, y) = (xy + a_{j-1} - p_{j-1}(x))u(x, xy + a_{j-1}).$$

Since $a_{j-1} - p_{j-1}(x) = -xp_j(x)$,

$$T_j(x, y) = x(y - p_j(x))u(x, xy + a_{j-1})$$

so that $(y - p_j(x)) \big| T_j(x, y)$. From (7.100), $T_j(x, y) = x^m Q_j(x, y)$ for some $m \geq 0$, so $(y-p_j(x)) \big| x^m Q_j(x, y)$. Since x^m and $y-p(x)$ are relatively prime, $(y-p_j(x)) \big| Q_j(x, y)$.

$\Leftarrow$: Assuming $(y - p_j(x)) \big| Q_j(x, y)$ and using (7.100), $(y - p_j(x)) \big| T_j(x, y)$. From (7.99), $(y - p_j(x)) \big| Q_{j-1}(x, xy + a_{j-1})$ so that

$$Q_{j-1}(x, xy + a_{j-1}) = (y - p_j(x))u(x, y) \tag{7.101}$$

for some quotient polynomial $u(x, y)$. Replace y by $(y - a_{j-1})/x$ in (7.101) to obtain

$$Q_{j-1}(x, y) = ((y - a_{j-1})/x - p_j(x))u(x, (y - a_{j-1})/x).$$

The fractions can be cleared by multiplying both sides by some sufficiently large power L of x. Then using the fact that $p_{j-1}(x) = a_{j-1} + xp_j(x)$ we obtain

$$x^L Q_{j-1}(x, y) = (y - p_{j-1}(x))v(x, y)$$

for some polynomial $v(x, y)$. Thus $(y - p_{j-1}(x)) \big| x^L Q_{j-1}(x, y)$ and so

$$(y - p_{j-1}(x)) \big| Q_{j-1}(x, y).$$

□

The following lemma is a repeat of Lemma 7.28 and is proved in a similar manner.

Lemma 7.30 *If $(y - p(x)) \big| Q(x, y)$, then $y = p_j(0)$ is a root of the equation $Q_j(0, y) = 0$ for $j = 0, 1, \ldots, v$.*

Since $p_j(0) = a_j$, this lemma indicates that the coefficient a_j can be found by finding the roots of the equation $Q_j(0, y) = 0$.

Finally, we need a termination criterion, provided by the following lemma.

Lemma 7.31 *If $y \big| Q_{v+1}(x, y)$, then $p(x) = a_0 + a_1 x + \cdots + a_v x^v$ is a y-root of $Q(x, y)$.*

Proof Note that if $y \big| Q_{v+1}(x, y)$, then $Q_{v+1}(x, 0) = 0$.

By the construction (7.98), $p_j(x) = 0$ for $j \geq v + 1$. The condition $y \big| Q_{v+1}(x, y)$ is equivalent to $(y - p_{v+1}(x)) \big| Q_{v+1}(x, y)$. Thus by Theorem 7.29, it must be the case that $(y - p(x)) \big| Q(x)$. □

The overall operation of the algorithm is outlined as follows.

for each a_0 in the set of roots of $Q_0(0, y)$
  for each a_1 in the set of roots of $Q_1(0, y)$
    for each a_2 in the set of roots of $Q_2(0, y)$
      ⋮
      for each a_u in the set of roots of $Q_u(0, y)$
        if $Q_u(x, 0) = 0$, then (by Lemma 7.31), $p(x) = a_0 + a_1 x + \cdots + a_u x^u$ is a y-root
      end for
      ⋮
    end for
  end for
end for

The iterated "for" loops (up to depth u) can be implemented using a recursive programming structure. The following algorithm uses a depth-first tree structure.

Algorithm 7.8 Roth-Ruckenstein Algorithm for Finding y-roots of $Q(x, y)$

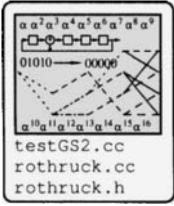
---

1 **Input:** $Q(x, y)$, D (maximum degree of $p(x)$)
2 **Output:** List of polynomials $p(x)$ of degree $\leq D$ such that $(y - p(x)) \mid Q(x, y)$
3 **Initialization:** Set $p(x) = 0$, $u = \deg(p) = -1$, $D =$ maximum degree (set as internal global)
4 Set up linked list where polynomials are saved.
5 Set $\nu = 0$ (the number of the node; global variable)
6 Call `rothrucktree`$(Q(x, y), u, p)$

7 Function `rothrucktree`(Q, u, p):
8 **Input:** $Q(x, y)$, $p(x)$ and u (degree of p)
9 **Output:** List of polynomials
10 $\nu = \nu + 1$ (increment node number)
11 if$(Q(x, 0) = 0)$
12   add $p(x)$ to the output list
13 end(if)
14 else if$(u < D)$ (try another branch of the tree)
15   R = list of roots of $Q(0, y)$
16   for each $\alpha \in R$
17     $Q_{\text{new}}(x, y) = Q(x, xy + \alpha)$ (shift the polynomial)
18     $p_{u+1} = \alpha$ (new coefficient of $p(x)$)
19   Call `rothrucktree`$(\langle\langle Q_{\text{new}}(x, y)\rangle\rangle, u + 1, p)$ (recursive call)
20   end (for)
21 else (leaf of tree reached with nonzero polynomial)
22   (no output)
23 end (if)
24 end

Example 7.24 Let

$$Q(x, y) = (4 + 4x^2 + 2x^3 + 3x^4 + x^5 + 3x^6 + 4x^7) + (1 + 2x + 2x^2 + 3x^4 + 3x^6)y + (1 + x + 2x^2 + x^3 + x^4)y^2 + (4 + 2x)y^3$$



by a polynomial in $GF(5)[x, y]$. Figures 7.5 and 7.6 illustrate the flow of the algorithm with $D = 2$.

- At Node 1, the polynomial $Q(0, y) = 4 + y + y^2 + y^3$ is formed (see Figure 7.5) and its roots are computed as $\{1, 4\}$ (1 is actually a repeated root).
- At Node 1, the root 1 is selected. Node 2 is called with $\langle\langle Q(x, xy + 1)\rangle\rangle$.
- At Node 2, the polynomial $Q(0, y) = 3 + 3y^2$ is formed, with roots $\{2, 3\}$.
- At Node 2, the root 2 is selected. Node 3 is called with $\langle\langle Q(x, xy + 2)\rangle\rangle$.
- At Node 3, the polynomial $Q(x, 0) = 0$, so the list of roots selected to this node $\{1, 2\}$ forms an output polynomial, $p(x) = 1 + 2x$.
- The recursion returns to Node 2, where the root 3 is selected, and Node 4 is called with $\langle\langle Q(x, xy + 3)\rangle\rangle$.
- At Node 4, the polynomial $Q(0, y) = 2 + 3y$ is formed, with roots $\{1\}$.
- At Node 4, the root 1 is selected. Node 5 is called with $\langle\langle Q(x, xy + 1)\rangle\rangle$.
- At Node 5, it is not the case that $Q(x, 0) = 0$. Since the level of the tree (3) is greater than D (2), no further searching is performed along this branch, so no output occurs. (However, if D had been equal to 3, the next branch would have been called with a root of 2 and a polynomial would have been found; the polynomial $p(x) = 1 + 3x + x^2 + 2x^3$ would have been added to the list.)
- The recursion returns to Node 1, where the root 4 is selected and Node 6 is called with $\langle\langle Q(x, xy + 4)\rangle\rangle$.
- At Node 6, the polynomial $Q(0, y) = 2 + y$ is formed, with root $\{3\}$. (See Figure 7.6.) Node 7 is called with $\langle\langle Q(x, xy + 3)\rangle\rangle$
- At Node 7, the polynomial $Q(0, y) = 3 + y$ is formed, with root $\{2\}$. Node 8 is called with $\langle\langle Q(x, xy + 2)\rangle\rangle$.
- At Node 8, $Q(x, 0) = 0$, so the list of roots selected to this node $\{4, 3, 2\}$ forms an output polynomial $p(x) = 4 + 3x + 2x^2$.

The set of polynomials produced is $\{1 + 2x, 4 + 3x + 2x^2\}$. □

Example 7.25 For the interpolating polynomial $Q(x, y)$ of Example 7.21, the Roth-Ruckenstein algorithm determines the following y-roots (see `testGS3.cc`):

$$\begin{aligned} p(x) \in \{&\alpha + \alpha^2 x + \alpha^3 x^2 + \alpha^4 x^3 + \alpha^5 x^4 + \alpha^6 x^5 + \alpha^7 x^6, \\ &\alpha^7 + \alpha^6 x + \alpha^{14} x^2 + \alpha^5 x^3 + \alpha^{10} x^4 + \alpha^{10} x^5 + \alpha^2 x^6, \\ &\alpha^{12} + \alpha^{10} x + \alpha^{11} x^3 + \alpha^{13} x^4 + \alpha^{11} x^5 + \alpha^{11} x^6\} = \mathcal{L}. \end{aligned}$$

Note that the original message $m(x)$ is among this list, resulting in a codeword a Hamming distance 4 away from the received $r(x)$. There are also two others (confer the fact that $L_m = 3$ was computed in Example 7.21). The other polynomials result in distances 5 and 6, respectively, away from $r(x)$. Being further than t_m away from $r(x)$, these can be eliminated from further consideration. □

What to Do with Lists of Factors?

The Guruswami-Sudan algorithm returns the message $m(x)$ on its list of polynomials $\mathcal{L}$, provided that the codeword for $m(x)$ is within a distance t_m of the received vector. This is called the *causal* codeword.It may also return other codewords at a Hamming distance $\leq t_m$, which are called*plausible* codewords. Other codewords, at a distance $> t_m$ from $r(x)$ may also be on the list; these are referred to as *noncausal* codewords (i.e., codewords not

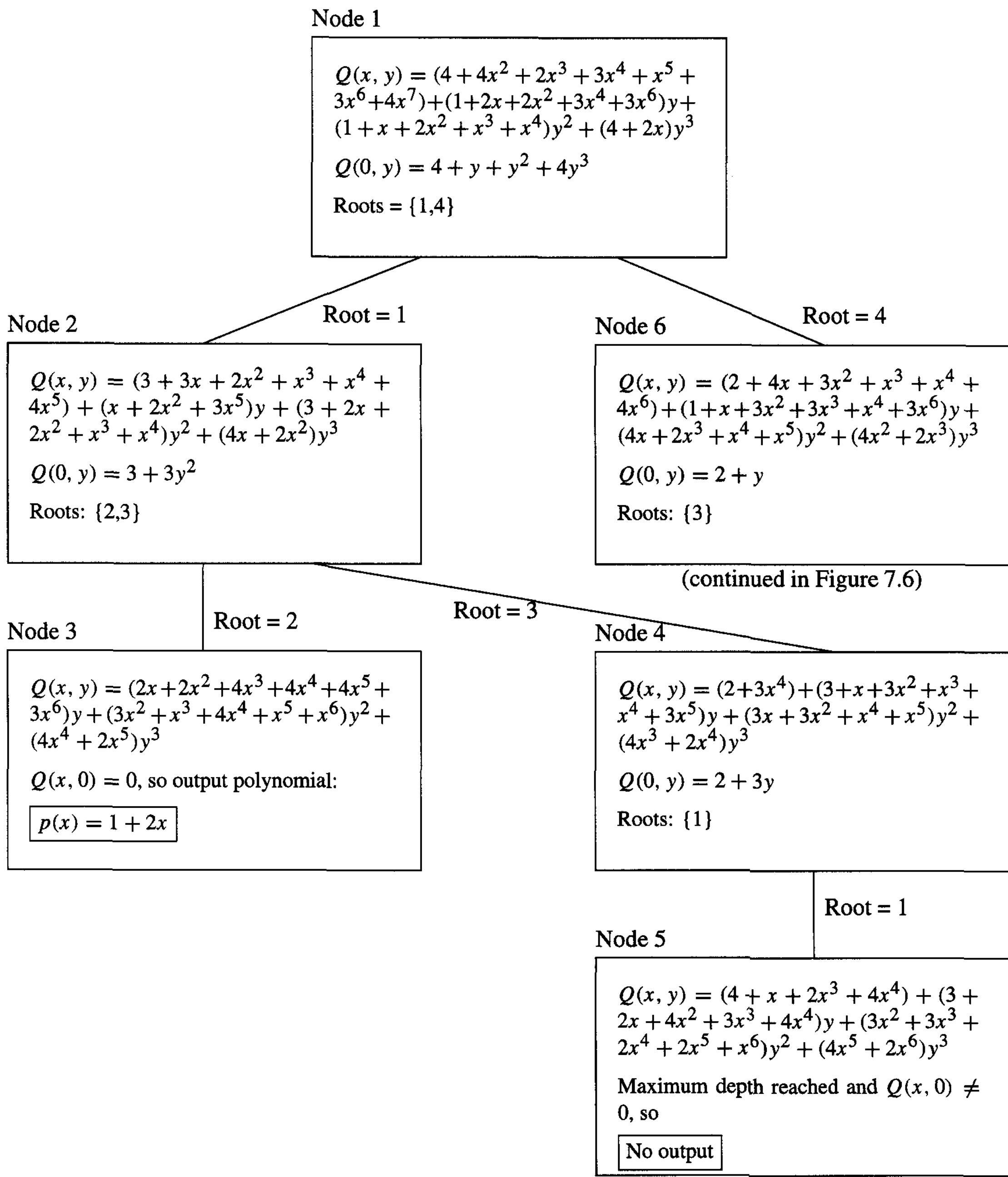


Figure 7.5: An example of the Roth-Ruckenstein Algorithm over $GF(5)$.

Node 6

$Q(x, y) = (2 + 4x + 3x^2 + x^3 + x^4 + 4x^6) + (1+x+3x^2+3x^3+x^4+3x^6)y + (4x+2x^3+x^4+x^5)y^2 + (4x^2+2x^3)y^3$

$Q(0, y) = 2 + y$

Roots: $\{3\}$

Root = 3

Node 7

$Q(x, y) = (3+2x^2+3x^3+4x^4+3x^5) + (1+x^2+4x^3+2x^4+x^5+3x^6)y + (4x^2 + x^3 + x^5 + x^6)y^2 + (4x^4 + 2x^5)y^3$

$Q(0, y) = 3 + y$

Roots: $\{2\}$

Root = 2

Node 8

$Q(x, y) = (x+2x^3+3x^4+4x^6+2x^7)y + (4x^4+x^5+4x^6+3x^7+x^8)y^2 + (4x^7 + 2x^8)y^3$

$Q(x, 0) = 0$, so output polynomial:

$\boxed{p(x) = 4 + 3x + 2x^2}$

Figure 7.6: An example of the Roth-Ruckenstein Algorithm over $GF(5)$ (cont'd).

caused by the original, true, codeword). We have essentially showed, by the results above, that all plausible codewords are in $\mathcal{L}$ and that the maximum number of codewords is $\leq L_m$.

In Example 7.25, it was possible to winnow the list down to a single plausible codeword. But in the general case, what should be done when there are multiple plausible codewords in $\mathcal{L}$? If the algorithm is used in conjunction with another coding scheme (in a concatenated scheme), it may be possible to eliminate one or more of the plausible codewords. Or there may be external ways of selecting a correct codeword from a short list. Another approach is to exploit *soft* decoding information, to determine using a more refined measure which codeword is closest to $r(x)$. But there is, in general, no universally satisfactory solution to this problem.

However, it turns out that for many codes, the number of elements in $\mathcal{L}$ is equal to 1: the list decoder may not actually return a list with more than one element in it. This concept has been explored in [230]. We summarize some key conclusions.

When a list decoder returns more than one plausible codeword, there is the possibility of a decoding failure. Let L denote the number of codewords on the list and let list L' denote the number of noncausal codewords. There may be a decoding error if $L' > 1$. Let P_E denote the probability of a decoding error. We can write

$$P_E < E[L'],$$

the *average* number of noncausal codewords on the list.

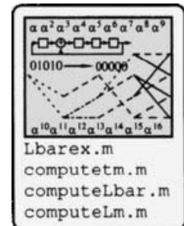
Now consider selecting a point at random in the space $GF(q)^n$ and placing a Hamming sphere of radius t_m around it. How many codewords of an (n, k) code, on average, are in this Hamming sphere? The "density" of codewords in the space is q^k/q^n. As we have seen (Section 3.3.1), the number of points in the Hamming sphere is

$$\sum_{s=0}^{t_m} \binom{n}{s} (q-1)^2.$$

Therefore, the average number of codewords in a sphere of radius t_m around a random point is

$$\overline{L}(t_m) = \frac{q^k}{q^n} \sum_{s=0}^{t_m} \binom{n}{s} (q-1)^2.$$

It can be shown [230] that $\overline{L}(t_m)$ is slightly less than a rigorous bound on the average number of noncausal codewords on the list.



Example 7.26 [230] For a (32, 8) RS code over $GF(32)$, we have the following results (only values of m are shown that lead to distinct values of t_m):

m	t_m	L_m	$\overline{L}(t_m)$
0	12	1	1.36305e-10
1	14	2	2.74982e-07
2	15	4	1.02619e-05
4	16	8	0.000339205
120	17	256	0.00993659

Thus, while the list for $m = q$ may have as many as $L_4 = 8$ polynomials, the probability $\overline{L}(t_4)$ is very small — the list is very likely to contain only one codeword, which will be the causal codeword. It is highly likely that this code is capable of correcting up to 16 errors. (Actually, even for $m = 120$, the probability of more than codeword is still quite small; however, the computational complexity for $m = 120$ precludes its use as a practical decoding option.) ☐

Example 7.27 [230] For a (32,15) code over $GF(32)$, we have

m	t_m	L_m	$\bar{L}(t_m)$
0	8	1	5.62584e-06
3	9	4	0.000446534
21	10	31	0.0305164

It is reasonable to argue that the code can correct up to 9 errors with very high probability. □

7.6.8 Soft-Decision Decoding of Reed-Solomon Codes

The Reed-Solomon decoding algorithms described up to this point in the book have been *hard* decision decoding algorithms, making explicit use of the algebraic structure of the code and employing symbols which can be interpreted as elements in a Galois field. A long outstanding problem in coding theory has been to develop a *soft-decision* decoding algorithm for Reed-Solomon codes, which is able to exploit soft channel outputs without mapping it to hard values, while still retaining the ability to exploit the algebraic structure of the code. This problem was solved [191] by an extension of the Guruswami-Sudan algorithm which we call the Koetter-Vardy (KV) algorithm.[3]

Recall that the GS algorithm has a parameter m representing the interpolation multiplicity, m. For most of this chapter, a fixed multiplicity has been employed at each point. It is possible, however, to employ a different multiplicity at each point. (In fact, the Algorithm 7.6 already handles multiple multiplicities.) In the KV algorithm, a mapping is found from posterior probabilities (soft information) to the multiplicities, after which the conventional interpolation and factorization steps of the GS are used to decode. (The mapping still results in some degree of "hardness," since probabilities exist on a continuum, while the multiplicities must be integers.) We present here their algorithm for *memoryless* channels; for further results on channels with memory and concatenated channels, the reader is referred to [191].

Notation

Recall (see Section 1.6) that a memoryless channel can be modeled as an input alphabet $\mathfrak{X}$, an output alphabet $\mathfrak{Y}$ and a set of $|\mathfrak{X}|$ functions $f(\cdot|x) : \mathfrak{Y} \rightarrow \mathbb{R}$. The channel input and output are conventionally viewed as a random variables $\mathcal{X}$ and $\mathcal{Y}$. If $\mathcal{Y}$ is continuous then $f(\cdot|x)$ is a probability density function (for example, for a Gaussian channel, $f(y|x)$ would be a Gaussian likelihood function). If $\mathcal{Y}$ is discrete then $f(\cdot|x)$ is a probability mass function. The memoryless nature of the channel is reflected by the assumption that the joint likelihood factors,

$$f(y_1, y_2, \ldots, y_n|x_1, x_2, \ldots, x_n) = \prod_{i=1}^{n} f(y_i|x_i).$$

It is assumed that the $\mathcal{X}$ random variable is uniformly distributed (i.e., that each codeword is selected with equal probability). Given an observation $y \in \mathfrak{Y}$, the probability that some $\alpha \in \mathfrak{X}$ was transmitted is found using Bayes' theorem,

$$P(\mathcal{X} = \alpha|\mathcal{Y} = y) = \frac{f(y|\alpha)}{\sum_{x \in \mathfrak{X}} f(y|x)},$$

where the assumption of uniformity of $\mathcal{X}$ is used.

[3]The name Koetter is simply a transliteration of the name Kötter.

For Reed-Solomon codes, the input alphabet is the field over which the symbols occur, $\mathcal{X} = GF(q)$. We have therefore $\mathcal{X} = \{\alpha_1, \alpha_2, \ldots, \alpha_q\}$, for some arbitrary ordering of the elements of $GF(q)$.

Let $\mathbf{y} = (y_1, y_2, \ldots, y_n) \in \mathfrak{Y}^n$ be a vector of observations. We define the posterior probabilities

$$\pi_{i,j} = P(\mathcal{X} = \alpha_i | \mathcal{Y} = y_j), \qquad i = 1, 2, \ldots, q, \quad j = 1, 2, \ldots, n$$

and form the $q \times n$ matrix Π with elements $\pi_{i,j}$. The matrix Π is called the **reliability matrix**. It is convenient below to use the notation $\Pi(\alpha, j)$ to refer the element in the row indexed by α and the column indexed by j. It is assumed that the reliability matrix is provided (somehow) as the input to the decoding algorithm.

A second matrix is also employed. Let M be a $q \times n$ *multiplicity matrix* with nonnegative elements $m_{i,j}$, where $m_{i,j}$ is the interpolation multiplicity associated with the point (α_i, y_j). The key step of the algorithm to be described below is to provide a mapping from the reliability matrix Π to the multiplicity matrix M.

Definition 7.17 Let M be a multiplicity matrix with elements $m_{i,j}$. We will denote by $Q_M(x, y)$ the polynomial of minimal $(1, k-1)$-weighted degree that has a zero of multiplicity at least $m_{i,j}$ at the point (α_i, y_j) for every i, j such that $m_{i,j} \neq 0$.

□

Recall that the main point of the interpolation theorem is that there must be more degrees of freedom (variables) than there are constraints. The number of constraints introduced by a multiplicity $m_{i,j}$ is equal to $\binom{m_{i,j}+1}{2}$. The total number of constraints associated with a multiplicity matrix M is called the *cost of M*, denoted $C(M)$, where

$$C(M) = \frac{1}{2} \sum_{i=1}^{q} \sum_{j=1}^{n} m_{i,j}(m_{i,j} + 1).$$

As before, let $C(v, l)$ be the number of monomials of weighted $(1, v)$-degree less than or equal to l. Then by the interpolation theorem an interpolating solution exists if

$$C(v, l) > C(M).$$

Let $\mathcal{K}_v(x) = \min\{l \in \mathbb{Z} : C(v, l) > x\}$ be the smallest $(1, v)$-weighted degree which has the number of monomials of lesser (or equal) degree exceeding x. Then $\mathcal{K}_{k-1}(C(M))$ is the smallest $(1, k-1)$-weighted degree which has a sufficiently large number of monomials to exceed the cost $C(M)$. (Confer with K_m defined in (7.76)). By (7.64), for a given cost $C(M)$ we must have

$$\mathcal{K}_v(C(M)) < \sqrt{2vC(M)}. \tag{7.102}$$

It will be convenient to represent vectors in $GF(q)^n$ as indicator matrices over the reals, as follows. Let $\mathbf{v} = (v_1, v_2, \ldots, v_n) \in GF(q)^n$ and let $[\mathbf{v}]$ denote the $q \times n$ matrix which has $[\mathbf{v}]_{i,j} = 1$ if $v_j = \alpha_i$, and $[\mathbf{v}]_{i,j} = 0$ otherwise. That is,

$$[\mathbf{v}]_{i,j} = I_{\alpha_i}(v_j),$$

where $I_{\alpha_i}(v_j)$ is the indicator function, $I_{\alpha_i}(v_j) = \begin{cases} 1 & v_j = \alpha_i \\ 0 & \text{otherwise.} \end{cases}$

Example 7.28 In the field $GF(3)$, let $\mathbf{v} = (1, 2, 0, 1)$. The matrix representation is

$$[\mathbf{v}] = \begin{matrix} 0: \\ 1: \\ 2: \end{matrix} \overset{\begin{matrix} v_1 & v_2 & v_3 & v_4 \end{matrix}}{\begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{bmatrix}}.$$

□

For two $q \times n$ matrices A and B, define the inner product

$$\langle A, B \rangle = \mathrm{tr}(AB^T) = \sum_{i=1}^{q}\sum_{j=1}^{n} a_{i,j}b_{i,j}.$$

Now using this, we define the *score* of a vector $\mathbf{v} = (v_1, v_2, \ldots, v_n)$ with respect to a given multiplicity matrix M as

$$S_M(\mathbf{v}) = \langle M, [\mathbf{v}] \rangle.$$

The score thus represents the total multiplicities of all the points associated with the vector $\mathbf{v}$.

Example 7.29 If $M = \begin{bmatrix} m_{1,1} & m_{1,2} & m_{1,3} & m_{1,4} \\ m_{2,1} & m_{2,2} & m_{2,3} & m_{2,4} \\ m_{3,1} & m_{3,2} & m_{3,3} & m_{3,4} \end{bmatrix}$ then the score of the vector $\mathbf{v}$ from the last example is

$$S_M(\mathbf{v}) = m_{2,1} + m_{3,2} + m_{1,3} + m_{2,4}.$$

□

A Factorization Theorem

Our key result is an extension of the factorization theorem for different multiplicities.

Theorem 7.32 *Let $C(M)$ be the cost of the matrix M (the number of constraints to satisfy). Let $\mathbf{c}$ be a codeword in a Reed-Solomon (n, k) code over $GF(q)$ and let $p(x)$ be a polynomial that evaluates to $\mathbf{c}$ (i.e., $\mathbf{c} = (p(x_1), p(x_2), \ldots, p(x_n))$ for code support set $\{x_1, x_2, \ldots, x_n\}$). If $S_M(\mathbf{c}) > \mathcal{K}_{k-1}(C(M))$, then $(y - p(x)) \mid Q_M(x, y)$.*

Proof Let $\mathbf{c} = (c_1, \ldots, c_n)$ be a codeword and let $p(x)$ be a polynomial of degree $< k$ that maps to the codeword: $p(x_j) = c_j$ for all x_j in the support set of the code. Let $g(x) = Q_M(x, p(x))$. We will show that $g(x)$ is identically 0, which will imply that $(y - p(x)) \mid Q_M(x, y)$.

Write $S_M(\mathbf{c}) = m_1 + m_2 + \cdots m_n$. The polynomial $Q_M(x, y)$ has a zero of order m_j at (x_j, c_j) (by the definition of Q_M). By Lemma 7.18, $(x - x_j)^{m_j} \mid Q_M(x, p(x))$, for $j = 1, 2, \ldots, n$. Thus $g(x) = Q_M(x, p(x))$ is divisible by the product

$$(x - x_1)^{m_1}(x - x_2)^{m_2} \cdots (x - x_n)^{m_n} \text{ having degree } m_1 + m_2 + \cdots + m_n = S_M(\mathbf{c}),$$

so that either $\deg(g(x)) \geq S_M(\mathbf{c})$, or $g(x)$ must be zero.

Since $\deg p(x) \leq k-1$, the degree of $Q_M(x, p(x))$ is less than or equal to the $(1, k-1)$-weighted degree of $Q_M(x, y)$. Furthermore, since $Q_M(x, y)$ is of minimal $(1, k-1)$-degree, $\deg_{1,k-1} Q_M(x, y) \leq \mathcal{K}_{k-1}(C(M))$:

$$\deg g(x) \leq \deg_{1,k-1} Q_M(x, y) \leq \mathcal{K}_{k-1}(C(M)).$$

By hypothesis we have $S_M(\mathbf{c}) > \mathcal{K}_{k-1}(C(M))$. Since we cannot have $\deg g(x) \geq S_M(\mathbf{c})$, we must therefore have $g(x)$ is zero (identically). As before, we write using the factorization theorem

$$Q(x, y) = (y - p(x))q(x, y) + r(x)$$

so that at $y = p(x)$ we have

$$Q(x, p(x)) = g(x) = 0 = 0q(x, y) + r(x)$$

so $r(x) = 0$, giving the stated divisibility. □

In light of (7.102), this theorem indicates that $Q_M(x, y)$ has a factor $y - p(x)$, where $p(x)$ evaluates to a codeword $\mathbf{c}$, if

$$S_M(\mathbf{c}) \geq \sqrt{2(k-1)C(M)}. \tag{7.103}$$

Mapping from Reliability to Multiplicity

Given a cost C, we define the set $\mathfrak{M}(C)$ as the set of all matrices with nonnegative elements whose cost is equal to C:

$$\mathfrak{M}(C) = \{M \in \mathbb{Z}^{q \times n} : m_{i,j} \geq 0 \text{ and } \frac{1}{2}\sum_{i=1}^{q}\sum_{j=1}^{n} m_{i,j}(m_{i,j}+1) = C\}.$$

The problem we would address now is to select a multiplicity matrix M which maximizes the score $S_M(\mathbf{c})$ for a transmitted codeword $\mathbf{c}$ (so that the condition $S_M(\mathbf{c}) > \mathcal{K}_{k-1}(C(M))$ required by Theorem 7.32 is satisfied). Not knowing which codeword was transmitted, however, the best that can be hoped for is to maximize some function of a codeword chosen at random. Let $\mathfrak{X} = (\mathfrak{X}_1, \mathfrak{X}_2, \ldots, \mathfrak{X}_n)$ be a random transmitted vector. The score for this vector is $S_M(\mathfrak{X}) = \langle M, [\mathfrak{X}] \rangle$. We choose to maximize the *expected value* of this score,[4]

$$E[S_M(\mathfrak{X})] = \sum_{\mathbf{x} \in \mathcal{X}^n} S_M(\mathbf{x}) P(\mathbf{x})$$

with respect to a probability distribution $P(\mathbf{x})$. We adopt the distribution determined by the channel output (using the memoryless channel model),

$$P(\mathbf{x}) = P(x_1, x_2, \ldots, x_n) = \prod_{j=1}^{n} P(\mathfrak{X}_j = x_j | \mathfrak{Y}_j = y_j) = \prod_{j=1}^{n} \Pi(x_j, j), \tag{7.104}$$

where Π is the reliability matrix. This would be the posterior distribution of $\mathfrak{X}$ given the observations if the prior distribution of $\mathfrak{X}$ were uniform over $GF(q)^n$. However, the $\mathfrak{X}$ are, in fact, drawn from the set of codewords, so this distribution model is not accurate. But computing the optimal distribution function, which takes into account all that the receiver knows about the code locations, can be shown to be NP complete [191]. We therefore adopt the suboptimal, but tractable, stance.

The problem can thus be stated as: Select the matrix $M \in \mathfrak{M}$ maximizing $E[S_M(\mathfrak{X})]$, where the expectation is with respect to the distribution P in (7.104). We will denote the solution as $M(\Pi, C)$, where

$$M(\Pi, C) = \arg\max_{M \in \mathfrak{M}(C)} E[S_M(\mathfrak{X})].$$

We have the following useful lemma.

[4] More correctly, one might want to maximize $P(S_M(\mathfrak{X}) > \mathcal{K}_{k-1}(C(M)))$, but it can be shown that this computation is, in general, very complicated.

Lemma 7.33

$$E[S_M(\mathcal{X})] = \langle M, \Pi \rangle.$$

Proof

$$E[S_M(\mathcal{X})] = E\langle M, [\mathcal{X}]\rangle = \langle M, E[\mathcal{X}]\rangle$$

by linearity. Now consider the (i, j)th element:

$$E[\mathcal{X}]_{i,j} = E[I_{\alpha_i}(\mathcal{X}_j)] = (1)P(\mathcal{X}_j = \alpha_i) + (0)P(\mathcal{X}_j \neq \alpha_i) = P(\mathcal{X}_j = \alpha_i) = \Pi(\alpha_i, j),$$

where the last equality follows from the assumed probability model in (7.104). □

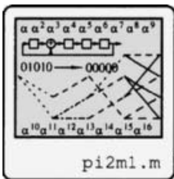
The following algorithm provides a mapping from a reliability matrix Π to a multiplicity matrix M.

Algorithm 7.9 Koetter-Vardy Algorithm for Mapping from Π to M

---

1 **Input:** A reliability matrix Π; a positive integer S indicating the number of interpolation points
2 **Output:** A multiplicity matrix M
3 **Initialization:** Set $\Pi^* = \Pi$ and $M = \mathbf{0}$.
4 Do:
5 Find the position (i, j) of the largest element $\pi^*_{i,j}$ of Π^*.
6 Set $\pi^*_{i,j} = \frac{\pi^*_{i,j}}{m_{i,j}+2}$
7 Set $m_{i,j} = m_{i,j} + 1$
8 Set $S = S - 1$
9 While $S > 0$

Let $\overline{M}$ be formed by normalizing the columns of M produced by this algorithm to sum to 1. It can be shown that $\overline{M} \to \Pi$ as $S \to \infty$. That is, the algorithm produces an integer matrix which, when normalized to look like a probability matrix, asymptotically approaches Π. Since one more multiplicity is introduced for every iteration, it is clear that the score increases essentially linearly with S and that the cost increases essentially quadratically with S.



Example 7.30 Suppose

$$\Pi = \begin{bmatrix} 0.1349 & 0.3046 & 0.2584 & 0.2335 \\ 0.2444 & 0.1578 & 0.1099 & 0.1816 \\ 0.2232 & 0.1337 & 0.2980 & 0.1478 \\ 0.1752 & 0.1574 & 0.2019 & 0.2307 \\ 0.2222 & 0.2464 & 0.1317 & 0.2064 \end{bmatrix}.$$

Figure 7.7 shows a plot of $\|\overline{M} - \Pi\|$ after S iterations as S varies up to 400 iterations. It also shows the score and the cost. The M matrix produced after 400 iterations, and its normalized equivalent, are

$$M = \begin{bmatrix} 13 & 31 & 26 & 23 \\ 25 & 16 & 11 & 18 \\ 22 & 13 & 30 & 15 \\ 17 & 16 & 20 & 23 \\ 22 & 25 & 13 & 21 \end{bmatrix} \qquad \overline{M} = \begin{bmatrix} 0.1313 & 0.3069 & 0.2600 & 0.2300 \\ 0.2525 & 0.1584 & 0.1100 & 0.1800 \\ 0.2222 & 0.1287 & 0.3000 & 0.1500 \\ 0.1717 & 0.1584 & 0.2000 & 0.2300 \\ 0.2222 & 0.2475 & 0.1300 & 0.2100 \end{bmatrix}.$$

□

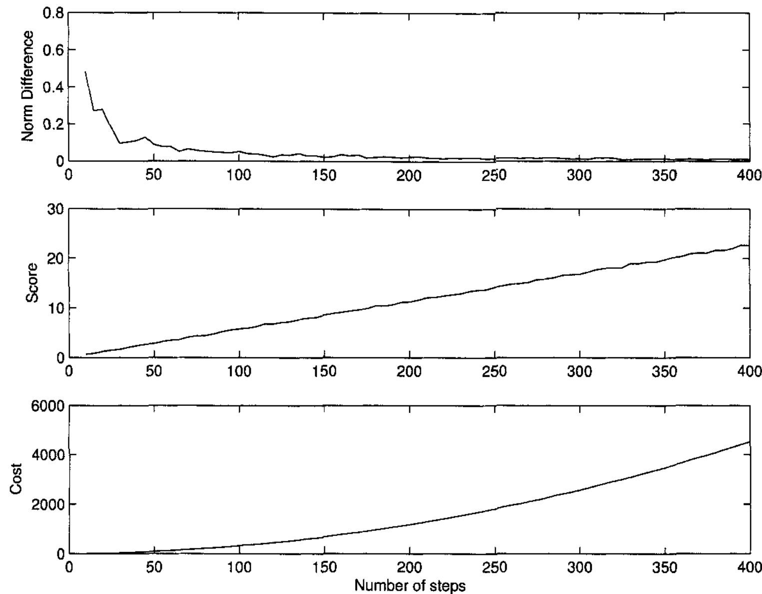


Figure 7.7: Convergence of $\overline{M}$ to Π, and the score and cost as a function of the number of iterations.

Some understanding of this convergence result — why M proportional to Π is appropriate — can be obtained as follows. We note that the cost $C(M)$ can be written as

$$C(M) = \frac{1}{2}(\langle M, M\rangle + \langle M, 1\rangle),$$

where 1 is the matrix of all ones. For sufficiently large M, $C(M)$ is close to $\frac{1}{2}\langle M, M\rangle$, which is (1/2) the Frobenius norm of M. For fixed norm $\langle M, M\rangle$, maximizing the expected score $\langle M, \Pi\rangle$ is accomplished (as indicated by the Cauchy-Schwartz inequality) by setting M to be proportional to Π.

The Geometry of the Decoding Regions

In the bound (7.103), write $S_M(\mathbf{c}) = \langle M, [\mathbf{c}]\rangle$ and $C(M) = \langle M, \Pi\rangle + \langle M, 1\rangle$; we thus see from Theorem 7.32 that the decoding algorithm outputs a codeword $\mathbf{c}$ if

$$\frac{\langle M, [\mathbf{c}]\rangle}{\sqrt{\langle M, M\rangle + \langle M, 1\rangle}} \geq \sqrt{k-1}.$$

Asymptotically (as $S \to \infty$ and the normalized $\overline{M} \to \Pi$) we have

$$\frac{\langle \Pi, [\mathbf{c}]\rangle}{\sqrt{\langle \Pi, \Pi\rangle}} \geq \sqrt{k-1} + o(1) \tag{7.105}$$

(where the $o(1)$ term accounts for the neglected $\langle \overline{M}, 1\rangle$ term).

Observe that for any codeword $\mathbf{c}$, $\langle [\mathbf{c}], [\mathbf{c}]\rangle = n$. Recall that in the Hilbert space $\mathbb{R}^{qn}$, the cosine of the angle β between vectors X and Y is

$$\cos\beta(X, Y) = \frac{\langle X, Y\rangle}{\sqrt{\langle X, X\rangle\langle Y, Y\rangle}}.$$

In light of (7.105), a codeword $\mathbf{c}$ is on the decoder list if

$$\frac{\langle \Pi, [\mathbf{c}] \rangle}{\sqrt{n \langle \Pi, \Pi \rangle}} \geq \sqrt{(k-1)/n} + o(1).$$

Asymptotically ($n \rightarrow \infty$), the codeword is on the list if

$$\cos \beta([\mathbf{c}], \Pi) \geq \sqrt{R} + o(1).$$

The asymptotic decoding regions are thus cones in Euclidean space $\mathbb{R}^{qn}$: the central axis of the cone is the line from the origin to the point Π. Codewords which lie within an angle of $\cos^{-1} \sqrt{R}$ of the central axis are included in the decoding list of Π.

Each codeword lies on a sphere S of radius $\sqrt{\langle [\mathbf{c}], [\mathbf{c}] \rangle} = \sqrt{n}$. To contrast the KV algorithm with the GS algorithm, the GS algorithm would take a reliability matrix Π and project it (by some nonlinear means) onto the sphere S, and then determine the codewords within an angle of $\cos^{-1} \sqrt{R}$ of this projected point. Conventional decoding is similar, except that the angle of inclusion is $\cos^{-1}(1 + R)/2$ and the decoding regions are nonoverlapping.

Computing the Reliability Matrix

We present here a suggested method for computing a reliability matrix for transmission using a large signal constellation.

Consider the constellation shown in Figure 7.8. Let the signal points in the constellation be $\mathbf{s}_0, \ldots, \mathbf{s}_{M-1}$ and let the received point be $\mathbf{r}$. Then the likelihood functions $f(\mathbf{r}|\mathbf{s}_i)$ can be computed. Rather than use all M points in computing the reliability, we compute using only the N largest likelihoods. From the N largest of these likelihoods (corresponding to the N closest points in Gaussian noise) $f(\mathbf{r}|\mathbf{s}_{i_1}), f(\mathbf{r}|\mathbf{s}_{i_2}), \ldots, f(\mathbf{r}|\mathbf{s}_{i_N})$ we form

$$P(\mathbf{r}|\mathbf{s}_{i_k}) = \frac{f(\mathbf{r}|\mathbf{s}_{i_k})}{\sum_{l=1}^{N} f(\mathbf{r}|\mathbf{s}_{i_l})}.$$

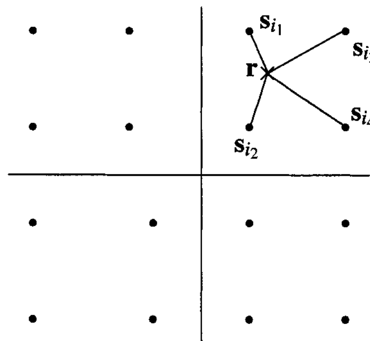


Figure 7.8: Computing the reliability function.

Using likelihoods computed this way, the authors of [191] have examined codes with a rate near 1/2 for a 256 QAM signal constellation. The soft-decision decoding algorithm achieved gains of up to 1.6 dB compared to conventional (hard-decision) decoding of the same RS codes.

7.7 Exercises

7.1 [61] Let M be a module over a ring R.

(a) Show that the additive identity $0 \in M$ is unique.

(b) Show that each $f \in M$ has a unique additive inverse.

(c) Show that $0f = 0$, where $0 \in R$ on the left-hand side is the zero element in R and $0 \in M$ on the right-hand side is the additive identity element in M.

7.2 Show that an ideal $I \subset R$ is an R-module.

7.3 Show that if a subset $M \subset R$ is a module over R, then M is an ideal in R.

7.4 Let I be an ideal in R. Show that the quotient $M = R/I$ is an R-module under the quotient ring sum operation and the scalar multiplication defined for cosets $[g] \in R/I$ and $f \in R$ by $f[g] = [fg] \in R/I$.

7.5 Show that the expression leading to (7.19) is true, that is, $g_0'(\alpha^k)\alpha^{kb} = g'(\alpha^{b+k})\alpha^{b(2-d+k)}$.

7.6 Show that $g'(\alpha^{b+k})\alpha^{b(k+2-d)}p_k\alpha^k = \tilde{C}$ for $k = 0, 1, \ldots, d-2$, where $\tilde{C}$ is a constant, using the following steps. Let $g^{[r]}(x) = \prod_{i=0}^{r}(x-\alpha^i)$ and $p^{[r]}(x) = \prod_{i=1}^{r}(x-\alpha^i) = \sum_{k=0}^{r} p_k^{[r]}x^k$. (That is, these correspond to polynomials with $b = 0$.) You may prove the result by induction, using

$$g^{[r]}(\alpha^k)p_k^{[r]}\alpha^k = C^{[r]} = \prod_{i=0}^{r-1}(\alpha^{r+1} - \alpha^{i+1}), \qquad k = 0, 1, \ldots, r$$

as the inductive hypothesis.

(a) Show that $g^{[r]\prime}(\alpha^k) = g^{[r]\prime}(\alpha^{k-1})\frac{\alpha^{r+k-1}-\alpha^{r-1}}{\alpha^{k-1}-\alpha^r}$.

(b) Show that $g^{[r+1]\prime}(\alpha^k) = \begin{cases} g^{[r]\prime}(\alpha^k)(\alpha^k - \alpha^{r+1}) & k = 0, 1, \ldots, r \\ g^{[r]}(\alpha^{r+1}) & k = r+1 \end{cases}$. *Hint*: $g^{[r+1]}(x) = g^{[r]}(x)(x - \alpha^{r+1})$.

(c) Show that $p_k^{[r+1]} = \begin{cases} -p_0^{[r]}\alpha^{r+1} & k = 0 \\ p_{k-1}^{[r]} - \alpha^{r+1}p_k^{[r]} & k = 1, 2, \ldots, r \\ 1 & k = r+1. \end{cases}$ *Hint*: $p^{[r+1]}(x) = p^{[r]}(x)(x-\alpha^{r+1})$.

(d) Show that for the case $k = r+1$, $g^{[r+1]\prime}(\alpha^{r+1})p_{r+1}^{[r+1]}\alpha^{r+1} = g^{[r+1]\prime}(\alpha^{r+1})\alpha^{r+1} = \prod_{i=0}^{r}\alpha^{r+2} - \alpha^{i+1} \triangleq C^{[r+1]}$.

(e) For the case that $k = 0$, show (using the inductive hypothesis) that $g^{[r+1]\prime}(\alpha^0)p_0^{[r+1]}\alpha^0 = C^{[r+1]}$.

(f) For the case that $k = 1, 2, \ldots, r$, show that $g^{[r+1]}(\alpha^k)p_k^{[r+1]}\alpha^k = C^{[r+1]}$.

(g) Now extend to the case that $b \neq 0$. Let $g_0(x) = \prod_{i=0}^{d-2}(x - \alpha^i)$ and $g(x) = \prod_{i=b}^{b+d-2}$ and let $p_0(x) = \prod_{i=1}^{d-2}(x - \alpha^i)$ and $p(x) = \prod_{i=b+1}^{b+d-2}(x - \alpha^i)$. Show that $g_0'(x) = g'(\alpha^b x)\alpha^{-b(d-2)}$ and $p_{0k} = p_k\alpha^{b(k-d+2)}$. Conclude that

$$g'(\alpha^{b+k})p_k\alpha^{k+b(k-r)} = C\alpha^{br} = \tilde{C}.$$

(h) Hence show that

$$\frac{N_2(\alpha^k)/W_2(\alpha^k)}{N_1(\alpha^k)/W_1(\alpha^k)} = \tilde{C}.$$

7.7 Show that $f(\alpha^k)g(\alpha^{b+k}) = \tilde{C}\alpha^{b(d-1-k)}$, $k = d-1, \ldots, n-1$, where $\tilde{C}$ is defined in Exercise 7.6. This may be done as follows:

(a) Let $f_0(x) = \sum_{i=0}^{d-2} \frac{\alpha^i p_{0i}}{\alpha_i - x}$ and $g_0(x) = \prod_{i=0}^{d-2}(x - \alpha^i)$. Show that $f_0(x)g_0(x) = -\tilde{C}$. *Hint*: Lagrange interpolation.

(b) Show that $f(x) = x^{-b} f_0(x)\alpha^{b(d-2)}$ to conclude the result.

7.8 Show that (7.23) and (7.24) follow from (7.21) and (7.22) and the results of Exercises 7.6 and 7.7.

7.9 Work through the steps of Lemma 7.5.

(a) Explain why there must by polynomials $Q_1(x)$ and $Q_2(x)$ such that $N(x) - W(x)P(x) = Q_1(x)\Pi(x)$ and $M(x) - V(x)P(x) = Q_2(x)\Pi(x)$.

(b) Show that $(NV - MW)P = (MQ_1 - NQ_2)\Pi$

(c) Explain why $(\Pi(x), P(x)) \big| N(x)$ and $(\Pi(x), P(x)) \big| M(x)$. (Here, $(\Pi(x), P(x))$ is the GCD.)

(d) Show that $(N(x)V(x) - M(x)W(x))P(x) = (M(x)Q_1(x) - N(x)Q_2(x))\Pi(x)$.

(e) Show that

$$(N(x)V(x) - M(x)W(x))\frac{P(x)}{(\Pi(x), P(x))} = \frac{M(x)Q_1(x) - N(x)Q_2(x)}{\Pi(x), P(x)}\Pi(x)$$

and hence that $\Pi(x) \big| (N(x)V(x) - M(x)W(x))$.

(f) Show that $\deg(N(x)V(x) - M(x)W(x)) < k$. Hence conclude that $N(x)V(x) - M(x)W(x) = 0$.

(g) Let $d(x) = (W(x), V(x))$ (the GCD), so that $W(x) = d(x)w(x)$ and $V(x) = d(x)v(x)$ for relatively prime polynomials $v(x)$ and $w(x)$. Show that $N(x) = h(x)w(x)$ and $M(x) = h(x)v(x)$ for a polynomial $h(x) = N(x)/w(x) = M(x)/v(x)$.

(h) Show that $h(x)w(x) - d(x)w(x)P(x) = Q_1(x)\Pi(x)$ and $h(x)v(x) - d(x)v(x)P(x) = Q_2(x)\Pi(x)$

(i) Show that there exist polynomials $s(x)$ and $t(x)$ such that $s(x)w(x) + t(x)v(x) = 1$. Then show that $h(x) - d(x)P(x) = (s(x)Q_1(x) + t(x)Q_2(x))\Pi(x)$.

(j) Conclude that $(h(x), d(x))$ is also a solution. Conclude that $\deg(w(x)) = 0$, so that only $(M(x), V(x))$ has been reduced.

7.10 [45] Explain how the Euclidean algorithm can be used to solve the rational interpolation problem (i.e., how it can be used to solve the Welch-Berlekamp key equation).

7.11 Show that when x_i is an error in a check location that $\Psi_{2,1}(x_i) \neq 0$ and that $h(x_i) \neq 0$. *Hint*: If $\Psi_{2,1}(x_i) = 0$, show that $\Psi_{2,2}(x_i)$ must also be zero; furthermore we have $\Psi_{1,1}(x_i) = 0$, which implies $\Psi_{1,2}(x_i) = 0$. Show that this leads to a contradiction, since $(x - x_i)^2$ cannot divide $\det(\Psi(x))$.

7.12 Write down the monomials up to weight 8 in the (1, 4)-revlex order. Compute $C(4, 8)$ and compare with the number of monomials you obtained.

7.13 Write down the polynomials up to weight 8 in the (1, 4)-lex order. Compute $C(4, 8)$ and compare with the number of monomials you obtained.

7.14 Write the polynomial $Q(x, y) = x^5 + x^2y + x^3 + y + 1 \in GF(5)[x, y]$ with the monomials in (1,3)-revlex order. Write the polynomial with the monomials in (1,3)-lex order.

7.15 Let $Q(x, y) = 2x^2y + 3x^3y^3 + 4x^2y^4$. Compute the Hasse derivatives $D_{1,0}Q(x, y)$, $D_{0,1}Q(x, y)$, $D_{1,1}Q(x, y)$, $D_{2,0}Q(x, y)$, $D_{2,1}Q(x, y)$ and $D_{3,2}Q(x, y)$.

7.16 For a (16,4) RS code over $GF(16)$, plot K_m as a function of m.

7.17 A polynomial $Q(x, y) \in GF(5)[x, y]$ is to be found such that:

$$Q(2,3) = 0 \quad D_{0,1}Q(2,3) = 0 \quad D_{1,0}Q(2,3) = 0$$

$$Q(3,4) = 0 \quad D_{0,1}Q(3,4) = 0 \quad D_{1,0}Q(3,4) = 0$$

(a) Determine the monomials that constitute $Q(x, y)$ in $(1, 4)$-revlex order.

(b) Determine the matrix $\mathcal{D}$ as in (7.89) representing the interpolation and multiplicity constraints for a polynomial.

7.18 In relation to the Feng-Tzeng algorithm, show that $[C(x)a^{(i)}(x)x^p]_n = [C(x)a^{(i)}(x)]_{n-p}$.

7.19 From Lemma 7.25, show that for $P(x, y), Q(x, y) \in \mathbb{F}[x, y]$, $[P(x, y), Q(x, y)]_D \in \ker D$.

7.20 Write down the proof to Lemma 7.15.

7.21 Show that (7.97) is true.

7.22 Bounds on L_m:

(a) Show that

$$\left\lfloor \sqrt{\left(\frac{nm(m+1)}{v}\right)} - \left(\frac{v+2}{2v}\right) \right\rfloor \le r_B(nm(m+1)/2) \le \left\lfloor \sqrt{\frac{nm(m+1)}{v}} \right\rfloor .$$

(b) Show that $L_m < (m + \frac{1}{2})\sqrt{n/v}$.

7.23 [230] Let $A(v, K)$ be the rank of the monomial x^K in $(1, v)$-revlex order. From Table 7.1 we have

K	0	1	2	3	4	5
$K(3, L)$	0	1	2	3	5	7

(a) Show that $A(v, K+1) = C(v, K)$.

(b) Show that $A(v, K) = |\{(i, j) : i + vj < K\}|$.

(c) Show that $B(L, v) = A(vL + 1, v) - 1$.

(d) Euler's integration formula [187, section 1.2.11.2, (3)] indicates that the sum of a function $f(k)$ can be represented in terms of an integral as

$$\sum_{k=0}^{n} f(k) = \int_0^n f(x)\,dx - \frac{1}{2}(f(n) - f(0)) + \int_0^n \{x - \frac{1}{2}\} f'(x)\,dx,$$

where $\{x\} = x - \lfloor x \rfloor$ is the fractional part of x. Based on this formula show that

$$A(v, K) = \frac{K^2}{2v} + \frac{K}{2} + \frac{r(v-r)}{2v}, \tag{7.106}$$

where $r = K \pmod{v}$.

(e) Show the following bound:

$$\frac{K^2}{2v} < A(K, v) \le \frac{(K + v/2)^2}{2v}. \tag{7.107}$$

7.24 Bounds on K_m:

(a) Show that

$$K_m = \lfloor r_A(n\binom{m+1}{2})/m \rfloor + 1,$$

where

$$r_A(x) = \max\{K : A(v, K) \le x\}.$$

(b) Show that

$$\left\lfloor \sqrt{vn(m+1)/m} - v/(2m) \right\rfloor + 1 \leq K_m \leq \left\lfloor \sqrt{vn(m+1)/m} \right\rfloor.$$

(c) Hence show that asymptotically (as $m \to \infty$)

$$t_\infty = n - K_\infty = n - 1 - \lfloor \sqrt{vn} \rfloor.$$

7.8 References

The original Welch-Berlekamp algorithm appeared in [369]. In addition to introducing the new key equation, it also describes using different symbols as check symbols in a novel application to magnetic recording systems. This was followed by [23], which uses generalized minimum distanceto improve the decoding behavior. The notation we use in Section 7.2.1 comes from this and from [247]. A comparison of Welch-Berlekamp key equations is in [244]. Our introduction to modules was drawn from [61]; see also [162, 155].

Our discussion of the DB form of the key equation, as well as the idea of "exact sequences" and the associated algorithms, is drawn closely from [63]. Other derivations of WB key equations are detailed in [214] and [247].

The development of the Welch-Berlekamp algorithm from Section 7.4.2 closely follows [214]. The modular form follows from [63]. Other related work is in [44] and [45].

The idea of list decoding goes back to [77]. The idea of this interpolating recovery is expressed in [369]. Work preparatory to the work here appears in [193] and was extended in [323], building in turn on [8], to decode beyond the RS design distance for some low rate codes. In particular, a form of Theorem 7.19 appeared originally in [8]; our statement and proof follows [128]. In [128], the restriction to low-rate codes was removed by employing higher multiplicity interpolating polynomials. The Feng-Tzeng algorithm appears in [83], which also shows how to use their algorithm for decoding up to the Hartmann-Tzeng and Roos BCH bounds. A preceding paper, [82] shows how to solve the multi-sequence problem using a generalization of the Euclidean algorithm, essentially producing a Gröbner basis approach. The algorithm attributed to Kötter [193] is clearly described in [230]. Other algorithms for the interpolation step are in [188] and in [252], which puts a variety of algorithms under the unifying framework of displacements.

The factorization step was efficiently expressed in [297]. The description presented of the Roth-Ruckenstein algorithm draws very closely in parts from the excellent tutorial paper [230]. Alternative factorization algorithms appear in [114, 9, 81, 379].

Chapter 8

Other Important Block Codes

8.1 Introduction

There are a variety of block codes of both historical and practical importance which are used either as building blocks or components of other systems, which we have not yet seen in this book. In this chapter we introduce some of the most important of these.

8.2 Hadamard Matrices, Codes, and Transforms

8.2.1 Introduction to Hadamard Matrices

A Hadamard matrix of order n is an $n \times n$ matrix H_n of ± 1 such that

$$H_n H_n^T = nI.$$

That is, by normalizing H_n by $1/\sqrt{n}$ an orthogonal matrix is obtained. The distinct columns of H are pairwise orthogonal, as are the rows. Some examples of Hadamard matrices are:

$$H_1 = [1] \quad H_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \quad H_4 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix}$$
$$H_8 = \begin{bmatrix} 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \end{bmatrix}. \tag{8.1}$$

The operation of computing $\mathbf{r}H_n$, where $\mathbf{r}$ is a row vector of length n, is sometimes called computing the **Hadamard transform** of $\mathbf{r}$. As we show in Section 8.3.3, there are fast algorithms for computing the Hadamard transform which are useful for decoding certain Reed-Muller codes (among other things). Furthermore, the Hadamard matrices can be used to define some error correction codes.

It is clear that multiplying a row or a column of H_n by -1 produces another Hadamard matrix. By a sequence of such operations, a Hadamard matrix can be obtained which has the first row and the first column equal to all ones. Such a matrix is said to be *normalized.*

Some of the operations associated with Hadamard matrices can be expressed using the Kronecker product.

Definition 8.1 The **Kronecker product** $A \otimes B$ of an $m \times n$ matrix A with a $p \times q$ matrix B is the $mp \times nq$ obtained by replacing every element a_{ij} of A with the matrix $a_{ij}B$. The Kronecker product is associative and distributive, but not commutative. □

Example 8.1 Let

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{bmatrix} \quad \text{and} \quad B = \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}.$$

Then

$$A \otimes B = \begin{bmatrix} a_{11}b_{11} & a_{11}b_{12} & a_{12}b_{11} & a_{12}b_{12} & a_{13}b_{11} & a_{13}b_{12} \\ a_{11}b_{21} & a_{11}b_{22} & a_{12}b_{21} & a_{12}b_{22} & a_{13}b_{21} & a_{13}b_{22} \\ a_{21}b_{11} & a_{21}b_{12} & a_{22}b_{11} & a_{22}b_{12} & a_{23}b_{11} & a_{23}b_{12} \\ a_{21}b_{21} & a_{21}b_{22} & a_{22}b_{21} & a_{22}b_{22} & a_{23}b_{21} & a_{23}b_{22} \end{bmatrix}.$$

□

Theorem 8.1 *The Kronecker product has the following properties [246, ch. 9]:*

1. $A \otimes B \neq B \otimes A$ *in general. (The Kronecker product does not commute.)*
2. *For a scalar* x, $(xA) \otimes B = A \otimes (xB) = x(A \otimes B)$.
3. *Distributive properties:*

$$(A + B) \otimes C = (A \otimes C) + (B \otimes C).$$

$$A \otimes (B + C) = (A \otimes B) + (A \otimes C).$$

4. *Associative property:* $(A \otimes B) \otimes C = A \otimes (B \otimes C)$.
5. *Transposes:* $(A \otimes B)^T = A^T \otimes B^T$.
6. *Trace (for square* A *and* B*):* $\operatorname{tr}(A \otimes B) = \operatorname{tr}(A)\operatorname{tr}(B)$.
7. *If* A *is diagonal and* B *is diagonal, then* $A \otimes B$ *is diagonal.*
8. *Determinant, where* A *is* $m \times m$ *and* B *is* $n \times n$*:* $\det(A \otimes B) = \det(A)^n \det(B)^m$.
9. *The Kronecker product theorem:*

$$(A \otimes B)(C \otimes D) = (AC) \otimes (BD), \tag{8.2}$$

provided that the matrices are shaped such that the indicated products are allowed.

10. *Inverses: If* A *and* B *are nonsingular then* $A \otimes B$ *is nonsingular and*

$$(A \otimes B)^{-1} = A^{-1} \otimes B^{-1}. \tag{8.3}$$

Returning now to Hadamard matrices, it may be observed that the Hadamard matrices in (8.1) have the structure

$$H_{2n} = H_2 \otimes H_n = \begin{bmatrix} H_n & H_n \\ H_n & -H_n \end{bmatrix}.$$

This works in general:

Theorem 8.2 *If* H_n *is a Hadamard matrix, then so is* $H_{2n} = H_2 \otimes H_n$.

Proof By the properties of the Kronecker product,

$$H_{2n}H_{2n}^T = (H_2 \otimes H_n)(H_2 \otimes H_n)^T = H_2H_2^T \otimes H_nH_n^T = (2I_2) \otimes (nI_n)$$
$$= 2n(I_2 \otimes I_n) = 2nI_{2n}.$$

□

This construction of Hadamard matrices is referred to as the Sylvester construction. By this construction, Hadamard matrices of sizes 1, 2, 4, 8, 16, 32, etc., exist. However, unless a Hadamard matrix of size 6 exists, for example, then this construction cannot be used to construct a Hadamard matrix of size 12. As the following theorem indicates, there is no Hadamard matrix of size 6.

Theorem 8.3 *A Hadamard matrix must have an order that is either 1, 2, or a multiple of 4.*

Proof [220, p. 44] Suppose without loss of generality that H_n is normalized. By column permutations, we can put the first three rows of H_n in the following form:

$$\underbrace{\begin{matrix} 1 & 1 & \cdots & 1 \\ 1 & 1 & \cdots & 1 \\ 1 & 1 & \cdots & 1 \end{matrix}}_{i} \quad \underbrace{\begin{matrix} 1 & 1 & \cdots & 1 \\ 1 & 1 & \cdots & 1 \\ -1 & -1 & \cdots & -1 \end{matrix}}_{j} \quad \underbrace{\begin{matrix} 1 & 1 & \cdots & 1 \\ -1 & -1 & \cdots & -1 \\ 1 & 1 & \cdots & 1 \end{matrix}}_{k} \quad \underbrace{\begin{matrix} 1 & 1 & \cdots & 1 \\ -1 & -1 & \cdots & -1 \\ -1 & -1 & \cdots & -1 \end{matrix}}_{l}$$

For example, j is the number of columns such that the first two rows of H_n have ones while the third row has negative ones. Since the rows are orthogonal, we have

$$\begin{aligned} i + j - k - l &= 0 \qquad &\text{(inner product of row 1 with row 2)} \\ i - j + k - l &= 0 \qquad &\text{(inner product of row 1 with row 3)} \\ i - j - k + l &= 0 \qquad &\text{(inner product of row 2 with row 3),} \end{aligned}$$

which collectively imply $i = j = k = l$. Thus $n = 4i$, so n must be a multiple of 4. (If $n = 1$ or 2, then there are not three rows to consider.) □

This theorem does not exclude the possibility of a Hadamard matrix of order 12. However, it cannot be obtained by the Sylvester construction.

8.2.2 The Paley Construction of Hadamard Matrices

Another method of constructing Hadamard matrices is by the Paley construction, which employs some number-theoretic concepts. This allows, for example, creation of the Hadamard matrix H_{12}. While in practice Hadamard matrices of order 4^k are most frequently employed, the Paley construction introduces the important concepts of quadratic residues and the Legendre symbol, both of which have application to other error correction codes.

Definition 8.2 For all numbers a such that $(a, p) = 1$, the number a is called a **quadratic residue** modulo p if the congruence $x^2 \equiv a \pmod{p}$ has some solution x. That is to say, a is the square of some number, modulo p. If a is not a quadratic residue, then a is called a **quadratic nonresidue**. □

If a is a quadratic residue modulo p, then so is $a + p$, so we consider as distinct residues only these which are distinct modulo p.

Example 8.2 The easiest way to find the quadratic residues modulo a prime p is to list the nonzero numbers modulo p, then square them.

Let $p = 7$. The set of nonzero numbers modulo p is $\{1, 2, 3, 4, 5, 6\}$. Squaring these numbers modulo p we obtain the list $\{1^2, 2^2, 3^2, 4^2, 5^2, 6^2\} = \{1, 4, 2, 2, 4, 1\}$. So the quadratic residues modulo 7 are $\{1, 2, 4\}$. The quadratic nonresidues are $\{3, 5, 6\}$. The number 9 is a quadratic residue modulo 7, since $9 = 7 + 2$, and 2 is a quadratic residue.

Now let $p = 11$. Forming the list of squares we have

$$\{1^2, 2^2, 3^2, 4^2, 5^2, 6^2, 7^2, 8^2, 9^2, 10^2\} = \{1, 4, 9, 5, 3, 3, 5, 9, 4, 1\}.$$

The quadratic residues modulo 11 are $\{1, 3, 4, 5, 9\}$. □

Theorem 8.4 *Quadratic residues have the following properties:*

1. *There are $(p - 1)/2$ quadratic residues modulo p for an odd prime p.*
2. *The product of two quadratic residues or two quadratic nonresidues is always a quadratic residue. The product of a quadratic residue and a quadratic nonresidue is a quadratic nonresidue.*
3. *If p is of the form $4k + 1$, then -1 is a quadratic residue modulo p. If p is of the form $4k + 3$, then -1 is a nonresidue modulo p.*

The Legendre symbol is a number theoretic function associated with quadratic residues.

Definition 8.3 Let p be an odd prime. The **Legendre symbol** $\chi_p(x)$ is defined as

$$\chi_p(x) = \begin{cases} 0 & \text{if } x \text{ is a multiple of } p \\ 1 & \text{if } x \text{ is a quadratic residue modulo } p \\ -1 & \text{if } x \text{ is a quadratic nonresidue modulo } p. \end{cases}$$

The Legendre symbol $\chi_p(x)$ is also denoted as $\left(\frac{x}{p}\right)$. □

Example 8.3 Let $p = 7$. The Legendre symbol values are

x:	0	1	2	3	4	5	6
$\chi_7(x)$:	0	1	1	−1	1	−1	−1

When $p = 11$ the Legendre symbol values are

x:	0	1	2	3	4	5	6	7	8	9	10
$\chi_{11}(x)$:	0	1	−1	1	1	1	−1	−1	−1	1	−1

□

The key to the Paley construction of Hadamard matrices is the following theorem.

Lemma 8.5 *[220, p. 46] For any $c \not\equiv 0 \pmod p$,*

$$\sum_{b=0}^{p-1} \chi_p(b)\chi_p(b + c) = -1. \tag{8.4}$$

Proof From Theorem 8.4 and the definition of the Legendre symbol, $\chi_p(xy) = \chi_p(x)\chi_p(y)$. Since $b = 0$ contributes nothing to the sum in (8.4), suppose $b \neq 0$. Let $z \equiv (b+c)b^{-1} \pmod{p}$. As b runs from $1, 2, \ldots, p-1$, z takes on distinct values in $0, 2, 3, \ldots, p-1$, but not the value 1. Then

$$\sum_{b=0}^{p-1} \chi_p(b)\chi_p(b+c) = \sum_{b=1}^{p-1} \chi_p(b)\chi_p(bz) = \sum_{b=1}^{p-1} \chi_p(b)\chi_p(b)\chi_p(z) = \sum_{b=1}^{p-1} \chi_p(b)^2\chi_p(z)$$
$$= \sum_{z=0}^{p-1} \chi_p(z) - \chi_p(1) = 0 - \chi_p(1) = -1,$$

where the last equality follows since half of the numbers z from 0 to $p-1$ have $\chi_p(z) = -1$ and the other half $\chi_p(z) = 1$, by Theorem 8.4. □

With this background, we can now define the Paley construction.

1. First, construct the $p \times p$ *Jacobsthal matrix* J_p, with elements q_{ij} given by $q_{ij} = \chi_p(j-i)$ (with zero-based indexing). Note that the first row of the matrix is $\chi_p(j)$, which is just the Legendre symbol sequence. The other rows are obtained by cyclic shifting.

2. Second, form the matrix

$$H_{p+1} = \begin{bmatrix} 1 & \mathbf{1}^T \\ \mathbf{1} & J_p - I_p \end{bmatrix},$$

where $\mathbf{1}$ is a column vector of length p containing all ones.

Example 8.4 Let $p = 7$. For the first row of the matrix, see Example 8.3.

$$J_7 = \begin{bmatrix} 0 & 1 & 1 & -1 & 1 & -1 & -1 \\ -1 & 0 & 1 & 1 & -1 & 1 & -1 \\ -1 & -1 & 0 & 1 & 1 & -1 & 1 \\ 1 & -1 & -1 & 0 & 1 & 1 & -1 \\ -1 & 1 & -1 & -1 & 0 & 1 & 1 \\ 1 & -1 & 1 & -1 & -1 & 0 & 1 \\ 1 & 1 & -1 & 1 & -1 & -1 & 0 \end{bmatrix} \quad H_8 = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & 1 & -1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & 1 & -1 \\ 1 & -1 & -1 & -1 & 1 & 1 & -1 & 1 \\ 1 & 1 & -1 & -1 & -1 & 1 & 1 & -1 \\ 1 & -1 & 1 & -1 & -1 & -1 & 1 & 1 \\ 1 & 1 & -1 & 1 & -1 & -1 & -1 & 1 \\ 1 & 1 & 1 & -1 & 1 & -1 & -1 & -1 \end{bmatrix}.$$

□

Example 8.5 We now show the construction of H_{12}. The 11×11 Jacobsthal matrix is

$$J_{11} = \begin{bmatrix} 0 & 1 & -1 & 1 & 1 & 1 & -1 & -1 & -1 & 1 & -1 \\ -1 & 0 & 1 & -1 & 1 & 1 & 1 & -1 & -1 & -1 & 1 \\ 1 & -1 & 0 & 1 & -1 & 1 & 1 & 1 & -1 & -1 & -1 \\ -1 & 1 & -1 & 0 & 1 & -1 & 1 & 1 & 1 & -1 & -1 \\ -1 & -1 & 1 & -1 & 0 & 1 & -1 & 1 & 1 & 1 & -1 \\ -1 & -1 & -1 & 1 & -1 & 0 & 1 & -1 & 1 & 1 & 1 \\ 1 & -1 & -1 & -1 & 1 & -1 & 0 & 1 & -1 & 1 & 1 \\ 1 & 1 & -1 & -1 & -1 & 1 & -1 & 0 & 1 & -1 & 1 \\ 1 & 1 & 1 & -1 & -1 & -1 & 1 & -1 & 0 & 1 & -1 \\ -1 & 1 & 1 & 1 & -1 & -1 & -1 & 1 & -1 & 0 & 1 \\ 1 & -1 & 1 & 1 & 1 & -1 & -1 & -1 & 1 & -1 & 0 \end{bmatrix}$$

and the Hadamard matrix is

$$H_{12} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 & 1 & 1 & -1 & -1 & -1 & 1 & -1 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & 1 & -1 & -1 & -1 & 1 \\ 1 & 1 & -1 & -1 & 1 & -1 & 1 & 1 & 1 & -1 & -1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 & 1 & 1 & -1 \\ 1 & -1 & -1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 & 1 \\ 1 & 1 & 1 & -1 & -1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & 1 & -1 & -1 & 1 & -1 \\ 1 & -1 & 1 & 1 & 1 & -1 & -1 & -1 & 1 & -1 & -1 & 1 \\ 1 & 1 & -1 & 1 & 1 & 1 & -1 & -1 & -1 & 1 & -1 & -1 \end{bmatrix} \tag{8.5}$$

□

The following lemma establishes that the Paley construction gives a Hadamard matrix.

Lemma 8.6 *Let J_p be a $p \times p$ Jacobsthal matrix. Then $J_p J_p^T = pI - U$ and $J_p U = U J_p = \mathbf{0}$, where U is the matrix of all ones.*

Proof Let $P = J_p J_p^T$. Then

$$p_{ii} = \sum_{k=0}^{p-1} q_{ik}^2 = p - 1 \qquad (\text{since } \chi_p^2(x) = 1 \text{ for } x \neq 0)$$

$$p_{ij} = \sum_{k=0}^{p-1} q_{ik} q_{jk} = \sum_{k=0}^{p-1} \chi_p(k-i)\chi_p(k-j)$$

$$= \sum_{b=0}^{p-1} \chi_p(b)\chi_p(b+c) = -1 \qquad (\text{subs. } b = k-i,\, c = i-j, \text{ then use Lemma 8.5}).$$

Also, $J_p U = \mathbf{0}$ since each row contains $(p-1)/2$ elements of 1 and $(p-1)/2$ elements of -1. □

Now

$$H_{p+1} H_{p+1}^T = \begin{bmatrix} 1 & \mathbf{1}^T \\ \mathbf{1} & J_p - I \end{bmatrix} \begin{bmatrix} 1 & \mathbf{1}^T \\ \mathbf{1} & J_p^T - I \end{bmatrix} = \begin{bmatrix} p+1 & \mathbf{0} \\ \mathbf{0} & U + (J_p - I)(J_p - I) \end{bmatrix}.$$

But from Lemma 8.6, $J + (J_p - I)(J_p^T - I) = U + pI - U - U - J_p - J_p^T + I = (p+1)I$. So $H_{p+1} H_{p+1}^T = (p+1) I_{p+1}$.

8.2.3 Hadamard Codes

Let A_n be the binary matrix obtained by replacing the 1s in a Hadamard matrix with 0s, and replacing the -1s with 1s. We have the following code constructions:

- By the orthogonality of H_n, any pair of distinct rows of A_n must agree in $n/2$ places and differ in $n/2$ places. Deleting the left column of A_n (since these bits are all the same and do not contribute anything to the code), the rows of the resulting matrix forms a code of length $n-1$ called the **Hadamard code**, denoted $\mathcal{A}_n$, having n codewords and minimum distance $n/2$. This is also known as the **simplex code**.
- By including the binary-complements of all codewords in $\mathcal{A}_n$ we obtain the code $\mathcal{B}_n$ which has $2n$ codewords of length $n-1$ and a minimum distance of $n/2 - 1$.

- Starting from A_n again, if we adjoin the binary complements of the rows of A_n, we obtain a code with code length n, $2n$ codewords, and minimum distance $n/2$. This code is denoted $\mathcal{C}$.

This book does not treat many nonlinear codes. However, if any of these codes are constructed using a Paley matrix with $n > 8$, then the codes are nonlinear. (The linear span of the nonlinear code is a quadratic residue code.) Interestingly, if the Paley Hadamard matrix is used in the construction of $\mathcal{A}_n$ or $\mathcal{B}_n$, then the codes are cyclic, but not necessarily linear. If the codes are constructed from Hadamard matrices constructed using the Sylvester construction, the codes are linear.

8.3 Reed-Muller Codes

Reed-Muller codes were among the first codes to be deployed in space applications, being used in the deep space probes flown from 1969 to 1977 [373, p. 149]. They were probably the first family of codes to provide a mechanism for obtaining a desired minimum distance. And, while they have been largely displaced by Reed-Solomon codes in volume of practice, they have a fast maximum likelihood decoding algorithm which is still very attractive. They are also used as components in several other systems. Furthermore, there are a variety of constructions for Reed-Muller codes which has made them useful in many theoretical developments.

8.3.1 Boolean Functions

Reed-Muller codes are closely tied to functions of Boolean variables and can be described as multinomials over the field $GF(2)$ [284]. Consider a Boolean function of m variables, $f(\nu_1, \nu_2, \ldots, \nu_m)$, which is a mapping from the vector space V_m of binary m-tuples to the binary numbers $\{0, 1\}$. Such functions can be represented using a *truth table*, which is an exhaustive listing of the input/output values. Boolean functions can also be written in terms of the variables.

Example 8.6 The table below is a truth table for two functions of the variables ν_1, ν_2, ν_3 and ν_4.

$\nu_4 =$	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
$\nu_3 =$	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
$\nu_2 =$	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
$\nu_1 =$	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
$f_1 =$	0	1	1	0	1	0	0	1	1	0	0	1	0	1	1	0
$f_2 =$	1	1	1	1	1	0	0	1	1	0	1	0	1	1	0	0

It can be verified (using, e.g., methods from elementary digital logic design) that

$$f_1(\nu_1, \nu_2, \nu_3, \nu_4) = \nu_1 + \nu_2 + \nu_3 + \nu_4$$

and that

$$f_2(\nu_1, \nu_2, \nu_3, \nu_4) = 1 + \nu_1\nu_4 + \nu_1\nu_3 + \nu_2\nu_3.$$

□

The columns of the truth table can be numbered from 0 to $2^m - 1$ using a base-2 representation with ν_1 as the least-significant bit. Then without ambiguity, the functions can be represented simply using their bit strings. From Example 8.6,

$$\mathbf{f}_1 = (0110100110010110)$$

$$\mathbf{f}_2 = (1111100110101100).$$

The number of distinct Boolean functions in m variables is the number of distinct binary sequences of length 2^m, which is 2^{2^m}. The set M of all Boolean functions in m variables forms a vector space that has a basis

$$\{1, v_1, v_2, \ldots, v_m, v_1v_2, v_1v_3, \ldots, v_{m-1}v_m, \cdots, v_1v_2v_3\cdots v_m\}.$$

Every function f in this space can be represented as a linear combination of these basis functions:

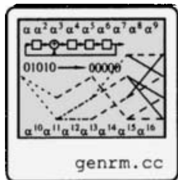
$$f = a_0 1 + a_1v_1 + a_2v_2 + \cdots a_mv_m + a_{12}v_1v_2 + \cdots + a_{12\ldots m}v_1v_2\cdots v_m.$$

Functional and vector notation can be used interchangeably. Here are some examples of some basic functions and their vector representations:

$$\begin{aligned}
1 &\leftrightarrow \mathbf{1} = 1111111111111111\\
v_1 &\leftrightarrow \boldsymbol{\nu}_1 = 0101010101010101\\
v_2 &\leftrightarrow \boldsymbol{\nu}_2 = 0011001100110011\\
v_3 &\leftrightarrow \boldsymbol{\nu}_3 = 0000111100001111\\
v_4 &\leftrightarrow \boldsymbol{\nu}_4 = 0000000011111111\\
v_1v_2 &\leftrightarrow \boldsymbol{\nu}_1\boldsymbol{\nu}_2 = 0001000100010001\\
v_1v_2v_3v_4 &\leftrightarrow \boldsymbol{\nu}_1\boldsymbol{\nu}_2\boldsymbol{\nu}_3\boldsymbol{\nu}_4 = 0000000000000001.
\end{aligned}$$

As this example demonstrates, juxtaposition of vectors represents the corresponding Boolean 'and' function, element by element. A vector representing a function can be written as

$$\mathbf{f} = a_0\mathbf{1} + a_1\boldsymbol{\nu}_1 + a_2\boldsymbol{\nu}_2 + \cdots a_m\boldsymbol{\nu}_m + a_{12}\boldsymbol{\nu}_1\boldsymbol{\nu}_2 + \cdots + a_{12\ldots m}\boldsymbol{\nu}_1\boldsymbol{\nu}_2\cdots\boldsymbol{\nu}_m.$$



8.3.2 Definition of the Reed-Muller Codes

Definition 8.4 [373, p. 151] The binary Reed-Muller code $RM(r, m)$ of order r and length 2^m consists of all linear combinations of vectors $\mathbf{f}$ associated with Boolean functions f that are monomials of degree $\leq r$ in m variables. □

Example 8.7 The $RM(1, 3)$ code has length $2^3 = 8$. The monomials of degree ≤ 1 are $\{1, v_1, v_2, v_3\}$, with associated vectors

$$\begin{array}{rcccccccc}
1 \leftrightarrow \mathbf{1} = & (1 & 1 & 1 & 1 & 1 & 1 & 1 & 1)\\
v_3 \leftrightarrow \mathbf{v}_3 = & (0 & 0 & 0 & 0 & 1 & 1 & 1 & 1)\\
v_2 \leftrightarrow \mathbf{v}_2 = & (0 & 0 & 1 & 1 & 0 & 0 & 1 & 1)\\
v_1 \leftrightarrow \mathbf{v}_1 = & (0 & 1 & 0 & 1 & 0 & 1 & 0 & 1).
\end{array}$$

It is natural to describe the code using a generator matrix having these vectors as rows.

$$G = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1\\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1\\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1\\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}. \tag{8.6}$$

This is an $(8, 4, 4)$ code; it is single-error correcting and double-error detecting. This is also the extended Hamming code (obtained by adding an extra parity bit to the $(7, 4)$ Hamming code). □

Example 8.8 The $RM(2, 4)$ code has length 16 and is obtained by linear combinations of the monomials up to degree 2, which are

$$\{1, v_1, v_2, v_3, v_4, v_1v_2, v_1v_3, v_1v_4, v_2v_3, v_2v_4, v_3v_4\}$$

with the following corresponding vector representations:

$\mathbf{1} =$	(1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1)
$\mathbf{v}_4 =$	(0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1)
$\mathbf{v}_3 =$	(0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1)
$\mathbf{v}_2 =$	(0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1)
$\mathbf{v}_1 =$	(0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1)
$\mathbf{v}_3\mathbf{v}_4 =$	(0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1)
$\mathbf{v}_2\mathbf{v}_4 =$	(0	0	0	0	0	0	0	0	0	0	1	1	0	0	1	1)
$\mathbf{v}_2\mathbf{v}_3 =$	(0	0	0	0	0	0	1	1	0	0	0	0	0	0	1	1)
$\mathbf{v}_1\mathbf{v}_4 =$	(0	0	0	0	0	0	0	0	0	1	0	1	0	1	0	1)
$\mathbf{v}_1\mathbf{v}_3 =$	(0	0	0	0	0	1	0	1	0	0	0	0	0	1	0	1)
$\mathbf{v}_1\mathbf{v}_2 =$	(0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1).

This is a (16, 11) code, with minimum distance 4. □

In general for an $RM(r, m)$ code, the dimension is

$$k = 1 + \binom{m}{1} + \binom{m}{2} + \cdots + \binom{m}{r}.$$

The codes are linear, but not cyclic.

As the following lemma states, we can recursively construct an $RM(r + 1, m + 1)$ code — twice the length — from an $RM(r, m)$ and $RM(r + 1, m)$ code. In this context, the notation $(\mathbf{f}, \mathbf{g})$ means the *concatenation* of the vectors $\mathbf{f}$ and $\mathbf{g}$.

Lemma 8.7 $RM(r+1, m+1) = \{(\mathbf{f}, \mathbf{f}+\mathbf{g})$ *for all* $\mathbf{f} \in RM(r+1, m)$ *and* $\mathbf{g} \in RM(r, m)\}$.

Proof The codewords of $RM(r + 1, m + 1)$ are associated with Boolean functions in $m + 1$ variables of degree $\leq r + 1$. If $c(v_1, \dots, v_{m+1})$ is such a function (i.e., it represents a codeword) we can write it as

$$c(v_1, \dots, v_{m+1}) = f(v_1, \dots, v_m) + v_{m+1}g(v_1, \dots, v_m),$$

where f is a Boolean function in m variables of degree $\leq r + 1$, and hence represents a codeword in $RM(r + 1, m)$, and g is a Boolean function in m variables with degree $\leq r$, representing a Boolean function in $RM(r, m)$. The corresponding functions $\mathbf{f}$ and $\mathbf{g}$ are thus in $RM(r + 1, m)$ and $RM(r, m)$, respectively.

Now let $\tilde{f}(v_1, v_2, \dots, v_{m+1}) = f(v_1, v_2, \dots, v_m) + 0 \cdot v_{m+1}$ represent a codeword in $RM(r + 1, m + 1)$ and let $\tilde{g}(v_1, v_2, \dots, v_{m+1}) = v_{m+1}g(v_1, v_2, \dots, v_m)$ represent a codeword in $RM(r + 1, m + 1)$. The associated vectors, which are codewords in $RM(r + 1, m + 1)$, are

$$\tilde{\mathbf{f}} = (\mathbf{f}, \mathbf{f}) \qquad \text{and} \qquad \tilde{\mathbf{g}} = (\mathbf{0}, \mathbf{g}).$$

Their linear combination $(\mathbf{f}, \mathbf{f} + \mathbf{g})$ must therefore also be a codeword in $RM(r + 1, m + 1)$. □

We now use this lemma to compute the minimum distance of an $RM(r, m)$ code.

Theorem 8.8 *The minimum distance of $RM(r, m)$ is 2^{m-r}.*

Proof By induction. When $m = 1$ the $RM(0, 1)$ code is built from the basis $\{1\}$, giving rise to two codewords: it is the length-2 repetition code. In this case $d_{\min} = 2$. The $RM(1, 1)$ code is built upon the basis vectors $\{1, v_1\}$ and has four codewords of length two: 00, 01, 10, 11. Hence $d_{\min} = 1$.

As an inductive hypothesis, assume that up to m and for $0 \le r \le m$ the minimum distance is 2^{m-r}. We will show that $d_{\min}$ for $RM(r, m + 1)$ is 2^{m-r+1}.

Let $\mathbf{f}$ and $\mathbf{f}'$ be in $RM(r, m)$ and let $\mathbf{g}$ and $\mathbf{g}'$ be in $RM(r - 1, m)$. By Lemma 8.7, the vectors $\mathbf{c}_1 = (\mathbf{f}, \mathbf{f} + \mathbf{g})$ and $\mathbf{c}_2 = (\mathbf{f}', \mathbf{f}' + \mathbf{g}')$ must be in $RM(r, m + 1)$.

If $\mathbf{g} = \mathbf{g}'$ then $d(\mathbf{c}_1, \mathbf{c}_2) = d((\mathbf{f}, \mathbf{f} + \mathbf{g}), (\mathbf{f}', \mathbf{f}' + \mathbf{g})) = d((\mathbf{f}, \mathbf{f}'), (\mathbf{f}, \mathbf{f}')) = 2d(\mathbf{f}, \mathbf{f}') \ge 22^{m-r}$ by the inductive hypothesis. If $\mathbf{g} \ne \mathbf{g}'$ then

$$d(\mathbf{c}_1, \mathbf{c}_2) = w(\mathbf{f} - \mathbf{f}') + w(\mathbf{g} - \mathbf{g}' + \mathbf{f} - \mathbf{f}').$$

Claim: $w(\mathbf{x} + \mathbf{y}) \ge w(\mathbf{x}) - w(\mathbf{y})$. Proof: Let w_{xy} be the number of places in which the nonzero digits of $\mathbf{x}$ and $\mathbf{y}$ overlap. Then $w(\mathbf{x} + \mathbf{y}) = (w(\mathbf{x}) - w_{xy}) + (w(\mathbf{y}) - w_{xy})$. But since $2w(\mathbf{y}) \ge 2w_{xy}$, the result follows.

By this result,

$$d(\mathbf{c}_1, \mathbf{c}_2) \ge w(\mathbf{f} - \mathbf{f}') + w(\mathbf{g} - \mathbf{g}') - w(\mathbf{f} - \mathbf{f}') = w(\mathbf{g} - \mathbf{g}').$$

But $\mathbf{g} - \mathbf{g}' \in RM(r - 1, m)$, so that $w(\mathbf{g} - \mathbf{g}') \ge 2^{m-(r-1)} = 2^{m-r+1}$. □

The following theorem is useful in characterizing the duals of RM codes.

Theorem 8.9 *For $0 \le r \le m - 1$, the $RM(m - r - 1, m)$ code is dual to the $RM(r, m)$ code.*

Proof [373, p. 154] Let $\mathbf{a}$ be a codeword in $RM(m - r - 1, m)$ and let $\mathbf{b}$ be a codeword in $RM(r, m)$. Associated with $\mathbf{a}$ is a polynomial $a(v_1, v_2, \ldots, v_m)$ of degree $\le m - r - 1$; associated with $\mathbf{b}$ is a polynomial $b(v_1, v_2, \ldots, v_m)$ of degree $\le r$. The product polynomial has degree $\le m - 1$, and thus corresponds to a codeword in the $RM(m - 1, m)$ code, with vector representation $\mathbf{ab}$. Since the minimum distance of $RM(m - r - 1, m)$ is 2^{r-1} and the minimum distance of $RM(m - 1, m)$ is 2^{m-r}, the codeword $\mathbf{ab}$ must have even weight. Thus $\mathbf{a} \cdot \mathbf{b} \equiv 0 \pmod 2$. From this, $RM(m - r - 1, m)$ must be a subset of the dual code to $RM(r, m)$. Note that

$$\begin{aligned}
&\dim(RM(r, m)) + \dim(RM(m - r - 1, m)) \\
&= 1 + \binom{m}{1} + \cdots + \binom{m}{r} + 1 + \binom{m}{1} + \binom{m}{2} + \cdots + \binom{m}{m-r-1} \\
&= 1 + \binom{m}{1} + \cdots \binom{m}{r} + \binom{m}{m} + \binom{m}{m-1} + \binom{m}{m-2} + \cdots + \binom{m}{r+1} \\
&= \sum_{i=0}^{m} \binom{m}{i} = 2^m.
\end{aligned}$$

By the theorem regarding the dimensionality of dual codes, Theorem 2.8, $RM(m-r-1, m)$ must be the dual to $RM(r, m)$. □

It is clear that the weight distribution of $RM(1, m)$ codes is $A_0 = 1$, $A_{2^m} = 1$ and $A_{2^{m-1}} = 2^{m+1} - 2$. Beyond these simple results, the weight distributions are more complicated.

8.3.3 Encoding and Decoding Algorithms for First-Order RM Codes

In this section we describe algorithms for encoding and decoding $RM(1, m)$ codes, which are $(2^m, m+1, 2^{m-1})$ codes. In Section 8.3.4 we present algorithms for more general RM codes.

Encoding $RM(1, m)$ Codes

Consider the $RM(1, 3)$ code generated by

$$\mathbf{c} = (c_0, c_1, \ldots, c_7) = \mathbf{m}G = (m_0, m_3, m_2, m_1) \begin{bmatrix} 1 \\ \mathbf{v}_3 \\ \mathbf{v}_2 \\ \mathbf{v}_1 \end{bmatrix}$$

$$= (m_0, m_3, m_2, m_1) \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}.$$

The columns of G consist of the numbers $(1, 0, 0, 0)$ through $(1, 1, 1, 1)$ in increasing binary counting order (with the least-significant bit on the right). This sequence of bit values can thus be obtained using a conventional binary digital counter. A block diagram of an encoding circuit embodying this idea is shown in Figure 8.1.

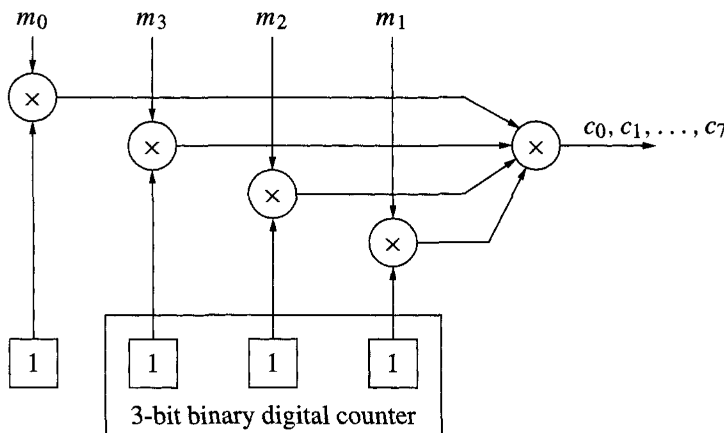


Figure 8.1: An encoder circuit for a $RM(1, 3)$ code.

Decoding $RM(1, m)$ Codes

The idea behind the decoder is to compare the received sequence $\mathbf{r}$ with every codeword in $RM(1, m)$ by means of correlation, then to select the codeword with the highest correlation. As we shall show, because of the structure of the code these correlations can be computed using a Hadamard transform. The existence of a fast Hadamard transform algorithms makes this an efficient decoding algorithm.

Let $\mathbf{r} = (r_0, r_1, \ldots, r_{2^m-1})$ be the received sequence, and let $\mathbf{c} = (c_0, c_1, \ldots, c_{2^m-1})$

be a codeword. We note that

$$\begin{aligned}\sum_{i=0}^{2^m-1}(-1)^{r_i}(-1)^{c_i} &= \sum_{i=0}^{2^m-1}(-1)^{r_i+c_i} = \sum_{i=0}^{2^m-1}(-1)^{r_i \oplus c_i} \\ &= \sum_{i=0}^{2^m-1}(-1)^{d(r_i,c_i)} = 2^m - 2d(\mathbf{r},\mathbf{c}),\end{aligned} \tag{8.7}$$

where $\oplus$ denotes addition modulo 2 and $d(r_i, c_i)$ is the Hamming distance between the arguments. A sequence which minimizes $d(\mathbf{r}, \mathbf{c})$ has the largest number of positive terms in the sum on the right of (8.7) and therefore maximizes the sum.

Let $\mathcal{F}(\mathbf{r})$ be the transformation that converts binary $\{0, 1\}$ elements of $\mathbf{r}$ to binary ± 1 values of a vector $\mathbf{R}$ according to

$$\mathcal{F}(\mathbf{r}) = \mathcal{F}(r_0, r_1, \ldots, r_{2^m-1}) = \mathbf{R} = ((-1)^{r_0}, (-1)^{r_1}, \ldots, (-1)^{r_{2^m-1}}).$$

We refer to $\mathbf{R}$ as the bipolar representation of $\mathbf{r}$. Similarly define $\mathcal{F}(\mathbf{c}) = \mathbf{C} = (C_0, C_1, \ldots, C_{2^m-1})$. We define the correlation function

$$T = \text{cor}(\mathbf{R}, \mathbf{C}) = \text{cor}((R_0, R_1, \ldots, R_{2^m-1}), (C_0, C_1, \ldots, C_{2^m-1})) = \sum_{i=0}^{2^m-1} R_i C_i.$$

By (8.7), the codeword $\mathbf{c}$ which minimizes $d(\mathbf{r}, \mathbf{c})$ maximizes the correlation $\text{cor}(\mathbf{R}, \mathbf{C})$.

The decoding algorithm is summarized as: Compute $T_i = \text{cor}(\mathbf{R}, \mathbf{C}_i)$, where $\mathbf{C}_i = \mathcal{F}(\mathbf{c}_i)$ for each of the 2^{m+1} codewords, then select that codeword for which $\text{cor}(\mathbf{R}, \mathbf{C}_i)$ is the largest. The simultaneous computation of all the correlations can be represented as a matrix. Let $\mathbf{C}_i$ be represented as a *column* vector and let

$$H = \begin{bmatrix}\mathbf{C}_0 & \mathbf{C}_1 & \ldots & \mathbf{C}_{2^{m+1}-1}\end{bmatrix}.$$

Then all the correlations can be computed by

$$\mathbf{T} = \begin{bmatrix}T_0 & T_1 & \cdots & T_{2^{m+1}-1}\end{bmatrix} = \mathbf{R}H.$$

Recall that the generator matrix for the $RM(1, m)$ code can be written as

$$G = \begin{bmatrix} 1 \\ \nu_m \\ \nu_{m-1} \\ \vdots \\ \nu_1 \end{bmatrix}.$$

We actually find it convenient to deal explicitly with those codewords formed as linear combinations of only the vectors $\nu_1, \nu_1, \ldots, \nu_m$, since $1 + \mathbf{c}$ complements all the elements of $\mathbf{c}$, which corresponds to negating the elements of the transform $\mathbf{C}$. We therefore deal with the $2^m \times 2^m$ matrix H_{2^m}. Let us examine one of these matrices in detail. For the $RM(1, 3)$ code with G as in (8.6), the matrix H_8 can be written as

$$H_8 = \begin{array}{c} \begin{matrix} \;0 & \;\;1 & \;\;2 & \;\;3 & \;\;4 & \;\;5 & \;\;6 & \;\;7 \end{matrix} \\ \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \end{bmatrix} \end{array}. \tag{8.8}$$

Examination of this reveals that, with this column ordering, column 1 corresponds to $\mathcal{F}(\mathbf{v}_1)$, column 2 corresponds to $\mathcal{F}(\mathbf{v}_2)$, and column 4 corresponds to $\mathcal{F}(\mathbf{v}_3)$. In general, the ith column corresponds to the linear combination of $i_1\mathbf{v}_1 + i_2\mathbf{v}_2 + i_3\mathbf{v}_3$, where i has the binary representation

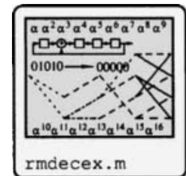
$$i = i_1 + 2i_2 + 4i_3.$$

We write the binary representation as $i = (i_3, i_2, i_1)_2$. In the general case, for an $2^m \times 2^m$ Hadamard matrix, we place $\mathcal{F}\left(\sum_{j=1}^m i_j \mathbf{v}_j\right)$ in the ith column where $i = (i_m, i_{m-1}, \ldots, i_1)_2$. The computation $\mathbf{R}H$ is referred to as the **Hadamard transform** of $\mathbf{R}$.

The decoding algorithm can be described as follows:

Algorithm 8.1 Decoding for $RM(1, m)$ Codes

1 **Input:** $\mathbf{r} = (r_0, r_1, \ldots, r_{2^m-1})$.
2 **Output:** A maximum-likelihood codeword $\hat{\mathbf{c}}$.
3 **Begin**
4 Find the bipolar representation $\mathbf{R} = \mathcal{F}(\mathbf{r})$.
5 Compute the Hadamard transform $\mathbf{T} = \mathbf{R}H_{2^m} = (t_0, t_1, \ldots, t_{2^m-1})$
6 Find the coordinate t_i with the largest magnitude
7 Let i have the binary expansion $(i_m, i_{m-1}, \ldots, i_1)_2$. ($i_1$ LSB)
8 if($t_i > 0$) (1 is not sent)
9 $\quad \hat{\mathbf{c}} = \sum_{j=1}^m i_j \mathbf{v}_j$
10 else (1 is sent – complement all the bits)
11 $\quad \hat{\mathbf{c}} = 1 + \sum_{j=1}^m i_j \mathbf{v}_j$
12 end (if)
13 End



Example 8.9 For the $RM(1, 3)$ code, suppose the received vector is

$$\mathbf{r} = \left[1, 0, 0, 1, 0, 0, 1, 0\right].$$

The steps of the algorithm follow:

1. Compute the transform: $\mathbf{R} = [-1, 1, 1, -1, 1, 1, -1, 1]$.
2. Compute $\mathbf{T} = \mathbf{R}H = [2, -2, 2, -2, -2, 2, -2, -6]$.
3. The maximum absolute element occurs at $t_7 = -6$, so $i = 7 = (1, 1, 1)_2$.
4. Since $t_7 < 0$, $\mathbf{c} = 1 + \mathbf{v}_1 + \mathbf{v}_2 + \mathbf{v}_3 = [1, 0, 0, 1, 0, 1, 1, 0]$.

□

Expediting Decoding Using the Fast Hadamard Transform

The main step of the algorithm is the computation of the Hadamard transform $\mathbf{R}H$. This can be considerably expedited by using a *fast Hadamard transform*, applicable to Hadamard matrices obtained via the Sylvester construction. This transform is analogous to the fast Fourier transform (FFT), but is over the set of numbers ± 1. It is built on some facts from linear algebra.

As we have seen, Sylvester Hadamard can be built by

$$H_{2^m} = H_2 \otimes H_{2^{m-1}}. \tag{8.9}$$

This gives the following factorization.

Theorem 8.10 *The matrix H_{2^m} can be written as*

$$H_{2^m} = M_{2^m}^{(1)} M_{2^m}^{(2)} \cdots M_{2^m}^{(m)}, \tag{8.10}$$

where

$$M_{2^m}^{(i)} = I_{2^{m-i}} \otimes H_2 \otimes I_{2^{i-1}},$$

and where I_p is a $p \times p$ identity matrix.

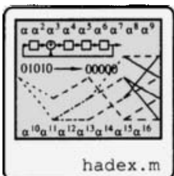
Proof By induction. When $m = 1$ the result holds, as may be easily verified. Assume, then, that (8.10) holds for m. We find that

$$\begin{aligned} M_{2^{m+1}}^{(i)} &= I_{2^{m+1-i}} \otimes H_2 \otimes I_{2^{i-1}} \\ &= (I_2 \otimes I_{2^{m-i}}) \otimes H_2 \otimes I_{2^{i-1}} && \text{(by the structure of the identity matrix)} \\ &= I_2 \otimes (I_{2^{m-i}} \otimes H_2 \otimes I_{2^{i-1}}) && \text{(associativity)} \\ &= I_2 \otimes M_{2^m}^{(i)} && \text{(definition).} \end{aligned}$$

Furthermore, by the definition, $M_{2^{m+1}}^{m+1} = H_2 \otimes I_{2^m}$. We have

$$\begin{aligned} H_{2^{m+1}} &= M_{2^{m+1}}^{(1)} M_{2^{m+1}}^{(2)} \cdots M_{2^{m+1}}^{(m+1)} \\ &= (I_2 \otimes M_{2^m}^{(1)})(I_2 \otimes M_{2^m}^{(2)}) \cdots (I_2 \otimes M_{2^m}^{(m)})(H_2 \otimes I_{2^m}) \\ &= (I_2^m H_2) \otimes (M_{2^m}^{(1)} M_{2^m}^{(2)} \cdots M_{2^m}^{(m)}) && \text{(Kronecker product theorem 8.2)} \\ &= H_2 \otimes H_{2^m}. \end{aligned}$$

□



Example 8.10 By the theorem, we have the factorization

$$H_8 = M_8^{(1)} M_8^{(2)} M_8^{(3)} = (I_{2^2} \otimes H_2 \otimes I_{2^0})(I_{2^1} \otimes H_2 \otimes I_{2^1})(I_{2^0} \otimes H_2 \otimes I_{2^2}).$$

Straightforward substitution and multiplication shows that this gives the matrix H_8 in (8.8).

Let $\mathbf{R} = [R_0, R_1, \ldots, R_7]$. Then the Hadamard transform can be written

$$\mathbf{T} = \mathbf{R}H_8 = \mathbf{R}(M_8^{(1)} M_8^{(2)} M_8^{(3)}) = \mathbf{R}(I_{2^2} \otimes H_2 \otimes I_{2^0})(I_{2^1} \otimes H_2 \otimes I_{2^1})(I_{2^0} \otimes H_2 \otimes I_{2^2}).$$

The matrices involved are

$$M_8^{(1)} = I_4 \otimes H_2 = \begin{bmatrix} 1 & 1 & & & & & & \\ 1 & -1 & & & & & & \\ & & 1 & 1 & & & & \\ & & 1 & -1 & & & & \\ & & & & 1 & 1 & & \\ & & & & 1 & -1 & & \\ & & & & & & 1 & 1 \\ & & & & & & 1 & -1 \end{bmatrix}$$

$$M_8^{(2)} = I_2 \otimes H_2 \otimes I_2 = \begin{bmatrix} 1 & 0 & 1 & 0 & & & & \\ 0 & 1 & 0 & 1 & & & & \\ 1 & 0 & -1 & 0 & & & & \\ 0 & 1 & 0 & -1 & & & & \\ & & & & 1 & 0 & 1 & 0 \\ & & & & 0 & 1 & 0 & 1 \\ & & & & 1 & 0 & -1 & 0 \\ & & & & 0 & 1 & 0 & -1 \end{bmatrix}$$

$$M_8^{(3)} = H_2 \otimes I_4 = \begin{bmatrix} 1 & & & & 1 & & & \\ & 1 & & & & 1 & & \\ & & 1 & & & & 1 & \\ & & & 1 & & & & 1 \\ 1 & & & & -1 & & & \\ & 1 & & & & -1 & & \\ & & 1 & & & & -1 & \\ & & & 1 & & & & -1 \end{bmatrix}.$$

Let $\mathbf{P}^{[0]} = \mathbf{R}$ and let $\mathbf{P}^{[k+1]} = \mathbf{P}^{[k]} M_8^{(k+1)}$. The stages of transform can be written

$$\mathbf{T} = \left(\mathbf{R} M_8^{(1)}\right) M_8^{(2)} M_8^{(3)} = \left(\mathbf{P}^{(1)} M_8^{(2)}\right) M_8^{(3)} = \mathbf{P}^{(2)} M_8^{(3)} = \mathbf{P}^{(3)}.$$

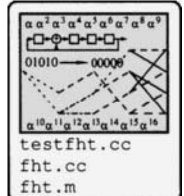


Figure 8.2 shows the flow diagram corresponding to the matrix multiplications, where arrows indicate the direction of flow, arrows incident along a line imply addition, and the coefficients -1 along the horizontal branches indicate the gain along their respective branch. At each stage, the two-point Hadamard transform is apparent. (At the first stage, the operations of H_2 are enclosed in the box to highlight the operation.) The interleaving of the various stages by virtue of the Kronecker product is similar to the "butterfly" pattern of the fast Fourier transform. ☐

The conventional computation of the Hadamard transform $\mathbf{R}H_{2^m}$ produces 2^m elements, each of which requires 2^m addition/subtraction operations, for a total complexity of $(2^m)^2$. The fast Hadamard transform has m stages, each of which requires 2^m addition/subtraction operations, for a total complexity of $m2^m$. This is still exponential in m (typical for maximum likelihood decoding), but much lower than brute force evaluation. Furthermore, as Figure 8.2 suggests, parallel/pipelined hardware architectures are possible.

The $RM(1, m)$ decoding algorithm employing the fast Hadamard transform is referred to as the "Green machine," after its developer at the Jet Propulsion Laboratory for the 1969 Mariner mission [373].

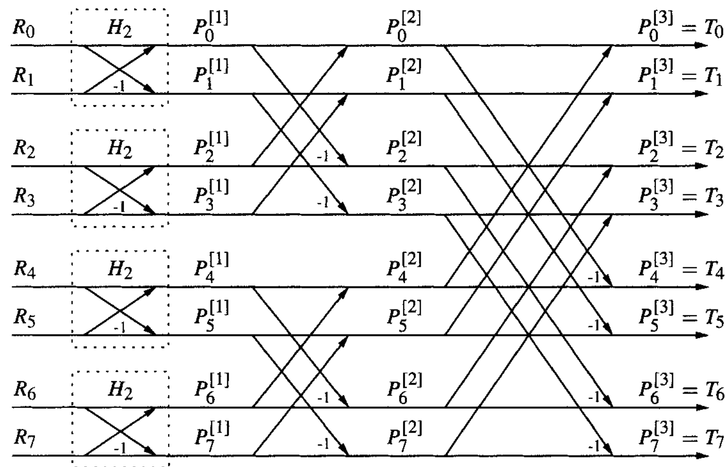


Figure 8.2: Signal flow diagram for the fast Hadamard transform.

8.3.4 The Reed Decoding Algorithm for $RM(r, m)$ Codes, $r \geq 1$

Efficient decoding algorithms for general RM codes rely upon the concept of *majority logic* decoding, in which multiple estimates of a bit value are obtained, and the decoded value is that value which occurs in the majority of estimates. We demonstrate this first for a $RM(2, 4)$ code, then develop a notation to extend this to other $RM(r, m)$ codes.

Details for an $RM(2, 4)$ Code

Let us write the generator for the $RM(2, 4)$ code as

$$G = \left[\begin{array}{cccccccccccccccc}
1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1 \\ \hline
0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1 \\
0&0&0&0&1&1&1&1&0&0&0&0&1&1&1&1 \\
0&0&1&1&0&0&1&1&0&0&1&1&0&0&1&1 \\
0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1 \\ \hline
0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1 \\
0&0&0&0&0&0&0&0&0&0&1&1&0&0&1&1 \\
0&0&0&0&0&0&1&1&0&0&0&0&0&0&1&1 \\
0&0&0&0&0&0&0&0&0&1&0&1&0&1&0&1 \\
0&0&0&0&0&1&0&1&0&0&0&0&0&1&0&1 \\
0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&1
\end{array}\right] = \left[\begin{array}{c}
1 \\ \hline
\mathbf{v}_4 \\ \mathbf{v}_3 \\ \mathbf{v}_2 \\ \mathbf{v}_1 \\ \hline
\mathbf{v}_3\mathbf{v}_4 \\ \mathbf{v}_2\mathbf{v}_4 \\ \vdots \\ \mathbf{v}_1\mathbf{v}_2
\end{array}\right] = \left[\begin{array}{c}
G_0 \\ \hline G_1 \\ \hline G_2
\end{array}\right].$$

We partition the 11 input bits to correspond to the rows of this matrix as

$$\mathbf{m} = (m_0, m_4, m_3, m_2, m_1, m_{34}, m_{24}, \ldots, m_{12}) = (\mathbf{m}_0, \mathbf{m}_1, \mathbf{m}_2).$$

Thus the bits in $\mathbf{m}_0$ are associated with the zeroth order term, the $\mathbf{m}_1$ bits are associated with the first order terms, and the second-order terms are associated with $\mathbf{m}_2$. The encoding

operation is

$$\mathbf{c} = (c_0, c_1, c_2, \ldots, c_{15}) = \mathbf{m}G = [\mathbf{m}_0, \mathbf{m}_1, \mathbf{m}_2] \begin{bmatrix} G_0 \\ \hline G_1 \\ \hline G_2 \end{bmatrix} = \mathbf{m}_0 G_0 + \mathbf{m}_1 G_1 + \mathbf{m}_2 G_2. \tag{8.11}$$

The general operation of the algorithm is as follows: Given a received vector $\mathbf{r}$, estimates are first obtained for the highest-order block of message bits, $\mathbf{m}_2$. Then $\mathbf{m}_2 G_2$ is subtracted off from $\mathbf{r}$, leaving only lower-order codewords. Then the message bits for $\mathbf{m}_1$ are obtained, then are subtracted, and so forth.

The key to finding the message bits comes from writing *multiple* equations for the same quantity and taking a majority vote. Selecting coded bits from (8.11) we have

$$\begin{aligned} c_0 &= m_0 \\ c_1 &= m_0 + m_1 \\ c_2 &= m_0 + m_2 \\ c_3 &= m_0 + m_1 + m_2 + m_{12}. \end{aligned}$$

Adding these code bits together (modulo 2) we obtain an equation for the message bit m_{12}:

$$c_0 + c_1 + c_2 + c_3 = m_{12}.$$

We can similarly obtain three other equations for the message bit m_{12},

$$\begin{aligned} c_4 + c_5 + c_6 + c_7 &= m_{12} \\ c_8 + c_9 + c_{10} + c_{11} &= m_{12} \\ c_{12} + c_{13} + c_{14} + c_{15} &= m_{12}. \end{aligned}$$

Given the code bits $c_0, \ldots, c_{15}$, we could compute m_{12} four *independent* ways. However, the code bits are not available at the receiver, only the received bits $\mathbf{r} = (r_0, r_1, \ldots, r_{15})$. We use this in conjunction with the equations above to obtain four estimates of m_{12}:

$$\begin{aligned} \hat{m}_{12}^{(1)} &= r_0 + r_1 + r_2 + r_3 \\ \hat{m}_{12}^{(2)} &= r_4 + r_5 + r_6 + r_7 \\ \hat{m}_{12}^{(3)} &= r_8 + r_9 + r_{10} + r_{11} \\ \hat{m}_{12}^{(4)} &= r_{12} + r_{13} + r_{14} + r_{15}. \end{aligned}$$

Expressions such as this, in which the check sums all yield the same message bit, are said to be *orthogonal*[1] on the message bit. From these four orthogonal equations, we determine the value of m_{12} by majority vote. Given $\hat{m}_{12}^{(i)}, i = 1, 2, 3, 4$, the decoded value $\hat{m}_{12}$ is

$$\hat{m}_{12} = \text{maj}(\hat{m}_{12}^{(1)}, \hat{m}_{12}^{(2)}, \hat{m}_{12}^{(3)}, \hat{m}_{12}^{(4)}),$$

where maj$(\cdots)$ returns the value that occurs most frequently among its arguments.

If errors occur such that only one of the $\hat{m}_{12}^{(i)}$ is incorrect, the majority vote gives the correct answer. If two of them are incorrect, then it is still possible to detect the occurrence of errors.

[1]This is a different usage from orthogonality in the vector-space sense.

It is similarly possible to set up multiple equations for the other second-order bits $m_{13}, m_{14}, \ldots, m_{34}$.

$$
\begin{aligned}
m_{13} &= c_0 + c_1 + c_4 + c_5 & m_{14} &= c_0 + c_1 + c_8 + c_9 \\
m_{13} &= c_2 + c_3 + c_6 + c_7 & m_{14} &= c_2 + c_3 + c_{10} + c_{11} \\
m_{13} &= c_8 + c_9 + c_{12} + c_{13} & m_{14} &= c_4 + c_5 + c_{12} + c_{13} \\
m_{13} &= c_{10} + c_{11} + c_{14} + c_{15} & m_{14} &= c_6 + c_7 + c_{14} + c_{15} \\
m_{23} &= c_0 + c_2 + c_4 + c_6 & m_{24} &= c_0 + c_2 + c_8 + c_{10} \\
m_{23} &= c_1 + c_3 + c_5 + c_7 & m_{24} &= c_1 + c_3 + c_9 + c_{11} \\
m_{23} &= c_8 + c_{10} + c_{12} + c_{14} & m_{24} &= c_4 + c_6 + c_{12} + c_{14} \\
m_{23} &= c_9 + c_{11} + c_{13} + c_{15} & m_{24} &= c_5 + c_7 + c_{13} + c_{15}
\end{aligned}
$$

$$
\begin{aligned}
m_{34} &= c_0 + c_4 + c_8 + c_{12} \\
m_{34} &= c_1 + c_5 + c_9 + c_{13} \\
m_{34} &= c_2 + c_6 + c_{10} + c_{14} \\
m_{34} &= c_3 + c_7 + c_{11} + c_{15}
\end{aligned}
\tag{8.12}
$$

Based upon these equations, majority logic decisions are computed for each element of the second-order block. These decisions are then stacked up to give the block

$$\hat{\mathbf{m}}_2 = (\hat{m}_{34}, \hat{m}_{24}, \hat{m}_{14}, \hat{m}_{23}, \hat{m}_{13}, \hat{m}_{12}).$$

We then "peel off" these decoded values to get

$$\mathbf{r}' = \mathbf{r} - \hat{\mathbf{m}}_2 G_2.$$

Now we repeat for the first-order bits. We have eight orthogonal check sums on each of the first-order message bits. For example,

$$
\begin{aligned}
m_1 &= c_0 + c_1 & m_1 &= c_2 + c_3 & m_1 &= c_4 + c_5 & m_1 &= c_6 + c_7 \\
m_1 &= c_8 + c_9 & m_1 &= c_{10} + c_{11} & m_1 &= c_{12} + c_{13} & m_1 &= c_{14} + c_{15}
\end{aligned}
$$

We use the bits of $\mathbf{r}'$ to obtain eight estimates,

$$
\begin{aligned}
m_1^{(1)} &= r_0' + r_1' & m_1^{(2)} &= r_2' + r_3' & m_1^{(3)} &= r_4' + r_5' & m_1^{(4)} &= r_6' + r_7' \\
m_1^{(5)} &= r_8' + r_9' & m_1^{(6)} &= r_{10}' + r_{11}' & m_1^{(7)} &= r_{12}' + r_{13}' & m_1^{(8)} &= r_{14}' + r_{15}'
\end{aligned}
$$

then make a decision using majority logic,

$$\hat{m}_1 = \operatorname{maj}(m_1^{(1)}, m_1^{(2)}, \ldots, m_1^{(8)}).$$

Similarly, eight orthogonal equations can be written on the bits m_2, m_3, and m_4, resulting in the estimate $\hat{\mathbf{m}}_1 = (\hat{m}_1, \hat{m}_2, \hat{m}_3, \hat{m}_4)$.

Having estimated $\hat{\mathbf{m}}_1$, we strip it off from the received signal,

$$\mathbf{r}'' = \mathbf{r}' - \hat{\mathbf{m}}_1 G_1$$

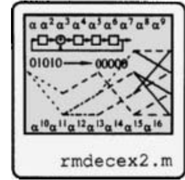
and look for m_0. But if the previous decodings are correct,

$$\mathbf{r}'' = m_0 \mathbf{1} + \mathbf{e}.$$

Then m_0 is obtained simply by majority vote:

$$\hat{m}_0 = \operatorname{maj}(r_0'', r_1'', \ldots, r_{15}'').$$

If at any stage there is no clear majority, then a decoding failure is declared.



Example 8.11 The computations described here are detailed in the indicated file. Suppose $\mathbf{m} =$ (00001001000), so the codeword is $\mathbf{c} = \mathbf{m}G =$ (0101011001010110). Suppose the received vector is $\mathbf{r} =$ (0101011011010110). The bit message estimates are

$$\hat{m}_{12}^{(1)} = r_0 + r_1 + r_2 + r_3 = 0 \qquad \hat{m}_{12}^{(2)} = r_4 + r_5 + r_6 + r_7 = 0$$
$$\hat{m}_{12}^{(3)} = r_8 + r_9 + r_{10} + r_{11} = 1 \qquad \hat{m}_{12}^{(4)} = r_{12} + r_{13} + r_{14} + r_{15} = 0$$

We obtain $\hat{m}_{12} = \text{maj}(0, 0, 1, 0) = 0$. We similarly find

$$\hat{m}_{13} = \text{maj}(0,0,1,0) = 0 \quad \hat{m}_{14} = \text{maj}(1,0,0,0) = 0 \quad \hat{m}_{23} = \text{maj}(1,1,0,1) = 1$$
$$\hat{m}_{24} = \text{maj}(1,0,0,0) = 0 \quad \hat{m}_{34} = \text{maj}(1,0,0,0) = 0,$$

so that $\hat{\mathbf{m}}_2 =$ (001000). Removing this decoded value from the received vector we obtain

$$\mathbf{r}' = \mathbf{r}' - \hat{\mathbf{m}}_2 G_2 = \mathbf{r}' - \hat{\mathbf{m}}_2 \begin{bmatrix} \mathbf{v}_3\mathbf{v}_4 \\ \mathbf{v}_2\mathbf{v}_4 \\ \mathbf{v}_2\mathbf{v}_3 \\ \mathbf{v}_1\mathbf{v}_4 \\ \mathbf{v}_1\mathbf{v}_3 \\ \mathbf{v}_1\mathbf{v}_2 \end{bmatrix} = (0101010111010101).$$

At the next block we have

$$\hat{m}_1 = \text{maj}(1,1,1,1,0,1,1,1) = 1 \quad \hat{m}_2 = \text{maj}(0,0,0,0,1,0,0,0) = 0$$
$$\hat{m}_3 = \text{maj}(0,0,0,0,1,0,0,0) = 0 \quad \hat{m}_4 = \text{maj}(1,0,0,0,0,0,0,0) = 0$$

so that $\hat{\mathbf{m}}_2 =$ (0001). We now remove this decoded block

$$\mathbf{r}'' = \mathbf{r}' - \hat{\mathbf{m}}_2 G_1 = \mathbf{r}' - \hat{\mathbf{m}}_2 \begin{bmatrix} \mathbf{v}_4 \\ \mathbf{v}_3 \\ \mathbf{v}_2 \\ \mathbf{v}_1 \end{bmatrix} = (0000000010000000).$$

The majority decision is $\hat{m}_0 = 0$. The overall decoded message is

$$\hat{\mathbf{m}} = (\hat{m}_0, \hat{\mathbf{m}}_1, \hat{\mathbf{m}}_2) = (00001001000).$$

This matches the message sent. □

A Geometric Viewpoint

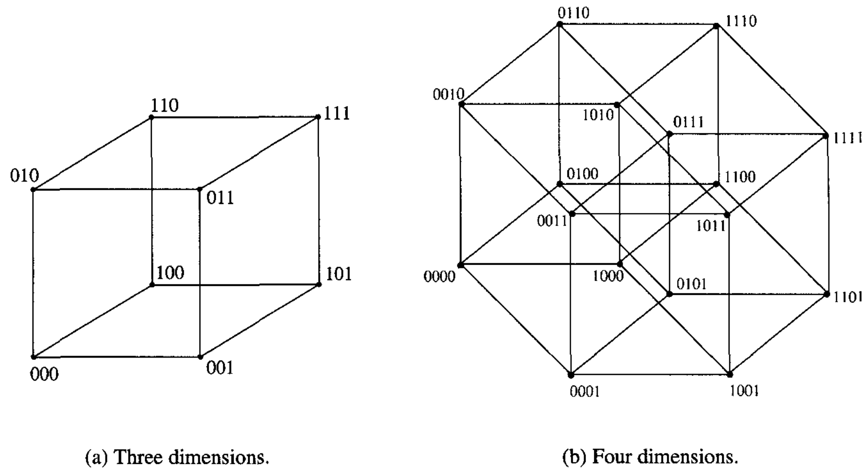
Clearly the key to employing majority logic decoding on an $RM(r, m)$ code is to find a description of the equations which are orthogonal on each bit. Consider, for example, the orthogonal equations for m_{34}, as seen in (8.12). Writing down the indices of the checking bits, we create the check set

$$S_{34} = \{\{0,4,8,12\}, \{1,5,9,13\}, \{2,6,10,14\}, \{3,7,11,15\}\}.$$

Now represent the indices in 4-bit binary,

$$S_{34} = \{\{(0000),(0100),(1000),(1100)\}, \{(0001),(0101),(1001),(1101)\},$$
$$\{(0010),(0110),(1010),(1110)\}, \{(0011),(0111),(1011),(1111)\}\}.$$

Within each subset there are pairs of binary numbers which are *adjacent*, differing by a single bit. We can represent this adjacency with a graph that has a vertex for each of the



(a) Three dimensions.

(b) Four dimensions.

Figure 8.3: Binary adjacency relationships in three and four dimensions.

numbers from 0000 to 1111, with edges between those vertices that are logically adjacent. The graph for a code with $n = 3$ is shown in Figure 8.3(a). It forms a conventional 3-dimensional cube. The graph for a code with $n = 4$ is shown in Figure 8.3(b); it forms a 4-dimensional hypercube. The check set S_{34} can be represented as subsets of the nodes in the graph. Figure 8.4 shows these sets by shading the "plane" defined by each of the fours check subsets. Similarly, the check sets for each the bits m_{12}, m_{13}, etc., form a set of planes.

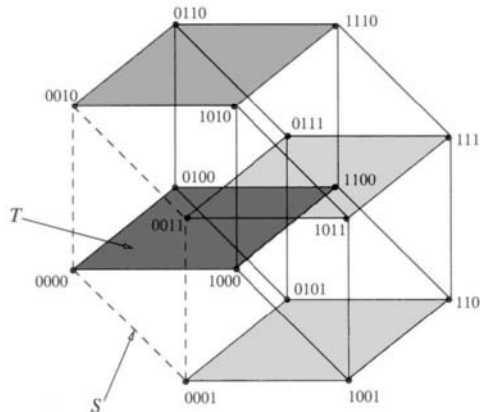


Figure 8.4: Planes shaded to represent the equations orthogonal on bit m_{34}.

With these observations, let us now develop the notation to describe the general situation. For a codeword $\mathbf{c} = (c_0, c_1, \ldots, c_{n-1})$, let the coordinate c_i be associated with the binary m-tuple P_i obtained by complementing the binary representation of the index i. For example, c_0 is associated with $P_0 = (1111)$ (since $0 = (0000)_2$) and c_6 is associated with $P_6 = (1001)$ (since $6 = (0110)_2$). We think of the P_i as points on the adjacency graph such as those

shown in Figure 8.3.

Each codeword $\mathbf{c}$ in $RM(r, m)$ forms an incidence vector for a subset of the graph, selecting points in the graph corresponding to 1 bits in the codeword. For example, the codeword $\mathbf{c} = (0101011001010110)$ is an incidence vector for the subset containing the points $\{P_1, P_3, P_5, P_6, P_9, P_{11}, P_{13}, P_{14}\}$.

Let $I = \{1, 2, \ldots, m\}$. We represent the basis vectors for the $RM(r, m)$ code as subsets of I. For example, the basis vector $\mathbf{v}_1$ is represented by the set $\{1\}$. The basis vector $\mathbf{v}_2\mathbf{v}_3$ is represented by the set $\{2, 3\}$. The basis vector $\mathbf{v}_2\mathbf{v}_3\mathbf{v}_4$ is represented by $\{2, 3, 4\}$. With this notation, we now define the procedure for finding the orthogonal check sums for the vector $\mathbf{v}_{i_1}\mathbf{v}_{i_2}\cdots\mathbf{v}_{i_p}$ [373, p. 160].

1. Let $S = \{S_1, S_2, \ldots, S_{2^{m-p}}\}$ be the subset of points associated with the incidence vector $\mathbf{v}_{i_1}\mathbf{v}_{i_2}\cdots\mathbf{v}_{i_p}$.
2. Let $\{j_1, j_2, \ldots, j_{m-p}\}$ be the set difference $I - \{i_1, i_2, \ldots, i_p\}$. Let T be the subset of points associated with the incidence vector $\mathbf{v}_{j_1}\mathbf{v}_{j_2}\cdots\mathbf{v}_{j_{m-p}}$. The set T is called the *complementary subspace to S*.
3. The first check sum consists of the sum of the coordinates specified by the points in T.
4. The other check sums are obtained by "translating" the set T by the points in S. That is, for each $S_i \in S$, we form the set $T + S_i$. The corresponding check sum consists of the sum of the coordinates specified by this set.

Example 8.12 Checksums for $RM(2, 4)$. Let us find checksums for $\mathbf{v}_3\mathbf{v}_4 = (0000000000001111)$.

1. The subset for which $\mathbf{v}_3\mathbf{v}_4$ is an incidence vector contains the points

$$S = \{P_{12}, P_{13}, P_{14}, P_{15}\} = \{(0011)(0010)(0001)(0000)\}.$$

   In Figure 8.4, the set S is indicated by the dashed lines.

2. The difference set is

$$\{j_1, j_2\} = \{1, 2, 3, 4\} - \{3, 4\} = \{1, 2\},$$

   which has the associated vector $\mathbf{v}_1\mathbf{v}_2 = (0001000100010001)$. This is the incidence vector for the set

$$T = \{P_3, P_7, P_{11}, P_{15}\} = \{(1100)(1000)(0100)(0000)\}.$$

   In Figure 8.4, the set T is the darkest of the shaded regions.

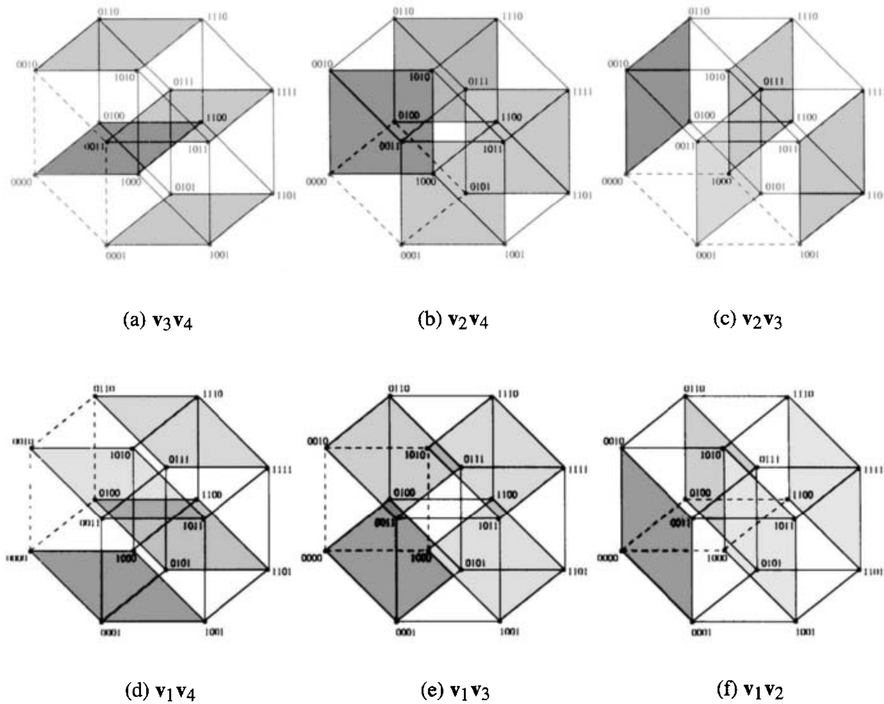
3. T represents the checksum $m_{34} = c_{12} + c_8 + c_4 + c_0$.
4. The translations of T by the nonzero elements of S are:

$$\begin{aligned}
\text{by } P_{12} = (0011) &\rightarrow \{(1111)(1011)(0111)(0011)\} = \{P_0, P_4, P_8, P_{12}\} \\
\text{by } P_{13} = (0010) &\rightarrow \{(1110)(1010)(0110)(0010)\} = \{P_1, P_5, P_9, P_{13}\} \\
\text{by } P_{14} = (0001) &\rightarrow \{(1101)(1001)(0101)(0001)\} = \{P_2, P_6, P_{10}, P_{14}\}.
\end{aligned}$$

   These correspond to the checksums

$$\begin{aligned}
m_{34} &= c_{15} + c_{11} + c_7 + c_3 \\
m_{34} &= c_{14} + c_{10} + c_6 + c_2 \\
m_{34} &= c_{13} + c_9 + c_5 + c_1,
\end{aligned}$$

   which are shown in the figure as shaded planes.



(a) $\mathbf{v}_3\mathbf{v}_4$ (b) $\mathbf{v}_2\mathbf{v}_4$ (c) $\mathbf{v}_2\mathbf{v}_3$

(d) $\mathbf{v}_1\mathbf{v}_4$ (e) $\mathbf{v}_1\mathbf{v}_3$ (f) $\mathbf{v}_1\mathbf{v}_2$

Figure 8.5: Geometric descriptions of parity check equations for second-order vectors of the $RM(2, 4)$ code.

Figure 8.5 indicates the check equations for all of the second-order vectors for the $RM(2, 4)$ code.

Now let us examine check sums for the first-order vectors.

1. For the vector $\mathbf{v}_4 = (0000000011111111)$ the set S is

$$S = \{P_8, P_9, P_{10}, P_{11}, P_{12}, P_{13}, P_{14}, P_{15}\}$$
$$= \{(0111), (0110), (0101), (0100), (0011), (0010), (0001), (0000)\}.$$

   These eight points are connected by the dashed lines in Figure 8.6(a).

2. The difference set is

$$\{1, 2, 3, 4\} - \{4\} = \{1, 2, 3\},$$

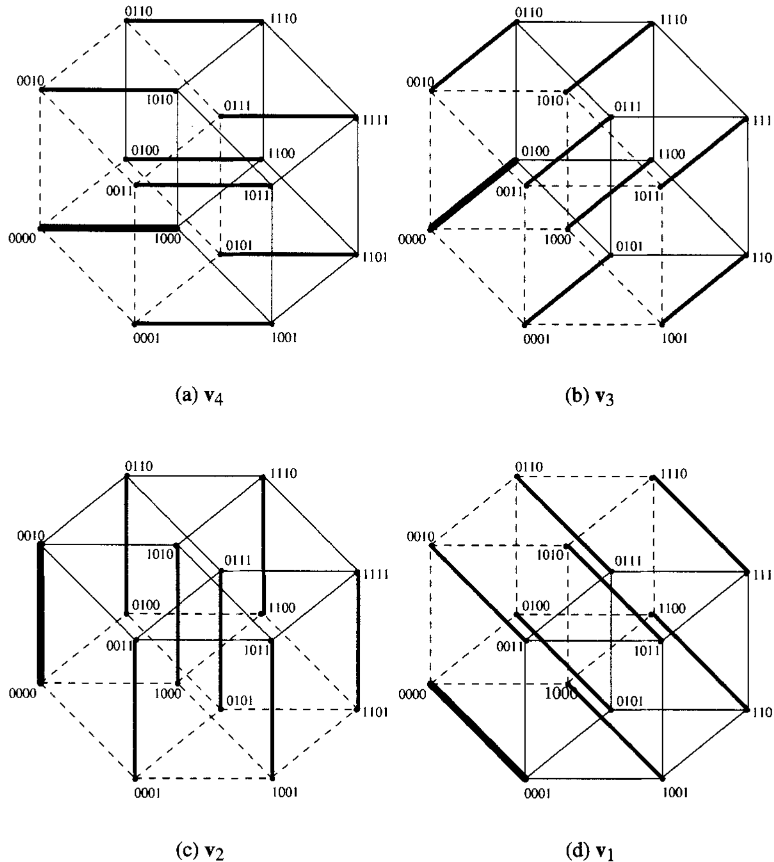
   which has the associated vector $\mathbf{v}_1\mathbf{v}_2\mathbf{v}_3 = (0000000100000001)$, which is the incidence vector for the set

$$T = \{P_7, P_{15}\} = \{(1000), (0000)\}.$$

   The corresponding equation is $m_4 = c_8 + c_0$. The subset is indicated by the widest line in Figure 8.6(a).

3. There are eight translations of T by the points in S. These are shown by the other wide lines in the figure.

□



(a) $\mathbf{v}_4$ (b) $\mathbf{v}_3$

(c) $\mathbf{v}_2$ (d) $\mathbf{v}_1$

Figure 8.6: Geometric descriptions of parity check equations for first-order vectors of the $RM(2, 4)$ code.

8.3.5 Other Constructions of Reed-Muller Codes

The $|\mathbf{u}|\mathbf{u}+\mathbf{v}|$ Construction The $|\mathbf{u}|\mathbf{u}+\mathbf{v}|$ introduced in Exercise 3.29 may be used to construct Reed-Muller codes. In fact,

$$RM(r, m) = \{[\mathbf{u}|\mathbf{u}+\mathbf{v}] : \mathbf{u} \in RM(r, m-1), \mathbf{v} \in RM(r-1, m-1)\}$$

having generator matrix

$$G_{RM}(r, m) = \begin{bmatrix} G_{RM}(r, m-1) & G_{RM}(r, m-1) \\ 0 & G_{RM}(r-1, m-1) \end{bmatrix}.$$

A Kronecker Construction Let $G_{(2,2)} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$. Define the m-fold Kronecker product of $G_{(2,2)}$ as

$$G_{(2^m, 2^m)} = \underbrace{G_{(2,2)} \otimes G_{(2,2)} \otimes \cdots \otimes G_{(2,2)}}_{m \text{ operands}},$$

which is a $2^m \times 2^m$ matrix. Then the generator for the $RM(r, m)$ code is obtained by selecting from $G_{(2^m,2^m)}$ those rows with weight greater than or equal to 2^{m-r}.

8.4 Building Long Codes from Short Codes: The Squaring Construction

There are several ways of combining short codes together to obtain codes with different properties. Among these are the $[\mathbf{u}|\mathbf{u}+\mathbf{v}]$ construction (outlined in Exercise 3.29) and concatenation (described in Section 10.6). In this section we present another one, called the squaring construction [220, 204].

We begin by examining partitions of codes into cosets by subcodes. Let $\mathcal{C}_0 = \mathcal{C}$ be a binary linear (n, k_0) block code with generator G and let $\mathcal{C}_1 \subset \mathcal{C}_0$ be a (n, k_1) subcode of $\mathcal{C}_0$. That is, $\mathcal{C}_1$ is a subgroup of $\mathcal{C}_0$. Recall that a coset of $\mathcal{C}_1$ is a set of the form

$$\mathbf{c}_l + \mathcal{C}_1 = \{\mathbf{c}_l + \mathbf{c} : \mathbf{c} \in \mathcal{C}_1\},$$

where $\mathbf{c}_l \in \mathcal{C}_0$ is a coset leader. We will take the nonzero coset leaders in $\mathcal{C} \setminus \mathcal{C}_1$. From Section 2.2.5, recall that $\mathcal{C}_0/\mathcal{C}_1$ forms a factor group, partitioning $\mathcal{C}_0$ into $2^{k_0-k_1}$ disjoint subsets each containing 2^{k_1} codewords. Each of these subsets can be represented by a coset leader. The set of coset leaders is called the coset representative space. The coset representative for the coset $\mathcal{C}_1$ is always chosen to be $\mathbf{0}$. Denote this coset representative space by $[\mathcal{C}_0/\mathcal{C}_1]$. The code $\mathcal{C}_1$ and the set $[\mathcal{C}/\mathcal{C}_1]$ share only the zero vector in common, $\mathcal{C}_1 \cap [\mathcal{C}/\mathcal{C}_1] = \mathbf{0}$.

Without loss of generality, let $G_0 = G$ be expressed in a form that k_1 rows of G_0 can be selected as a generator G_1 for $\mathcal{C}_1$. The $2^{k_0-k_1}$ codewords generated by the remaining $k_0 - k_1$ rows of $G_0 \setminus G_1$ can be used as to generate representatives for the cosets in $\mathcal{C}/\mathcal{C}_1$. Let $G_{0\setminus 1} = G_0 \setminus G_1$ (that is, the set difference, thinking of the rows as individual elements of the set). The $2^{k_0-k_1}$ codewords generated by $G_{0\setminus 1}$ form a $(n, k - k_1)$ subcode of $\mathcal{C}_0$.

Every codeword in $\mathcal{C}$ can be expressed as the sum of a codeword in $\mathcal{C}_1$ and a vector in $[\mathcal{C}_0/\mathcal{C}_1]$. We denote this as

$$\mathcal{C}_0 = \mathcal{C}_1 \oplus [\mathcal{C}_0/\mathcal{C}_1] = \{\mathbf{u} + \mathbf{v} : \mathbf{u} \in \mathcal{C}_1, \mathbf{v} \in [\mathcal{C}_0/\mathcal{C}_1]\}.$$

The set-operand sum $\oplus$ is called the *direct sum.*

Example 8.13 While the squaring construction can be applied to any linear block code, we demonstrate it here for a Reed-Muller code. Consider the $RM(1, 3)$ code with

$$G = G_0 = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}.$$

Let $\mathcal{C}_1$ be the (8, 3) code generated by the first $k_1 = 3$ rows of the generator G_0,

$$G_1 = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}.$$

The cosets in $\mathcal{C}/\mathcal{C}_1$ are

$$[0, 0, 0, 0, 0, 0, 0, 0] + \mathcal{C}_1, [0, 1, 0, 1, 0, 1, 0, 1] + \mathcal{C}_1.$$

The coset representatives are

$$[\mathcal{C}_0/\mathcal{C}_1] = \{[0,0,0,0,0,0,0,0], [0,1,0,1,0,1,0,1]\}$$

generated by the rows of the matrix

$$G_{0\setminus 1} = G_0 \setminus G_1 = [0,1,0,1,0,1,0,1].$$

□

One-level squaring is based on $\mathcal{C}_1$ and the partition $\mathcal{C}_0/\mathcal{C}_1$. Let $|\mathcal{C}_0/\mathcal{C}_1|^2$ denote the code $\mathfrak{C}_{0/1}$ of length $2n$ obtained by the squaring construction, defined as

$$\mathfrak{C}_{0/1} = |\mathcal{C}_0/\mathcal{C}_1|^2 = \{(\mathbf{a}+\mathbf{x}, \mathbf{b}+\mathbf{x}) : \mathbf{a}, \mathbf{b} \in \mathcal{C}_1 \text{ and } \mathbf{x} \in [\mathcal{C}_0/\mathcal{C}_1]\}. \tag{8.13}$$

Since there are $2^{k_0-k_1}$ vectors in $[\mathcal{C}_0/\mathcal{C}_1]$ and 2^{k_1} choices each for $\mathbf{a}$ and $\mathbf{b}$, there are $2^{k_0-k_1}2^{k_1}2^{k_1} = 2^{k_0+k_1}$ codewords in $\mathfrak{C}_{0/1}$. The code $\mathfrak{C}_{0/1}$ is thus a (n, k_0+k_1) code. Let

$$\mathbf{m} = [m_{1,0}, m_{1,1}, \ldots, m_{1,k_1-1}, m_{2,0}, m_{2,1}, \ldots, m_{2,k_1-1}, m_{3,0}, m_{3,1}, \ldots, m_{3,k_0-k_1-1}]$$

be a message vector. A coded message of the form $\mathbf{c} = (\mathbf{a}+\mathbf{x}, \mathbf{b}+\mathbf{x})$ from (8.13) can be obtained by

$$\mathbf{c} = \mathbf{m}\begin{bmatrix} G_1 & 0 \\ 0 & G_1 \\ G_{0\setminus 1} & G_{0\setminus 1}\end{bmatrix} \stackrel{\Delta}{=} \mathbf{m}\mathfrak{G}_{0/1}$$

so the matrix $\mathfrak{G}_{0/1}$ is the generator for the code. The minimum weight for the code is $d_{0/1} = \min(2d_0, d_1)$.

We can express the generator matrix $\mathfrak{G}_{0/1}$ in the following way. For two matrices M_1 and M_2 having the same number of columns, let $M_1 \oplus M_2$ denote the stacking operation

$$M_1 \oplus M_2 = \begin{bmatrix} M_1 \\ M_2 \end{bmatrix}.$$

This is called the matrix direct sum. Let I_2 be the 2×2 identity. Then

$$I_2 \otimes G_1 = \begin{bmatrix} G_1 & 0 \\ 0 & G_1 \end{bmatrix},$$

where $\otimes$ is the Kronecker product. We also have

$$\begin{bmatrix} 1 & 1 \end{bmatrix} \otimes G_{0\setminus 1} = \begin{bmatrix} G_{0\setminus 1} & G_{0\setminus 1} \end{bmatrix}.$$

Then we can write

$$\mathfrak{G}_{0/1} = I_2 \otimes G_1 \oplus \begin{bmatrix} 1 & 1 \end{bmatrix} \otimes G_{0\setminus 1}.$$

Example 8.14 Continuing the previous example, the generator for the code $|\mathcal{C}_0/\mathcal{C}_1|^2$ is

$$\mathfrak{G}_{0/1} = \begin{bmatrix}
1&1&1&1&1&1&1&1 & 0&0&0&0&0&0&0&0 \\
0&0&1&1&0&0&1&1 & 0&0&0&0&0&0&0&0 \\
0&0&0&0&1&1&1&1 & 0&0&0&0&0&0&0&0 \\
0&0&0&0&0&0&0&0 & 1&1&1&1&1&1&1&1 \\
0&0&0&0&0&0&0&0 & 0&0&1&1&0&0&1&1 \\
0&0&0&0&0&0&0&0 & 0&0&0&0&1&1&1&1 \\
0&1&0&1&0&1&0&1 & 0&1&0&1&0&1&0&1
\end{bmatrix}.$$

□

We can further partition the cosets as follows. Let $\mathcal{C}_2$ be a (n, k_2) subcode of $\mathcal{C}_1$ with generator G_2, with $0 \le k_2 \le k_1$. Then each of the $2^{k_0-k_1}$ cosets $\mathbf{c}_l + \mathcal{C}_1$ in the partition $\mathcal{C}_0/\mathcal{C}_1$ can be partitioned into $2^{k_1-k_2}$ cosets consisting of the following codewords

$$\mathbf{c}_l + \mathbf{d}_p + \mathcal{C}_2 = \{\mathbf{c}_l + \mathbf{d}_p + \mathbf{c} : \mathbf{c} \in \mathcal{C}_2\}$$

for each l in $0, 1, 2, \ldots, 2^{k-k_1}$ and each p in $1, 2, \ldots, 2^{k_1-k_2}$, where $\mathbf{d}_p$ is a codeword in $\mathcal{C}_1$ but not in $\mathcal{C}_2$. This partition is denoted as $\mathcal{C}/\mathcal{C}_1/\mathcal{C}_2$. We can express the entire code as the direct sum

$$\mathcal{C}_0 = [\mathcal{C}/\mathcal{C}_1] \oplus [\mathcal{C}_1/\mathcal{C}_2] \oplus \mathcal{C}_2.$$

Let $G_{1\backslash 2}$ denote the generator matrix for the coset representative space $[\mathcal{C}_1/\mathcal{C}_2]$. Then $G_{1\backslash 2} = G_1 \setminus G_2$.

Example 8.15 Let $\mathcal{C}_2$ be generated by the first two rows of G_1, so

$$G_2 = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \end{bmatrix}.$$

There are two cosets in $\mathcal{C}_1/\mathcal{C}_2$,

$$[0, 0, 0, 0, 0, 0, 0, 0] + \mathcal{C}_2, [0, 0, 0, 0, 1, 1, 1, 1] + \mathcal{C}_2.$$

The set of coset representatives $[\mathcal{C}_1/\mathcal{C}_2]$ is generated by

$$G_{1\backslash 2} = G_1 \setminus G_2 = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}.$$

□

Two-level squaring begins by forming the two one-level squaring construction codes $\mathfrak{C}_{0/1} = |\mathcal{C}_0/\mathcal{C}_1|^2$ and $\mathfrak{C}_{1/2} = |\mathcal{C}_1/\mathcal{C}_2|^2$, with generators $\mathfrak{G}_{0/1}$ and $\mathfrak{G}_{1/2}$, respectively, given by

$$\mathfrak{G}_{0/1} = \begin{bmatrix} G_1 & 0 \\ 0 & G_1 \\ G_{0\backslash 1} & G_{0\backslash 1} \end{bmatrix} \qquad \mathfrak{G}_{1/2} = \begin{bmatrix} G_2 & 0 \\ 0 & G_2 \\ G_{1\backslash 2} & G_{1\backslash 2} \end{bmatrix}. \tag{8.14}$$

Note that $\mathfrak{C}_{1/2}$ is a subcode (subgroup) of $\mathfrak{C}_{0/1}$. The coset representatives for $\mathfrak{C}_{0/1}/\mathfrak{C}_{1/2}$, which are denoted by $[\mathfrak{C}_{0/1}/\mathfrak{C}_{1/2}]$, form a linear code. Let $\mathfrak{G}_{\mathfrak{C}_{0/1}\backslash\mathfrak{C}_{1/2}}$ denote the generator matrix for the coset representatives $[\mathfrak{C}_{0/1}/\mathfrak{C}_{1/2}]$. Then form the code $\mathfrak{C}_{0/1/2} = |\mathcal{C}_0/\mathcal{C}_1/\mathcal{C}_2|^4$ by

$$\mathfrak{C}_{0/1/2} = |\mathcal{C}_0/\mathcal{C}_1/\mathcal{C}_2|^4 = \{(\mathbf{a}+\mathbf{x}, \mathbf{b}+\mathbf{x}) : \mathbf{a}, \mathbf{b} \in \mathfrak{C}_{1/2} \text{ and } \mathbf{x} \in [\mathfrak{C}_{0/1}/\mathfrak{C}_{1/2}]\}.$$

That is, it is obtained by the squaring construction of $\mathfrak{C}_{0/1}$ and $\mathfrak{C}_{0/1}/\mathfrak{C}_{1/2}$. The generator matrix for $\mathfrak{C}_{0/1/2}$ is

$$\mathfrak{G}_{0/1/2} = \begin{bmatrix} \mathfrak{G}_{1/2} & 0 \\ 0 & \mathfrak{G}_{1/2} \\ \mathfrak{G}_{1\backslash 2} & \mathfrak{G}_{1\backslash 2} \end{bmatrix}.$$

This gives a $(4n, k_0 + 2k_1 + k_2)$ linear block code with minimum distance

$$d_{0/1/2} = \min(4d_0, 2d_1, d_2).$$

Writing $\mathfrak{G}_{0/1}$ and $\mathfrak{G}_{1/2}$ as in (8.14), rearranging rows and columns and performing some simple row operations, we can write $\mathfrak{G}_{0/1/2}$ as

$$\mathfrak{G}_{0/1/2} = \begin{bmatrix} G_2 & 0 & 0 & 0 \\ 0 & G_2 & 0 & 0 \\ 0 & 0 & G_2 & 0 \\ 0 & 0 & 0 & G_1 \\ G_{0\backslash 1} & G_{0\backslash 1} & G_{0\backslash 1} & G_{0\backslash 1} \\ G_{1\backslash 2} & G_{1\backslash 2} & G_{1\backslash 2} & G_{1\backslash 2} \\ 0 & 0 & G_{1\backslash 2} & G_{1\backslash 2} \\ 0 & G_{1\backslash 2} & 0 & G_{1\backslash 2} \end{bmatrix},$$

which can be expressed as

$$\mathfrak{G}_{0/1/2} = I_4 \otimes G_2 \oplus \begin{bmatrix} 1 & 1 & 1 & 1 \end{bmatrix} \otimes G_{0\backslash 1} \oplus \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{bmatrix} \otimes G_{1\backslash 2}.$$

Note that

$$\begin{bmatrix} 1 & 1 & 1 & 1 \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{bmatrix}$$

are the generator matrices for the zeroth and first order Reed-Muller codes of length 4.

More generally, let $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_m$ be a sequence of linear subcodes of $\mathcal{C} = \mathcal{C}_0$ with generators G_i, minimum distance d_i, and dimensions $k_1, k_2, \ldots, k_m$ satisfying

$$\mathcal{C}_0 \supseteq \mathcal{C}_1 \supseteq \cdots \supseteq \mathcal{C}_m$$
$$k \geq k_1 \geq \cdots \geq k_m \geq 0.$$

Then form the chain of partitions

$$\mathcal{C}_0/\mathcal{C}_1, \mathcal{C}_0/\mathcal{C}_1/\mathcal{C}_2, \ldots, \mathcal{C}_0/\mathcal{C}_1/\cdots/\mathcal{C}_m,$$

such that the code can be expressed as the direct sum

$$\mathcal{C}_0 = [\mathcal{C}/\mathcal{C}_1] \oplus [\mathcal{C}_1/\mathcal{C}_2] \oplus \cdots \oplus [\mathcal{C}_{q-1}/\mathcal{C}_m].$$

Assume that the generator matrix is represented in a way that $G_0 \supseteq G_1 \cdots \supseteq G_m$. Let $G_{i\backslash i+1}$ denote the generator matrix for the coset representative space $[\mathcal{C}_i/\mathcal{C}_{i+1}]$, with

$$\text{rank}(G_{i/i+1}) = \text{rank}(G_i) - \text{rank}(G_{i+1})$$

and $G_{i\backslash i+1} = G_i \setminus G_{i+1}$. Then higher-level squaring is performed recursively. From the codes

$$\mathfrak{G}_{0/1/\cdots/m-1} \triangleq |\mathcal{C}_0/\mathcal{C}_1/\cdots/\mathcal{C}_{m-1}|^{2^{m-1}}$$

and the code

$$\mathfrak{G}_{1/2/\cdots/m} \triangleq |\mathcal{C}_1/\mathcal{C}_2/\cdots/\mathcal{C}_m|^{2^{m-1}},$$

form the code

$$\begin{aligned} \mathfrak{G}_{0/1/\cdots/m} &\triangleq [\mathcal{C}_0/\mathcal{C}_1/\cdots/\mathcal{C}_m]^{2^m} \\ &= \{(\mathbf{a}+\mathbf{x}, \mathbf{b}+\mathbf{x}) : \mathbf{a}, \mathbf{b} \in \mathfrak{G}_{1/2/\cdots/m}, \mathbf{x} \in [\mathfrak{G}_{0/1/\cdots/m-1}/\mathfrak{G}_{1/2/\cdots/m}]\}. \end{aligned}$$

The generator matrix can be written (after appropriate rearrangement of the rows) as

$$\mathfrak{G}_{0/1/\ldots/m} = I_{2^m} \otimes G_m \oplus \sum_{r=0}^{m}\oplus G_{RM}(r, m) \otimes G_{r\backslash r+1},$$

where $G_{RM}(r, m)$ is the generator of the $RM(r, m)$ code of length 2^m.

8.5 Quadratic Residue Codes

Quadratic residue codes are codes of length p, where p is a prime with coefficients in $GF(s)$, where s is a quadratic residue of p. They have rather good distance properties, being among the best codes known of their size and dimension.

We begin the construction with the following notation. Let p be prime. Denote the set of quadratic residues of p by Q_p and the set of quadratic nonresidues by N_p. Then the elements in $GF(p)$ are partitioned into sets as

$$GF(p) = Q_p \cup N_p \cup \{0\}.$$

As we have seen $GF(p)$ is cyclic. This gives rise to the following observation:

Lemma 8.11 *A primitive element of $GF(p)$ must be a quadratic nonresidue. That is, it is in N_p.*

Proof Let γ be a primitive element of $GF(p)$. We know $\gamma^{p-1} = 1$, and $p - 1$ is the smallest such power. Suppose γ is a quadratic residue. Then there is a number σ (square root) such that $\sigma^2 = \gamma$. Taking powers of σ, we have $\sigma^{2(p-1)} = 1$. Furthermore, the powers $\sigma, \sigma^2, \sigma^3, \ldots, \sigma^{2(p-1)}$ can be shown to all be distinct. But this contradicts the order p of the field. ☐

So a primitive element $\gamma \in GF(p)$ satisfies $\gamma^e \in Q_p$ if and only if e is even, and $\gamma^e \in N_p$ if and only if e is odd. The elements of Q_p correspond to the first $(p-1)/2$ consecutive powers of γ^2; that is, Q_p is a cyclic group under multiplication modulo p, generated by γ^2.

The quadratic residue codes are designed as follows. Choose a field $GF(s)$ as the field for the coefficients, where s is a quadratic residue modulo p. We choose an extension field $GF(s^m)$ so that it has a primitive pth root of unity; from Lemma 5.16 we must have $p \mid s^m - 1$. (It can be shown [220, p. 519] that if $s = 2$, then p must be of the form $p = 8k \pm 1$.)

Let β be a primitive pth root of unity in $GF(s^m)$. Then the conjugates with respect to $GF(s)$ are

$$\beta^1, \beta^s, \beta^{s^2}, \beta^{s^3}, \ldots$$

The cyclotomic coset is $\{1, s, s^2, s^3, \ldots, \}$. Since $s \in Q_p$ and Q_p is a group under multiplication modulo p, Q_p is closed under multiplication by s. So all of the elements in the cyclotomic coset are in Q_p. Thus Q_p is a cyclotomic coset or the union of cyclotomic cosets.

Example 8.16 Let $p = 11$, which has quadratic residues $Q_p = \{1, 3, 4, 5, 9\}$. Let $s = 3$. A field having a primitive 11th root of unity is $GF(3^5)$. Let $\beta \in GF(3^5)$ be a primitive 11th root of unity. The conjugates of β are:

$$\beta, \beta^3, \beta^9, \beta^{27} = 5, \beta^{81} = \beta^4,$$

so the cyclotomic coset is

$$\{1, 3, 9, 5, 4\},$$

which is identical to Q_p. □

Now let β be a primitive pth root of unity in $GF(s^m)$. Because of the results above,

$$q(x) = \prod_{i \in Q_p} (x - \beta^i)$$

is a polynomial with coefficients in $GF(s)$. Furthermore,

$$n(x) = \prod_{i \in N_p} (x - \beta^i) \tag{8.15}$$

also has coefficients in $GF(s)$. We thus have the factorization

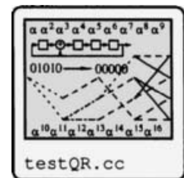
$$x^p - 1 = q(x)n(x)(x - 1).$$

Let R be the ring $GF(s)[x]/(x^p - 1)$.

Definition 8.5 [220, p. 481] For a prime p, the quadratic residue codes of length $\mathcal{Q}$, $\overline{\mathcal{Q}}$, $\mathcal{N}$ and $\overline{\mathcal{N}}$ are the cyclic codes (or ideals of R) with generator polynomials

$$q(x), \qquad (x - 1)q(x), \qquad n(x), \qquad (x - 1)n(x),$$

respectively. The codes $\mathcal{Q}$ and $\mathcal{N}$ have dimension $\frac{1}{2}(p + 1)$; the codes $\overline{\mathcal{Q}}$ and $\overline{\mathcal{N}}$ have dimension $\frac{1}{2}(p - 1)$. The codes $\mathcal{Q}$ and $\mathcal{N}$ are sometimes called *augmented* QR codes, while $\overline{\mathcal{Q}}$ and $\overline{\mathcal{Q}}$ are called *expurgated* QR codes. □



Example 8.17 Let $p = 17$ and $s = 2$. The field $GF(2^8)$ has a primitive 17th root of unity, which is $\beta = \alpha^{15}$. The quadratic residues modulo 17 are $\{1, 2, 4, 8, 9, 13, 15, 16\}$. Then

$$\begin{aligned}
q(x) &= (x - \beta)(x - \beta^2)(x - \beta^4)(x - \beta^8)(x - \beta^9)(x - \beta^{13})(x - \beta^{15})(x - \beta^{16}) \\
&= 1 + x + x^2 + x^4 + x^6 + x^7 + x^8 \\
n(x) &= (x - \beta^3)(x - \beta^5)(x - \beta^6)(x - \beta^7)(x - \beta^{10})(x - \beta^{11})(x - \beta^{12})(x - \beta^{14}) \\
&= 1 + x^3 + x^4 + x^5 + x^8.
\end{aligned}$$

□

QR codes tend to have rather good distance properties. Some binary QR codes are the best codes known for their particular values of n and k. A bound on the distance is provided by the following.

Theorem 8.12 *The minimum distance d of the codes $\mathcal{Q}$ or $\mathcal{N}$ satisfies $d^2 \geq p$. If, additionally, $p = 4l - 1$ for some l, then $d^2 - d + 1 \geq p$.*

The proof relies on the following lemma.

Lemma 8.13 *Let $\tilde{q}(x) = q(x^n)$, where $n \in N_p$ (where the operations are in the ring R). Then the roots of $\tilde{q}(x)$ are in the set $\{\alpha^i, i \in N_p\}$. That is, $\tilde{q}(x)$ is a scalar multiple of $n(x)$. Similarly, $n(x^n)$ is a scalar multiple of $q(x)$.*

Proof Let ρ be a generator of the nonzero elements of $GF(p)$. From the discussion around Lemma 8.11, Q is generated by even powers of ρ and N_p is generated by odd powers of ρ.

Write $\tilde{q}(x) = \prod_{i \in N_p}(x^n - \alpha^i)$. Let $m \in N_p$. Then for any $m \in N_p$,

$$\tilde{q}(\alpha^m) = \prod_{i \in N_p} (\alpha^{mn} - \alpha^i).$$

But since $m \in N_p$ and $n \in N_p$, $mn \in N_p$ (being both odd powers of ρ. So $\alpha^i = \alpha^{mn}$ for some value of i, so α^m is a root of $\tilde{q}(x)$. □

The effect of evaluation at $q(x^n)$ is to permute the coefficients of $q(x)$.

Proof of Theorem 8.12. [220, p. 483]. Let $a(x)$ be a codeword of minimum nonzero weight d in $\mathcal{Q}$. Then by Lemma 8.13, the polynomial $\tilde{a}(x) = a(x^n)$ is a codeword in $\mathcal{N}$. Since the coefficients of $\tilde{a}(x)$ are simply a permutation (and possible scaling) of those of $a(x)$, $\tilde{a}(x)$ must be a codeword of minimum weight in $\mathcal{N}$. The product $a(x)\tilde{a}(x)$ must be a multiple of the polynomial

$$\prod_{q \in Q_p}(x - \alpha^q) \prod_{n \in N_p}(x - \alpha^n) = \prod_{i=1}^{p-1}(x - \alpha^i) = \frac{x^p - 1}{x - 1} = \sum_{j=0}^{p-1} x^j.$$

Thus $a(x)\tilde{a}(x)$ has weight p. Since $a(x)$ has weight d, the maximum weight of $a(x)\tilde{a}(x)$ is d^2. We obtain the bound $d^2 \geq p$.

If $p = 4k - 1$ then $n = 1$ is a quadratic nonresidue. In the product $a(x)\tilde{a}(x) = a(x)a(x^{-1})$ there are d terms equal to 1, so the maximum weight of the product is $d^2 - d + 1$. □

Table 8.1 summarizes known distance properties for some augmented binary QR codes, with indications of best known codes. In some cases, d is expressed in terms of upper and lower bounds.

Table 8.1: Extended Quadratic Residue Codes $\mathcal{Q}$ [220, 373]

n	k	d	n	k	d	n	k	d
8	4	4*	74	37	14	138	69	14–22
18	9	6*	80	40	16*	152	76	20
24	12	8*	90	45	18*	168	84	16–24
32	16	8*	98	49	16	192	96	16–28
42	21	10*	104	52	20*	194	97	16–28
48	24	12*	114	57	12–16	200	100	16–32
72	36	12	128	64	20			

* Indicates that the code is as good as the best known for this n and k

While general decoding techniques have been developed for QR codes, we present only a decoding algorithm for a particular QR code, the Golay code presented in the next section. Decoding algorithms for other QR codes are discussed in [220], [287], [283], and [75].

8.6 Golay Codes

Of these codes it was said, "The Golay code is probably the most important of all codes, for both practical and theoretical reasons." [220, p. 64]. While the Golay codes have not

supported the burden of applications this alleged importance would suggest, they do lie at the confluence of several routes of theoretical development and are worth studying.

Let us take $p = 23$ and form the binary QR code. The field $GF(2^{11})$ has a primitive 23rd root of unity. The quadratic residues are

$$Q_p = \{1, 2, 3, 4, 6, 8, 9, 12, 13, 16, 18\}$$

and the corresponding generators for $\mathcal{Q}$ and $\mathcal{N}$ are

$$q(x) = \prod_{i \in Q_p} (x - \beta^i) = 1 + x + x^5 + x^6 + x^7 + x^9 + x^{11}$$

$$n(x) = 1 + x^2 + x^4 + x^5 + x^6 + x^{10} + x^{11}.$$

This produces a (23,12,7) code, the Golay code $\mathcal{G}_{23}$. It is straightforward to verify that this code is a perfect code: the number of points out to a distance $t = 3$ is equal to

$$V_2(23, 3) = \binom{23}{0} + \binom{23}{1} + \binom{23}{2} + \binom{23}{3} = 2^{11}.$$

This satisfies the Hamming bound. It is said that Golay discovered this code by observing this relationship in Pascal's triangle. Adding an extra overall parity bit to the (23, 12) code produces the Golay $\mathcal{G}_{24}$ (24,12,8) code.

The binary Golay code also has some interesting generator matrix representations. The generator matrix for the $\mathcal{G}_{24}$ code can be written as

$$G = \left[\begin{array}{c|ccccccccccc|c|ccccccccccc}
1 & 1 & & & & & & & & & & & & 1 & 1 & & 1 & 1 & 1 & & & & 1 & \\
1 & & 1 & & & & & & & & & & & & 1 & 1 & & 1 & 1 & 1 & & & & 1 \\
1 & & & 1 & & & & & & & & & & 1 & & 1 & 1 & & 1 & 1 & 1 & & & \\
1 & & & & 1 & & & & & & & & & & 1 & & 1 & 1 & & 1 & 1 & 1 & & \\
1 & & & & & 1 & & & & & & & & & & 1 & & 1 & 1 & & 1 & 1 & 1 & \\
1 & & & & & & 1 & & & & & & & & & & 1 & & 1 & 1 & & 1 & 1 & 1 \\
1 & & & & & & & 1 & & & & & & 1 & & & & 1 & & 1 & 1 & & 1 & 1 \\
1 & & & & & & & & 1 & & & & & 1 & 1 & & & & 1 & & 1 & 1 & & 1 \\
1 & & & & & & & & & 1 & & & & 1 & 1 & 1 & & & & 1 & & 1 & 1 & \\
1 & & & & & & & & & & 1 & & & & 1 & 1 & 1 & & & & 1 & & 1 & 1 \\
1 & & & & & & & & & & & 1 & & 1 & & 1 & 1 & 1 & & & & 1 & & 1 \\
\hline
1 & & & & & & & & & & & & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1
\end{array}\right]$$

$$= \begin{bmatrix} \mathbf{1}_{11} & I_{11} & \mathbf{0}_{11} & A_{11}^T \\ 0 & \mathbf{0}_{11}^T & 1 & \mathbf{1}_{11}^T \end{bmatrix}. \tag{8.16}$$

We note the following facts about $\mathcal{G}_{24}$.

- In this representation, the 11×11 A_{11}^T matrix on the upper right is obtained from the transpose of 12×12 Hadamard matrix of Paley type (8.5) by removing the first row and column of H_{12}, then replacing -1 by 1 and 1 by 0. Since the rows of H_{12} differ in six places, the rows of A_{11}^T differ by six places. Because of the identity block, the sum of any two rows of G has weight 8.
- If $\mathbf{u}$ and $\mathbf{v}$ are rows of G (not necessarily distinct), then $\text{wt}(\mathbf{u} \cdot \mathbf{v}) \equiv 0 \pmod 2$. So every row of G is orthogonal to every other row. Therefore, G is also the parity check matrix H of $\mathcal{G}_{24}$. Also $\mathcal{G}$ is dual to itself: $\mathcal{G}_{24} = \mathcal{G}_{24}^{\perp}$. Such a code is call **self-dual.**
- Every codeword has even weight: If there were a codeword $\mathbf{u}$ of odd weight then $\text{wt}(\mathbf{u} \cdot \mathbf{u}) = 1$. Furthermore, since every row of the generator has weight divisible by 4, every codeword is even.
- The weight distributions for the (23, 12) and the (24, 12) codes are shown in Table 8.2.

Table 8.2: Weight Distributions for the $\mathcal{G}_{23}$ and $\mathcal{G}_{24}$ Codes [373]

$\mathcal{G}_{23}$:	i: 0	7	8	11	12	15	16	23
	A_i: 1	253	506	1288	1288	506	253	1

$\mathcal{G}_{24}$:	i: 0	8	12	16	24
	A_i: 1	759	2576	759	1

8.6.1 Decoding the Golay Code

We present here two decoding algorithms for the Golay code. The first decoder, due to [75], is algebraic and is similar in spirit to the decoding algorithms used for BCH and Reed-Solomon codes. The second decoder is arithmetic, being similar in spirit to the Hamming decoder presented in Section 1.9.1.

Algebraic Decoding of the $\mathcal{G}_{23}$ Golay Code

The algebraic decoder works similar to those we have seen for BCH codes. An algebraic syndrome is first computed, which is used to construct an error locator polynomial. The roots of the error locator polynomial determine the error locations, which for a binary code is sufficient for the decoding. Having minimum distance 7, $\mathcal{G}_{23}$ is capable of correcting up to three errors.

Let β be a primitive 23rd root of unity in $GF(2^{11})$. Recall that the quadratic residues modulo 23 are $Q_p = \{1, 2, 3, 4, 6, 8, 9, 12, 13, 16, 18\}$ and the generator polynomial is

$$g(x) = \prod_{i \in Q_p} (x - \beta^i).$$

Thus β, β^3, and β^9 are all roots of $g(x)$, and hence of any codeword $c(x) = m(x)g(x)$. Let $c(x)$ be the transmitted codeword, and let $r(x) = c(x) + e(x)$ be the received polynomial. We define the syndrome as

$$s_i = r(\beta^i) = e(\beta^i).$$

If there are no errors, then $s_i = 0$ for $i \in Q_p$. Thus, for example, if $s_1 = s_3 = s_9 = 0$, no errors are detected. If there is a single error, $e(x) = x^{j_1}$, then

$$s_1 = \beta^{j_1}, \qquad s_3 = \beta^{3j_1}, \qquad s_9 = \beta^{9j_1}.$$

When this condition is detected, single-error correction can proceed.

Suppose there are two or three errors, $e(x) = x^{j_1} + x^{j_2} + x^{j_3}$. Let $z_1 = \beta^{j_1}$, $z_2 = \beta^{j_2}$ and $z_3 = \beta^{j_3}$ be the error locators, where $z_3 = 0$ in the case of only two errors.) The syndromes in this case are

$$s_i = z_1^i + z_2^i + z_3^i.$$

Define the *error locator* polynomial as

$$L(x) = (x - z_1)(x - z_2)(x - z_3) = x^3 + \sigma_1 x^2 + \sigma_2 x + \sigma_3,$$

where, by polynomial multiplication,

$$\sigma_1 = z_1 + z_2 + z_3$$
$$\sigma_2 = z_1 z_2 + z_1 z_3 + z_2 z_3$$
$$\sigma_3 = z_1 z_2 z_3.$$

The problem now is to compute the coefficients of the error locator polynomial using the syndrome values. By substitution of the definitions, it can be shown that

$$s_9 + s_1^9 = \sigma_2 s_7 + \sigma_3 s_3^2 \qquad s_7 = s_1 s_3^2 + \sigma_2 s_5 + \sigma_3 s_1^4$$
$$s_5 = s_1^5 + \sigma_2 s_3 + \sigma_3 s_1^2 \qquad s_1^3 + s_3 = \sigma_3 + \sigma_2 s_1.$$

golaysimp.m

By application of these equivalences (see golaysimp.m) it can be shown that

$$D \triangleq (s_1^3 + s_3)^2 + \frac{s_1^9 + s_9}{s_1^3 + s_3} = (\sigma_2 + s_1^2)^3. \tag{8.17}$$

The quantity D thus has a cube root in $GF(2^{11})$. From (8.17) we obtain $\sigma_2 = s_1^2 + D^{1/3}$; similarly for σ_3. Combining these results, we obtain the following equations:

$$\sigma_1 = s_1 \qquad \sigma_2 = s_1^2 + D^{1/3} \qquad \sigma_3 = s_3 + s_1 D^{1/3}$$

testGolay.cc

An example of the decoder is shown in testGolay.cc.

Arithmetic Decoding of the $\mathcal{G}_{24}$ Code

In this section we present an arithmetic coder, which uses the weight structure of the syndrome to determine the error patterns.

Besides the generator matrix representation of (8.16), it is convenient to employ a systematic generator. The generator matrix can be written in the form

$$G = \left[\begin{array}{cccccccccccc|cccccccccccc}
1 &&&&&&&&&&& & & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
& 1 &&&&&&&&&& & 1 & 1 & 1 & & 1 & 1 & 1 & & & & 1 & \\
&& 1 &&&&&&&&& & 1 & 1 & & 1 & 1 & 1 & & & & 1 & & 1 \\
&&& 1 &&&&&&&& & 1 & & 1 & 1 & 1 & & & & 1 & & 1 & 1 \\
&&&& 1 &&&&&&& & 1 & 1 & 1 & 1 & & & & 1 & & 1 & 1 & \\
&&&&& 1 &&&&&& & 1 & 1 & 1 & & & & 1 & & 1 & 1 & & 1 \\
&&&&&& 1 &&&&& & 1 & 1 & & & & 1 & & 1 & 1 & & 1 & 1 \\
&&&&&&& 1 &&&& & 1 & & & & 1 & & 1 & 1 & & 1 & 1 & 1 \\
&&&&&&&& 1 &&& & 1 & & & 1 & & 1 & 1 & & 1 & 1 & 1 & \\
&&&&&&&&& 1 && & 1 & & 1 & & 1 & 1 & & 1 & 1 & 1 & & \\
&&&&&&&&&& 1 & & 1 & 1 & & 1 & 1 & & 1 & 1 & 1 & & & \\
&&&&&&&&&&& 1 & 1 & & 1 & 1 & & 1 & 1 & 1 & & & & 1
\end{array}\right]$$

$$= \begin{bmatrix} I_{12} & B \end{bmatrix}.$$

It may be observed that B is orthogonal,

$$B^T B = I.$$

Let $\mathbf{r} = \mathbf{c} + \mathbf{e}$ and let $\mathbf{e} = (\mathbf{x}, \mathbf{y})$, where $\mathbf{x}$ and $\mathbf{y}$ are each vectors of length 12. Since the code is capable of correcting up to three errors, there are only a few possible weight distributions of $\mathbf{x}$ and $\mathbf{y}$ to consider:

$$\begin{array}{ll}
\text{wt}(\mathbf{x}) \leq 3 & \text{wt}(\mathbf{y}) = 0 \\
\text{wt}(\mathbf{x}) \leq 2 & \text{wt}(\mathbf{y}) = 1 \\
\text{wt}(\mathbf{x}) \leq 1 & \text{wt}(\mathbf{y}) = 2 \\
\text{wt}(\mathbf{x}) = 0 & \text{wt}(\mathbf{y}) = 3.
\end{array}$$

Since the code is self-dual, the generator matrix is also the parity check matrix. We can compute a syndrome by

$$\mathbf{s} = G\mathbf{r}^T = G(\mathbf{e}^T) = G[\mathbf{x}, \mathbf{y}]^T = \mathbf{x}^T + B\mathbf{y}^T.$$

If $\mathbf{y} = \mathbf{0}$, then $\mathbf{s} = \mathbf{x}^T$. If $\mathbf{s}$ has weight ≤ 3, we conclude that $\mathbf{y} = \mathbf{0}$. The error pattern is $\mathbf{e} = (\mathbf{x}, \mathbf{0}) = (\mathbf{s}^T, \mathbf{0})$.

Suppose now that $\text{wt}(\mathbf{y}) = 1$, where the error is in the ith coordinate of $\mathbf{y}$ and that $\text{wt}(\mathbf{x}) \leq 2$. The syndrome in this case is

$$\mathbf{s} = \mathbf{x}^T + \mathbf{b}_i,$$

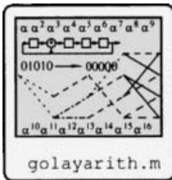
where $\mathbf{b}_i$ is the ith column of B. The position i is found by identifying the position such that $\text{wt}(\mathbf{s} + \mathbf{b}_i) = \text{wt}(\mathbf{x}) \leq 2$. Having thus identified i, the error pattern is $\mathbf{e} = ((\mathbf{s} + \mathbf{b}_i)^T, \mathbf{y}_i)$. Here, the notation $\mathbf{y}_i$ is the vector of length 12 having a 1 in position i and zeros elsewhere.

If $\text{wt}(\mathbf{x}) = 0$ and $\text{wt}(\mathbf{y}) = 2$ or 3 then $\mathbf{s} = \mathbf{b}_i + \mathbf{b}_j$ or $\mathbf{s} = \mathbf{b}_i + \mathbf{b}_j + \mathbf{b}_k$. Since B is an orthogonal matrix,

$$B^T\mathbf{s} = B^T(B\mathbf{y}^T) = \mathbf{y}^T.$$

The error pattern is $\mathbf{e} = (\mathbf{0}, (B^T\mathbf{s})^T)$.

Finally, if $\text{wt}(\mathbf{x}) = 1$ and $\text{wt}(\mathbf{y}) = 2$, let the nonzero coordinate of $\mathbf{x}$ be at index i. Then



$$B^T\mathbf{s} = B^T(\mathbf{x}^T + B\mathbf{y}^T) = B^T\mathbf{x}^T + B^TB\mathbf{y}^T = \mathbf{r}_i^T + \mathbf{y}^T,$$

where $\mathbf{r}_i$ is the ith row of B. The error pattern is $\mathbf{e} = (\mathbf{x}_i, (B^T\mathbf{s})^T + \mathbf{r}_i)$.

Combining all these cases together, we obtain the following decoding algorithm.

Algorithm 8.2 Arithmetic Decoding of the Golay $\mathcal{G}_{24}$ Code

(This presentation is due to Wicker [373])

1 **Input:** $\mathbf{r} = \mathbf{e} + \mathbf{c}$, the received vector
2 **Output:** $\mathbf{c}$, the decoded vector
3 Compute $\mathbf{s} = G\mathbf{r}$ (compute the syndrome)
4 if $\text{wt}(\mathbf{s}) \leq 3$
5 $\quad \mathbf{e} = (\mathbf{s}^T, \mathbf{0})$
6 else if $\text{wt}(\mathbf{s} + \mathbf{b}_i) \leq 2$ for some column vector $\mathbf{b}_i$
7 $\quad \mathbf{e} = ((\mathbf{s} + \mathbf{b}_i)^T, \mathbf{y}_i)$
8 else
9 $\quad$ Compute $B^T\mathbf{s}$
10 $\quad$ if $\text{wt}(B^T\mathbf{s}) \leq 3$
11 $\quad\quad \mathbf{e} = (\mathbf{0}, (B^T\mathbf{s})^T)$
12 $\quad$ else if $\text{wt}(B^T\mathbf{s} + \mathbf{r}_i^T) \leq 2$ for some row vector $\mathbf{r}_i$
13 $\quad\quad \mathbf{e} = (\mathbf{x}_i, (B^T\mathbf{s})^T + \mathbf{r}_i)$
14 $\quad$ else
15 $\quad\quad$ Too many errors: declare uncorrectable error pattern and stop.
16 $\quad$ end
17 end
18 $\mathbf{c} = \mathbf{r} + \mathbf{e}$

A Matlab implementation that may be used to generate examples is in `golayarith.m`

8.7 Exercises

8.1 Verify items 1, 9, and 10 of Theorem 8.1.

8.2 Show that

$$\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \otimes H_n \quad \text{and} \quad H_n \otimes \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

are Hadamard matrices.

8.3 Prove the first two parts of Theorem 8.4.

8.4 Compute the quadratic residues modulo 19. Compute the values of the Legendre symbol $\chi_{19}(x)$ for $x = 1, 2, \ldots, 18$.

8.5 The Legendre symbol has the following properties. Prove them. In all cases, take $(a, p) = 1$ and $(b, p) = 1$.

(a) $\chi_p(a) \equiv a^{(p-1)/2} \pmod{p}$. *Hint*: if $\chi_p(a) = 1$ then $x^2 \equiv a \pmod{p}$ has a solution, say x_0. Then $a^{(p-1)/2} \equiv x_0^{p-1}$. Then use Fermat's theorem.

(b) $\chi_p(a)\chi_p(b) = \chi_p(ab)$.

(c) $a \equiv b \pmod{p}$ implies that $\chi_p(a) = \chi_p(b)$.

(d) $\chi_p(a^2) = 1$. $\chi_p(a^2 b) = \chi_p(b)$. $\chi_p(1) = 1$. $\chi_p(-1) = (-1)^{(p-1)/2}$.

8.6 Construct a Hadamard matrix of order 20.

8.7 Construct the Hadamard codes $\mathcal{A}_{20}$, $\mathcal{B}_{20}$ and $\mathcal{C}_{20}$. Which of these are linear codes? Which are cyclic?

8.8 Construct a generator and parity check matrix for $RM(2, 4)$.

8.9 Show that $RM(r, m)$ is a subcode of $RM(r+1, m)$.

8.10 Show that $RM(0, m)$ is a repetition code.

8.11 Show that $RM(m-1, m)$ is a simple parity check code.

8.12 Show that if $\mathbf{c} \in RM(r, m)$, then $(\mathbf{c}, \mathbf{c}) \in RM(r, m+1)$.

8.13 For each of the following received sequences received from $RM(1, 3)$ codes, determine the transmitted codeword $\mathbf{c}$.

(a) $\mathbf{r} = [1, 0, 1, 0, 1, 1, 0, 1]$.

(b) $\mathbf{r} = [0, 1, 0, 0, 1, 1, 1, 1]$.

8.14 Prove that all codewords in $RM(1, m)$ have weight 0, 2^{m-1} or 2^m. *Hint*: By induction.

8.15 Show that the $RM(1, 3)$ and $RM(2, 5)$ codes are self-dual. Are there other self-dual RM codes?

8.16 For the $RM(1, 4)$ code:

(a) Write the generator G.

(b) Determine the minimum distance.

(c) Write down the parity checks for $\hat{m}_4$, $\hat{m}_3$, $\hat{m}_2$, $\hat{m}_1$ and $\hat{m}_0$.

(d) Decode the received vector $\mathbf{r} = [0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1]$, if possible.

(e) Decode the received vector $\mathbf{r} = [1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0]$, if possible.

(f) Decode the received vector $\mathbf{r} = [1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0]$, if possible.

8.17 Verify the parity check equations for Figure 8.6(a) and (b).

8.18 For the inverse Hadamard transform:

(a) Provide a matrix decomposition analogous to (8.10).

(b) Draw the signal flow diagram for the fast inverse Hadamard computation.

(c) Implement your algorithm in Matlab.

8.19 Construct the generator for a $RM(2, 4)$ code using the $[\mathbf{u}|\mathbf{u}+\mathbf{v}]$ construction.

8.20 Using the $[\mathbf{u}|\mathbf{u}+\mathbf{v}]$ construction show that the generator for a $RM(r, m)$ code can be constructed as

$$G = \begin{bmatrix} G_{RM}(r, m-2) & G_{RM}(r, m-2) & G_{RM}(r, m-2) & G_{RM}(r, m-2) \\ 0 & G_{RM}(r-1, m-2) & 0 & G_{RM}(r-1, m-2) \\ 0 & 0 & G_{RM}(r-1, m-2) & G_{RM}(r-1, m-2) \\ 0 & 0 & 0 & G_{RM}(r-2, m-2) \end{bmatrix}.$$

8.21 Construct the generator for a $RM(2, 4)$ code using the Kronecker construction.

8.22 Let $G = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix}$ be the generator for a (7, 4) Hamming code $\mathcal{C}_0$. Let G_1 be formed from the first two rows of G. Let G_2 be formed from the first row of G.

(a) Identify $G_{0\backslash 1}$ and the elements of $[\mathcal{C}_0/\mathcal{C}_1]$.

(b) Write down the generator $\mathfrak{G}_{0/1}$ for the code $\mathcal{C}_{0/1}$.

(c) Write down the generator $\mathfrak{G}_{1/2}$ for the code $\mathcal{C}_{1/2}$.

(d) Write down the generator $\mathfrak{G}_{0/1/2}$ for the code $\mathcal{C}_{0/1/2}$.

8.23 Quadratic residue code designs.

(a) Find the generator polynomials for binary quadratic residue codes of length 7 and dimensions 4 and 3. Also, list the quadratic residues modulo 7 and compare with the cyclotomic coset for β.

(b) Are there binary quadratic residue codes of length 11? Why or why not?

(c) Find the generator polynomials for binary quadratic residue codes of length 23 and dimensions 12 and 11. Also, list the quadratic residues modulo 23 and compare with the cyclotomic coset for β.

8.24 Find quadratic residue codes with $s = 3$ of length 11 having dimensions 5 and 6.

8.25 Show that $n(x)$ defined in (8.15) has coefficients in $GF(s)$.

8.26 In decoding the Golay code, show that the cube root of D may be computed by finding $x = D^{1365}$.

8.27 Show that the Golay (24,12) code is self-dual.

8.28 Let $\mathbf{r} = [1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1]$ be a received vector from a Golay (24, 12) code. Determine the transmitted codeword using the arithmetic decoder.

8.29 Let $\mathbf{r} = [1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1]$ be the received vector from a Golay (23, 12) code. Determine the transmitted codeword using the algebraic decoder.

8.8 References

This chapter was developed largely out of course notes based on [373] and closely follows it. The discussion of Hadamard matrices and codes is drawn from [220] and [373]. A more complete discussion on quadratic residues may be found in [250]. Considerably more detail about Reed-Muller codes is available in [220]. The Reed-Muller codes were first described in [248], with work by Reed immediately following [284] which reinforced the Boolean function idea and provided an efficient decoding algorithm. The graph employed in developing the Reed-Muller decoding orthogonal equations is an example of a Euclidean

geometry $EG(m, r)$, a finite geometry. This area is developed and explored, giving rise to generalizations of the majority logic decoding, in [203, Chapter 8]. Chapter 7 of [203] also develops majority logic decoding for cyclic codes. Majority logic decoding can also be employed on convolutional codes [203, Chapter 13], but increasing hardware capabilities has made this less-complex alternative to the Viterbi algorithm less attractive.

The Golay codes are covered in lavish and fascinating detail in Chapters 2, 16 and 20 of [220]. Interesting connections between $\mathcal{G}_{24}$ and the 24-dimensional Leech lattice are presented in [56].

Another type of decoder which has worked well for the Golay code is an *error trapping decoder*. Such decoders employ the cycle structure of the codes, just as the Meggitt decoders do, but they simplify the number of syndromes the decoder must recognize. A thorough discussion of error trapping decoders is in [203] and [185]. Other Golay code decoders include conventional coset decoding; a method due to Berlekamp [350, p. 35]; and majority logic decoding [220].

Chapter 9

Bounds on Codes

Let C be an (n, k) block code with minimum distance d over a field with q elements with redundancy $r = n - k$. There are relationships that must be satisfied among the code length n, the dimension k, the minimum distance $d_{\min}$, and the field size q. We have already met two of these: the Singleton bound of theorem 3.4,

$$d \leq n - k + 1 = r + 1,$$

and the Hamming bound of Theorem 3.5,

$$r \geq \log_q V_q(n, t),$$

where $V_q(n, t)$ is the number of points in a Hamming sphere of radius $t = \lfloor (d-1)/2 \rfloor$,

$$V_q(n, t) = \sum_{i=0}^{t} \binom{n}{i} (q-1)^i \tag{9.1}$$

(see (3.9)). In this chapter we present other bounds which govern the relationships among the parameters defining a code. In this, we are seeking theoretical limits without regard to the feasibility of a code for any particular use, such as having efficient encoding or decoding algorithms. This chapter is perhaps the most mathematical of the book. It does not introduce any good codes or decoding algorithms, but the bounds introduced here have been of both historical and practical importance, as they have played a part in motivating the search for good codes and helped direct where the search should take place.

Definition 9.1 Let $A_q(n, d)$ be the maximum number of codewords in any code over $GF(q)$ of length n with minimum distance d. □

For a linear code the dimension of the code is $k = \log_q A_q(n, d)$.

Consider what happens as n gets long in a channel with probability of symbol error equal to p_c. The average number of errors in a received vector is np_c. Thus, in order for a sequence of codes to be asymptotically effective, providing capability to correct the increasing number of errors in longer codewords, the minimum distance must grow at least as fast as $2np_c$. We will frequently be interested in the relative distance and the rate k/n as $n \to \infty$.

Definition 9.2 For a code with length n and minimum distance d, let $\delta = d/n$ be the **relative distance** of the code. □

For a code with relative distance δ, the distance is $d \approx \delta n = \delta n + O(1)$.

Definition 9.3 Let[1]

$$\alpha_q(\delta) = \limsup_{n\to\infty} \frac{1}{n} \log_q A_q(n, \lfloor \delta n \rfloor). \tag{9.2}$$

[1] The lim sup is the least upper bound of the values that its argument function returns to infinitely often. Initially it may be helpful think of lim sup as "sup" or "max."

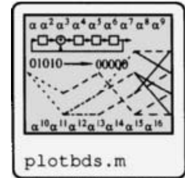
For a linear code, $\log_q A_q(n, d)$ is the dimension of the code and $\frac{1}{n}\log_q A_q(n, d)$ is the code rate, so $\alpha_q(\delta)$ is the maximum possible code rate that an arbitrarily long code can have while maintaining a relative distance δ. We call this the **asymptotic rate**. □

The functions $A_q(n, d)$ and $\alpha_q(\delta)$ are not known in general, but upper and lower bounds on these functions can be established. For example, the Singleton bound can be expressed in terms of these functions as

$$A_q(n, d) \leq q^{n-d+1}$$

and, asymptotically,

$$\alpha_q(\delta) \leq 1 - \delta.$$



Many of the bounds presented here are expressed in terms of $\alpha_q(\delta)$. A lower bound is the Gilbert-Varshamov bound (sometimes called the Varshamov-Gilbert bound). As upper bounds on $\alpha_q(\delta)$, we also have the Hamming and Singleton bounds, the Plotkin bound, the Elias bound, and the McEliece-Rodemich-Rumsey-Welch bound (in two forms). Figure 9.1 shows a comparison of the lower bound and these upper bounds. Codes exist which fall between the lower bound and the smallest of the upper bounds, that is, in the shaded region.

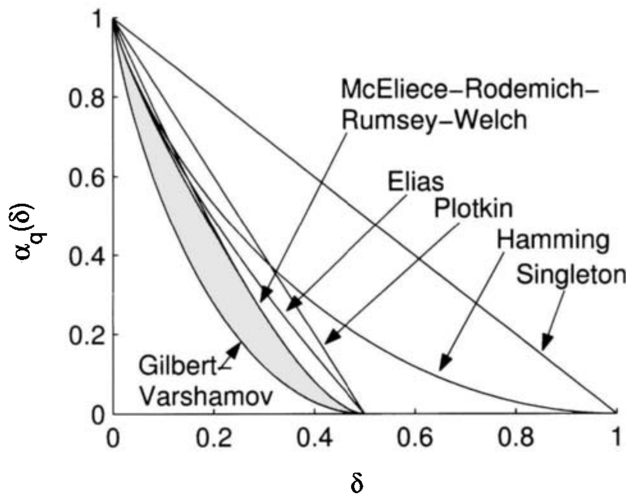


Figure 9.1: Comparison of lower bound (Gilbert-Varshamov) and various upper bounds.

A function which figures into some of the bounds is the entropy function. The binary entropy function H_2 was introduced in Section 1.3. Here we generalize that definition.

Definition 9.4 Let $\rho = (q-1)/q$. Define the entropy function H_q on $[0, \rho]$ by

$$H_q(x) = x\log_q(q-1) - x\log_q x - (1-x)\log_q(1-x), \qquad x \in (0, \rho]$$

and $H_q(0) = 0$. □

The entropy and the number of points in the Hamming sphere are asymptotically related.

Lemma 9.1 *Let* $0 \leq x \leq \rho = (q-1)/q$. *Then*

$$\lim_{n\to\infty} \frac{1}{n}\log_q V_q(n, \lfloor xn \rfloor) = H_q(x). \tag{9.3}$$

Box 9.1: O and o Notation

The O and o notation are used to represent "orders" of functions. Roughly, these may be interpreted as follows [152]:

- $f = O(1)$ as $x \to x_0 \Leftrightarrow f$ is bounded as $x \to x_0$.
- $f = o(1)$ as $x \to x_0 \Leftrightarrow f \to 0$ as $x \to x_0$.
- $f = O(g)$ as $x \to x_0 \Leftrightarrow f/g$ is bounded as $x \to x_0$.
- $f = o(g)$ as $x \to x_0 \Leftrightarrow f/g \to 0$ as $x \to x_0$.

It follows from the definitions that $o(1) + o(1) = o(1)$ and $O(x^n) + O(x^m) = O(x^{\max(n,m)})$.

Proof First we need a way to approximate the binomial. Stirling's formula (see Exercise 9.10) says that

$$n! = \sqrt{2\pi n}n^n e^{-n} + o(1).$$

Thus

$$\log n! = \log\sqrt{2\pi} + (n + \frac{1}{2})\log n - n + o(1) = n\log n - n + O(\log n). \tag{9.4}$$

Now let $m = \lfloor xn \rfloor$. Then by (9.1),

$$V_q(n, m) = \sum_{i=0}^{m} \binom{n}{i}(q-1)^i.$$

The last term of the summation is the largest over this index range. Also, it is clear that

$$\sum_{i=0}^{m-1} \binom{n}{i}(q-1)^i \leq m\binom{n}{m}(q-1)^m.$$

We thus obtain

$$\binom{n}{m}(q-1)^m \leq V_q(n, m) \leq (1+m)\binom{n}{m}(q-1)^m.$$

Take $\log_q$ throughout and divide through by n to obtain

$$\begin{aligned}\frac{1}{n}\log_q\binom{n}{m} + \frac{m}{n}\log_q(q-1) &\leq \frac{1}{n}\log_q V_q(n, m) \\ &\leq \frac{1}{n}\log_q(1+m) + \frac{1}{n}\log_q\binom{n}{m} + \frac{m}{n}\log_q(q-1).\end{aligned} \tag{9.5}$$

As $n \to \infty$, $\frac{1}{n}\log_q(1+m) \to 0$. Using the fact that $\frac{m}{n} = \delta + o(1)$, we obtain

$$\begin{aligned}\lim_{n\to\infty}\frac{1}{n}\log_q V_q(n, m) &= \lim_{n\to\infty}\frac{1}{n}\log_q\binom{n}{m} + \frac{m}{n}\log_q(q-1) \\ &= \lim_{n\to\infty}\frac{1}{n}\log_q\binom{n}{m} + \delta\log_q(q-1) + o(1).\end{aligned}$$

Using (9.4) we have

$$\begin{aligned}
\lim_{n\to\infty} \frac{1}{n} \log_q V_q(n,m)& \\
&= \log_q n - \delta \log_q m - (1-\delta)\log_q(n-m) + \delta \log_q(q-1) + o(1) \\
&= \log_q n - \delta \log_q \delta - \delta \log_q n - (1-\delta)\log_q(1-\delta) - (1-\delta)\log_q n \\
&+ \quad \delta \log_q(q-1) + o(1) \\
&= -\delta \log_q \delta - (1-\delta)\log_q(1-\delta) + \delta\log_q(q-1) + o(1) = H_q(\delta) + o(1).
\end{aligned}$$

□

9.1 The Gilbert-Varshamov Bound

The Gilbert-Varshamov bound is a lower bound on $A_q(n,d)$.

Theorem 9.2 *For natural numbers n and d, with $d \le n$,*

$$A_q(n,d) \ge \frac{q^n}{V_q(n,d-1)}. \tag{9.6}$$

Proof [350] Let C be a code of length n and distance d with the maximum number of codewords . Then of all the q^n possible n-tuples, there is none with distance d or more to some codeword in C. (Otherwise, that n-tuple could be added to the code and C would not have had the maximum number of codewords.) Thus, the Hamming spheres of radius $d-1$ around the codewords cover all the n-tuples, so that the sum of their volumes is $\ge$ the number of points. That is,

$$|C| V_q(n,d-1) \ge q^n.$$

This is equivalent to (9.6). □

For a linear code, the Gilbert-Varshamov bound can be manipulated as follows:

$$\log_q A_q(n,d) \ge n - \log_q V_q(n,d-1)$$

or

$$n - \log_q A_q(n,d) \le \log_q V_q(n,d-1).$$

The correction capability satisfies $d = 2t+1$. The redundancy can be written as $r = n - k = n - \log_q A_q(n,d)$. We obtain

$$r \le \log_q V_q(n,2t). \tag{9.7}$$

The Gilbert-Varshamov bound can thus be viewed as an *upper bound* on the necessary redundancy for a code: there exists a t-error correcting q-ary code with redundancy r bounded as in (9.7).

The Gilbert-Varshamov bound also has an asymptotic form.

Theorem 9.3 *If $0 \le \delta \le \rho = (q-1)/q$, then*

$$\alpha_q(\delta) \ge 1 - H_q(\delta).$$

Proof [350] Using (9.2), (9.6) and Lemma 9.1 we have

$$\alpha_q(\delta) = \limsup_{n\to\infty} \frac{1}{n} \log_q A_q(n, \lfloor \delta n \rfloor) \geq \lim_{n\to\infty} (1 - \frac{1}{n} \log_q V_q(n, \delta n)) = 1 - H_q(\delta).$$

□

The Gilbert-Varshamov bound is a lower bound: it should be possible to do at least as well as this bound predicts for long codes. However, for many years it was assumed that $\alpha_q(\delta)$ would, in fact, be equal to the lower bound for long codes, since no families of codes were known that were capable of exceeding the Gilbert-Varshamov bound as the code length increased. In 1982, a family of codes based on algebraic geometry was reported, however, which exceeded the lower bound [342]. Unfortunately, algebraic geometry codes fall beyond the scope of this book. (See [341] for a comprehensive introduction to algebraic geometry codes, or [349] or [274]. For mathematical background of these codes, see [321].)

9.2 The Plotkin Bound

Theorem 9.4 *Let C be a q-ary code of length n and minimum distance d. Then if $d > \rho n$,*

$$A_q(n, d) \leq \frac{d}{d - \rho n}, \tag{9.8}$$

where $\rho = (q-1)/q$.

Proof [350] Consider a code C with M codewords in it. Form a list with the M codewords as the rows, and consider a column in this list. Let q_j denote the number of times that the jth symbol in the code alphabet, $0 \leq j < q$, appears in this column. Clearly, $\sum_{j=0}^{q-1} q_j = M$.

Let the rows of the table be arranged so that the q_0 codewords with the 0th symbol are listed first and call that set of codewords R_0, the q_1 codewords with the 1st symbol are listed second and call that set of codewords R_1, and so forth. Consider the Hamming distance between all $M(M-1)$ pairs of codewords, as perceived by this selected column. For pairs of codewords within a single set R_i, all the symbols are the same, so there is no contribution to the Hamming distance. For pairs of codewords drawn from different sets, there is a contribution of 1 to the Hamming distance. Thus, for each of the q_j codewords drawn from set R_j, there is a total contribution of $M - q_j$ to the Hamming distance between the codewords in R_j and all the other sets. Summing these up, the contribution of this column to the sum of the distances between all pairs of codewords is

$$\sum_{j=0}^{q-1} q_j(M - q_j) = M \sum_{j=0}^{q-1} q_j - \sum_{j=0}^{q-1} q_j^2 = M^2 - \sum_{j=0}^{q-1} q_j^2.$$

Now use the Cauchy-Schwartz inequality (see Box 9.2 and Exercise 9.6) to write

$$\sum_{j=0}^{q-1} q_j(M - q_j) \leq M^2 - \frac{1}{q}\left(\sum_{j=0}^{q-1} q_j\right)^2 = M^2\left(1 - \frac{1}{q}\right).$$

Now total this result over all n columns. There are $M(M-1)$ pairs of codewords, each a distance at least d apart. We obtain

$$M(M-1)d \leq n\left(1 - \frac{1}{q}\right)M^2 = n\rho M^2$$

Box 9.2: The Cauchy-Schwartz Inequality

For our purposes, the Cauchy-Schwartz inequality can be expressed as follows (see [246] for extensions and discussions): Let $\mathbf{a} = (a_1, a_2, \ldots, a_n)$ and $\mathbf{b} = (b_1, b_2, \ldots, b_n)$ be sequences of real or complex numbers. Then

$$\left|\sum_{i=1}^{n} a_i b_i\right|^2 \leq \sum_{i=1}^{n} |a_i|^2 \sum_{i=1}^{n} |b_i|^2.$$

or

$$M \leq \frac{d}{d - n\rho}.$$

Since this result holds for any code, since the $\mathcal{C}$ was arbitrary, it must hold for the code with $A_q(n, d)$ codewords. □

Equivalently,

$$d \leq \frac{n\rho M}{M - 1}.$$

The Plotkin bound provides an upper bound on the distance of a code with given length n and size M.

9.3 The Griesmer Bound

Theorem 9.5 *For a linear block (n, k) q-ary code $\mathcal{C}$ with minimum distance d,*

$$n \geq \sum_{i=0}^{k-1} \lceil d/q^i \rceil.$$

Proof Let $N(k, d)$ be the length of the shortest q-ary linear code of dimension k and minimum distance d. Let $\mathcal{C}$ be an $(N(k, d), k, d)$ code and let G be a generator matrix of the code. Assume (without loss of generality, by row and/or column interchanges and/or row operations) that G is written with the first row as follows:

$$G = \left[\begin{array}{cccc|cccc} 1 & 1 & \cdots & 1 & 0 & 0 & \cdots & 0 \\ \hline & G_1 & & & & G_2 & & \end{array}\right],$$

where G_1 is $(k-1) \times d$ and G_2 is $(k-1) \times (N(k, d) - d)$. Claim: G_2 has rank $k - 1$. Otherwise, it would be possible to make the first row of G_2 equal to 0 (by row operations). Then an appropriate input message could produce a codeword of weight $< d$, by canceling one of the ones from the first row with some linear combination of rows of G_1, resulting in a codeword of minimum distance $< d$.

Let G_2 be the generator for an $(N(k, d) - d, k - 1, d_1)$ code $\mathcal{C}'$. We will now determine a bound on d_1.

Now let $[\mathbf{u}_0|\mathbf{v}]$ (the concatenation of two vectors) be a codeword in $\mathcal{C}$,

$$[\mathbf{u}_0|\mathbf{v}] = \mathbf{m}_0 G,$$

where $\mathbf{v} \in \mathcal{C}'$ has weight d_1 and where $\mathbf{m}_0$ is a message vector with 0 in the first position,

$$\mathbf{m}_0 = [0, m_2, \ldots, m_k].$$

Let z_0 be the number of zeros in $\mathbf{u}_0$, $z_0 = d - \mathrm{wt}(\mathbf{u}_0)$. Then we have

$$\mathrm{wt}(\mathbf{u}_0) + d_1 = (d - z_0) + d_1 \geq d.$$

Let $\mathbf{m}_i = [i, m_2, \ldots, m_k]$, for $i \in GF(q)$, and let

$$\mathbf{u}_i = \mathbf{m}_i G.$$

Let z_i be the number of zeros in $\mathbf{u}_i$, $z_i = d - \mathrm{wt}(\mathbf{u}_0)$. As i varies over the elements in $GF(q)$, eventually every element in $\mathbf{u}_i$ will be set to 0. Thus

$$\sum_{i=0}^{q-1} z_i = d.$$

Writing down the weight equation for each i we have

$$\begin{aligned} d_1 + d - z_0 &\geq d \\ d_1 + d - z_1 &\geq d \\ &\vdots \\ d_1 + d - z_{q-1} &\geq d. \end{aligned}$$

Summing all these equations, we obtain

$$qd_1 + qd - d \geq qd$$

or

$$d_1 \geq \lceil d/q \rceil.$$

G_2 therefore generates an $(N(k, d) - d, k - 1, \lceil d/q \rceil)$ code. We conclude that

$$N(k - 1, \lceil d/q \rceil) \leq N(k, d) - d.$$

Now we simply proceed inductively:

$$\begin{aligned} N(k, d) &\geq d + N(k - 1, \lceil d/q \rceil) \geq d + \lceil d/q \rceil + N(k - 2, \lceil d/q^2 \rceil) \\ &\vdots \\ &\geq \sum_{i=0}^{k-2} \lceil d/q^i \rceil + N(1, \lceil d/2^{k-1} \rceil) \\ &= \sum_{i=0}^{k-1} \lceil d/q^i \rceil. \end{aligned}$$

Since $N(k, d) < n$ for any actual code, the result follows. □

9.4 The Linear Programming and Related Bounds

The linear programming bound takes the most development to produce of the bounds introduced so far. However, the tools introduced are interesting and useful in their own right. Furthermore, the programming bound technique leads to one of the tightest bounds known. We introduce first what is meant by linear programming, then present the main theorem, still leaving some definitions unstated. This is followed by the definition of Krawtchouk polynomials, the character (needed for a couple of key lemmas), and finally the proof of the theorem.

Let $\mathbf{x}$ be a vector, $\mathbf{c}$ a vector, $\mathbf{b}$ a vector, and A a matrix. A problem of the form

$$\begin{aligned}&\text{maximize } \mathbf{c}^T\mathbf{x}\\&\text{subject to } A\mathbf{x} \leq \mathbf{b}\end{aligned} \tag{9.9}$$

(or other equivalent forms) is said to be a *linear programming problem*. The maximum quantity $\mathbf{c}^T\mathbf{x}$ from the solution is said to be the *value* of the linear programming problem.

Example 9.1 Consider the following problem.

$$\begin{aligned}&\text{maximize } x_1 + x_2\\&\text{subject to } x_1 + 2x_2 \leq 10\\&\qquad\frac{15}{2}x_1 + 5x_2 \leq 45\\&\qquad x_1 \geq 0 \quad x_2 \geq 0.\end{aligned}$$

Figure 9.2 illustrates the geometry. The shaded region is the *feasible region*, the region where all the constraints are satisfied. The function $x_1 + x_2$ increases in the direction shown, so that the point in the feasible region maximizing $x_1 + x_2$ is as shown.

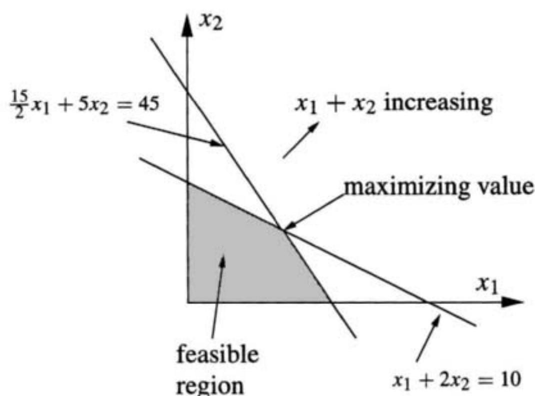
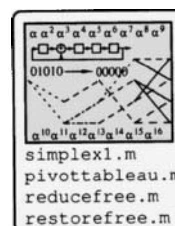


Figure 9.2: A linear programming problem.



The solution to this problem is $x_1 = 4$, $x_2 = 3$ and the value is $x_1 + x_2 = 7$. □

The feasible region always forms a polytope and, due to the linear nature of the function being optimized, the solution is always found on a vertex or along an edge of the feasible region. Linear programming problems arise in a variety of applied contexts and algorithmic

methods of solving them are well established. (See [246] and the references therein for an introduction.)

The linear programming bound applies to both linear and nonlinear codes.

Definition 9.5 Let $\mathcal{C}$ be a q-ary code with M codewords of length n. Define

$$A_i = \frac{1}{M}|\{(\mathbf{c}, \mathbf{d})|\mathbf{c} \in \mathcal{C}, \mathbf{d} \in \mathcal{C}, d_H(\mathbf{c}, \mathbf{d}) = i\}|.$$

That is, A_i is the (normalized) number of codewords in the code at a distance i from each other. The sequence $(A_0, A_1, \ldots, A_n)$ is called the **distance distribution** of the code. For a linear code, the distance distribution is the weight distribution. □

In Section 9.4.1, we will introduce a family of polynomials $K_k(x)$, known as *Krawtchouk polynomials*. The theorem is expressed in terms of these polynomials as follows.

Theorem 9.6 *[350] (Linear programming bound) For a q-ary code of length n and minimum distance d*

$$A_q(n, d) \leq M,$$

where M is value of the linear programming problem

$$\begin{aligned} \textit{maximize} \ & \sum_{i=0}^{n} A_i \\ \textit{subject to} \ & A_0 = 1, \\ & A_i = 0 \textit{ for } 1 \leq i < d \\ & \sum_{i=0}^{n} A_i K_k(i) \geq 0 \textit{ for } k \in \{0, 1, \ldots, n\} \\ & A_i \geq 0 \textit{ for } i \in \{0, 1, \ldots, n\}. \end{aligned}$$

Furthermore, if $q = 2$ and d is even, then we may take $A_i = 0$ for odd i.

Solution of the linear programming problem in this theorem not only provides a bound on $A_q(n, d)$, but also the distance distribution of the code.

Let us begin with an example that demonstrates what is involved in setting up the linear programming problem, leaving the $K_k(i)$ functions still undefined.

Example 9.2 Determine a bound on $A_2(14, 6)$ for binary codes.

Since $q = 2$, we have $A_1 = A_3 = A_5 = A_7 = A_9 = A_{11} = A_{13} = 0$. Furthermore since the minimum distance is 6, we also have $A_2 = A_4 = 0$. All the A_i are ≥ 0. The condition $\sum_{i=0}^{n} A_i K_k(i) \geq 0$ in the theorem becomes

$$\begin{aligned} & K_0(0) + K_0(6)A_6 + K_0(8)A_8 + K_0(10)A_{10} + K_0(12)A_{12} + K_0(14)A_{14} \geq 0 \\ & K_1(0) + K_1(6)A_6 + K_1(8)A_8 + K_1(10)A_{10} + K_1(12)A_{12} + K_1(14)A_{14} \geq 0 \\ & K_2(0) + K_2(6)A_6 + K_2(8)A_8 + K_2(10)A_{10} + K_2(12)A_{12} + K_2(14)A_{14} \geq 0 \\ & \vdots \\ & K_{14}(0) + K_{14}(6)A_6 + K_{14}(8)A_8 + K_{14}(10)A_{10} + K_{14}(12)A_{12} + K_{14}(14)A_{14} \geq 0 \end{aligned} \tag{9.10}$$

This problem can clearly be expressed in the form (9.9). □

9.4.1 Krawtchouk Polynomials

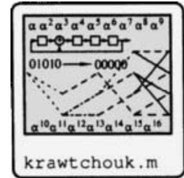
The Krawtchouk polynomials mentioned above and used in the linear programming bound are now introduced.

Definition 9.6 The Krawtchouk polynomial $K_k(x; n, q)$ is defined by

$$K_k(x; n, q) = \sum_{j=0}^{k} (-1)^j \binom{x}{j} \binom{n-x}{k-j} (q-1)^{k-j},$$

where

$$\binom{x}{j} = \frac{x(x-1)\cdots(x-j+1)}{j!} \text{ for } x \in \mathbb{R}.$$



Usually, $K_k(x; n, q)$ is used in the context of a fixed n and q, so the abbreviated notation $K_k(x)$ is used. □

Some of the important properties of Krawtchouk polynomials are developed in Exercise 9.16.

9.4.2 Character

We also need the idea of a **character**.

Definition 9.7 Let $\langle G, +\rangle$ be a group and let $\langle T, \cdot\rangle$ be the group of complex numbers which have absolute value 1, with multiplication as the operation. A **character** is a homomorphism $\chi : G \to T$. That is, for all $g_1, g_2 \in G$,

$$\chi(g_1 + g_2) = \chi(g_1)\chi(g_2). \tag{9.11}$$

If $\chi(g) = 1$ for every $g \in G$, then χ is called the **principal character**. □

It is straightforward to show that $\chi(0) = 1$, where 0 is the identity of G.

The lemma which we need for our development is the following.

Lemma 9.7 *If χ is a character for $\langle G, +\rangle$, then*

$$\sum_{g \in G} \chi(g) = \begin{cases} |G| & \text{if } \chi \text{ is the principal character} \\ 0 & \text{otherwise.} \end{cases}$$

Proof If χ is principal, the first part is obvious.

Let h be an arbitrary element of G. Then

$$\chi(h) \sum_{g \in G} \chi(g) = \sum_{g \in G} \chi(h+g) = \sum_{k \in G} \chi(k),$$

where the first equality follows from the homomorphism (9.11) and the second equality follows since $h + g$ sweeps out all elements of G as g sweeps over all the elements of G. We thus have

$$(\chi(h) - 1) \sum_{g \in G} \chi(g) = 0.$$

Since h was arbitrary, then if χ is not principal it is possible to choose an $h \in G$ such that $\chi(h) \neq 1$. We must therefore have $\sum_{g \in G} \chi(g) = 0$. □

Example 9.3 The character property of Lemma 9.7 is actually familiar from signal processing, under a slightly different guise. Let $G = \mathbb{Z}_n$, the set of integers modulo n, and let $\chi_l(g) = e^{-j2\pi lg/n}$ for $g \in G$. Then

$$\sum_{g\in G} \chi_l(g) = \sum_{i=0}^{n-1} e^{-j2\pi lg/n} = \begin{cases} n & l \equiv 0 \pmod{n} \\ 0 & \text{otherwise.} \end{cases}$$

Thus, $\chi_l(g)$ is principal if $l = 0$, and not principal if $l \neq 0$. □

Now let G be the additive group in $GF(q)$. Let $\omega = e^{j2\pi/q}$ be a primitive qth root of unity in $\mathbb{C}$. We want to define a character by

$$\chi(g) = \omega^g$$

for $g \in GF(q)$, but g is not an integer. However, for the purposes of defining a character, we can *interpret* the elements of $GF(q)$ as integers. The only property we need to enforce is the homomorphism property, $\omega^{g_1+g_2} = \omega^{g_1}\omega^{g_2}$, which will follow if we make the integer interpretation. Thus, we can interpret the alphabet over which a q-ary code exists as $\mathbb{Z}/q\mathbb{Z} \cong \mathbb{Z}_q$, which we will denote as Q.

Note that this character is not principal, so that, by Lemma 9.7,

$$\sum_{g\in Q} \omega^g = 0$$

or

$$\sum_{g\in Q\setminus\{0\}} \omega^g = -1. \tag{9.12}$$

Also note that

$$\sum_{g\in Q\setminus\{0\}} \omega^{0g} = (q-1). \tag{9.13}$$

Let $\langle \mathbf{x}, \mathbf{y}\rangle$ be the conventional inner product (see Definition 2.27)

$$\langle \mathbf{x}, \mathbf{y}\rangle = \sum_{i=1}^{n} x_i y_i.$$

9.4.3 Krawtchouk Polynomials and Characters

We now present an important lemma which relates Krawtchouk polynomials and characters.

Lemma 9.8 *[350] Let ω be a primitive qth root of unity in $\mathbb{C}$ and let $\mathbf{x} \in Q^n$ be a fixed codevector with weight i. Then*

$$\sum_{\substack{\mathbf{y}\in Q^n: \\ \text{wt}(\mathbf{y})=k}} \omega^{\langle \mathbf{x},\mathbf{y}\rangle} = K_k(i).$$

Proof Assume that all the weight of $\mathbf{x}$ is in the first i positions,

$$\mathbf{x} = [x_1, x_2, \ldots, x_i, 0, 0, \ldots, 0],$$

with x_1 through x_i not equal to 0. In the vector $\mathbf{y}$ of weight k, choose positions $h_1, h_2, \ldots, h_k$ such that

$$0 < h_1 < h_2 < \cdots < h_j \leq i < h_{j+1} < \cdots < h_k \leq n$$

with $y_{h_k} \neq 0$. That is, the first j nonzero positions of $\mathbf{y}$ overlap the nonzero positions of $\mathbf{x}$. Let D be the set of all words of weight k that have their nonzero coordinates at these k fixed positions. Then

$$\begin{aligned}\sum_{\mathbf{y}\in D} \omega^{\langle \mathbf{x},\mathbf{y}\rangle} &= \sum_{\mathbf{y}\in D} \omega^{x_{h_1}y_{h_1}+\cdots+x_{h_j}y_{h_j}+0+\cdots+0} \\ &= \sum_{y_{h_1}\in Q\setminus\{0\}} \cdots \sum_{y_{h_k}\in Q\setminus\{0\}} \omega^{x_{h_1}y_{h_1}+\cdots+x_{h_j}y_{h_j}+0+\cdots+0} \\ &= \sum_{y_{h_1}\in Q\setminus\{0\}} \omega^{x_{h_1}y_{h_1}} \cdots \sum_{y_{h_j}\in Q\setminus\{0\}} \omega^{x_{h_j}y_{h_j}} \sum_{y_{h_{j+1}}\in Q\setminus\{0\}} \omega^0 \cdots \sum_{y_{h_k}\in Q\setminus\{0\}} \omega^0 \\ &= (-1)^j (q-1)^{n-j},\end{aligned}$$

where the last equality follows from (9.12) and (9.13).

The set D may be chosen in $\binom{i}{j}\binom{n-i}{k-j}$ different ways for each fixed position j. We have

$$\begin{aligned}\sum_{\substack{\mathbf{y}\in Q^n:\\ \mathrm{wt}(\mathbf{y})=k}} \omega^{\langle \mathbf{x},\mathbf{y}\rangle} &= \sum_{\substack{\text{Different}\\ \text{choices of}\\ D}} \sum_{\mathbf{y}\in D} \omega^{\langle \mathbf{x},\mathbf{y}\rangle} = \sum_{j=0}^{k} \binom{i}{j}\binom{n-i}{k-j} \sum_{\mathbf{y}\in D} \omega^{\langle \mathbf{x},\mathbf{y}\rangle} \\ &= \sum_{j=0}^{k} \binom{i}{j}\binom{n-i}{k-j} (-1)^j (q-1)^{k-j} = K_k(i),\end{aligned}$$

by the definition of the Krawtchouk polynomial. □

The final lemma we need in preparation for proving the linear programming bound is the following.

Lemma 9.9 *Let $\{A_0, A_1, \ldots, A_n\}$ be the distance distribution of a code C of length n with M codewords. Then*

$$\sum_{i=0}^{n} A_i K_k(i) \geq 0.$$

Proof From Lemma 9.8 and the definition of A_i we can write

$$\begin{aligned}\sum_{i=0}^{n} A_i K_k(i) &= \sum_{i=0}^{n} \frac{1}{M} \sum_{\substack{(\mathbf{x},\mathbf{y})\in C^2:\\ d(\mathbf{x},\mathbf{y})=i}} \sum_{\substack{\mathbf{z}\in C:\\ \mathrm{wt}(\mathbf{z})=k}} \omega^{\langle \mathbf{x}-\mathbf{y},\mathbf{z}\rangle} \\ &= \frac{1}{M} \sum_{\substack{\mathbf{z}\in C:\\ \mathrm{wt}(\mathbf{z})=k}} \sum_{(\mathbf{x},\mathbf{y})\in C^2} \omega^{\langle \mathbf{x}-\mathbf{y},\mathbf{z}\rangle} \\ &= \frac{1}{M} \sum_{\substack{\mathbf{z}\in C:\\ \mathrm{wt}(\mathbf{z})=k}} \left(\sum_{\mathbf{x}\in C} \omega^{\langle \mathbf{x},\mathbf{z}\rangle}\right)\left(\sum_{\mathbf{y}\in C} \omega^{-\langle \mathbf{y},\mathbf{z}\rangle}\right) \\ &= \frac{1}{M} \sum_{\substack{\mathbf{z}\in C:\\ \mathrm{wt}(\mathbf{z})=k}} \left|\sum_{\mathbf{x}\in C} \omega^{\langle \mathbf{x},\mathbf{z}\rangle}\right|^2 \geq 0.\end{aligned}$$

□

We are now ready to prove the linear programming bound.

Proof of Theorem 9.6. Lemma 9.9 shows that the distance distribution must satisfy

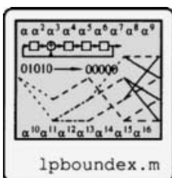
$$\sum_{i=0}^{n} A_i K_k(i) \geq 0.$$

Clearly the A_i are nonnegative, $A_0 = 1$ and the $A_i = 0$ for $1 \leq i < d$ to obtain the distance properties. By definition, we also have $\sum_{i=0}^{n} A_i = M$, the number of codewords. Hence, any code must have its number of codewords less than the largest possible value of $\sum_{i=0}^{n} A_i = M$.

For binary codes with d even we may take the $A_i = 0$ for i odd, since any codeword with odd weight can be modified to a codeword with even weight by flipping one bit without changing the minimum distance of the code. □

Example 9.4 Let us return to the problem of Example 9.2. Now that we know about the Krawtchouk polynomials, the inequalities in (9.10) can be explicitly computed. These become

$$\begin{aligned}
k=0 &: \; 1 + A_6 + A_8 + A_{10} + A_{12} + A_{14} \geq 0 \\
k=1 &: \; 14 + 2A_6 - 2A_8 - 6A_{10} - 10A_{12} - 14A_{14} \geq 0 \\
k=2 &: \; 91 - 5A_6 - 5A_8 + 11A_{10} + 43A_{12} + 91A_{14} \geq 0 \\
k=3 &: \; 364 - 12A_6 + 12A_8 + 4A_{10} - 100A_{12} - 364A_{14} \geq 0 \\
k=4 &: \; 1001 + 9A_6 + 9A_8 - 39A_{10} + 121A_{12} + 1001A_{14} \geq 0 \\
k=5 &: \; 2002 + 30A_6 - 30A_8 + 38A_{10} - 22A_{12} - 2002A_{14} \geq 0 \\
k=6 &: \; 3003 - 5A_6 - 5A_8 + 27A_{10} - 165A_{12} + 3003A_{14} \geq 0 \\
k=7 &: \; 3432 - 40A_6 + 40A_8 - 72A_{10} + 264A_{12} - 3432A_{14} \geq 0.
\end{aligned}$$



The other inequalities are duplicates, by symmetry in the polynomials. Also note that the $k = 0$ inequality is implicit in maximizing $A_6 + A_8 + \cdots + A_{14}$, so it does not need to be included. Solving the linear programming problem, we obtain the solution

$$A_6 = 42 \quad A_8 = 7 \quad A_{10} = 14 \quad A_{12} = 0 \quad A_{14} = 0.$$

Hence $A_2(14, 6) \leq 1 + 42 + 7 + 14 = 64$. □

9.5 The McEliece-Rodemich-Rumsey-Welch Bound

The McEliece-Rodemich-Rumsey-Welch bound is a bound that is quite tight. It applies only to binary codes and is derived using the linear programming bound.

Theorem 9.10 *(The McEliece-Rodemich-Rumsey-Welch bound) For a binary code,*

$$\alpha_2(\delta) \leq H_2\left(\frac{1}{2} - \sqrt{\delta(1-\delta)}\right), \qquad 0 \leq \delta \leq \frac{1}{2}. \tag{9.14}$$

The proof is subtle and makes use of some properties of Krawtchouk polynomials introduced in Exercise 9.16, an extension of the linear programming bound in Exercise 9.17, as well as other properties that follow from the fact that Krawtchouk polynomials are orthogonal. What is provided here is a sketch; the reader may want to fill in some of the details.

Proof [227, 350] Let t be an integer in the range $1 \leq t \leq \frac{n}{2}$ and let a be a real number in $[0, n]$. Define the polynomial

$$\alpha(x) = \frac{1}{a-x}(K_t(a)K_{t+1}(x) - K_{t+1}(a)K_t(x))^2. \tag{9.15}$$

Because the $\{K_k(x)\}$ form an orthogonal set, they satisfy the Christoffel-Darboux formula

$$\frac{K_{k+1}(x)K_k(y) - K_k(x)K_{k+1}(y)}{y-x} = \frac{2}{k+1}\binom{n}{k}\sum_{i=0}^{k}\frac{K_i(x)K_i(y)}{\binom{n}{i}} \tag{9.16}$$

(see [246, p. 224] for an introduction, or a reference on orthogonal polynomials such as [110]). Using (9.16), equation (9.15) can be written as

$$\alpha(x) = \frac{2}{t+1}\binom{n}{t}(K_t(a)K_{t+1}(x) - K_{t+1}(a)K_t(x))\sum_{k=0}^{t}\frac{K_k(a)K_k(x)}{\binom{n}{k}}. \tag{9.17}$$

Since $K_k(x)$ is a polynomial in x of degree k, it follows that $\alpha(x)$ is a polynomial in x of degree $2t+1$. Then $\alpha(x)$ can also be written as a series in $\{K_k(x)\}$ as

$$\alpha(x) = \sum_{k=0}^{2t+1} \alpha_k K_k(x)$$

(since both are polynomials of degree $2t+1$). Let $\beta(x) = \alpha(x)/\alpha_0 = 1 + \sum_{k=1}^{2t+1} \beta_k K_k(x)$. We desire to choose a and t such that $\beta(x)$ satisfies the conditions of the theorem in Exercise 9.17: $\beta_k \geq 0$ and $\beta(j) \leq 0$ for $j = d, d+1, \ldots, n$.

Note that if $a \leq d$, then by (9.15) $\alpha(j) \leq j$ for $j = d, d+1, \ldots, n$, so the only thing to be verified is whether $\alpha_i \geq 0, i = 1, \ldots, n$ and $\alpha_0 > 0$. This is established using the interlacing property of the roots of the Krawtchouk polynomials: $K_{t+1}(x)$ has $t+1$ distinct real zeros on $(0, n)$. Denote these as $x_1^{(t+1)}, \ldots, x_{t+1}^{(t+1)}$, with $x_1^{(t+1)} < x_2^{(t+1)} < \cdots < x_{t+1}^{(t+1)}$. Similarly $K_t(x)$ has t distinct real zeros on $(0, n)$, which we denote as $x_1^{(t)}, \ldots, x_t^{(t)}$ and order similarly. These roots are interlaced:

$$0 < x_1^{(t+1)} < x_1^{(t)} < x_2^{(t+1)} < x_2^{(t)} < \cdots < x_t^{(t)} < x_{t+1}^{(t+1)} < n.$$

(This interlacing property follows as a result of the orthogonality properties of the Krawtchouk polynomials; see [110].)

So choose t such that $x_1^{(t)} < d$ and choose a between $x_1^{(t+1)}$ and $x_1^{(t)}$ in such a way that $K_t(a) = -K_{t+1}(a) > 0$. Then $\alpha(x)$ can be written in the form $\alpha(x) = \sum c_{kl}K_k(x)K_l(x)$ where all the $c_{kl} \geq 0$. Thus the $\alpha_i \geq 0$. It is clear that $\alpha_0 = -\frac{2}{t+1}\binom{n}{t}K_t(a)K_{t+1}(a) > 0$.

Thus, the theorem in Exercise 9.17 can be applied:

$$A_q(n, d) \leq \beta(0) = \frac{\alpha(0)}{\alpha_0} = \frac{(n+1)x^2}{2a(t+1)}\binom{n}{t}. \tag{9.18}$$

We now invoke another result about orthogonal polynomials. It is known (see [110]) that as $n \to \infty$, if $t \to \infty$ in such a way that $t/n \to \tau$ for some $0 < \tau < \frac{1}{2}$, then $x_1^{(t)}/n \to \frac{1}{2} - \sqrt{\tau(1-\tau)}$. So let $n \to \infty$ and $d/n \to \delta$ in such a way that $t/n \to \frac{1}{2} - \sqrt{\delta(1-\delta)}$. Taking the logarithm of (9.18), dividing by n and using the results of Lemma 9.1, the result follows. $\square$

In [227], a slightly stronger bound is given (which we do not prove here):

$$\alpha_2(\delta) \le \min_{0 \le u \le 1-2\delta} \{g(u^2) - g(u^2 + 2\delta u - 2\delta)\}, \tag{9.19}$$

where

$$g(x) = H_2((1 - \sqrt{1-x})/2).$$

The bound (9.14) actually follows from this with $u = 1 - 2\delta$. For $0.273 \le \delta \le \frac{1}{2}$, the bounds (9.14) and (9.19) coincide, but for $\delta < 0.273$ (9.19) gives a slightly tighter bound.

Another bound is obtained from (9.19) when $u = 0$. This gives rise to the *Elias bound* for binary codes. In its more general form, it can be expressed as follows.

Theorem 9.11 *(Elias bound) For a q-ary code,*

$$\alpha_q(\delta) \le 1 - H_q(\rho - \sqrt{\rho(1-\rho)}) \qquad 0 \le \delta \le \rho,$$

where $\rho = (q-1)/q$.

The Elias bound also has a nonasymptotic form: Let $r \le \rho n$, where $\rho = (q-1)/q$. Then [350, p. 65]

$$A_q(n,d) \le \frac{\rho n d}{r^2 - 2\rho n r + \rho n d} \frac{q^n}{V_q(n,r)}.$$

The value of r can be adjusted to find the tightest bound.

9.6 Exercises

9.1 Using the Hamming and Gilbert-Varshamov bounds, determine lower and upper bounds on the redundancy $r = n - k$ for the following codes.

(a) A single-error correcting binary code of length 7.
(b) A single-error correcting binary code of length 15.
(c) A triple-error correcting binary code of length 23.
(d) A triple-error correcting ternary code of length 23.
(e) A triple-error correcting 4-ary code of length 23.
(f) A triple-error correcting 16-ary code of length 23.

9.2 With $H_2(x) = -x\log_2 x - (1-x)\log_2(1-x)$, show that

$$2^{-nH_2(\lambda)} = \lambda^{n\lambda}(1-\lambda)^{n(1-\lambda)}.$$

9.3 With $H_2(x)$ as defined in the previous exercise, show that for any λ in the range $0 \le \lambda \le \frac{1}{2}$,

$$\sum_{i=0}^{\lambda n} \binom{n}{i} \le 2^{nH_2(\lambda)}.$$

Hint: Use the binomial expansion on $(\lambda + (1-\lambda))^n = 1$; truncate the sum up to λn and use the fact that $(\lambda/(1-\lambda))^i < (\lambda/(1-\lambda))^{\lambda n}$.

9.4 [33] Prove: In any set of M distinct nonzero binary words of length n having weight at most $n\lambda$, the sum of the weights W satisfies

$$W \geq n\lambda(M - 2^{nH_2(\lambda)})$$

for any $\lambda \in (0, 1/2)$. *Hint*: Use the fact that $\sum_{k=0}^{\lambda n} \binom{n}{k} \leq 2^{nH_2(\lambda)}$. Then argue that for each λ there are at least $M - 2^{nH_2(\lambda)}$ words of weight exceeding $n\lambda$.

9.5 For binary codes, show that $A_2(n, 2l - 1) = A_2(n + 1, 2l)$.

9.6 Using the Cauchy-Schwartz inequality, show that

$$\sum_{i=0}^{q-1} q_i^2 \geq \frac{1}{q}\left(\sum_{i=0}^{q-1} q_i\right)^2.$$

9.7 What is the largest possible number of codewords of a ternary ($q = 3$) code having $n = 13$ and $d = 9$?

9.8 Examine the proof of Theorem 9.4. Under what condition can equality be obtained in (9.8)?

9.9 Prove the asymptotic Plotkin bound:

$$\alpha_q(\delta) = 0 \text{ if } \rho \leq \delta \leq 1$$
$$\alpha_q(\delta) \leq 1 - \delta/\rho \text{ if } 0 \leq \delta \leq \rho.$$

Hint: For the second part, let $n' = \lfloor (d - 1)/\rho \rfloor$. Show that $1 \leq d - \rho n' \leq 1 + \rho$. Shorten a code of length n with M codewords to a code of length n' with M' codewords, both having the same distance. Show that $M' \geq q^{n'-n} M$ and apply the Plotkin bound.

9.10 [136, 30] Stirling's formula. In this problem, you will derive the approximation

$$n! \approx n^n e^{-n} \sqrt{2\pi n}.$$

(a) Show that

$$\int_1^n \log x \, dx = n \log n - n + 1.$$

(b) Use the trapezoidal rule, as suggested by Figure 9.3(a), to show that

$$\int_1^n \log x \, dx \geq \log n! - \frac{1}{2} \log n,$$

where the overbound is the area between the function $\log x$ and its trapezoidal approximation. Conclude that

$$n! \leq n^n e^{-n} \sqrt{n} e.$$

(c) Using the integration regions suggested in Figure 9.3(b), show that

$$\int_1^n \log x \, dx \leq \log n! + \frac{1}{8} - \frac{1}{2} \log n$$

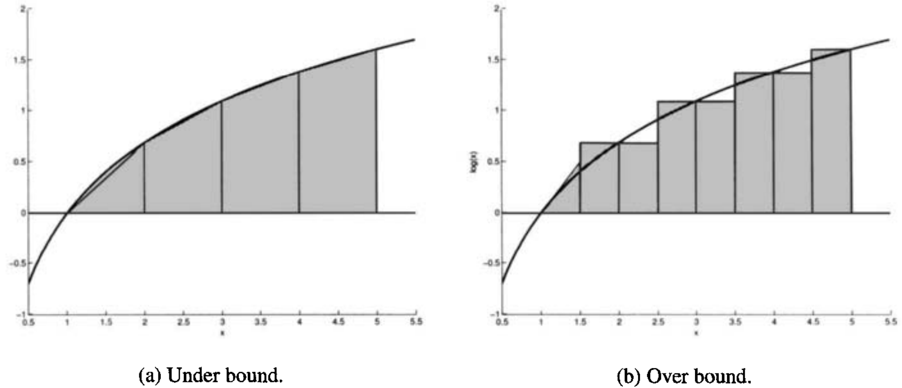
and that

$$n! \geq n^n e^{-n} \sqrt{n} e^{7/8}.$$

In this integral approximation, for the triangular region, use a diagonal line tangent to the function at 1.

In the approximation

$$n! \approx n^n e^{-n} \sqrt{n} C, \tag{9.20}$$



(a) Under bound. (b) Over bound.

Figure 9.3: Finding Stirling's formula.

we thus observe that

$$e^{7/8} \leq C \leq e.$$

It will turn out that $C = \sqrt{2\pi}$ works as $n \to \infty$. To see this, we take a nonobvious detour. Define the function

$$I_k = \int_0^{\pi/2} \cos^k x \, dx.$$

(d) Show that $I_k = I_{k-2} - I_k/(k-1)$, and hence that $I_k = (k-1)I_{k-2}/k$ for $k \geq 2$. *Hint*: $\cos^k(x) = \cos^{k-2}(x)(1 - \sin^2 x)$.

(e) Show that $I_0 = \pi/2$ and $I_1 = 1$.

(f) Show that $I_{2k-1} > I_{2k} > I_{2k+1}$, and hence that

$$1 > \frac{I_{2k}}{I_{2k-1}} > \frac{I_{2k+1}}{I_{2k-1}}.$$

(g) Show that

$$1 > 2k\left[\frac{2k-1}{2k}\frac{2k-3}{2k-2}\cdots\frac{3}{2}\right]^2\frac{\pi}{2} > \frac{2k}{2k+1}.$$

(h) Show that

$$1 > \pi k\left[\frac{(2k)!}{2^k k! 2^k k!}\right]^2 > \frac{2k}{2k+1}.$$

(i) Now substitute (9.20) for each factorial to show that

$$1 > \sqrt{\pi k}\left[\frac{(2k)^{2k}e^{-2k}\sqrt{2k}C}{2^{2k}k^k e^{-k}\sqrt{k}Ck^k e^{-k}\sqrt{k}C}\right] > \sqrt{1 - \frac{1}{2k+1}}.$$

(j) Show that $C \to \sqrt{2\pi}$ as $k \to \infty$.

9.11 Show that asymptotically, the Hamming bound can be expressed as

$$\alpha_q(\delta) \leq 1 - H_q(\delta/2).$$

9.12 Use the Griesmer bound to determine the largest possible value of k for a $(13, k, 5)$ binary code.

9.13 Use the Griesmer bound to determine the largest possible value of k for a $(14, k, 9)$ ternary code.

9.14 Find a bound on the length of shortest triple-error correcting ($d = 7$) binary code of dimension 5.

9.15 Let $d = 2^{k-1}$. Determine $N(k, d)$. Is there a code that reaches the bound? (*Hint*: simplex.)

9.16 Properties of Krawtchouk polynomials.

(a) Show that for $x \in \{0, 1, \ldots, k\}$,

$$\sum_{k=0}^{\infty} K_k(x) z^k = (1 + (q-1)z)^{n-x}(1-z)^x. \tag{9.21}$$

(b) Show that $K_k(x)$ is an orthogonal polynomial with weighting function $\binom{n}{i}(q-1)^i$, in that

$$\sum_{i=0}^{n} \binom{n}{i} (q-i)^i K_k(i) K_l(i) = \delta_{kl} \binom{n}{k} (q-1)^k q^n. \tag{9.22}$$

(c) Show that for $q = 2$,

$$K_k(x) = (-1)^k K_k(n-x).$$

(d) Show that

$$(q-1)^i \binom{n}{i} K_k(i) = (q-1)^k \binom{n}{k} K_i(k).$$

(e) Use this result to show another orthogonality relation.

$$\sum_{i=0}^{n} K_l(i) K_i(k) = \delta_{lk} q^n.$$

(f) One way to compute the Krawtchouk polynomials is to use the recursion

$$(k+1)K_{k+1}(x) = (k + (q-1)(n-k) - qx)K_x(x) - (q-1)(n-k+1)K_{k-1}(x),$$

which can be initialized with $K_0(x) = 1$ and $K_1(x) = n(q-1) - qx$ from the definition. Derive this recursion. *Hint*: Differentiate both sides of (9.21) with respect to z, multiply both sides by $(1 + (q-1)z)(1-z)$ and match coefficients of z^k.

(g) Another recursion is

$$K_k(x) = K_k(x-1) - (q-1)K_{k-1}(x) - K_{k-1}(x-1).$$

Show that this is true. *Hint*: In (9.21), replace x by $x - 1$.

9.17 [350] (Extension of Theorem 9.6) Theorem: Let $\beta(x) = 1 + \sum_{k=1}^{n} \beta_k K_k(x)$ be a polynomial with $\beta_i \geq 0$ for $1 \leq k \leq n$ such that $\beta(j) \leq 0$ for $j = d, d+1, \ldots, n$. Then $A_q(n, d) \leq \beta(0)$. Justify the following steps of the proof:

(a) $\sum_{i=d}^{n} A_i \beta(i) \leq 0$.

(b) $-\sum_{i=d}^{n} A_i \geq \sum_{k=1}^{n} \beta_k \sum_{i=d}^{n} A_k K_k(i)$

(c) $\sum_{k=1}^{n} \beta_k \sum_{i=d}^{n} A_k K_k(i) \geq -\sum_{k=1}^{n} \beta_k K_k(0)$.

(d) $-\sum_{k=1}^{n} \beta_k K_k(0) = 1 - \beta(0)$.

(e) Hence conclude that $A_q(n, d) \leq \beta(0)$.

(f) Now put the theorem to work. Let $q = 2$, $n = 2l + 1$ and $d = l + 1$. Let $\beta(x) = 1 + \beta(1)K_1(x) + \beta(2)K_2(x)$.

i. Show that $\beta(x) = 1 + \beta_1(n - 2x) + \beta_2(2x^2 - 2nx + \frac{1}{2}n(n - 1))$.
ii. Show that choosing to set $\beta(d) = \beta(n) = 0$ leads to $\beta_1 = (n + 1)/2n$ and $\beta_2 = 1/n$.
iii. Show that the conditions of the theorem in this problem are satisfied.
iv. Hence conclude that $A_q(2l + 1, l + 1) \leq 2l + 2$.

9.18 Let χ be a character, $\chi : G \to T$. Show that $\chi(0) = 1$, where 0 is the identity of G.

9.19 Show that (9.17) follows from (9.15) and (9.16).

9.7 References

Extensive discussions of bounds appear in [220]. Our discussion has benefited immensely from [350]. The Hamming bound appears in [137]. The Singleton bound appears in [314]. The Plotkin bound appears in [266]. The linear programming bound was developed in [68].

An introduction to orthogonal polynomials is in [246]. More extensive treatment of general facts about orthogonal polynomials is in [110].

Chapter 10

Bursty Channels, Interleavers, and Concatenation

10.1 Introduction to Bursty Channels

The coding techniques introduced to this point have been appropriate for channels with independent random errors, such as a memoryless binary symmetric channel, or an AWGN channel. In such channels, each transmitted symbol is affected independently by the noise. We refer to the codes that are appropriate for such channels as random error correcting codes. However, in many channels of practical interest, the channel errors tend to be clustered together in "bursts." For example, on a compact disc, a scratch on the media may cause errors in several consecutive bits. On a magnetic medium such as a hard disk or a tape, a blemish on the magnetic surface may introduce many errors. A wireless channel may experience fading over several symbol times, or a stroke of lightning might affect multiple digits. In a concatenated coding scheme employing a convolutional code as the inner code, a single incorrect decoding decision might give rise to a burst of decoding errors.

Using a conventional random error correcting block code in a bursty channel leads to inefficiencies. A burst of errors may introduce several errors into a small number codewords, which therefore need strong correction capability, while the majority of codewords are not subjected to error and therefore waste error correction capabilities.

In this chapter we introduce techniques for dealing with errors on bursty channels. The straightforward but important concept of interleaving is presented. The use of Reed-Solomon codes to handle bursts of bit errors is described. We describe methods of concatenating codes. Finally, Fire codes are introduced, which is a family of cyclic codes specifically designed to handle bursts of errors.

Definition 10.1 In a sequence of symbols, a **burst** of length l is a sequence of symbols confined to l consecutive symbols of which the first and last are in error. □

For example, in the error vector

$$\mathbf{e} = (0\,0\,0\,0\,\underbrace{1\,1\,0\,1\,0\,1\,1\,0\,0\,1\,0\,0\,1\,0\,1}\,0\,0\,0\,0\,0\,0\,0)$$

is a burst of length 15.

A code C is said to have burst-error-correcting capability l if it is capable of correcting all bursts up to length l.

10.2 Interleavers

An **interleaver** takes a sequence of symbols and permutes them. At the receiver, the sequence is permuted back into the original order by a **deinterleaver**. Interleavers are efficacious in dealing with bursts of errors because, by shuffling the symbols at the receiver,

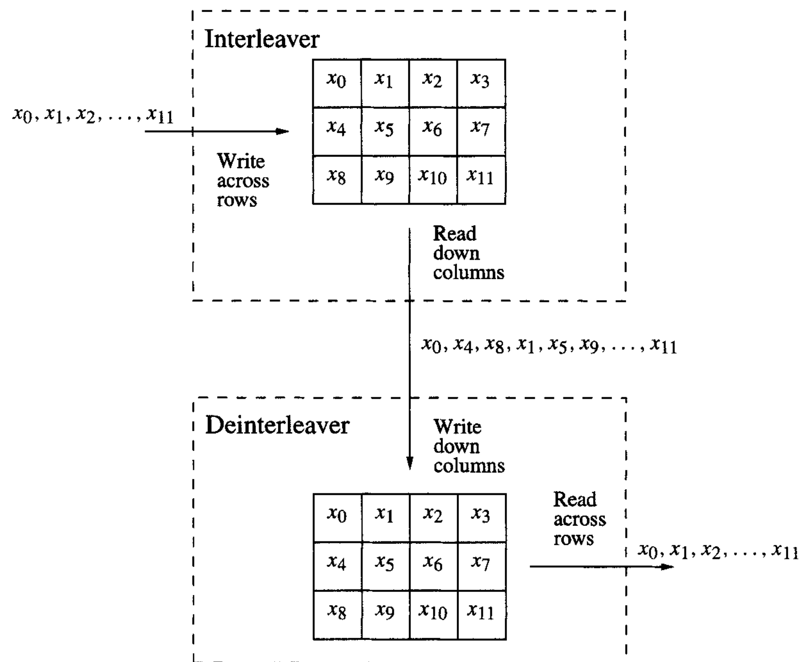


Figure 10.1: A 3×4 interleaver and deinterleaver.

a burst of errors appearing in close proximity may be broken up and spread around, thereby creating an effectively random channel.

A common way to interleave is with a **block interleaver**. This is simply an $N \times M$ array which can be read and written in different orders. Typically the incoming sequence of symbols is written into the interleaver in row order and read out in column order. Figure 10.1 shows a 3×4 interleaver. The input sequence $x_0, x_1, \ldots, x_{11}$ is read into the rows of array, as shown, and read off as the sequence

$$x_0, x_4, x_8, x_1, x_5, x_9, x_2, x_6, x_{10}, x_3, x_7, x_{11}.$$

Frequently, the width M is chosen to be the length of a codeword.

Example 10.1 We present here an application of interleaving. The UDP (user datagram protocol) internet protocol is one of the TCP/IP protocol suite which does not guarantee packet delivery, but experiences lower network latency. In this protocol, each packet carries with it a sequential packet number, so that any missing packets may be identified. Because of its lower latency, UDP is of interest in near real-time internet applications. Error correction coding can be used to significantly increase the probability that all packets are correctly received with only a moderate decrease in rate.

The data stream is blocked into message blocks of 249 bytes and the data are encoded with a (255,249) Reed-Solomon code having $d_{\min} = 7$ and capable of correcting 6 erasures. The codewords are written into an $N \times 255$ matrix as rows and read out in column order. Each column is transmitted as a data packet of length N.

Suppose that the third packet associated with this interleaving block, corresponding to the third column, is lost in transmission, as suggested by the shading on this matrix.

$c_{1,1}$	$c_{1,2}$	$c_{1,3}$	$\cdots$	$c_{1,255}$
$c_{2,1}$	$c_{2,2}$	$c_{2,3}$	$\cdots$	$c_{2,255}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$c_{N,1}$	$c_{N,2}$	$c_{N,3}$	$\cdots$	$c_{N,255}$

By the protocol, the fact that the third packet is lost is known. When the data are written into the matrix for deinterleaving, the missing column is left blank and recorded as an *erasure*. Then each of the N codewords is subjected to erasure decoding, recovering the lost byte in each codeword. For this code, up to six packets out of N may be lost in transmission (erased) and fully recovered at the receiver. □

In general, when a burst of length l is deinterleaved, it causes a maximum of $\lceil l/N \rceil$ errors to occur among the received codewords. If the code used can correct up to t errors, a decoding failure may occur if the burst length exceeds $Nt + 1$.

The **efficiency** γ of an interleaver can be defined as the ratio of the length of the smallest burst of errors that exceeds the correction capability of a block code to the amount of memory in the interleaver. Based on our discussion, for the block interleaver the efficiency is

$$\gamma = \frac{Nt+1}{NM} \approx \frac{t}{M}.$$

Another kind of interleaver is the **cross interleaver** or **convolutional interleaver** [282], which consists of a bank of delay lines of successively increasing length. An example is shown in Figure 10.2. This figure shows the input stream, and the state of the interleaver at a particular time as an aid in understanding its operation. The cross interleaver is parameterized by (M, D), where M is the number of delay lines and D the number of samples each delay element introduces. It is clear that adjacent symbols input to the interleaver are separated by MD symbols. If M is chosen to be greater than or equal to the length of the code, then each symbol in the codeword is placed on a different delay line. If a burst error of length l occurs, then $\lceil l/(MD + 1) \rceil$ errors may be introduced into the deinterleaved codewords. For a t-error correcting code, a decoding failure may be possible when l exceeds $(MD + 1)(t - 1) + 1$. The total memory of the interleaver is $(0 + 1 + 2 + \ldots + (M - 1))D = DM(M - 1)/2$. The efficiency is thus

$$\gamma = \frac{(MD+1)(t-1)+1}{DM(M-1)/2} \approx \frac{2t}{M-1}.$$

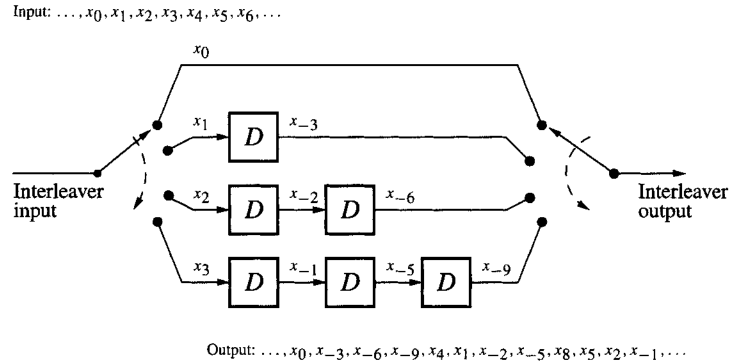
Comparison with the block interleaver shows that cross interleavers are approximately twice as efficient as block interleavers.

While block interleaving can be accomplished in a straightforward way with an array, for cyclic codes there is another approach. If C is a cyclic code of length n with generator $g(x)$, then the code obtained after interleaving with an $M \times n$ interleaver matrix is also cyclic, with generator polynomial $g(x^M)$. The encoding and syndrome computation can thus be implemented in conventional fashion using shift registers [203, p. 272].

10.3 An Application of Interleaved RS Codes: Compact Discs

By far the most common application of RS codes is to compact discs. The data on the compact disc (CD) is protected with a Reed-Solomon code, providing resilience to scratches

Interleaver



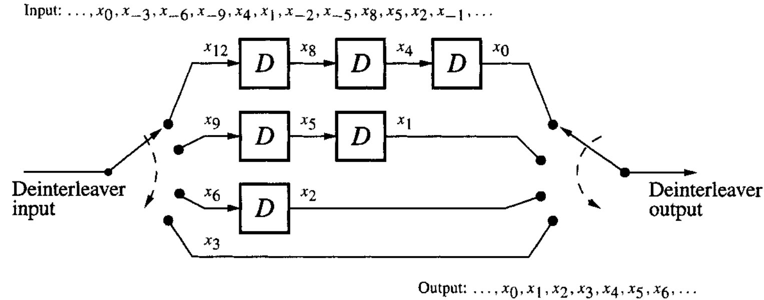


Figure 10.2: A cross interleaver and deinterleaver system.

on the surface of the disc. (Of course, scratches may still impede CD playback, as many listeners can attest, but these problems are more often the result of tracking problems with the read laser and not the problem of errors in the decoding stream read.) Because this medium is so pervasive, it is important to be familiar with the data representation. This presentation also brings out another important point. The error correction is only one aspect of the data processing that takes place. To be of practical value, error correction coding must work as a component within a larger system design. In addition to basic error correction provided by Reed-Solomon codes, protection against burst errors due to scratches on the disc is provided by the use of a cross interleaver. The data stream is also formatted with an eight-to-fourteen modulation (EFM) code which prevents excessively long runs of ones. This is necessary because the laser motion control system employs a phase-locked loop (PLL) which is triggered by bit transitions. If there is a run of ones that is too long, the PLL may drift. Details on the data stream and the EFM code are in [159]. Our summary here follows [373].

The overall block diagram of the CD recording process is shown in Figure 10.3. The 1.41 Mbps sampled data stream passes through an error correction system resulting in a data rate of 1.88 Mbps. The encoder system, referred to as CIRC, uses two interleaved, shortened, Reed-Solomon codes, C_1 and C_2. Both codes are built on codes over $GF(256)$. The eight-bit symbols of the field fit naturally with the 16-bit samples used by the A/D

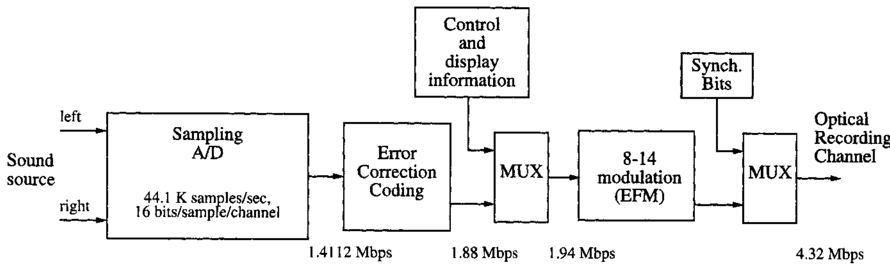


Figure 10.3: The CD recording and data formatting process.

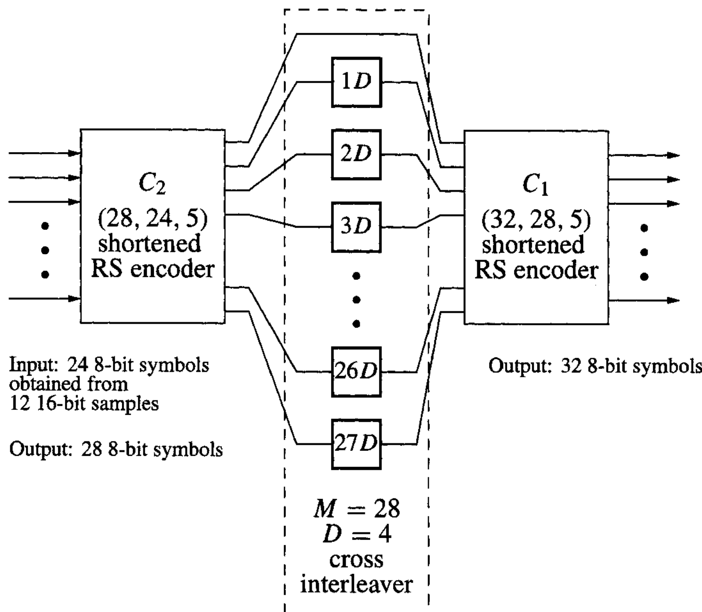


Figure 10.4: The error correction encoding in the compact disc standard.

converter. However, the codes are significantly shortened: C_1 is a (32,28) code and C_2 is a (28,24) code. For every 24 input symbols there are 32 output symbols, resulting in a rate $R = 24/32 = 3/4$. Both codes have minimum distance 5.

Encoding The outer code, C_2, uses 12 16-bit samples to create 24 8-bit symbols as its message word. The 28-symbol codeword is passed through a (28, 4) cross interleaver. The resulting 28 interleaved symbols are passed through the code C_1, resulting in 32 coded output symbols, as shown in Figure 10.4.

Decoding At the decoder (in the CD player), the data that reaches the CIRC decoder first passes through the outer decoder C_1. Since C_1 has minimum distance 5, it is capable of correcting two errors, or correcting one error and detecting two errors in the codeword, or detecting up to four errors. If the C_1 decoder detects a high-weight error pattern, such as

Table 10.1: Performance Specification of the Compact Disc Coding System [159, p. 57]

Maximum completely correctable burst length	$\approx$ 4000 bits ($\approx$ 2.5 mm track length)
Maximum interpolatable burst length in the worst case	$\approx$ 12,300 data bits ($\approx$ 7.7 mm track length)
Sample interpolation rate	One sample every 10 hours at a bit error rate (BER) of 10^{-4}. 1000 samples per minute at BER of 10^{-3}
Undetected error samples (producing click in output)	less than one every 750 hours at BER=10^{-3}. Negligible at BER $\leq 10^{-4}$
Code rate	$R = 3/4$
Implementation	One LSI chip plus one random-access memory of 2048 bytes.

a double error pattern or any error pattern causing a decoder failure, the decoder outputs 28 erased symbols. The deinterleaver spreads these erased symbols over 28 C_2 codewords, where the erasure correction capability of the C_2 code can be used.

The C_2 decoder can correct any combination of e errors and f erasures satisfying $2e + f < 5$. Since C_1 is likely to be able to produce error-free output, but will declare erasures when there seem to be too many errors for it to correct, the C_2 decoder is frequently built to be an erasures-only decoder. Since erasure decoding involves only straightforward linear algebra (e.g., Forney's algorithm) and does not require finding an error locater polynomial, this can be a low-complexity decoder. In the rare event that a vector is presented having more than four erasures that C_2 is not able to correct the C_2 decoder outputs 24 erasures. In this case, the playback system uses an "error concealment" system which either mutes the corresponding 12 samples of music or performs some kind of interpolation.

The performance specifications for this code are summarized in Table 10.1.

10.4 Product Codes

Let C_1 be an (n_1, k_1) linear block code and let C_2 be an (n_2, k_2) linear block code over $GF(q)$. An $(n_1 n_2, k_1 k_2)$ linear code called the **product code**, denoted $C_1 \times C_2$, can be formed as diagrammed in Figure 10.5. A set of $k_1 k_2$ symbols is written into a $k_2 \times k_1$ array. Each of the k_2 rows of the array is (systematically) encoded using code C_1, forming n_1 columns. Each of the n_1 columns of the array is then (systematically) encoded using code C_2, forming an $n_2 \times n_1$ array. Because of linearity, it does not matter whether the encoding procedure is reversed (C_2 encoding followed by C_1 encoding).

Theorem 10.1 *If C_1 has minimum distance d_1 and C_2 has minimum distance d_2, then the product code $C_1 \times C_2$ has minimum distance $d_1 d_2$.*

Proof The minimum weight cannot be less than $d_1 d_2$: Each nonzero row of the matrix in Figure 10.5 must have weight $\geq d_1$ and there must be at least d_2 nonzero rows.

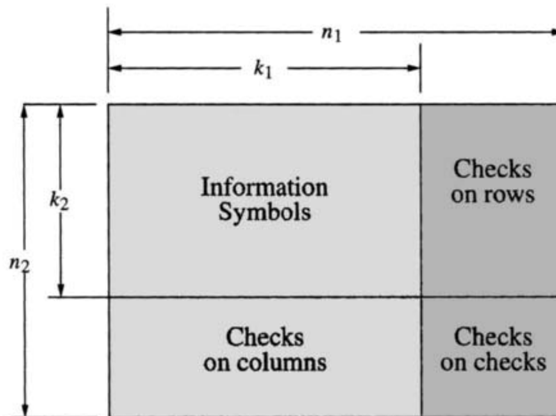


Figure 10.5: The product code $\mathcal{C}_1 \times \mathcal{C}_2$.

To show that there is a codeword of weight $d_1 d_2$, let $\mathbf{c}_1$ be a codeword in $\mathcal{C}_1$ of minimum weight and let $\mathbf{c}_2$ be a codeword in $\mathcal{C}_2$ of minimum weight. Form an array in which all columns corresponding to zeros in $\mathbf{c}_1$ are zeros and all columns corresponding to ones in $\mathbf{c}_1$ are $\mathbf{c}_2$. □

Using the Chinese remainder theorem it is straightforward to show [220, p. 570] that if $\mathcal{C}_1$ and $\mathcal{C}_2$ are cyclic and $(n_1, n_2) = 1$, then $\mathcal{C}_1 \times \mathcal{C}_2$ is also cyclic.

The product code construction can be iterated. For example, the code $\mathcal{C}_1 \times \mathcal{C}_2 \times \mathcal{C}_3$ can be produced. It may be observed that by taking such multiple products, codes with large distance can be obtained. However, the rate of the product code is the product of the rates, so that the product code construction produces codes with low rate.

We do not present a detailed decoding algorithm for product codes here. However, decoding of codes similar to product codes is discussed in chapter 14. Product codes are seen there to be an instance of turbo codes, so a turbo code decoding algorithm is used. However, we discuss here the burst error correction capability of product codes [203, p. 275]. Suppose $\mathcal{C}_1$ has burst error correction capability l_1 and $\mathcal{C}_2$ has burst error correction capability l_2. Suppose that the code is transmitted out of the matrix row by row. At the receiver the data are written back into an array in row by row order. A burst of length $n_1 l_2$ or less can affect no more than $l_2 + 1$ consecutive rows, since when the symbols are arranged into the array, each column is affected by a burst of length at most l_2. By decoding first on the columns, the burst can be corrected, so the burst correction capability of the product code is at least $n_1 l_2$. Similarly, it can be shown that bursts of length at least $n_2 l_1$ can be corrected, so the overall burst error correction capability of the code is $\max(n_1 l_2, n_2 l_1)$.

10.5 Reed-Solomon Codes

Reed-Solomon codes and other codes based on larger-than-binary fields have some intrinsic ability to correct bursts of binary errors. For a code over a field $GF(2^m)$, each coded symbol can be envisioned as a sequence of m bits. Under this interpretation, a (n, k) Reed-Solomon code over $GF(2^m)$ is a binary (mn, mk) code.

The RS code is capable of correcting up to t *symbols* of error. It does not matter that a

single symbol might have multiple bits in error — it still is a single symbol error from the perspective of the RS decoder. A single symbol might have up to m bits in error. Under the best of circumstances then, when all the errors affect adjacent bits of a symbol, a RS code may correct up to mt bits in error. This means that RS codes are naturally effective for transmitting over bursty binary channels: since bursts of errors tend to cluster together, there may be several binary errors contributing to a single erroneous symbol. As an example [203, p. 278], a burst of length $3m + 1$ cannot affect more than 4 symbols, so a RS code capable of correcting 4 errors can correct any burst of length $3m + 1$. Or any burst of length $m + 1$ cannot affect more than two bytes, so the 4-error correcting code could correct up to two bursts of length $m + 1$. In general, a t-error correcting RS code over $GF(2^m)$ can correct any combination of

$$\frac{t}{1 + \lfloor l + m - 2)/m \rfloor}$$

or fewer bursts of length l, or correcting a single burst up to length $(t - 1)m + 1$. And, naturally, it also corrects any combination of t or fewer random errors.

10.6 Concatenated Codes

Concatenated codes were proposed by Forney [87] as a means of obtaining long codes (as required by the Shannon channel coding theorem for capacity-approaching performance) with modest decoding complexity. The basic concatenation coding scheme is shown in Figure 10.6. The inner code is conventionally a binary code. The outer code is typically a (n_2, k_2) Reed-Solomon code over $GF(2^k)$. The outer code uses k_2 k-tuples of bits from the inner code as the message sequence. In the encoding, the outer code takes kk_2 bits divided into k-tuples which are employed as k_2 symbols in $GF(2^k)$ and encodes them as a Reed-Solomon codeword $(c_0, c_1, \ldots, c_{n_2})$. These symbols, now envisioned as k-tuples of binary numbers, are encoded by the inner encoder to produce a binary sequence transmitted over the channel.

The inner code is frequently a convolutional code. The purpose of the inner code is to improve the quality of the "superchannel" (consisting of the inner encoder, the channel, and the inner decoder) that the outer RS code sees so that the RS code can be used very effectively. When the Viterbi decoder (the inner decoder) makes a decoding error, it typically involves a few consecutive stages of the decoding trellis, which results in a short burst of errors. The bursts of bit errors which tend to be produced by the inner decoder are handled by the RS decoder with its inherent burst-error correction capability.

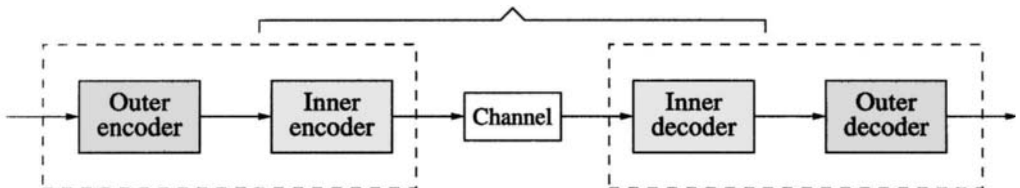


Figure 10.6: A concatenated code.

Example 10.2 [373, p. 432] Figure 10.7 shows the block diagram of a concatenated coding system employed by some NASA deep-space missions. The outer code is a (255, 223) Reed-Solomon code followed by a block interleaver. The inner code is a rate $1/2$ convolutional code, where the generator polynomials are

$$\text{Rate 1/2: } d_{\text{free}} = 10 \quad g_0(x) = 1 + x + x^3 + x^4 + x^6 \qquad g_1(x) = 1 + x^3 + x^4 + x^5 + x^6.$$

The RS code is capable of correcting up to 16 8-bit symbols. The d_{free} path through the trellis traverses 7 branches, so error bursts most frequently have length seven, which in the best case can be trapped by a single RS code symbol.

To provide for the possibility of decoding bursts exceeding $16 \times 8 = 128$ bits, a symbol interleaver is placed between the RS encoder and the convolutional encoder. Since it is a symbol interleaver, burst errors which occupy a single byte are still clustered together. But bursts crossing several bytes are randomized. Block interleavers holding from 2 to 8 Reed-Solomon codewords have been employed. By simulation studies [133], it is shown that to achieve a bit error rate of 10^{-5}, with interleavers of sizes of 2, 4, and 8, respectively, an E_b/N_0 of 2.6 dB, 2.45 dB, and 2.35 dB, respectively, are required. Uncoded BPSK performance would require 9.6 dB; using only the rate $1/2$ convolutional code would require 5.1 dB, so the concatenated system provides approximately 2.5 dB of gain compared to the convolutional code alone. □

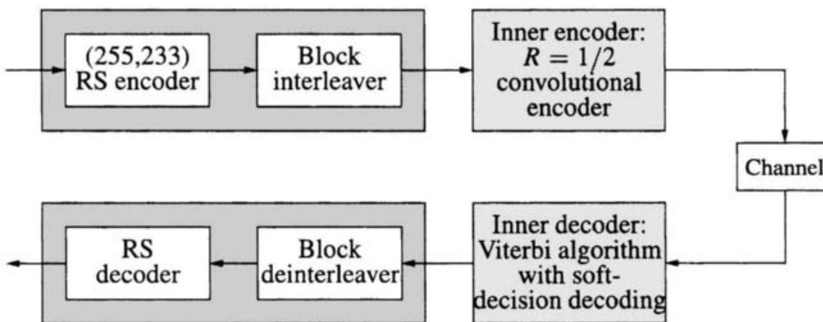


Figure 10.7: Deep-space concatenated coding system.

10.7 Fire Codes

10.7.1 Fire Code Definition

Fire codes, named after their inventor [85], are binary cyclic codes designed specifically to be able to correct a single burst of errors. They are designed as follows. Let $p(x)$ be an irreducible polynomial of degree m over $GF(2)$. Let ρ be the smallest integer such that $p(x)$ divides $x^\rho + 1$. ρ is called the *period* of $p(x)$. Let l be a positive integer such that $l \le m$ and $\rho \nmid 2l - 1$. Let $g(x)$ be the generator polynomial defined by

$$g(x) = (x^{2l-1} + 1)p(x). \tag{10.1}$$

Observe that the factors $p(x)$ and $x^{2l-1} + 1$ are relatively prime. The length n of the code is the least common multiple of $2l - 1$ and the period:

$$n = \text{LCM}(2l - 1, \rho)$$

and the dimension of the code is $k = n - m - 2l + 1$.

Example 10.3 Let $p(x) = 1 + x + x^4$. This is a primitive polynomial, so $\rho = 2^4 - 1 = 15$. Let $l = 4$ and note that $2l - 1 = 7$ is not divisible by 15. The Fire code has generator

$$g(x) = (x^7 + 1)(1 + x + x^4) = 1 + x + x^4 + x^7 + x^8 + x^{11}$$

with length and dimension

$$n = \mathrm{LCM}(7, 15) = 105 \quad \text{and} \quad k = 94.$$

□

The burst error correction capabilities of the Fire code are established by the following theorem.

Theorem 10.2 *The Fire code is capable of correcting any burst up to length l.*

Proof [203, p. 262] We will show that bursts of different lengths reside in different cosets, so they can be employed as coset leaders and form correctable error patterns. Let $a(x)$ and $b(x)$ be polynomials of degree $l_1 - 1$ and $l_2 - 1$, representing bursts of length l_1 and l_2, respectively,

$$a(x) = 1 + a_1 x + a_2 x^2 + \cdots + a_{l_1-2} x^{l_1-2} + x^{l_1-1}$$
$$b(x) = 1 + b_1 x + b_2 x^2 + \cdots + b_{l_2-2} x^{l_2-2} + x^{l_2-1},$$

with $l_1 \le l$ and $l_2 \le l$. Since a burst error can occur anywhere within the length of the code, we represent bursts as $x^i a(x)$ and $x^j b(x)$, where i and j are less than n and represent the starting position of the burst.

Suppose (contrary to the theorem) that $x^i a(x)$ and $x^j b(x)$ are in the same coset of the code. Then the polynomial

$$v(x) = x^i a(x) + x^j b(x)$$

must be a code polynomial in the code. We show that this cannot occur. Without loss of generality, take $i \le j$. By the division algorithm, dividing $j - i$ by $2l - 1$ we obtain

$$j - i = q(2l - 1) + b \tag{10.2}$$

for some quotient q and remainder b, with $0 \le b < 2l - 1$. Using this, we can write

$$v(x) = x^i(a(x) + x^b b(x)) + x^{i+b} b(x)(x^{q(2l-1)} + 1). \tag{10.3}$$

Since (by our contrary assumption) $v(x)$ is a codeword, it must be divisible by $g(x)$ and, since the factors of $g(x)$ in (10.1) are relatively prime, $v(x)$ must be divisible by $x^{2l-1} + 1$. Since $x^{q(2l-1)} + 1$ is divisible by $x^{2l-1} + 1$, it follows that $a(x) + x^b b(x)$ is either divisible by $x^{2l-1} + 1$ or is 0. Let us write

$$a(x) + x^b b(x) = d(x)(x^{2l-1} + 1) \tag{10.4}$$

for some quotient polynomial $d(x)$. Let δ be the degree of $d(x)$. The degree of $d(x)(x^{2l-1} + 1)$ is $\delta + 2l - 1$. The degree of $a(x)$ is $l_1 - 1 < 2l - 1$, so the degree of $a(x) + x^b b(x)$ must be established by the degree of $x^b b(x)$. That is, we must have

$$b + l_2 - 1 = 2l - 1 + \delta. \tag{10.5}$$

Since $l_1 \le l$ and $l_2 \le l$, subtracting l_2 from both sides of (10.5) we obtain

$$b \ge l_1 + \delta.$$

From this inequality we trivially observe that

$$b > l_1 - 1 \quad \text{and} \quad b > d.$$

Writing out $a(x) + x^b b(x)$ we have

$$a(x) + x^b b(x) = 1 + a_1 x + a_2 x^2 + \cdots + a_{l_1-2} x^{l_1-2} + x^{l_1-1}$$
$$+ x^b (1 + b_1 x + b_2 x^2 + \cdots + b_{l_2-2} x^{l_2-2} + x^{l_2-1})$$

so that x^b is one of the terms in $a(x) + x^b b(x)$. On the other hand, since $\delta < b < 2l - 1$, the expression $d(x)(x^{2l-1} + 1)$ from (10.4) does not have the term x^b, contradicting the factorization of (10.4). We must therefore have $d(x) = 0$ and $a(x) + x^b b(x) = 0$. In order to cancel the constant terms in each polynomial we must have $b = 0$, so we conclude that

$$a(x) = b(x).$$

Since b must be 0, (10.2) gives

$$j - i = q(2l - 1). \tag{10.6}$$

Substituting this into (10.3) we obtain

$$v(x) = x^i b(x)(x^{j-i} + 1).$$

Now the degree $b(x)$ is $l_2 - 1 < l$, so $\deg(p(x)) < m = \deg(p(x))$. But since $p(x)$ is irreducible, $b(x)$ and $p(x)$ must be relatively prime. Therefore, since $v(x)$ is (assumed to be) a code polynomial, $x^{j-i} + 1$ must be divisible by $p(x)$ (since it cannot be divisible by $x^{2l-1} + 1$). Therefore, $j - i$ must be a multiple of ρ. By (10.6), $j - i$ must also be multiple of $2l - 1$. So $j - i$ must be a multiple of the least common multiple of $2l - 1$ and m. But this least common multiple is n. We now reach the contradiction which leads to the final conclusion: $j - i$ cannot be a multiple of n, since j and i are both less than n.

We conclude, therefore, that $v(x)$ is not a codeword, so the bursts $x^i a(x)$ and $x^j b(x)$ are in different cosets. Hence they are correctable error patterns. □

10.7.2 Decoding Fire Codes: Error Trapping Decoding

There are several decoding algorithms which have been developed for Fire codes. We present here the **error trapping decoder**. Error trapping decoding is a method which works for many different cyclic codes, but is particularly suited to the structure of Fire codes.

Let $r(x) = c(x) + e(x)$ be a received polynomial. Let us recall that for a cyclic code, the syndrome may be computed by dividing $r(x)$ by $g(x)$, $e(x) = q(x)g(x) + s(x)$. The syndrome is a polynomial of degree up to $n - k - 1$,

$$s(x) = s_0 + s_1 x + \cdots + s_{n-k-1} x^{n-k-1}.$$

Also recall that if $r(x)$ is cyclically shifted i times to produce $r^{(i)}(x)$, the syndrome $s^{(i)}(x)$ may be obtained either by dividing $r^{(i)}(x)$ by $g(x)$, or by dividing $x^i s(x)$ by $g(x)$. Suppose that an l burst-error correcting code is employed and that the errors occur in a burst confined to the l digits,

$$e(x) = e_{n-k-l} x^{n-k-l} + e_{n-k-l+1} x^{n-k-l+1} + \cdots + e_{n-k-1} x^{n-k-1}.$$

Then the syndrome digits $s_{n-k-l}, s_{n-k-l+1}, \ldots, s_{n-k-1}$ match the error values and the syndrome digits $s_0, s_1, \ldots, s_{n-k-l-1}$ are zeros.

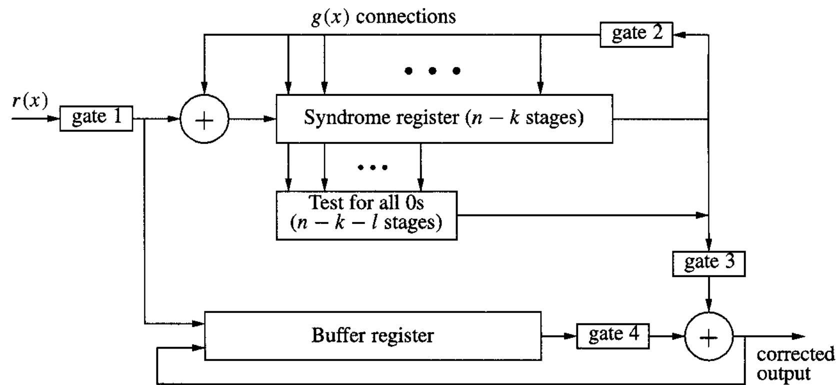


Figure 10.8: Error trapping decoder for burst-error correcting codes [203, p. 260].

If the errors occur in a burst of l consecutive positions at some other location, then after some number i of cyclic shifts, the errors are shifted to the positions $x^{n-k-l}, x^{n-k-l+1}, \dots, x^{n-k-1}$ of $r^{(i)}(x)$. Then the corresponding syndrome $s^{(i)}(x)$ of $r^{(i)}(x)$ matches the errors at positions $x^{n-k-l}, x^{n-k-l+1}, \dots, x^{n-k-1}$ of $r^{(i)}(x)$ and the digits at positions $x_0, x^1, \dots, x^{n-k-l-1}$ are zeros. This fact allows us to "trap" the errors: when the condition of zeros is detected among the lower syndrome digits, we conclude that the shifted errors are trapped in the other digits of the syndrome register.

An error trapping decoder is diagrammed in Figure 10.8. The operation is as follows: [203]

1. With gate 1 and gate 2 open, the received vector $r(x)$ is shifted into the syndrome register, where the division by $g(x)$ takes place by virtue of the feedback connections, so that when $r(x)$ has been shifted in, the syndrome register contains $s(x)$. $r(x)$ is also simultaneously shifted into a buffer register.
2. Successive shifts of the syndrome register occur with gate 2 still open. When the left $n-k-l$ memory elements contain only zeros, the right l stages are deemed to have "trapped" the burst error pattern and error correction begins. The exact correction actions depends upon how many shifts were necessary.
    - If the $n-k-l$ left stages of the syndrome register are all zero after the ith shift for $0 \leq i \leq n-k-l$, then the errors in $e(x)$ are confined only to the parity check positions of $r(x)$, so that the message bits are error free. There is no need to correct the parity bits. In this case, gate 4 is open, and the buffer register is simply shifted out. If, for this range of shifts the $n-k-l$ left stages are never zero, then the error burst is not confined to the parity check positions of $r(x)$.
    - If the $n-k-l$ left stages of the syndrome register are all zero after the $(n-k-l+i)$th shift, for $1 \leq i \leq l$, then the error burst is confined to positions $x^{n-i}, \dots, x^{n-1}, x^0, \dots, x^{l-i-1}$ or $r(x)$. (This burst is contiguous in a cyclic sense.) In this case, $l-i$ right digits of the syndrome buffer register match the errors at the locations $x^0, x^1, \dots, x^{l-i-1}$ of $r(x)$, which are parity check positions and the next i stages of the syndrome register match the errors at locations $x^{n-i}, \dots, x^{n-2}, x^{n-1}$, which are message locations. The syndrome

register is shifted $l - i$ times with gate 2 closed (no feedback) so that the errors align with the message digits in the buffer register. Then gate 3 and gate 4 are opened and the message bits are shifted out of the buffer register, being corrected by the error bits shifted out of the syndrome register.

- If the $n-k-l$ left stages of the syndrome register are never all zeros by the time the syndrome register has been shifted $n-k$ times, the bits are shifted out of the buffer register with gate 4 open while the syndrome register is simultaneously shifted with gate 2 open. In the event that the $n-k-l$ left stages of the syndrome register become equal to all zeros, the digits in the l right stages of the syndrome register match the errors of the next l message bits. Gate 3 is then opened and the message bits are corrected as they are shifted out of the buffer register.
- If the $n-k-l$ left stages of the syndrome register are never all zeros by the time the k message bits have been shifted out of the buffer, an uncorrectable error burst has been detected.

10.8 Exercises

10.1 Let an (n, k) cyclic code with generator $g(x)$ be interleaved by writing its codewords into an $M \times n$ array, then reading out the columns. The resulting code is an (Mn, kn) code. Show that this code is cyclic with generator $g(x^M)$.

10.2 Let G_1 and G_2 be generator matrices for C_1 and C_2, respectively. Show that $G_1 \otimes G_2$ is a generator matrix for $C_1 \times C_2$, where $\otimes$ is the Kronecker product introduced in chapter 8.

10.3 Let C_1 and C_2 be cyclic codes of length n_1 and n_2, respectively, with $(n_1, n_2) = 1$. Form the product code $C_1 \times C_2$. In this problem you will argue that $C_1 \times C_2$ is also cyclic. Denote the codeword represented by the matrix

$$\mathbf{c} = \begin{bmatrix} c_{00} & c_{01} & \cdots & c_{0\,n_2-1} \\ c_{10} & c_{11} & \cdots & c_{1\,n_2-1} \\ \vdots & & & \\ c_{n_1-1\,0} & c_{n_1-1\,1} & \cdots & c_{n_1-1\,n_2-1} \end{bmatrix}$$

by the polynomial

$$c(x, y) = \sum_{i=0}^{n_1-1} \sum_{j=0}^{n_2-1} c_{ij} x^i y^j,$$

where $c(x, y) \in \mathbb{F}[x]/(x^{n_1} - 1, y^{n_2} - 1)$. That is, in the ring where $x^{n_1} = 1$ and $y^{n_2} = 1$ (since the codes are cyclic). Thus $xc(x, y)$ and $yc(x, y)$ represent cyclic shifts of the rows and columns, respectively, of $\mathbf{c}$.

(a) Show that there exists a function $I(i, j)$ such that for each pair (i, j) with $0 \le i < n_1$ and $0 \le j < n_2$, $I(i, j)$ is a unique integer in the range $0 \le I(i, j) < n_1 n_2$, such that

$$I(i, j) \equiv i \pmod{n_1}$$
$$I(i, j) \equiv j \pmod{n_2}.$$

(b) Using $I(i, j)$, rewrite $c(x, y)$ in terms of a single variable $z = xy$ by replacing each $x^i y^j$ by $z^{I(i,j)}$ to obtain the representation $d(z)$.

(c) Show that the set of code polynomials $d(z)$ so obtained is cyclic.

10.4 For each of the following (M, D) pairs, draw the cross interleaver and the corresponding deinterleaver. Also, for the sequence $x_0, x_1, x_2, \ldots,$, determine the interleaved output sequence.

(a) $(M, D) = (2, 1)$

(b) $(M, D) = (2, 2)$

(c) $(M, D) = (3, 2)$

(d) $(M, D) = (4, 2)$

10.5 [203] Find a generator polynomial of a Fire code capable of correcting any single error burst of length 4 or less. What is the length of the code? Devise an error trapping decoder for this code.

10.9 References

The topic of burst-error correcting codes is much more fully developed in [203] where, in addition to the cyclic codes introduced here, several other codes are presented which were found by computer search. Fire codes were introduced in [85]. The cross interleaver is examined in [282]. Error trapping decoding is also fully developed in [203]; it was originally developed in [184, 240, 241, 298, 185]. The compact disc system is described in [158]. A thorough summary is provided in [159]. Product codes are discussed in [220, ch. 18]. Application to burst error correction is described in [203]. Cyclic product codes are explored in [39, 202]. The interpretation of Reed-Solomon codes as binary (mn, mk) codes is discussed in [220]. Decoders which attempt binary-level decoding of Reed-Solomon codes are in [243] and references therein.

Chapter 11

Soft-Decision Decoding Algorithms

11.1 Introduction and General Notation

Most of the decoding methods described to this point in the book have been based on discrete field values, usually *bits* obtained by quantizing the output of the matched filter. However, the actual value of the matched filter output might be used, instead of just its quantization, to determine the reliability of the bit decision. For example, in BPSK modulation if the matched filter output is very near to zero, then any bit decision made based on only that output would have low reliability. A decoding algorithm which takes into account reliability information or uses probabilistic or likelihood values rather than quantized data is called a *soft-decision* decoding algorithm. Decoding which uses only the (quantized) received bit values and not their reliabilities is referred to as hard-decision decoding. As a general rule of thumb, soft-decision decoding can provide as much as 3 dB of gain over hard-decision decoding. In this short chapter, we introduce some of the most commonly-used historical methods for soft-decision decoding, particularly for binary codes transmitted using BPSK modulation over the AWGN channel. Some modern soft-decision decoding techniques are discussed in the context of turbo codes (chapter 14) and LDPC codes (chapter 15).

Some clarification in the terminology is needed. The algorithms discussed in the chapter actually provide *hard* output decisions. That is, the decoded values are provided without any reliability information. However, they rely on "soft" input decisions — matched filter outputs or reliabilities. They should thus be called soft-input hard-output algorithms. A soft-output decoder would provide decoded values accompanied by an associated reliability measure, or a probability distribution for the decoded bits. Such decoders are called soft-input, soft-output decoders. The turbo and LDPC decoders provide this capability.

Let C be a code and let a codeword $\mathbf{c} = (c_0, c_1, \ldots, c_{n-1}) \in C$ be modulated as the vector

$$\tilde{\mathbf{c}} = (\tilde{c}_0, \tilde{c}_1, \ldots, \tilde{c}_{n-1})$$

(assuming for convenience a one-dimensional signal space; modifications for two- or higher-dimensional signal spaces are straightforward). We will denote the operation of modulation — mapping into the signal space for transmission — by M, so that we can write

$$\tilde{\mathbf{c}} = \mathsf{M}(\mathbf{c}).$$

The modulated signal $\tilde{\mathbf{c}}$ is passed through a memoryless channel to form the received vector $\mathbf{r} = (r_0, r_1, \ldots, r_{n-1})$. For example, for an AWGN

$$r_i = \tilde{c}_i + n_i,$$

where $n_i \sim \mathcal{N}(0, \sigma^2)$, with $\sigma^2 = N_0/2$.

The operation of "slicing" the received signal into signal constellation values, the detection problem, can be thought of as an "inverse" modulation. We denote the "sliced" values

as v_i. Thus

$$v_i = \mathsf{M}^{-1}(r_i)$$

or

$$\mathbf{v} = \mathsf{M}^{-1}(\mathbf{r}).$$

If the r_i values are sliced into discrete detected values v_i and only this information is used by the decoder, then hard-decision decoding occurs.

Example 11.1 For example, if BPSK modulation is used, then $\tilde{c}_i = \mathsf{M}(c_i) = \sqrt{E_c}(2c_i - 1)$ is the modulated signal point. The received values are sliced into detected bits by

$$v_i = \mathsf{M}^{-1}(r_i) = \begin{cases} 0 & r_i < 0 \\ 1 & r_i \geq 0 \end{cases} \in \{0, 1\}.$$

In this case, as described in Section 1.5.7, there is effectively a BSC model between transmitter and receiver. Figure 11.1 illustrates the signal labels.

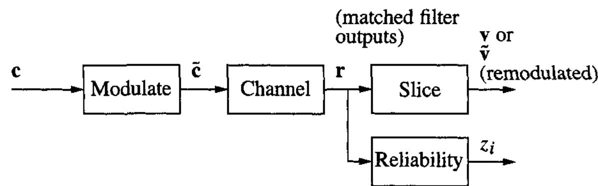
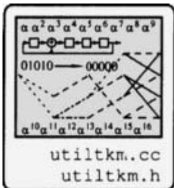


Figure 11.1: Signal labels for soft-decision decoding.

□

It is possible to associate with each sliced value v_i a *reliability* z_i, which indicates the quality of the decision. The reliabilities are ordered such that $z_i > z_j$ if the ith symbol is more reliable — capable of producing better decisions — than the jth symbol. If the channel is AWGN, then

$$z_i = |r_i| \in \mathbb{R}$$



can be used as the reliability measure, since the absolute log likelihood ratio

$$\left| \log \frac{p(r_i|c_i = 1)}{p(r_i|c_i = 0)} \right| \tag{11.1}$$

is proportional to $|r_i|$. Associated with each channel is a distance measure. For the BSC, the appropriate distance measure is the Hamming distance,

$$d_H(\mathbf{v}, \mathbf{c}) = \sum_{i=0}^{n-1} [v_i \neq c_i].$$

However, for soft-decision decoding over the AWGN, the Euclidean metric between the (soft) received vector and the transmitted codeword

$$d_E(\mathbf{r}, \tilde{\mathbf{c}}) = d_E(\mathbf{r}, \mathsf{M}(\mathbf{c})) = \|\mathbf{r} - \tilde{\mathbf{c}}\|^2.$$

is more appropriate. (See Section 1.5 for further discussion.)

A key idea in soft-decision decoding is *sorting* the symbols in order of reliability. When the symbols are sorted in order of decreasing reliability, then the first sorted symbols are more likely to be correct than the last sorted symbols. The set of symbols which have the highest reliability (appearing first in the sorted list) are referred to as being in the most reliable positions and the set of symbols appearing at the end of the sorted list are said to be in the least reliable positions.

11.2 Generalized Minimum Distance Decoding

Recall (see Section 3.8) that an erasure-and-error decoder is capable of correcting twice as many erasures as errors. That is, a code with minimum Hamming distance $d_{\min}$ can simultaneously correct e errors and f erasures provided that $2e + f \leq d_{\min} - 1$. The generalized minimum distance (GMD) decoder devised by Forney [88] makes use of this fact, deliberately erasing symbols which have the least reliability, then correcting them using an erasure-and-error decoder. The GMD decoding considers all possible patterns of up to $f \leq d_{\min} - 1$ erasures in the least reliable positions. The decoder operates as follows.

Algorithm 11.1 Generalized Minimum Distance (GMD) Decoding

Initialize: For the (soft) received sequence $\mathbf{r} = (r_0, r_1, \ldots, r_{n-1})$, form
the hard-decision vector $\mathbf{v} = (v_0, v_1, \ldots, v_{n-1})$ and the reliabilities $z_i = |r_i|$.
Sort the reliabilities to find the $d_{\min} - 1$ least reliable positions
if $d_{\min}$ is even:
  for $j = 1$ to $d_{\min} - 1$ by 2
    erase the j least reliable symbols in $\mathbf{v}$ to form a modified vector $\hat{\mathbf{v}}$
    Decode and Select the best codeword
else if $d_{\min}$ is odd:
  for $j = 0$ to $d_{\min} - 1$ by 2
    erase the j least reliable symbols in $\mathbf{v}$ to form a modified vector $\hat{\mathbf{v}}$
    Decode and Select the best codeword
end if

Decode and Select the best codeword
  Decode $\hat{\mathbf{v}}$ using an erasures-and-errors decoding algorithm to obtain a codeword $\mathbf{c}$.
  Compute the soft-decision (e.g., Euclidean) distance between $\mathsf{M}(\mathbf{c})$ and $\mathbf{r}$, $d_E(\mathbf{r}, \mathsf{M}(\mathbf{c}))$,
  and select the codeword with the best distance.

As discussed in Exercise 2, the correlation discrepancy λ can be computed instead of the distance.

Since hard-decision values are actually used in the erasures-and-errors decoding algorithm, an algebraic decoder can be used, if it exists for the particular code being decoded. (The Chase algorithms described below are also compatible with algebraic decoding algorithms.) Note that for $d_{\min}$ either even or odd, there are only $\lfloor (d_{\min} + 1)/2 \rfloor$ different vectors that must be decoded.

While the GMD algorithm is straightforward and conceptually simple, justifying it in detail will require a bit of work, which follows in the next section.

11.2.1 Distance Measures and Properties

There are two theorems which will establish the correctness of the GMD algorithm, which will require some additional notation. We first generalize the concept of Hamming distance. In defining the Hamming distance between elements of a codeword $\mathbf{c}$ and another vector $\mathbf{v}$, there were essentially two "classes" of outcomes, those where v_i matches c_i, and those where v_i does not match c_i. We generalize this by introducing *J reliability classes* $C_1, C_2, \ldots, C_J$, each of which has associated with it two parameters β_{cj} and β_{ej} such that

$$0 \leq \beta_{cj} \leq \beta_{ej} \leq 1.$$

We also introduce the *weight* of the class, $\alpha_j = \beta_{ej} - \beta_{cj}$. Then β_{ej} is the "cost" when v_i is in class j and $v_i \neq c_i$, and β_{cj} is the "cost" when v_i is in class j and $v_i = c_i$. It is clear that $0 \leq \alpha_j \leq 1$.

We write

$$d_G(v_i, c_i) = \begin{cases} \beta_{ej} & v_i \in C_j \text{ and } v_i \neq c_i \\ \beta_{cj} & v_i \in C_j \text{ and } v_i = c_i. \end{cases}$$

Then the *generalized distance* $d_G(\mathbf{v}, \mathbf{c})$ is defined as

$$d_G(\mathbf{v}, \mathbf{c}) = \sum_{i=0}^{n-1} d_G(v_i, c_i). \tag{11.2}$$

(Note that this is not a true distance, since it is not symmetric in $\mathbf{v}$ and $\mathbf{c}$.) Now let n_{cj} be the number of symbols received correctly (i.e., $v_i = c_i$) and put into class C_j. Let n_{ej} be the number of symbols received incorrectly (so that $v_i \neq c_i$) and put into class C_j. Then (11.2) can be written as

$$d_G(\mathbf{v}, \mathbf{c}) = \sum_{j=1}^{J} \beta_{cj} n_{cj} + \beta_{ej} n_{ej}. \tag{11.3}$$

Example 11.2 For conventional, errors-only decoding, there is only one reliability class, C_1, having $\beta_{c1} = 0$ and $\beta_{e1} = 1$, so $\alpha_1 = 1$. In this case, the generalized distance specializes to the Hamming distance.

Introducing erasures introduces a second reliability class C_2 having $\beta_{c2} = \beta_{e2} = 0$, or $\alpha_2 = 0$. Symbols in the erasure class can be considered to be equally distant from all transmitted symbols. □

A class for which $\alpha_j = 1$ is said to be fully reliable.

The first theorem provides a basis for declaring when the decoding algorithm can declare a correct decoded value.

Theorem 11.1 *[88] For a code having minimum distance $d_{\min}$, if $\mathbf{c}$ is sent and n_{cj} and n_{ej} are such that*

$$\sum_{j=1}^{J} [(1 - \alpha_j) n_{cj} + (1 + \alpha_j) n_{ej}] < d_{\min}, \tag{11.4}$$

then

$$d_G(\mathbf{v}, \mathbf{c}) < d_G(\mathbf{v}, \mathbf{c}')$$

for all codewords $\mathbf{c}' \neq \mathbf{c}$. That is, $\mathbf{v}$ is closer to $\mathbf{c}$ in the d_G measure than to all other codewords.

Note that for errors-only decoding, this theorem specializes as follows: if $2n_{e1} < d_{\min}$, then $d_H(\mathbf{r}, \mathbf{c}) < d_H(\mathbf{r}, \mathbf{c}')$, which is the familiar statement regarding the decoding capability of a code, with n_{e1} the number of correctable errors. For erasures-and-errors decoding, the theorem specializes as follows: if $2n_{e1} + (n_{c2} + n_{e2}) < d_{\min}$ then $d_G(\mathbf{v}, \mathbf{c}) < d_G(\mathbf{v}, \mathbf{c}')$. Letting $f = n_{c2} + n_{e2}$ be the number of erasures, this recovers the familiar condition for erasures-and-errors decoding.

Proof Let $\mathbf{c}'$ be any codeword not equal to $\mathbf{c}$. Partition the n symbol locations into sets such that

$$i \in \begin{cases} S_0 & \text{if } c_i = c_i' \\ S_{cj} & \text{if } c_i \neq c_i' \text{ and } v_i = c_i \text{ and } v_i \in C_j \\ S_{ej} & \text{if } c_i \neq c_i' \text{ and } v_i \neq c_i \text{ and } v_i \in C_j. \end{cases}$$

Clearly we must have $|S_{cj}| \le n_{cj}$ and $|S_{ej}| \le n_{ej}$.

- For $i \in S_0$, $d_G(v_i, c_i') \ge 0 = d_H(c_i, c_i')$.
- For $i \in S_{cj}$, $d_G(v_i, c_i') = \beta_{ej} = d_H(c_i', c_i) - 1 + \beta_{ej}$
- For $i \in S_{ej}$, $d_G(v_i, c_i') \ge \beta_{cj} = d_H(c_i', c_i) - 1 + \beta_{cj}$.

Summing both sides of these equalities/inequalities over $j = 1, \ldots, J$, we obtain

$$\begin{aligned} d_G(\mathbf{v}, \mathbf{c}') &\ge d_H(\mathbf{c}, \mathbf{c}') - \sum_{i=0}^{n-1}[(1 - \beta_{ej})|S_{cj}| + (1 - \beta_{cj})|S_{ej}|] \\ &\ge d_{\min} - \sum_{i=0}^{n-1}[(1 - \beta_{ej})n_{cj} + (1 - \beta_{cj})n_{ej})]. \end{aligned}$$

If, as stated in the hypothesis of the theorem,

$$d_{\min} > \sum_{j=1}^{J}[(1 - \beta_{ej} + \beta_{cj})n_{cj} + (1 - \beta_{cj} + \beta_{ej})n_{ej}],$$

then

$$d_G(\mathbf{v}, \mathbf{c}') > \sum_{j=1}^{J} n_{cj}\beta_{cj} + n_{ej}\beta_j = d_G(\mathbf{v}, \mathbf{c}).$$

□

This theorem allows us to draw the following conclusions:

- If generalized distance d_G is used as a decoding criterion, then no decoding error will be made when n_{cj} and n_{ej} are such that (11.4) is satisfied.

  Let us say that $\mathbf{c}$ is *within the minimum distance* of $\mathbf{v}$ if (11.4) is satisfied.
- The theorem also says that there can be at most one codeword within the minimum distance of any received word $\mathbf{v}$.

Thus, if by some means a codeword $\mathbf{c}$ can be found within the minimum distance of the received word $\mathbf{v}$, then it can be concluded that this is the decoded value.

The next theorem will determine the maximum number of different decoded values necessary and suggest how to obtain them.

Let the classes be ordered according to decreasing reliability (or weight), so that $\alpha_j \geq \alpha_k$ if $j < k$. Let

$$\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \ldots, \alpha_J)$$

be the vector of all class weights. Let

$$R_b = \{0, 1, 2, \ldots, b\} \text{ and } E_b = \{b+1, b+2, b+3, \ldots, J\}.$$

Let

$$\boldsymbol{\alpha}_b = \{\underbrace{1, 1, \ldots, 1}_{b \text{ ones}}, 0, 0, \ldots, 0\}.$$

Theorem 11.2 *[88] Let the weights be ordered according to decreasing reliability. If*

$$\sum_{j=1}^{J}[(1-\alpha_j)n_{cj} + (1+\alpha_j)n_{ej}] < d_{\min},$$

then there is some integer b such that

$$2\sum_{j=1}^{b} n_{ej} + \sum_{i=b+1}^{J}(n_{cj} + n_{ej}) < d_{\min}.$$

Since erased symbols are associated with a class having $\alpha_j = 0$, which would occur last in the ordered reliability list, the import of the theorem is that if there is some codeword such that the inequality (11.4) is satisfied, then there must be some assignment in which (only) the least reliable classes are erased which will enable an erasures-and-errors decoder to succeed in finding that codeword.

Proof Let

$$f(\boldsymbol{\alpha}) = \sum_{j=1}^{J}[(1-\alpha_j)n_{cj} + (1+\alpha_j)n_{ej}].$$

Note that $f(\boldsymbol{\alpha}_b) = 2\sum_{j=1}^{b} n_{ej} + \sum_{j=b+1}^{J}(n_{cj} + n_{ej})$. The proof is by contradiction.

Suppose (contrary to the theorem) that $f(\boldsymbol{\alpha}_b) \geq d_{\min}$. Let

$$\lambda_0 = 1 - \alpha_1 \qquad \lambda_b = \alpha_b - \alpha_{b+1} \qquad 1 \leq b \leq J-1 \qquad \lambda_J = \alpha_J.$$

By the ordering, $0 \leq \lambda_b \leq 1$ for $0 \leq b \leq J$, and $\sum_{b=0}^{J} \lambda_b = 1$.

Now let

$$\boldsymbol{\alpha} = \sum_{b=0}^{J} \lambda_b \boldsymbol{\alpha}.$$

Then

$$f(\boldsymbol{\alpha}) = f\left(\sum_{b=0}^{J} \lambda_b \boldsymbol{\alpha}_b\right) = \sum_{b=0}^{J} \lambda_b f(\boldsymbol{\alpha}_b) \geq d_{\min} \sum_{b=0}^{J} \lambda_b = d_{\min}.$$

Thus if $f(\boldsymbol{\alpha}_b) \geq d$ for all b, then $f(\boldsymbol{\alpha}) \geq d$. But the hypothesis of the theorem is that $f(\boldsymbol{\alpha}) < d$, which is a contradiction. Therefore, there must be at least one b such that $f(\boldsymbol{\alpha}_b) < d$. □

Let us examine conditions under which an erasures-and-errors decoder can succeed. The decoder can succeed if there are apparently no errors and $d_{\min} - 1$ erasures, or one error and $d_{\min} - 3$ erasures, and so forth up to t_0 errors and $d_{\min} - 2t_0 - 1$ erasures, where t_0 is the largest integer such that $2t_0 \leq d_{\min} - 1$. These possibilities are exactly those examined by the statement of the GMD decoder in Algorithm 11.1.

11.3 The Chase Decoding Algorithms

In [46], three other soft-decision decoding algorithms were presented, now referred to as Chase-1, Chase-2, and Chase-3. These algorithms provide varying levels of decoder capability for varying degrees of decoder effort.

In the Chase-1 algorithm, *all* patterns of up to $d_{\min} - 1$ errors are added to the received signal vector $\mathbf{v}$ to form $\mathbf{w} = \mathbf{v} + \mathbf{e}$. Then $\mathbf{w}$ is decoded, if possible, and compared using a soft-decision metric to $\mathbf{r}$. The decoded codeword closest to $\mathbf{r}$ is selected. However, since all patterns of up to $d_{\min} - 1$ errors are used this is very complex for codes of appreciable length or distance, so Chase-1 decoding has attracted very little interest.

In the Chase-2 algorithm, the $\lfloor d_{\min}/2 \rfloor$ least reliable positions are identified from the sorted reliabilities. The set E consisting of all errors in these $\lfloor d_{\min}/2 \rfloor$ least reliable positions is generated. Then the Chase-2 algorithm can be summarized as follows.

Algorithm 11.2 Chase-2 Decoder

Initialize: For the (soft) received sequence $\mathbf{r} = (r_0, r_1, \ldots, r_{n-1})$, form the hard-decision vector $\mathbf{v} = (v_0, v_1, \ldots, v_{n-1})$ and the reliabilities $z_i = |r_i|$.
Identify the $\lfloor d_{\min}/2 \rfloor$ least reliable positions
Generate the set E of error vectors (one at a time, in practice)
for each $\mathbf{e} \in E$
  Decode $\mathbf{v} + \mathbf{e}$ using an errors-only decoder to produce the codeword $\mathbf{c}$
  Compute $d_E(\mathbf{r}, \mathsf{M}(\mathbf{c}))$ and select the candidate codeword with the best metric.
end for

The Chase-3 decoder has lower complexity than the Chase-2 algorithm, and is similar to the GMD.

Algorithm 11.3 Chase-3 Decoder

Initialize: For the (soft) received sequence $\mathbf{r} = (r_0, r_1, \ldots, r_{n-1})$, form the hard-decision vector $\mathbf{v} = (v_0, v_1, \ldots, v_{n-1})$ and the reliabilities $z_i = |r_i|$.
Sort the reliabilities to find the $\lfloor d_{\min} - 1 \rfloor$ least reliable positions

Generate a list of at most $\lfloor d_{\min}/2 + 1 \rfloor$ sequences by modifying $\mathbf{v}$:
If $d_{\min}$ is even, modify $\mathbf{v}$ by complementing no symbols, then the least reliable symbol, then the three least reliable symbols, ..., the $d_{\min-1}$ least reliable symbols.
If $d_{\min}$ is odd, modify $\mathbf{v}$ by complementing no symbols, then the two least reliable symbols, then the four least reliable symbols, ..., the $d_{\min-1}$ least reliable symbols.
Decode each modified $\mathbf{v}$ into a codeword $\mathbf{c}$ using an error-only decoder.
Compute $d_E(\mathbf{r}, \mathsf{M}(\mathbf{c}))$ and select the codeword that is closest.

11.4 Halting the Search: An Optimality Condition

The soft-decision decoding algorithms presented so far require searching over an entire list of candidate codewords to select the best. It is of interest to know if a given codeword is the best that is going to be found, without having to complete the search. In this section we discuss an optimality condition appropriate for binary codes transmitted using BPSK which can be tested which establishes exactly this condition.

Let $\mathbf{r}$ be the received vector, with hard-decision values $\mathbf{v}$. Let $\mathbf{c} = (c_0, c_1, \ldots, c_{n-1})$ be a binary codeword in C and let $\tilde{\mathbf{c}} = (\tilde{c}_0, \tilde{c}_1, \ldots, \tilde{c}_{n-1})$ be its bipolar representation (i.e.., for BPSK modulation), with $\tilde{c}_i = 2c_i - 1$. In this section, whenever we indicate a codeword $\mathbf{c}$

its corresponding bipolar representation $\tilde{\mathbf{c}}$ is also implied. We define the index sets $D_0(\mathbf{c})$ and $D_1(\mathbf{c})$ with respect to a codevector $\mathbf{c}$ by

$$D_0(\mathbf{c}) = \{i : 0 \leq i < n \text{ and } c_i = v_i\} \qquad \text{and} \qquad D_1(\mathbf{c}) = \{i : 0 \leq i < n \text{ and } c_i \neq v_i\}.$$

Let $n(\mathbf{c}) = |D_1(\mathbf{c})|$; this is the Hamming distance between the transmitted codeword $\mathbf{c}$ and the hard-decision vector $\mathbf{v}$.

It is clear from the definitions that $r_i\tilde{c}_i < 0$ if and only if $v_i \neq c_i$. In Exercise 2, the correlation discrepancy $\lambda(\mathbf{r}, \tilde{\mathbf{c}})$ is defined as

$$\lambda(\mathbf{r}, \tilde{\mathbf{c}}) = \sum_{i: r_i \tilde{\mathbf{c}}_i < 0} |r_i|.$$

Based on the index sets just defined, λ can be expressed as

$$\lambda(\mathbf{r}, \tilde{\mathbf{c}}) = \sum_{i \in D_1(\mathbf{c})} |r_i|.$$

As discussed in Exercise 2, ML decoding seeks a codeword $\mathbf{c}$ such that $\lambda(\mathbf{r}, \tilde{\mathbf{c}})$ is minimized: a ML codeword $\mathbf{c}^*$ is one satisfying

$$\lambda(\mathbf{r}, \tilde{\mathbf{c}}^*) \leq \min_{\mathbf{c} \in \mathcal{C}, \mathbf{c} \neq \mathbf{c}^*} \lambda(\mathbf{r}, \mathbf{c}).$$

While determining the minimum requires search over all $\mathbf{c} \in \mathcal{C}$, we will establish a tight bound λ^* on $\min_{\mathbf{c} \in \mathcal{C}, \mathbf{c} \neq \mathbf{c}^*}(\lambda(\mathbf{r}, \mathbf{c}))$ such that $\lambda(\mathbf{r}, \mathbf{c}) \leq \lambda^*$ represents a sufficient condition for the optimality of the candidate codeword in the list of candidate codewords generated by a reliability-based decoding algorithm such as those presented above.

Let the indices in $D_0(\mathbf{c})$ be written as

$$D_0(\mathbf{c}) = \{l_1, l_2, \ldots, l_{n-n(\mathbf{c})}\},$$

where the indices are ordered such that for $i < j$, the reliabilities are ordered as

$$|r_{l_i}| < |r_{l_j}|.$$

Let $D_0^{(j)}(\mathbf{c})$ denote the first j indices in the ordered set,

$$D_0^{(j)}(\mathbf{c}) = \{l_1, l_2, \ldots, l_j\}. \tag{11.5}$$

For $j \leq 0$, define $D_0^{(j)}(\mathbf{c}) = \emptyset$, and for $j \geq n - n(\mathbf{c})$, define $D_0^{(j)}(\mathbf{c}) = D_0(\mathbf{c})$.

Let w_i be the ith weight in the weight profile of the code $\mathcal{C}$. That is, $w_0 = 0$, $w_1 = d_{\min}$, and $w_1 < w_2 < \cdots < w_m$ for some m. That is, w_m is the weight of the heaviest codeword in $\mathcal{C}$. For a codeword $\mathbf{c} \in \mathcal{C}$, let

$$q_i = w_i - n(\mathbf{c})$$

and define

$$G(\mathbf{c}, w_j) = \sum_{i \in D_0^{(j)}(\mathbf{c})} |r_i|$$

and

$$R(\mathbf{c}, w_j) = \{\mathbf{c}' \in \mathcal{C} : d_H(\mathbf{c}, \mathbf{c}') < w_j\}.$$

The set $R(\mathbf{c}, w_j)$ is the set of codewords within a distance w_{j-1} of $\mathbf{c}$. When $j = 1$, $R(\mathbf{c}, w_1)$ is simply the codeword $\{\mathbf{c}\}$. When $j = 2$, $R(\mathbf{c}, w_2)$ is the codeword $\mathbf{c}$ and all codewords in $\mathcal{C}$ which are at a distance $d_{\min}$ from $\mathbf{c}$.

With this notation, we are now ready for the theorem.

Theorem 11.3 *[204, p. 404] Let* $\mathbf{r}$ *be the received vector, with corresponding hard-decision vector* $\mathbf{v}$*. For a codeword* $\mathbf{c} \in C$ *and a nonzero weight* w_j *in the weight profile of* C*, if the correlation discrepancy* $\lambda(\mathbf{r}, \mathsf{M}(\mathbf{c}))$ *satisfies*

$$\lambda(\mathbf{r}, \mathsf{M}(\mathbf{c})) \leq G(\mathbf{r}, w_j),$$

then the maximum likelihood codeword $\mathbf{c}_{\text{ML}}$ *for* $\mathbf{r}$ *is contained in* $R(\mathbf{c}, w_j)$*.*

The theorem thus establishes a sufficient condition for $R(\mathbf{v}, w_j)$ to contain the maximum likelihood codeword for $\mathbf{r}$.

Proof Let $\mathbf{c}'$ be a codeword outside $R(\mathbf{c}, w_j)$. Then, by the definition of R, $d_H(\mathbf{c}, \mathbf{c}') \geq w_j$. The theorem is established if we can show that $\lambda(\mathbf{r}, \mathsf{M}(\mathbf{c}')) \geq \lambda(\mathbf{r}, \mathsf{M}(\mathbf{c}))$, since this would show that $\mathbf{c}'$ has lower likelihood than $\mathbf{c}$. Thus no codeword outside $R(\mathbf{r}, w_j)$ is more likely than $\mathbf{c}$, so that the maximum likelihood codeword must be in $R(\mathbf{c}, w_j)$.

Define

$$n_{01} = |D_0(\mathbf{c}) \cap D_1(\mathbf{c}')| \quad \text{and} \quad n_{10} = |D_1(\mathbf{c}) \cap D_0(\mathbf{c}')|.$$

From the definitions of D_0 and D_1, it follows that $d_H(\mathbf{c}, \mathbf{c}') = n_{01} + n_{10}$. As we have just observed, we must also have

$$w_j \leq d_H(\mathbf{c}, \mathbf{c}') = n_{01} + n_{10}. \tag{11.6}$$

It also follows that

$$\begin{aligned} |D_1(\mathbf{c}')| \geq |D_0(\mathbf{c}) \cap D_1(\mathbf{c}')| &\geq w_j - n_{10} \\ &\geq w_j - |D_1(\mathbf{c})| \\ &= w_j - n(\mathbf{c}), \end{aligned} \tag{11.7}$$

where the first inequality is immediate from the nature of intersection, the second from (11.6), and the third from the definition of n_{10} and the intersection. Now we have

$$\begin{aligned} \lambda(\mathbf{r}, \mathsf{M}(\mathbf{c})) = \sum_{i \in D_1(\mathbf{c})} |r_i| &\geq \sum_{i \in D_0(\mathbf{c}) \cap D_1(\mathbf{c}')} |r_i| \quad \text{(since } D_0(\mathbf{c}) \cap D_1(\mathbf{c}') \subset D_1(\mathbf{c}')\text{)} \\ &\geq \sum_{i \in D_0^{(w_j - n(\mathbf{c}))}(\mathbf{c})} |r_i| \quad \text{(by (11.7))} \\ &= G(\mathbf{c}, w_j) \quad \text{(by definition)} \\ &\geq \lambda(\mathbf{c}, \mathbf{c}) \quad \text{(by hypothesis)}. \end{aligned}$$

We thus have $\lambda(\mathbf{r}, \mathbf{c}') \geq \lambda(\mathbf{r}, \mathbf{c})$: no codeword outside $R(\mathbf{r}, w_j)$ is more likely than $\mathbf{c}$. Hence $\mathbf{c}_{\text{ML}}$ must lie in $R(\mathbf{r}, w_j)$. □

For $j = 1$, this theorem says that the condition $\lambda(\mathbf{r}, \mathsf{M}(\mathbf{c})) < G(\mathbf{c}, w_1)$ guarantees that $\mathbf{c}$ is, in fact, the maximum likelihood codeword. This provides a sufficient condition for optimality, which can be used to terminate the GMD or Chase algorithms before exhausting over all codewords in the lists these algorithms produce.

11.5 Ordered Statistic Decoding

The GMD and Chase algorithms make use of the least reliable symbols. By contrast, ordered statistic decoding [104] uses the *most* reliable symbols.

Let C be an (n, k) linear block code with $k \times n$ generator matrix

$$G = \begin{bmatrix} \mathbf{g}_0 & \mathbf{g}_1 & \cdots & \mathbf{g}_{n-1} \end{bmatrix}.$$

Let $\mathbf{r}$ be the received vector of matched filter outputs, with corresponding hard-decoded values $\mathbf{v}$. Let $\mathbf{z}$ denote the corresponding reliability values and let $\bar{\mathbf{z}} = (\bar{z}_0, \bar{z}_1, \ldots, \bar{z}_{n-1})$ denote the sorted reliability values, with $\bar{z}_0 \geq \bar{z}_1 \geq \cdots \geq \bar{z}_{n-1}$. This ordering establishes a permutation mapping π_1, with

$$\bar{\mathbf{z}} = \pi_1(\mathbf{z}).$$

In ordered statistic decoding, the columns of G are reordered by the permutation mapping to form a generator matrix G', where

$$G' = \pi_1(G) = \begin{bmatrix} \mathbf{g}'_0 & \mathbf{g}'_1 & \cdots & \mathbf{g}'_{n-1} \end{bmatrix}.$$

Now a *most reliable basis* for the codeword is obtained as follows. The first k linearly independent columns of G' (associated with the largest reliability values) are found. Then these k columns are used as the first k columns of a new generator matrix G'', in reliability order. The remaining $n - k$ columns of G'' are placed in decreasing reliability order. This ordering establishes another permutation mapping π_2, such that

$$G'' = \pi_2(G') = \pi_2(\pi_1(G)).$$

Applying π_2 to the ordered reliabilities $\bar{\mathbf{z}}$ results in another reliability vector $\tilde{\mathbf{z}}$,

$$\tilde{\mathbf{z}} = \pi_2(\bar{\mathbf{z}}) = \pi_2(\pi_1(\mathbf{z}))$$

satisfying

$$\tilde{z}_1 \geq \tilde{z}_2 \geq \cdots \geq \tilde{z}_k \text{ and } \tilde{z}_{k+1} \geq \tilde{z}_{k+2} \geq \cdots \geq \tilde{z}_n.$$

Now perform elementary row operations on G'' to obtain an equivalent generator matrix $\tilde{G}$ in systematic form,

$$\tilde{G} = \begin{bmatrix} 1 & 0 & \cdots & 0 & p_{1,1} & \cdots & p_{1,n-k} \\ 0 & 1 & \cdots & 0 & p_{2,1} & \cdots & p_{2,n-k} \\ \vdots & & & & & & \\ 0 & 0 & \cdots & 1 & p_{k,1} & \cdots & p_{k,n-k} \end{bmatrix} = \begin{bmatrix} I_k & P \end{bmatrix}.$$

Let $\tilde{C}$ denote the code generated by $\tilde{G}$. The code $\tilde{C}$ is equivalent to C (see Definition 3.5).

The next decoding step is to use the k most reliable elements of $\mathbf{v}$. Let us denote these as $\tilde{\mathbf{v}}_k = [\pi_2(\pi_1(\mathbf{v}))]_{1:k} = (\tilde{v}_1, \tilde{v}_2, \ldots, \tilde{v}_k)$. Since these are the most reliable symbols, these symbols should contain very few errors. Since $\tilde{G}$ is systematically represented, we will take these symbols as the message symbols for a codeword. The corresponding codeword $\tilde{\mathbf{c}}$ in $\tilde{C}$ is obtained by

$$\tilde{\mathbf{c}} = \tilde{\mathbf{v}}_k \tilde{G} \in \tilde{C}.$$

Finally, the codeword $\hat{\mathbf{c}}$ in the original code C can be obtained by unpermuting by both permutations:

$$\hat{\mathbf{c}} = \pi_1^{-1}(\pi_2^{-1}(\tilde{\mathbf{c}})) \in C.$$

This gives a single candidate decoded value. Then $\hat{\mathbf{c}}$ is compared with $\mathbf{r}$ by computing $d_E(\mathbf{r}, \mathsf{M}(\hat{\mathbf{c}}))$ (at least, this is the appropriate distance for the AWGN channel).

Now some additional search is performed. Fix a "search depth parameter" $l \leq k$. For each i in $1 \leq i \leq l$, make all possible changes of i of the k *most* reliable symbols in $\tilde{\mathbf{v}}_k$. Denote the modified vector as $\tilde{\mathbf{v}}'_k$. For each such $\tilde{\mathbf{v}}'_k$, find the corresponding codeword

$$\tilde{\mathbf{c}}' = \tilde{\mathbf{v}}'_k \tilde{G} \in \tilde{\mathcal{C}}$$

and its corresponding codeword $\hat{\mathbf{c}}' = \pi_1^{-1}(\pi_2^{-1}(\tilde{\mathbf{c}}')) \in \mathcal{C}$. Then compute $d_E(\mathbf{r}, \mathsf{M}(\tilde{\mathbf{c}}'))$. Select the codeword $\tilde{\mathbf{c}}'$ with the best metric. The number of codewords to try is $\sum_{i=0}^{l} \binom{k}{i}$.

The ordered statistic decoding algorithm is summarized below.

Algorithm 11.4 Ordered Statistic Decoding

Initialize: For the (soft) received sequence $\mathbf{r} = (r_0, r_1, \ldots, r_{n-1})$, form the hard-decision vector $\mathbf{v} = (v_0, v_1, \ldots, v_{n-1})$ and the reliabilities $z_i = |r_i|$.
Sort the reliabilities by π_1, such that

$$\bar{\mathbf{z}} = \pi_1(\mathbf{z})$$

such that $\bar{z}_0 \geq \bar{z}_1 \geq \cdots \geq \bar{z}_{n-1}$
Order the columns of G to produce $G' = \pi_1(G)$.
Find the first k linearly independent columns of G' and retain the order of the other columns. Let $G'' = \pi_2(G')$ have these first k linearly independent columns.
Reduce G'' by row operations to produce $\tilde{G}$.
Let $\tilde{\mathbf{v}}_k = \pi_2(\pi_1(\mathbf{v}))_{1:k}$.
for $i = 0$ to l
  form all patterns of i errors and add them to the most reliable positions of $\tilde{\mathbf{v}}_k$ to form $\tilde{\mathbf{v}}'_k$
    for each such $\tilde{\mathbf{v}}'_k$, find $\tilde{\mathbf{c}}' = \tilde{\mathbf{v}}'_k \tilde{G}$ and the corresponding $\hat{\mathbf{c}}' = \pi_1^{-1}(\pi_2^{-1}(\tilde{\mathbf{c}}'))$
    for each $\hat{\mathbf{c}}'$, compute $d_E(\mathbf{r}, \mathsf{M}(\hat{\mathbf{c}}))$ and retain the codeword with the smallest distance.

When $l = k$, then the algorithm will compute a maximum likelihood decision, but the decoding complexity is $O(2^k)$. However, since the vector $\tilde{\mathbf{v}}_k$ contains the most reliable symbols, it is likely to have very few errors, so that making a change in only a small number of locations is likely to produce the ML codeword. It is shown in [104] (see also [204, p. 433]) that for most block codes of length up to $n = 128$ and rates $R \geq \frac{1}{2}$ that $i = \lfloor d_{\min}/4 \rfloor$ achieves nearly ML decoding performance.

11.6 Exercises

11.1 Show that for a signal transmitted over an AWGN, the absolute log likelihood ratio (11.1) is proportional to $|r_i|$.

11.2 Let $\tilde{\mathbf{c}}$ is a BPSK signal transmitted through an AWGN to produce a received signal $\mathbf{r}$.

(a) Show that, as far as ML decoding is concerned, $d_E(\mathbf{r}, \tilde{\mathbf{c}})$ is equivalent to $m(\mathbf{r}, \tilde{\mathbf{v}}) = \sum_{i=0}^{n-1} r_i \tilde{v}_i$, where $\tilde{v}_i = \begin{cases} -1 & v_i = 0 \\ 1 & v_i = 1. \end{cases}$

(b) Show that $m(\mathbf{r}, \tilde{\mathbf{v}})$ can be written as

$$m(\mathbf{r}, \tilde{\mathbf{v}}) = \sum_{i=0}^{n-1} |r_i| - 2\lambda(\mathbf{r}, \tilde{\mathbf{v}}),$$

where

$$\lambda(\mathbf{r}, \tilde{\mathbf{v}}) = \sum_{i: r_i \tilde{v}_i < 0} |r_i|.$$

Thus, minimizing $d_E(\mathbf{r}, \tilde{\mathbf{c}})$ is equivalent to minimizing $\lambda(\mathbf{r}, \tilde{\mathbf{v}})$. $\lambda(\mathbf{r}, \tilde{\mathbf{v}})$ is called the *correlation discrepancy.*

11.3 [204, p. 406] In this exercise, a sufficient condition for determining the optimality of a codeword based on two codewords is derived. Let $\mathbf{c}_1, \mathbf{c}_2 \in \mathcal{C}$. Define

$$\delta_1 = w_1 - n(\mathbf{c}_1) \qquad \delta_2 = w_1 - n(\mathbf{c}_2),$$

where $n(\mathbf{c}) = |\{i : 0 \leq i < n \text{ and } c_i \neq v_i\}|$ is the Hamming distance between $\mathbf{c}$ and $\mathbf{v}$, and w_i is the ith weight in the weight profile of the code. Assume that the codewords are ordered such that $\delta_1 \geq \delta_2$. Also define

$$D_{00} = D_0(\mathbf{c}_1) \cap D_0(\mathbf{c}_2) \qquad D_{01} = D_0(\mathbf{c}_1) \cap D_1(\mathbf{c}_2).$$

Let $X^{(q)}$ denote the first q indices of an ordered index set X, as was done in (11.5). Define

$$I(\mathbf{c}_1, \mathbf{c}_2) = \left(D_{00} \cup D_{01}^{\lfloor(\delta_1 - \delta_2)/2\rfloor}\right)^{(\delta_1)}.$$

Also define

$$G(\mathbf{c}_1, w_1, \mathbf{c}_2, w_1) = \sum_{i \in I(\mathbf{c}_1, \mathbf{c}_2)} |r_i|.$$

Let $\mathbf{c}$ be the codeword among $\mathbf{c}_1$, $\mathbf{c}_2$, which has the smaller discrepancy.

Show that: If $\lambda(\mathbf{c}, \mathbf{c}) \leq G(\mathbf{c}_1, w_1, \mathbf{c}_2, w_1)$, then $\mathbf{c}$ is the maximum likelihood codeword for $\mathbf{r}$.

11.4 [204] In GMD decoding, only $\lfloor(d_{\min} + 1)/2\rfloor$ erasure patterns in the $d_{\min} - 1$ least reliable positions are examined. Explain why not all $d_{\min} - 1$ possible erasure patterns are considered.

11.7 References

Generalized minimum distance decoding was presented first in [88]. The statement of the algorithm presented here follows [204]. The Chase decoding algorithms were presented in [46]; our statement of Chase-3 is essentially identical to that in [204]. Generalizations of the Chase algorithm presented in [103] circumscribe Chase-2 and Chase-3, and are capable of achieving bounded distance decoding. An iterative method of soft decision decoding which is capable of finding the ML codeword is proposed in [183, 182]. Our discussion of the optimality criterion very closely follows [204, section 10.3], which in turn is derived from [327]. Chapter 10 of [204], in fact, provides a very substantial discussion of soft-decision decoding, including topics not covered here such as reduced list syndrome decoding [317, 318], priority-first search decoding [139, 140, 73, 16, 348], and majority-logic decoding [224, 192]. Soft-decision decoding is also discussed in this book in chapter 15 for LDPC codes, chapter 12 using the trellis representation of codes, chapter 7 for Reed-Solomon codes, and chapter 14 for turbo codes.

Part III

Codes on Graphs

Chapter 12

Convolutional Codes

12.1 Introduction and Basic Notation

Convolutional codes are linear codes that have additional structure in the generator matrix so that the encoding operation can be viewed as a filtering — or convolution — operation. Convolutional codes are widely used in practice, with several hardware implementations available for encoding and decoding. A convolutional encoder may be viewed as nothing more than a set of digital filters — linear, time-invariant systems — with the code sequence being the interleaved output of the filter outputs. Convolutional codes are often preferred in practice over block codes, because they provide excellent performance when compared with block codes of comparable encode/decode complexity. Furthermore, they were among the earliest codes for which effective soft-decision decoding algorithms were developed.

Whereas block codes take discrete blocks of k symbols and produce therefrom blocks of n symbols that depend only on the k input symbols, convolutional codes are frequently viewed as *stream codes*, in that they often operate on continuous streams of symbols not partitioned into discrete message blocks. However, they are still rate $R = k/n$ codes, accepting k new symbols at each time step and producing n new symbols. The arithmetic can, of course, be carried out over any field, but throughout this chapter and, in fact, in most of the convolutional coding literature, the field $GF(2)$ is employed.

We represent sequences and transfer functions as power series in the variable x.[1] A sequence $\{\ldots, m_{-2}, m_{-1}, m_0, m_1, m_2, \ldots\}$ with elements from a field $\mathbb{F}$ is represented as a formal **Laurent series** $m(x) = \sum_{l=-\infty}^{\infty} m_l x^l$. The set of all Laurent series over $\mathbb{F}$ is a field, which is usually denoted as $\mathbb{F}[[x]]$. Thus, $m(x) \in \mathbb{F}[[x]]$.

For multiple input streams we use a superscript, so $m^{(1)}(x)$ represents the first input stream and $m^{(2)}(x)$ represents the second input stream. For multiple input streams, it is convenient to collect the input streams into a single (row) vector, as in

$$m(x) = \begin{bmatrix} m^{(1)}(x) & m^{(2)}(x) \end{bmatrix} \in \mathbb{F}[[x]]^2.$$

A convolutional encoder is typically represented as sets of digital (binary) filters.

Example 12.1 Figure 12.1 shows an example of a convolutional encoder. (Recall that the D blocks represent 1-bit storage devices, or D flip-flops.) The input stream m_k passes through two filters (sharing memory elements) producing two output streams

$$c_k^{(1)} = m_k + m_{k-2} \quad \text{and} \quad c_k^{(2)} = m_k + m_{k-1} + m_{k-2}.$$

These two streams are interleaved together to produce the coded stream c_k. Thus, for every bit of input, there are two coded output bits, resulting in a rate $R = 1/2$ code.

---

[1] The symbol D is sometimes used instead of x. The Laurent series representation may be called the D-transform in this case.

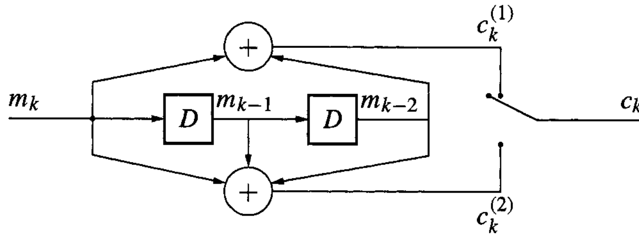


Figure 12.1: A rate $R = 1/2$ convolutional encoder.

It is conventional to assume that the memory elements are initialized with all zeros at the beginning of transmission.

For the input stream $\mathbf{m} = \{1, 1, 0, 0, 1, 0, 1\}$, the outputs are

$$\mathbf{c}^{(1)} = \{1, 1, 1, 1, 1, 0, 0, 0, 1\} \quad \text{and} \quad \mathbf{c}^{(2)} = \{1, 0, 0, 1, 1, 1, 0, 1, 1\}$$

and the interleaved stream is

$$\mathbf{c} = \{11, 10, 10, 11, 11, 01, 00, 01, 11\}$$

(where commas separate the pairs of outputs at a single input time). We can represent the transfer function from input $m(x)$ to output $c^{(1)}(x)$ as $g^{(1)}(x) = 1+x^2$, and the transfer function from $m(x)$ to output $c^{(2)}(x)$ as $g^{(2)}(x) = 1+x+x^2$. The input stream $\mathbf{m} = \{1, 1, 0, 0, 1, 0, 1\}$ can be represented as $m(x) = 1 + x + x^4 + x^6 \in GF(2)[[x]]$. The outputs are

$$c^{(1)}(x) = m(x)g_1(x) = (1 + x + x^4 + x^6)(1 + x^2) = 1 + x + x^2 + x^3 + x^4 + x^8$$
$$c^{(2)}(x) = m(x)g_2(x) = (1 + x + x^4 + x^6)(1 + x + x^2) = 1 + x^3 + x^4 + x^5 + x^7 + x^8.$$

□

A rate $R = k/n$ convolutional code has associated with it an encoder, a $k \times n$ matrix transfer function $G(x)$ called the *transfer function matrix*. For the rate $R = 1/2$ code of this example,

$$G_a(x) = \begin{bmatrix} 1 + x^2 & 1 + x + x^2 \end{bmatrix}.$$

The transfer function matrix of a convolutional code does not always have only polynomial entries, as the following example illustrates.

Example 12.2 Consider the convolutional transfer function matrix

$$G_b(x) = \begin{bmatrix} 1 & \frac{1+x+x^2}{1+x^2}. \end{bmatrix}.$$

Since there is a 1 in the first column the input stream appears explicitly in the interleaved output data; this is a *systematic* convolutional encoder.

A realization (in controller form) for this encoder is shown in Figure 12.2. For the input sequence $m(x) = 1 + x + x^2 + x^3 + x^4 + x^8$, the first output is

$$c^{(1)}(x) = m(x) = 1 + x + x^2 + x^3 + x^4 + x^8$$

and the second output is

$$c^{(2)}(x) = \frac{(1 + x + x^2 + x^3 + x^4 + x^8)(1 + x + x^2)}{1 + x^2} = 1 + x^3 + x^4 + x^5 + x^7 + x^8 + x^{10} + \cdots$$

as can be verified by long division. □

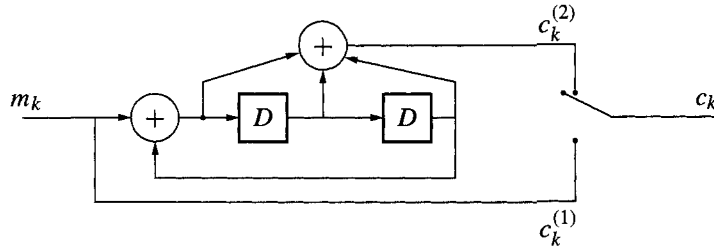


Figure 12.2: A systematic $R = 1/2$ encoder.

An encoder that has only polynomial entries in its transfer function matrix is said to be a **feedforward** encoder or an FIR encoder. An encoder that has rational functions in its transfer function matrix is said to be a **feedback** or IIR encoder.

For a rate $R = k/n$ code with $k > 1$, there are k input message sequences (usually obtained by splitting a single message sequence into k streams). Let

$$\mathbf{m}(x) = [m^{(1)}(x), m^{(2)}(x), \ldots, m^{(k)}(x)]$$

and

$$G(x) = \begin{bmatrix} g^{(1,1)}(x) & g^{(1,2)}(x) & \cdots & g^{(1,n)}(x) \\ g^{(2,1)}(x) & g^{(2,2)}(x) & \cdots & g^{(2,n)}(x) \\ \vdots & & & \\ g^{(k,1)}(x) & g^{(k,2)}(x) & \cdots & g^{(k,n)}(x) \end{bmatrix}. \tag{12.1}$$

The output sequences are represented as

$$\mathbf{c}(x) = [c^{(1)}(x), c^{(2)}(x), \ldots, c^{(n)}(x)] = \mathbf{m}(x)G(x).$$

A transfer function matrix $G(x)$ is said to be systematic if an identity matrix can be identified among the elements of $G(x)$. (That is, if by row and/or column permutations of $G(x)$, an identity matrix can be obtained.)

Example 12.3 For a rate $R = 2/3$ code, a systematic transfer function matrix might be

$$G_1(x) = \begin{bmatrix} 1 & 0 & \frac{x}{1+x^3} \\ 0 & 1 & \frac{x^2}{1+x^3} \end{bmatrix}, \tag{12.2}$$

with a possible realization as shown in Figure 12.3. This is based on the controller form of Figure 4.7. Another more efficient realization based on the observability form from Figure 4.8, is shown in figure 12.4. In this case, only a single set of memory elements is used, employing linearity. With $m(x) = [1 + x^2 + x^4 + x^5 + x^7 + \cdots, x^2 + x^5 + x^6 + x^7 + \cdots]$, the output is

$$\mathbf{c}(x) = [1 + x^2 + x^4 + x^5 + x^7 + \cdots, x^2 + x^5 + x^6 + x^7 + \cdots, x + x^3 + x^5 + \cdots].$$

The corresponding bit sequences are

$$[\{1, 0, 1, 0, 1, 1, 0, 1, \ldots\}, \{0, 0, 1, 0, 0, 1, 1, 1, \ldots\}, \{0, 1, 0, 1, 0, 1, 0, 0, \ldots,\}]$$

which, when interleaved, produce the output sequence

$$\{100, 001, 110, 001, 100, 111, 010, 110\}.$$

□

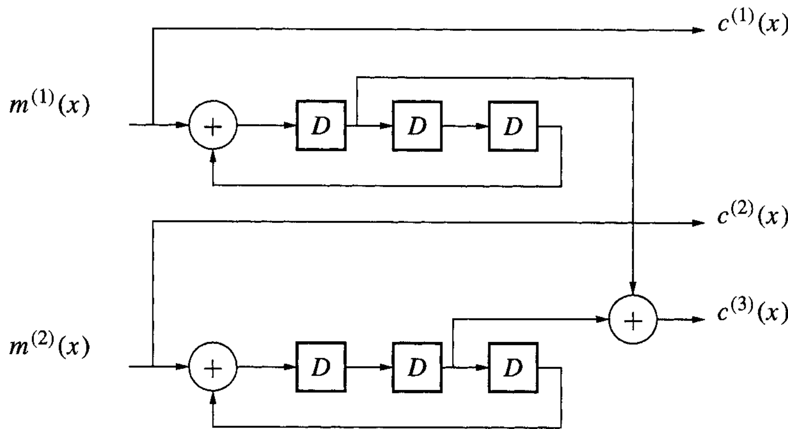


Figure 12.3: A systematic $R = 2/3$ encoder.

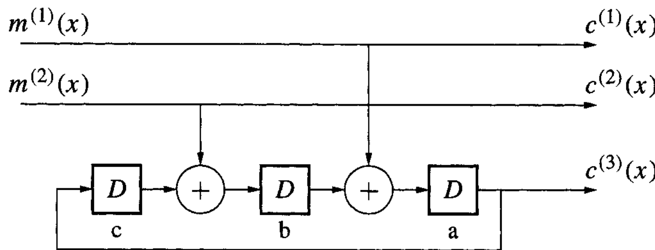


Figure 12.4: A systematic $R = 2/3$ encoder with more efficient hardware.

For feedforward encoders, it is common to indicate the connection polynomials as vectors of numbers representing the *impulse response* of the encoder, rather than polynomials. The transfer function matrix $G(x) = [1 + x^2, 1 + x + x^2]$ is represented by the vectors

$$\mathbf{g}^{(1)} = [101] \quad \text{and} \quad \mathbf{g}^{(2)} = [111].$$

These are often expressed compactly (e.g., in tables of codes) in octal form, where triples of bits are represented using the integers from 0 to 7. In this form, the encoder is represented using $g^{(1)} = 5$, $g^{(2)} = 7$.

For an impulse response $\mathbf{g}^{(j)} = [g_0^{(j)}, g_1^{(j)}, \ldots, g_r^{(j)}]$, the output at time i due to the input sequence m_i is

$$c_i = \sum_{l=0}^{r} m_{i-l} g_l^{(j)},$$

which is, of course, a convolution sum (hence the name of the codes). For an input sequence $\mathbf{m}$, the output sequence can be written as $\mathbf{c}^{(j)} = \mathbf{m} * \mathbf{g}^{(j)}$, where $*$ denotes discrete-time convolution. The operation of convolution can also be represented using matrices. Let $\mathbf{m} = [m_0, m_1, m_2, \ldots]$. Then for $\mathbf{g}^{(j)} = [g_0^{(j)}, g_1^{(j)}, \ldots, g_r^{(j)}]$, the convolution $\mathbf{c} = \mathbf{m}\mathbf{g}^{(j)}$

can be represented as

$$\mathbf{c} = [m_0, m_1, \ldots, m_L] \begin{bmatrix} g_0^{(j)} & g_1^{(j)} & g_2^{(j)} & \cdots & g_L^{(j)} & & & \\ & g_0^{(j)} & g_1^{(j)} & \cdots & g_{L-1}^{(j)} & g_L^{(j)} & & \\ & & g_0^{(j)} & \cdots & g_{L-2}^{(j)} & g_{L-1}^{(j)} & g_L^{(j)} & \\ & & & \ddots & \ddots & \ddots & \ddots & \ddots \end{bmatrix},$$

where empty entries in the matrix indicate zeros.

For a rate 1/2 code, the operation of convolution and interleaving the output sequences is represented by the following matrix, where the columns of different matrices are interleaved:

$$G = \begin{bmatrix} g_0^{(1)} g_0^{(2)} & g_1^{(1)} g_1^{(2)} & g_2^{(1)} g_2^{(2)} & \cdots & g_L^{(1)} g_L^{(2)} & & & \\ & g_0^{(1)} g_0^{(2)} & g_1^{(1)} g_1^{(2)} & \cdots & g_{L-1}^{(1)} g_{L-1}^{(2)} & g_L^{(1)} g_L^{(2)} & & \\ & & g_0^{(1)} g_0^{(2)} & \cdots & g_{L-2}^{(1)} g_{L-2}^{(2)} & g_{L-1}^{(1)} g_{L-1}^{(2)} & g_L^{(1)} g_L^{(2)} & \\ & & & \ddots & \ddots & \ddots & \ddots & \ddots \end{bmatrix}. \tag{12.3}$$

It is the shift (Toeplitz) structure of these generator matrices that gives rise to some of the desirable attributes of convolutional codes.

For a k-input, n-output code with impulse response vectors $\mathbf{g}^{(i,j)}$, $i = 1, 2, \ldots, k$ and $j = 1, 2, \ldots, n$, where $\mathbf{g}^{(i,j)}$ is the impulse response of the encoder connecting input i with output j, the output can be written as

$$c_t^{(j)} = \sum_{q=1}^{k} \sum_{l=0}^{r} m_{i-l}^{(q)} g_{l-t}^{(q,j)}, \qquad j = 1, 2, \ldots, n.$$

A matrix description of the these codes can also be given, but the transfer function matrix $G(x)$ is usually more convenient.

We always deal with *delay free* transfer function matrices, for which it is not possible to factor out a common multiple x^i from $G(x)$. That is, it is not possible to write $G(x) = x^i \tilde{G}(x)$ for some $i > 0$ and any $\tilde{G}(x)$.

12.1.1 The State

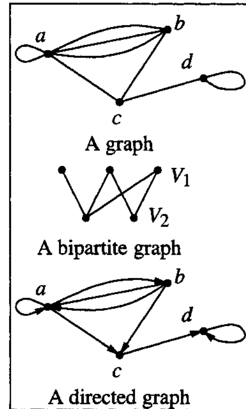
A convolutional encoder is a state machine. For both encoding and decoding purposes, it is frequently helpful to think of the *state diagrams* of the state machines, that is, a representation of the temporal relationships between the states portraying state/next-state relationships as a function of the inputs and the outputs. For an implementation with ν memory elements, there are 2^ν states in the state diagram.

Another representation which is extremely useful is a graph representing the connections from states at one time instant to states at the next time instant. The graph is a bipartite graph, that is, a graph which contains two subsets of nodes, with edges running only between the two subsets. By stacking these bipartite graphs to show several time steps one obtains a graph known as a *trellis*, so-called because of its resemblance to a trellis that might be used for decorative purposes in landscaping.

It may be convenient for some implementations to provide a table indicating the state/next state information explicitly. From the state/next state table the state/previous state information can be extracted.

Box 12.1: Graphs: Basic Definitions

Graph concepts are used throughout the remainder of the book. The following definitions summarize some graph concepts [375]. A **graph** G is a pair (V, E), where V is a nonempty finite set of **vertices** or **nodes** often called the vertex set and E is a finite family of unordered pairs of elements of V called **edges**. (A family is like a set, but some elements may be repeated.) In the graph here, the vertex set is $V = \{a, b, c, d\}$ and the edge family is $E = \{\{a, a\}, \{a, b\}, \{a, b\}, \{a, b\}, \{a, c\}, \{b, c\}, \{c, d\}, \{d, d\}\}$.



A **loop** is an edge joining a vertex to itself. If the edge family is in fact a set (no repeated elements) and there are no loops then the graph is a **simple graph.**

Two vertices of a graph G are **adjacent** if there is an edge joining them. We also say that adjacent nodes are **neighbors**. The two vertices are said to be **incident** to such an edge. Two distinct edges of a graph are adjacent if they have at least one vertex in common. The **degree** of a vertex is the number of edges incident to it.

The vertex set V of a **bipartite** graph can be split into two disjoint sets $V = V_1 \cup V_2$ in such a way that every edge of G joins a vertex of V_1 to a vertex of V_2.

A **walk** through a graph $G = (V, E)$ is a finite sequence of edges $\{v_0, v_1\}, \{v_1, v_2\}, \ldots, \{v_{m-1}, v_m\}$, where each $v_i \in V$ and each $\{v_i, v_j\} \in E$. The **length** of a walk is the number of edges in it. If all the edges are distinct the walk is a **trail**; if all the vertices are also distinct (except possibly $v_0 = v_m$) the walk is a **path**. A path or trail is **closed** if $v_0 = v_m$. A closed path containing at least one edge is called a **circuit**. The **girth** of a graph is the number of edges in its shortest circuit.

A graph is **connected** if there is a path between any pair of vertices $v, w \in V$. A connected graph which contains no circuits is a **tree**. A node of a tree is a **leaf** if its degree is equal to 1.

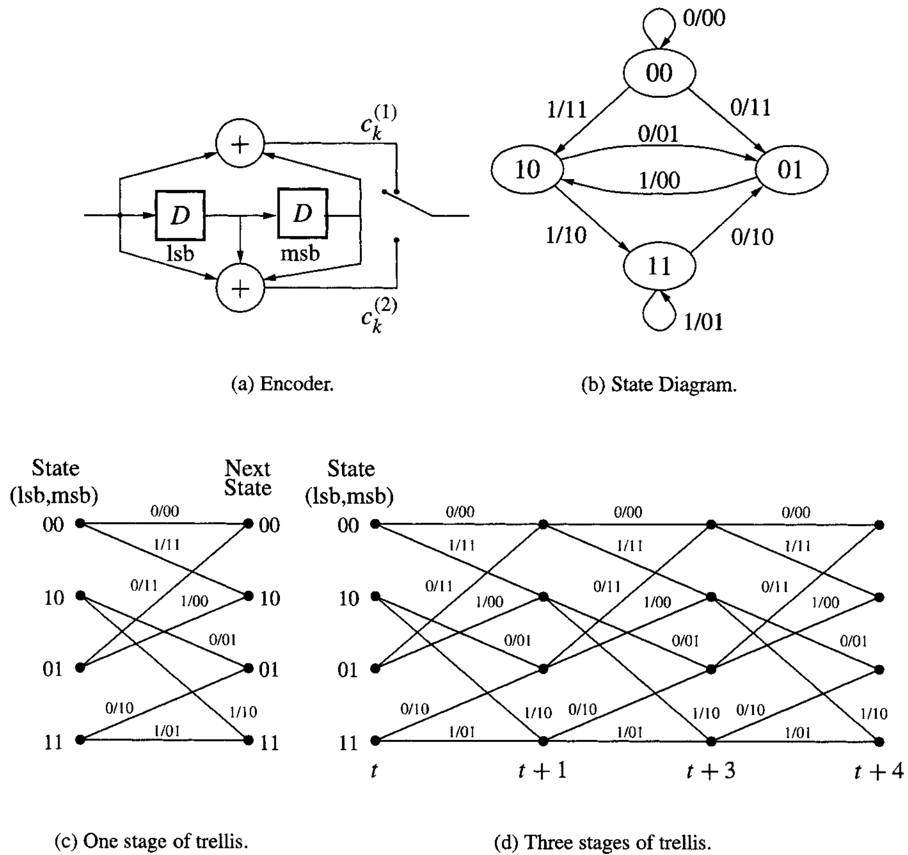
A **directed graph** (digraph) G is a pair (V, E), where E is a finite family of *ordered pairs* of elements of V. Such graphs are frequently represented using arrows to represent edges. In the directed graph here, the edge set is $E = \{(a, a), (a, b), (b, a), (b, a), (a, c), (b, c), (c, d), (d, d)\}$.

Example 12.4 Consider again the convolutional encoder of Example 12.1 with transfer function matrix $G(x) = [1 + x^2, 1 + x + x^2]$. A realization and its corresponding state diagram are shown in Figure 12.5(a) and (b). The state is indicated as a pair of bits, with the first bit representing the least significant bit (lsb). The branches along the state diagram indicate input/output values. One stage of the trellis (corresponding to the transition between two time instants) is shown in Figure 12.5(c). Three trellis stages are shown in Figure 12.5(d). □

Example 12.5 For the rational systematic encoder with matrix transfer function

$$G(x) = \begin{bmatrix} 1 & 0 & \frac{x}{1+x^3} \\ 0 & 1 & \frac{x^2}{1+x^3} \end{bmatrix}, \tag{12.4}$$

with the circuit realization of Figure 12.4, the state diagram, and trellis are shown in Figure 12.6. In the state diagram, the states are represented as integers from 0 to 7 using the numbers (a, b, c)



(a) Encoder.

(b) State Diagram.

(c) One stage of trellis.

(d) Three stages of trellis.

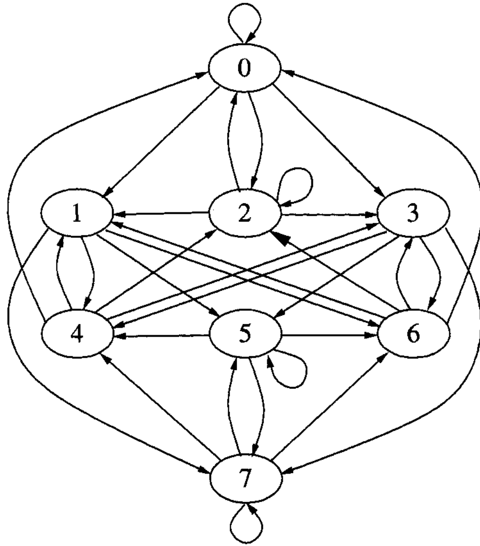
Figure 12.5: Encoder, state diagram, and trellis for $G(x) = [1 + x^2, 1 + x + x^2]$.

corresponding to the registers shown on the circuit. (That is, the lsb is on the right of the diagram this time.) Only the state transitions are shown, not the inputs or outputs, as that would excessively clutter the diagram. The corresponding trellis is also shown, with the branches input/output information listed on the left, with the order of the listing corresponding to the sequence of branches emerging from the corresponding state in top-to-bottom order.

□

12.2 Definition of Codes and Equivalent Codes

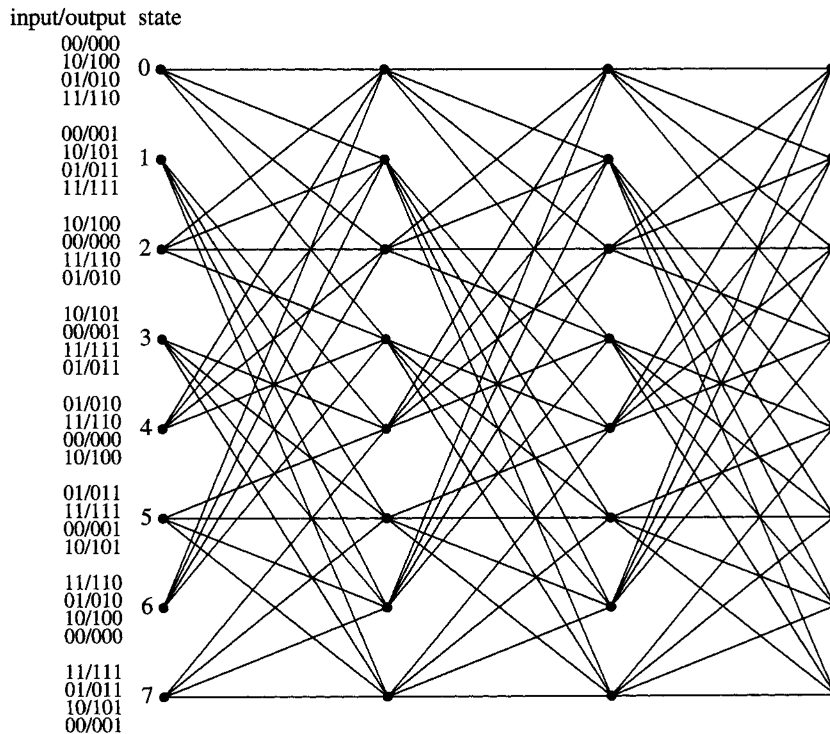
Having now seen several examples of codes, it is now time to formalize the definition and examine some structural properties of the codes. It is no coincidence that the code sequences $(c^{(1)}(x), c^{(2)}(x))$ are the same for Examples 12.1 and 12.2. The sets of sequences that lie in the range of the transfer function matrices $G_a(x)$ and $G_b(x)$ are identical: even though the encoders are different, they encode to the same code. (This is analogous to having different generator matrices to represent the same block code.)



(a) State diagram.

State	Next State			
0	0	1	2	3
1	4	5	6	7
2	1	0	3	2
3	5	4	7	6
4	2	3	0	1
5	6	7	4	5
6	3	2	1	0
7	7	6	5	4

(b) State/Next State Information.



(c) Trellis.

Figure 12.6: State diagram and trellis for a rate $R = 2/3$ systematic encoder.

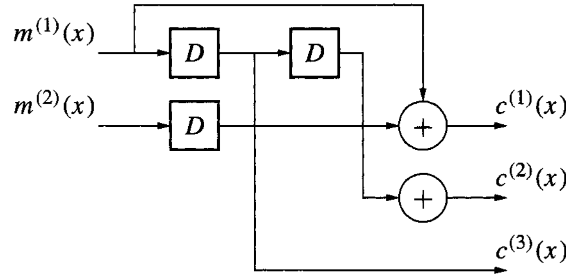


Figure 12.7: A feedforward $R = 2/3$ encoder.

We formally define a convolutional code as follows:

Definition 12.1 [303, p. 94] A rate $R = k/n$ code over the field of rational Laurent series $F[[x]]$ over the field $\mathbb{F}$ is the image of an injective linear mapping of the k-dimensional Laurent series $\mathbf{m}(x) \in \mathbb{F}[[x]]^k$ into the n-dimensional Laurent series $\mathbf{c}(x) \in \mathbb{F}[[x]]^n$. □

In other words, the convolutional code is the set $\{\mathbf{c}(x)\}$ of all possible output sequences as all possible input sequences $\{\mathbf{m}(x)\}$ are applied to the encoder. The code is the image set or (row) range of the linear operator $G(x)$, *not* the linear operator $G(x)$ itself.

Example 12.6 Let

$$G_2(x) = \begin{bmatrix} 1 & x^2 & x \\ x & 1 & 0 \end{bmatrix} \tag{12.5}$$

and note that

$$G_2(x) = \begin{bmatrix} 1 & x^2 \\ x & 1 \end{bmatrix} G_1(x) = T_2(x)G_1(x),$$

(where $G_1(x)$ was defined in (12.2)) and consider the encoding operation

$$m(x)G_2(x) = m(x)\begin{bmatrix} 1 & x^2 \\ x & 1 \end{bmatrix} G_1(x) = m'(x)G_1(x),$$

where

$$m'(x) = m(x)\begin{bmatrix} 1 & x^2 \\ x & 1 \end{bmatrix} = m(x)T_2(x).$$

Corresponding to each $m(x)$ there is a unique $m'(x)$, since $T_2(x)$ is invertible. Hence, as $m'(x)$ varies over all possible input sequences, $m(x)$ also varies over all possible input sequences. The set of output sequences $\{c(x)\}$ produced is the same for $G_2(x)$ as $G_1(x)$: that is, both encoders produce the same code.

Figure 12.7 shows a schematic representation of this encoder. Note that the implementation of both $G_1(x)$ (of Figure 12.4) and $G_2(x)$ have three one-bit memory elements in them. The contents of these memory elements may be thought of as the *state* of the devices. Since these are binary circuits, there are $2^3 = 8$ distinct states in either implementation. □

Example 12.7 Another encoder for the code of Example 12.1 is

$$G_3(x) = \begin{bmatrix} 1+x & 1+x^2 & x \\ x+x^2 & 1+x & 0 \end{bmatrix}, \tag{12.6}$$

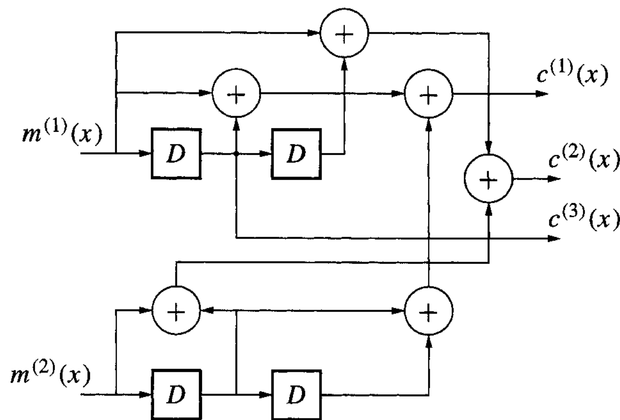


Figure 12.8: A less efficient feedforward $R = 2/3$ encoder.

where, it may be observed,

$$G_3(x) = \begin{bmatrix} 1+x & 1+x^2 \\ x+x^2 & 1+x \end{bmatrix} G_1(x) = T_3(x)G_1(x).$$

The encoding operation

$$m(x)G_3(x) = m(x)T_3(x)G_1(x) = m'(x)G_1(x),$$

with $m'(x) = m(x)T_3(x)$, again results in the same code, since $T_3(x)$ is invertible.

The schematic for $G_3(x)$ in Figure 12.8 would require more storage blocks than either $G_1(x)$ or $G_2(x)$: it is not as efficient in terms of hardware. □

These examples motivate the following definition.

Definition 12.2 Two transfer function matrices $G(x)$ and $G'(x)$ are said to be **equivalent** if they generate the same convolutional code. Two transfer function matrices $G(x)$ and $G'(x)$ are equivalent if $G(x) = T(x)G'(x)$ for an *invertible* matrix $T(x)$. □

These examples also motivate other considerations: For a given a code, is there always a feedforward transfer matrix representation? Is there always a systematic representation? What it the "minimal" representation, requiring the least amount of memory? As the following section reveals, another question is whether the representation is catastrophic.

12.2.1 Catastrophic Encoders

Besides the hardware inefficiency, there is another fundamental problem with the encoder $G_3(x)$ of (12.6). Suppose that the input is

$$\mathbf{m}(x) = \begin{bmatrix} 0 & \frac{1}{1+x} \end{bmatrix},$$

where, expanding the formal series by long division,

$$\frac{1}{1+x} = 1 + x + x^2 + x^3 + \cdots$$

The input sequence thus has infinite Hamming weight. The corresponding output sequence is

$$\mathbf{c}(x) = m(x)G_3(x) = \begin{bmatrix} x & 1 & 0 \end{bmatrix},$$

a sequence with total Hamming weight 2. Suppose now that $\mathbf{c}(x)$ is passed through a channel and that *two* errors occur at precisely the locations of the nonzero code elements. Then the received sequences is exactly zero, which would decode (under any reasonable decoding scheme) to $\hat{\mathbf{m}}(x) = \begin{bmatrix} 0 & 0 \end{bmatrix}$. Thus, a *finite* number of errors in the channel result in an *infinite* number of decoder errors. Such an encoder is called a *catastrophic* encoder. It may be emphasized, however, that the problem is not with the code but the particular encoder, since $G_1(x)$, $G_2(x)$ and $G_3(x)$ all produce the same code but, $G_1(x)$ and $G_2(x)$ do not exhibit catastrophic behavior.

Letting $\mathrm{wt}(c(x))$ denote the weight of the sequence $c(x)$, we have the following definition:

Definition 12.3 [303, p.97] An encoder $G(x)$ for a convolutional code is **catastrophic** if there exists a message sequence $\mathbf{m}(x)$ such that $\mathrm{wt}(\mathbf{m}(x)) = \infty$ and the weight of the coded sequence $\mathrm{wt}(\mathbf{m}(x)G(x)) < \infty$. □

To understand more of the nature of catastrophic codes, we introduce the idea of a right inverse of a matrix.

Definition 12.4 Let $k < n$. A **right inverse** of a $k \times n$ matrix G is a $n \times k$ matrix G^{-1} such that $GG^{-1} = I_{k,k}$, the $k \times k$ identity matrix. (This is not the same as the inverse, which cannot exist when G is not square.) A right inverse of G can exist only if G is full rank. □

Example 12.8 For $G_1(x)$ of (12.2), a right inverse is

$$G_1(x)^{-1} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix}.$$

For $G_2(x)$ of (12.5), a right inverse is

$$G_2(x)^{-1} = \begin{bmatrix} 1 & x \\ x & 1+x^2 \\ x^2 & 1+x+x^3 \end{bmatrix}.$$

For $G_3(x)$ of (12.6), a right inverse is

$$G_3(x)^{-1} = \frac{1}{1+x+x^3+x^4} \begin{bmatrix} 1+x & 1+x^2 \\ x+x^2 & 1+x \\ 0 & 0 \end{bmatrix}.$$

□

Note that $G_1(x)^{-1}$ and $G_2(x)^{-1}$ are *polynomial* matrices — they have only polynomial entries — while $G_3(x)^{-1}$ has non-polynomial entries — some of its entries involve rational functions.

Example 12.9 It should be observed that right inverses are not necessarily unique. For example, the matrix $[1+x^2, 1+x+x^2]$ has the right inverses

$$\begin{bmatrix} 1+x & x \end{bmatrix}^T \quad \text{and} \quad \begin{bmatrix} \frac{x}{x+1} & 1 \end{bmatrix}^T.$$

Of these, one has all polynomial elements. □

Definition 12.5 A transfer function matrix with only polynomial entries is said to be a **polynomial encoder** (i.e., it uses FIR filters). More briefly, such an encoder is said to be **polynomial**. A transfer function matrix with rational entries is said to be a **rational encoder** (i.e., it uses IIR filters), or simply **rational**. □

For an encoder $G(x)$ with right inverse $G(x)^{-1}$, the message may be recovered (in a theoretical sense when there is no noise corrupting the code — this is not a decoding algorithm!) by

$$\mathbf{c}(x)G(x)^{-1} = \mathbf{m}(x)G(x)G(x)^{-1} = \mathbf{m}(x). \tag{12.7}$$

Now suppose that $\mathbf{c}(x)$ has finite weight, but $\mathbf{m}(x)$ has infinite weight: from (12.7) this can only happen if one or more elements of the right inverse $G(x)^{-1}$ has an infinite number of coefficients, that is, they are rational functions. It turns out that this is a necessary and sufficient condition:

Theorem 12.1 *A transfer function matrix $G(x)$ is not catastrophic if and only if it has a right inverse $G(x)^{-1}$ having only polynomial entries.*

From the right inverses in Example 12.8, we see that $G_1(x)$ and $G_2(x)$ have polynomial right inverses, while $G_3(x)$ has non-polynomial entries, indicating that $G_3(x)$ is a catastrophic generator.

Definition 12.6 A transfer function matrix $G(x)$ is **basic** if it is polynomial and has a polynomial right inverse. □

$G_2(x)$ is an example of a basic transfer function matrix.

Example 12.10 Another example of a transfer function matrix for the code is

$$G_4(x) = \begin{bmatrix} 1+x+x^2+x^3 & 1+x & x \\ x & 1 & 0 \end{bmatrix}. \tag{12.8}$$

The invariant factor decomposition (presented below) can be used to show that this is basic. However, for sufficiently small matrices finding a right inverse may be done by hand. We seek a polynomial matrix such that

$$\begin{bmatrix} 1+x+x^2+x^3 & 1+x & x \\ x & 1 & 0 \end{bmatrix} \begin{bmatrix} a & d \\ b & e \\ c & f \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

Writing out the implied equations we have

$$a(1+x+x^2+x^3) + b(1+x) + cx = 1 \qquad ax + b = 0$$

$$d(1+x+x^2+x^3) + e(1+x) + fx = 0 \qquad dx + e = 1.$$

From the second we obtain $b = ax$; substituting this into the first we find $a(1+x^3) + cx = 1$. By setting $c = x^2$ and $a = 1$ we can solve this using polynomials.

From the fourth equation we obtain $e = 1 + dx$, so that from the third equation

$$d(1+x+x^2+x^3) + (1+dx)(1+x) + fx = 0$$

or $d(1+x^3) + fx = 1+x$. This yields $d = 1$ and $f = x^2 + 1$. This gives a polynomial right inverse, so $G_4(x)$ is basic. Note that the encoder requires four memory elements in its implementation. □

Two *basic* encoders $G(x)$ and $G'(x)$ are equivalent if and only if $G(x) = T(x)G'(x)$, where

1. $T(x)$ is not only invertible (as required by mere equivalence),
2. But also $\det(T(x)) = 1$,

so that when the right inverse is computed all the elements remain polynomial.

12.2.2 Polynomial and Rational Encoders

We show in this section that every rational encoder has an equivalent basic encoder. The implication is that it is sufficient to use only feedforward (polynomial) encoders to represent every code. There is, however, an important caveat: there may not be an equivalent basic (or even polynomial) *systematic* encoder. Thus, if a systematic coder is desired, it may be necessary to use a rational encoder. This is relevant because the very powerful behavior of turbo codes relies on good *systematic* convolutional codes.

Our results make use of the **invariant factor decomposition** of a matrix [162, section 3.7]. Let $G(x)$ be a $k \times n$ polynomial matrix. Then[2] $G(x)$ can be written as

$$G(x) = A(x)\Gamma(x)B(x),$$

where $A(x)$ is a $k \times k$ polynomial matrix and $B(x)$ is a $n \times n$ polynomial matrix and where $\det(A(x)) = 1$, $\det(B(x)) = 1$ (i.e., they are **unimodular matrices**); and $\Gamma(x)$ is the $k \times n$ diagonal matrix

$$\Gamma(x) = \begin{bmatrix} \gamma_1(x) & & & & & \\ & \gamma_2(x) & & & & \\ & & \gamma_3(x) & & & \mathbf{0}_{k,n-k} \\ & & & \ddots & & \\ & & & & \gamma_k(x) & \end{bmatrix}.$$

The nonzero elements $\gamma_i(x)$ of $\Gamma(x)$ are polynomials and are called the **invariant factors** of $G(x)$. (If any of the $\gamma_i(x)$ are zero, they are included in the zero block, so k is the number of nonzero elements.) Furthermore, the invariant factors satisfy

$$\gamma_i(x) \mid \gamma_{i+1}(x).$$

(Since we are expressing a theoretical result here, we won't pursue the algorithm for actually computing the invariant factor decomposition[3]); it is detailed in [162].

Extending the invariant factor theorem to rational matrices, a rational matrix $G(x)$ can be written as

$$G(x) = A(x)\Gamma(x)B(x),$$

where $A(x)$ and $B(x)$ are again polynomial unimodular matrices and $\Gamma(x)$ is diagonal with rational entries $\alpha_i(x)/\beta_i(x)$, such that $\gamma_i(x) = \alpha_i(x) \mid \alpha_{i+1}(x)$ and $\beta_{i+1}(x) \mid \beta_i(x)$.

Let $G(x)$ be a rational encoding matrix, with invariant factor decomposition $G(x) = A(x)\Gamma(x)B(x)$. Let us decompose $B(x)$ into the blocks

$$B(x) = \begin{bmatrix} G'(x) \\ B_2(x) \end{bmatrix},$$

[2]The invariant factor decomposition has a technical requirement: The factorization in the ring must be unique, up to ordering and units. This technical requirement *is* met in our case, since the polynomials form a principal ideal domain, which implies unique factorization. See, e.g., [106, Chapter 32].

[3]The invariant factor decomposition can be thought of as a sort of singular value decomposition for modules.

where $G'(x)$ is $k \times n$. Then, since the last k columns of $\Gamma(x)$ are zero, we can write

$$G(x) = A(x) \begin{bmatrix} \frac{\alpha_1(x)}{\beta_1(x)} & & & \\ & \frac{\alpha_2(x)}{\beta_2(x)} & & \\ & & \ddots & \\ & & & \frac{\alpha_k(x)}{\beta_k(x)} \end{bmatrix} G'(x) \triangleq A(x)\Gamma'(x)G'(x).$$

Since $A(x)$ and $\Gamma'(x)$ are nonsingular matrices, $G(x)$ and $G'(x)$ are equivalent encoders: they describe the same convolutional code. But $G'(x)$ is polynomial (since $B(x)$ is polynomial) and since $B(x)$ is unimodular (and thus has a polynomial inverse) it follows that $G'(x)$ has a polynomial right inverse. We have thus proved the following:

Theorem 12.2 *Every rational encoder has an equivalent basic transfer function matrix.*

The proof of the theorem is constructive: To obtain a basic encoding matrix from a rational transfer function $G(x)$, compute the invariant factor decomposition $G(x) = A(x)\Gamma(x)B(x)$ and take the first k rows of $B(x)$.

12.2.3 Constraint Length and Minimal Encoders

Comparing the encoders for the code we have been examining, we have seen that the encoders for $G_1(x)$ or $G_2(x)$ use three memory elements, while the encoder $G_3(x)$ uses four memory elements. We investigate in this section aspects of the question of the smallest amount of memory that a code requires of its encoder.

Let $G(x)$ be a basic encoder (so that the elements of $G(x)$ are polynomials). Let

$$\nu_i = \max_j \deg(g_{ij}(x))$$

denote the maximum degree of the polynomials in row i of $G(x)$. This is the number of memory elements necessary to store the portion of a realization (circuit) of the encoder corresponding to input i. The number

$$\nu = \sum_{i=1}^{k} \nu_i \tag{12.9}$$

represents the total amount of storage required for all inputs. This quantity is called the **constraint length** of the encoder.

Note: In other sources (e.g., [373]), the constraint length is defined as the maximum number of bits in a single output stream that can be affected by any input bit (for a polynomial encoder). This is taken as the highest degree of the encoder plus one: $\nu = 1 + \max_{i,j} \deg(g_{i,j}(x))$. The reader should be aware that different definitions are used. Ours suits the current purposes.

We make the following definition:

Definition 12.7 A **minimal basic** encoder is a basic encoder that has the smallest constraint length among all equivalent basic encoders. □

Typically we are interested in minimal encoders: they require the least amount of hardware to build and they have the fewest evident states. We now explore the question of when an encoder is minimal basic.

The first theorem involves a particular decomposition of the decoder matrix. We demonstrate first with some examples. Let $G(x) = G_2(x)$ from (12.2). Write

$$G_2(x) = \begin{bmatrix} 1 & x^2 & x \\ x & 1 & 0 \end{bmatrix} = \begin{bmatrix} x^2 & \\ & x \end{bmatrix} \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 1 & 0 & x \\ 0 & 1 & 0 \end{bmatrix}. \tag{12.10}$$

As another example, when $G(x) = G_4(x)$ from (12.8)

$$\begin{aligned} G_4(x) &= \begin{bmatrix} 1+x+x^2+x^3 & 1+x & x \\ x & 1 & 0 \end{bmatrix} \\ &= \begin{bmatrix} x^3 & \\ & x \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 1+x+x^2 & 1+x & x \\ 0 & 1 & 0 \end{bmatrix}. \end{aligned} \tag{12.11}$$

In general, given a basic encoder $G(x)$ we write it as

$$G(x) = \begin{bmatrix} x^{\nu_1} & & & \\ & x^{\nu_2} & & \\ & & \cdots & \\ & & & x^{\nu_k} \end{bmatrix} G_h + \tilde{G}(x) = \Lambda(x) G_h + \tilde{G}(x), \tag{12.12}$$

where G_h is a binary matrix with a 1 indicating the position where the highest degree term x^{ν_i} occurs in row i and each row of $\tilde{G}(x)$ contains all the terms of degree less than ν_i. Using this notation, we have the following:

Theorem 12.3 *[175] Let $G(x)$ be a $k \times n$ basic encoding matrix with constraint length ν. The following statements are equivalent:*

(a) $G(x)$ is a minimal basic encoding matrix.

(b) The maximum degree μ among the $k \times k$ subdeterminants of $G(x)$ is equal to the overall constraint length ν.

(c) G_h is full rank.

To illustrate this theorem, consider the decomposition of $G_4(x)$ in (12.11). The 2×2 subdeterminants of $G_4(x)$ are obtained by taking the determinant of the two 2×2 submatrices of $G_4(x)$,

$$\det \begin{bmatrix} 1+x+x^2+x^3 & 1+x \\ x & 1 \end{bmatrix} \qquad \det \begin{bmatrix} 1+x & x \\ 1 & 0 \end{bmatrix},$$

the maximum degree of which is 3. Also, we note that G_h is not full rank. Hence, we conclude that $G_4(x)$ is not a minimal basic encoding matrix.

Proof To show the equivalence of (b) and (c): Observe that the degree of a subdeterminant of $G(x)$ is determined by the $k \times k$ submatrices of $\Lambda(x)G_h$ (which have the largest degree terms) and not by $\tilde{G}(x)$. The degree of the determinants of the $k \times k$ submatrices of $G(x)$ are then determined by the subdeterminants of $\Lambda(x)$ and the $k \times k$ submatrices of G_h. Since $\det \Lambda(x) \neq 0$, if G_h is full rank, then at least one of its $k \times k$ submatrices has nonzero determinant, so that at least one of the determinants of the $k \times k$ submatrices of $\Lambda(x)G_h$ has degree μ equal to $\deg \Lambda(x) = \nu$. On the other hand, if G_h is rank deficient, then none of the determinants of the submatrices of $\Lambda(x)G_h$ can be equal to the determinant of $\Lambda(x)$.

To show that (a) implies (b): Assume that $G(x)$ is minimal basic. Suppose that $\text{rank}(G_h) < k$. Let the rows of $G(x)$ be denoted by $\mathbf{g}_1, \mathbf{g}_2, \ldots, \mathbf{g}_k$, let the rows of G_h

be denoted by $\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_k$, and let the rows of $\tilde{G}(x)$ be denoted by $\tilde{\mathbf{g}}_1, \tilde{\mathbf{g}}_2, \ldots, \tilde{\mathbf{g}}_k$. Then the decomposition (12.12) is

$$\mathbf{g}_i = x^{\nu_i}\mathbf{h}_i + \tilde{\mathbf{g}}_i.$$

By the rank-deficiency there is a linear combination of rows of G_h such that

$$\mathbf{h}_{i_1} + \mathbf{h}_{i_2} + \cdots + \mathbf{h}_{i_d} = 0$$

for some $d \leq k$. Assume (without loss of generality) that the rows of $G(x)$ are ordered such that $\nu_1 \geq \nu_2 \geq \cdots \geq \nu_k$. The ith row of $\Lambda(x)G_h$ is $x^{\nu_i}\mathbf{h}_i$. Adding

$$x^{\nu_{i_1}}[\mathbf{h}_{i_2} + \mathbf{h}_{i_3} + \cdots + \mathbf{h}_{i_d}]$$

to the i_1st row of $\Lambda(x)G_h$ (which is $x^{\nu_i}\mathbf{h}_i$) reduces it to an all-zero row. Note that

$$x^{\nu_{i_1}}[\mathbf{h}_{i_2} + \mathbf{h}_{i_3} + \cdots + \mathbf{h}_{i_d}] = x^{\nu_{i_1}-\nu_{i_2}}x^{\nu_{i_2}}\mathbf{h}_{i_2} + x^{\nu_{i_1}-\nu_{i_3}}x^{\nu_{i_3}}\mathbf{h}_{i_3} + \cdots + x^{\nu_{i_1}-\nu_{i_d}}x^{\nu_{i_d}}\mathbf{h}_{i_d}.$$

Now consider computing $G'(x) = T(x)G(x)$, where $T(x)$ is the invertible matrix

$$T = \begin{array}{r} \\ \\ \\ i_1: \\ \\ \\ \\ \end{array} \begin{bmatrix} 1 & & & & i_2 & & i_3 & \cdots & i_d \\ & 1 & & & & & & & \\ & & \ddots & & & & & & \\ & & & 1 & \cdots\; x^{\nu_{i_1}-\nu_{i_2}} & & x^{\nu_{i_1}-\nu_{i_3}} & \cdots & x^{\nu_{i_1}-\nu_{i_d}} \\ & & & & & \ddots & & & \\ & & & & & & 1 & & \\ & & & & & & & \ddots & \\ & & & & & & & & 1 \end{bmatrix},$$

with an identity on the diagonal. This has the effect of adding

$$x^{\nu_{i_1}-\nu_{i_2}}\mathbf{g}_{i_2} + x^{\nu_{i_1}-\nu_{i_3}}\mathbf{g}_{i_3} + \cdots + x^{\nu_{i_1}-\nu_{i_d}}\mathbf{g}_{i_d}$$

to the i_1st row of $G(x)$, which reduces the highest degree of the i_1st row of $G(x)$ (because the term $x^{\nu_{i_1}}\mathbf{h}_{i_1}$ is eliminated) but leaves other rows of $G(x)$ unchanged. But $G'(x)$ is an equivalent transfer function matrix. We thus obtain a basic encoding matrix $G'(x)$ equivalent to $G(x)$ with an overall constraint length less than that of $G(x)$. This contradicts the assumption that $G(x)$ is minimal basic, which implies that G_h must be full rank. From the equivalence of (b) and (c), $\mu = \nu$.

Conversely, to show (b) implies (a): Let $G'(x)$ be a basic encoding matrix equivalent to $G(x)$. Then $G'(x) = T(x)G(x)$, where $T(x)$ is a $k \times k$ polynomial matrix with $\det T(x) = 1$. The maximum degree among the $k \times k$ subdeterminants of $G'(x)$ is equal to that of $G(x)$ (since $\det T(x) = 1$). Hence, $\text{rank}(G_h)$ is invariant over all equivalent basic encoding matrices. Since $\text{rank}(G_h)$ is less than or equal to the overall constraint length, if $\mu = \nu$, it follows that $G(x)$ is minimal basic. □

The proof is essentially constructive: given a non-minimal basic $G(x)$, a minimal basic encoder can be constructed by finding rows of G_h such that

$$\mathbf{h}_{i_1} + \mathbf{h}_{i_2} + \mathbf{h}_{i_3} + \cdots + \mathbf{h}_{i_d} = 0,$$

where the indices are ordered such that $\nu_{i_d} \geq \nu_{i_j}$, $1 \leq j < d$, then adding

$$x^{\nu_{i_d}-\nu_{i_1}}\mathbf{g}_{i_1} + \cdots x^{\nu_{i_d}-\nu_{i_{d-1}}}\mathbf{g}_{i_{d-1}} \tag{12.13}$$

to the i_dth row of $G(x)$.

Example 12.11 Let $G(x) = G_4(x)$, as before. Then

$$\mathbf{h}_1 = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \qquad \mathbf{h}_2 = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}$$

so that $\mathbf{h}_1 + \mathbf{h}_2 = 0$. We have $i_1 = 2$ and $i_2 = 1$. We thus add

$$x^{3-1}\mathbf{g}_2 = x^2\begin{bmatrix} x & 1 & 0 \end{bmatrix} = \begin{bmatrix} x^3 & x^2 & 0 \end{bmatrix}$$

to row 1 of $G(x)$ to obtain the transfer function matrix

$$G_5(x) = \begin{bmatrix} 1+x+x^2 & 1+x+x^2 & x \\ x & 1 & 0 \end{bmatrix}$$

to obtain an equivalent minimal basic encoder. □

Comparing $G_5(x)$ with $G_2(x)$, we make the observation that minimal basic encoders are not unique.

As implied by its name, the advantage of a basic minimal encoder is that it is "smallest" in some sense. It may be built in such a way that the number of memory elements in the device is the smallest possible and the number of states of the device is the smallest possible. There is another advantage to minimal encoders: it can be shown that a minimal encoder is not catastrophic.

12.2.4 Systematic Encoders

Given an encoder $G(x)$, it may be turned into a systematic decoder by identifying a full-rank $k \times k$ submatrix $T(x)$. Then form

$$G'(x) = T(x)^{-1}G(x).$$

Then $G'(x)$ is of the form (perhaps after column permutations)

$$G'(x) = \begin{bmatrix} I_{k,k} & P_{k,n-k}(x) \end{bmatrix},$$

where $P_{k,n-k}(x)$ is a (generally) rational matrix. The outputs produced by $P_{k,n-k}$ — that is, the non-systematic part of the generator — are frequently referred to as the parity bits, or check bits, of the coded sequence.

Example 12.12 [175] Suppose

$$G(x) = \begin{bmatrix} 1+x & x & 1 \\ x^2 & 1 & 1+x+x^2 \end{bmatrix}$$

and $T(x)$ is taken as the first two columns:

$$T(x) = \begin{bmatrix} 1+x & x \\ x^2 & 1 \end{bmatrix} \qquad T^{-1}(x) = \frac{1}{1+x+x^3}\begin{bmatrix} 1 & x \\ x^2 & 1+x \end{bmatrix}.$$

Then

$$G'(x) = T^{-1}(x)G(x) = \begin{bmatrix} 1 & 0 & \frac{1+x+x^2+x^3}{1+x+x^3} \\ 0 & 1 & \frac{1+x^2+x^3}{1+x+x^3} \end{bmatrix}.$$

□

Historically, polynomial encoders (i.e., those implemented using FIR filters) have been much more commonly used than systematic encoders (employing IIR filters). However, there are some advantages to using systematic codes. First, it can be shown that every systematic encoding matrix is minimal. Second, systematic codes cannot be catastrophic (since the data appears explicitly in the codeword).

For a given constraint length, the set of systematic codes with polynomial transfer matrices has generally inferior distance properties compared with the set of systematic codes with rational transfer matrices. In fact, it has been observed [357, p. 252] that for large constraint lengths ν, the performance of a polynomial systematic code of constraint length K is approximately the same as that of a nonsystematic code of constraint length $K(1-k/n)$. (See Table 12.2) For example, for a rate $R = 1/2$ code, polynomial systematic codes have about the performance of nonsystematic codes of half the constraint length, while requiring exactly the same optimal decoder complexity. Because of these reasons, recent work in turbo codes has relied almost exclusively on systematic encoders.

12.3 Decoding Convolutional Codes

12.3.1 Introduction and Notation

Several algorithms have been developed for decoding convolutional codes. The one most commonly used is the Viterbi algorithm, which is a maximum likelihood sequence estimator (MLSE). A variation on the Viterbi algorithm, known as the soft-output Viterbi algorithm (SOVA), which provides not only decoded symbols but also an indication of the *reliability* of the decoded values, is presented in Section 14.3.17 in conjunction with turbo codes. Another decoding algorithm is the maximum a posteriori (MAP) decoder frequently referred to as the BCJR algorithm, which computes probabilities of decoded bits. The BCJR algorithm is somewhat more complex than the Viterbi algorithm, without significant performance gains compared to Viterbi codes. It is, however, ideally suited for decoding turbo codes, and so is also detailed in chapter 14. It is also shown there that the BCJR and the Viterbi are fundamentally equivalent at a deeper level.

Suboptimal decoding algorithms are also occasionally of interest, particularly when the constraint length is large. These provide most of the performance of the Viterbi algorithm, but typically have substantially lower computational complexity. In Section 12.8 the stack algorithm (also known as the ZJ algorithm), Fano's algorithm, and the M-algorithm are presented as instances of suboptimal decoders.

To set the stage for the decoding algorithm, we introduce some notation for the stages of processing. Consider the block diagram in Figure 12.9. The time index is denoted by t, which indexes the times at which states are distinguished in the state diagram; there are thus k bits input to the encoder and n bits output from the encoder at each time step t.

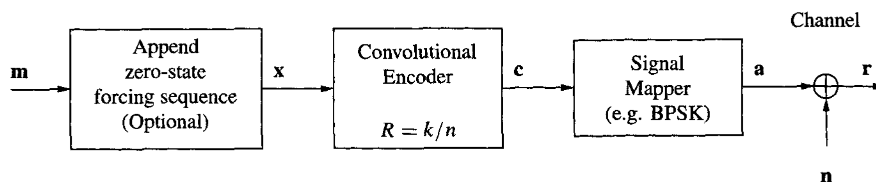


Figure 12.9: Processing stages for a convolutional code.

- The input — message — data may have a sequence appended to drive the state to 0 at the end of some block of input. At time t there are k input bits, denoted as $m_t^{(i)}$ or $x_t^{(i)}$, $i = 1, 2, \ldots, k$. The set of k input bits at time t is denoted as $\mathbf{m}_t = (m_t^{(1)}, m_t^{(2)}, \ldots, m_t^{(k)})$ and those with the (optional) appended sequence are $\mathbf{x}_t$. An input sequence consisting of L blocks is denoted as $\mathbf{x}$:

$$\mathbf{x} = \{\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_{L-1}\}.$$

- The corresponding coded output bits are denoted as $c_t^{(i)}$, $i = 1, 2, \ldots, n$, or collectively at time t as $\mathbf{c}_t$. The entire coded output sequence is $\mathbf{c} = \{\mathbf{c}_0, \mathbf{c}_1, \ldots, \mathbf{c}_{L-1}\}$.
- The coded output sequence is mapped to a sequence of M symbols selected from a signal constellation with Q points in some signal space, with $Q = 2^p$. We must have 2^{nL} (the number of coded bits in the sequence) equal to 2^{pM}, so that $M = nL/p$. For convenience in notation, we assume that $p = 1$ (e.g., BPSK modulation), so that $M = L$; we use M as identical to L in this development, although it does not have to be.

  The mapped signals at time t are denoted as $a_t^{(i)}$, $i = 1, 2, \ldots, n$. The entire coded sequence is $\mathbf{a} = \{\mathbf{a}_0, \mathbf{a}_1, \ldots, \mathbf{a}_{L-1}\}$
- The symbols $\mathbf{a}_t$ pass through a channel, resulting in a received symbol $r_t^{(i)}$, $i = 1, 2, \ldots, n$, or a block $\mathbf{r}_t$. We consider explicitly two channel models: an AWGN and a BSC. For the AWGN we have

$$r_t^{(i)} = a_t^{(i)} + n_t^{(i)}, \qquad \text{where } n_t^{(i)} \sim \mathcal{N}(0, \sigma^2), \text{ and where } \sigma^2 = \frac{N_0}{2}.$$

  For the AWGN the received data are real- or complex-valued. For the BSC, the mapped signals are equal to the coded data, $\mathbf{a}_t = \mathbf{c}_t$. The received signal is

$$r_t^{(i)} = c_t^{(i)} \oplus n_t^{(i)}, \qquad \text{where } n_t \sim \mathcal{B}(p_c),$$

  where $\oplus$ denotes addition modulo 2 and p_c is the channel crossover probability and $\mathcal{B}(p_c)$ indicates a Bernoulli random variable. For both channels it is assumed that the $n_t^{(i)}$ are mutually independent for all i and t, resulting in a memoryless channel. We denote the likelihood function for these channels as $f(\mathbf{r}_t|\mathbf{a}_t)$. For the AWGN channel,

$$f(\mathbf{r}_t|\mathbf{a}_t) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi}\sigma} \exp\left[-\frac{1}{2\sigma^2}(r_t^{(i)} - a_t^{(i)})^2\right]$$
$$= (\sqrt{2\pi}\sigma)^{-n} \exp\left[-\frac{1}{2\sigma^2}\|\mathbf{r}_t - \mathbf{a}_t\|^2\right],$$

  where $\|\cdot\|^2$ denotes the usual Euclidean distance,

$$\|\mathbf{r}_t - \mathbf{a}_t\|^2 = \sum_{i=1}^{n}(r_t^{(i)} - a_t^{(i)})^2.$$

  For the BSC,

$$f(\mathbf{r}_t|\mathbf{a}_t) = \prod_{i=1}^{n} p_c^{[r_t^{(i)} \neq a_t^{(i)}]}(1 - p_c)^{[r_t^{(i)} = a_t^{(i)}]} = p_c^{d_H(\mathbf{r}_t, \mathbf{a}_t)}(1 - p_c)^{n - d_H(\mathbf{r}_t, \mathbf{a}_t)}$$
$$= \left(\frac{p_c}{1 - p_c}\right)^{d_H(\mathbf{r}_t, \mathbf{a}_t)} (1 - p_c)^n,$$

where $[r_t^{(i)} \neq a_t^{(i)}]$ returns 1 if the indicated condition is true and $d_H(\mathbf{r}_t, \mathbf{a}_t)$ is the Hamming distance.

Since the sequence of inputs uniquely determines the sequence of outputs, mapped outputs, and states, the likelihood function can be equivalently expressed as $f(\mathbf{r}|\mathbf{c})$ or $f(\mathbf{r}|\mathbf{a})$ or $f(\mathbf{r}|\{\Psi_0, \Psi_1, \ldots, \Psi_L\})$.

- Maximizing the likelihood is obviously equivalent to minimizing the negative log likelihood. We deal with negative log likelihood functions and throw away terms and/or factors that do not depend upon the conditioning values. For the Gaussian channel, since

$$-\log f(\mathbf{r}_t|\mathbf{a}_t) = +n \log \sqrt{2\pi}\sigma + \frac{1}{2\sigma^2}\|\mathbf{r}_t - \mathbf{a}_t\|^2$$

we use

$$\|\mathbf{r}_t - \mathbf{a}_t\|^2 \tag{12.14}$$

as the "negative log likelihood" function. For the BSC, since

$$-\log f(\mathbf{r}_t|\mathbf{a}_t) = -d_H(\mathbf{r}_t, \mathbf{a}_t) \log \frac{p_c}{1 - p_c} - n \log(1 - p_c)$$

we use

$$d_H(\mathbf{r}_t, \mathbf{a}_t) \tag{12.15}$$

as the "negative log likelihood" (since $\log(p_c/(1 - p_c)) < 0$).

More generally, the affine transformation

$$a[-\log f(\mathbf{r}_t|\mathbf{a}) - b] \tag{12.16}$$

provides a function equivalent for purposes of detection to the log likelihood function for any $a > 0$ and any b. The parameters a and b can be chosen to simplify computations.

- The state at time t in the trellis of the encoder is denoted as Ψ_t. States are represented with integer values in the range $0 \leq \Psi_t < 2^\nu$, where ν is the constraint length for the encoder. (We use 2^ν since we are assuming binary encoders for convenience. For a q-ary code, the number of states is q^ν.) It is always assumed that the initial state is $\Psi_0 = 0$.
- As suggested by Figure 12.10, quantities associated with the transition from state p to state q are denoted with $^{(p,q)}$. For example, the input which causes the transition from state $\Psi_t = p$ to the state $\Psi_{t+1} = q$ is denoted as $\mathbf{x}^{(p,q)}$. (If the trellis had different structure at different times, one might use the notation $\mathbf{x}_t^{(p,q)}$.) The code bits output sequence as a result of this state transition is $\mathbf{c}^{(p,q)}$ and the mapped symbols are $\mathbf{a}^{(p,q)}$.
- A sequence of symbols such as $\{\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_l\}$ is denoted as $\mathbf{x}_0^l$.

12.3.2 The Viterbi Algorithm

The Viterbi algorithm was originally proposed by Andrew Viterbi [358], but its optimality as a maximum likelihood sequence decoder was not originally appreciated. In [89] it was established that the Viterbi algorithm computes the maximum likelihood code sequence given the received data. The Viterbi algorithm is essentially a shortest path algorithm, roughly

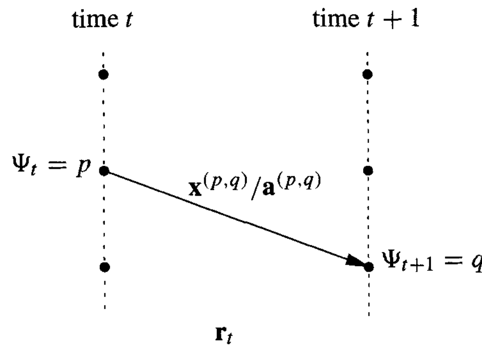


Figure 12.10: Notation associated with a state transition.

analogous to Dijkstra's shortest path algorithm (see, e.g., [305, p. 415]) for computing the shortest path through the trellis associated with the code. The Viterbi algorithm has been applied in a variety of other communications problems, including maximum likelihood sequence estimation in the presence of intersymbol interference [96] and optimal reception of spread-spectrum multiple access communication (see, e.g., [354]). It also appears in many other problems where a "state" can be defined, such as in hidden Markov modeling (see, e.g., [67]). See also [246] for a survey of applications. The decoder takes the input sequence $\mathbf{r} = \{\mathbf{r}_0, \mathbf{r}_1, \ldots\}$ and determines an estimate of the transmitted data $\{\mathbf{a}_0, \mathbf{a}_1, \ldots\}$ and from that an estimate of the sequence of input data $\{\mathbf{x}_0, \mathbf{x}_1, \ldots\}$.

The basic idea behind the Viterbi algorithm is as follows. A coded sequence $\{\mathbf{c}_0, \mathbf{c}_1, \ldots\}$, or its signal-mapped equivalent $\{\mathbf{a}_0, \mathbf{a}_1, \ldots\}$, corresponds to a path through the encoder trellis. Due to noise in the channel, the received sequence $\mathbf{r}$ may not correspond exactly to a path through the trellis. The decoder finds a path through the trellis which is closest to the received sequence, where the measure of "closest" is determined by the likelihood function appropriate for the channel. In light of (12.14), for an AWGN channel the maximum likelihood path corresponds to the path through the trellis which is closest in *Euclidean* distance to $\mathbf{r}$. In light of (12.15), for a BSC the maximum likelihood path corresponds to the path through the trellis which is closest in *Hamming* distance to $\mathbf{r}$. Naively, one could find the maximum likelihood path by computing separately the path lengths of all of the possible paths through the trellis. This, however, is computationally intractable. The Viterbi algorithm organizes the computations in an efficient recursive form.

For an input $\mathbf{x}_t$ the output $\mathbf{c}_t$ depends on the state of the encoder Ψ_t, which in turn depends upon previous inputs. This dependency among inputs means that optimal decisions cannot be made based upon a likelihood function for a single time $f(\mathbf{r}_t|\mathbf{x}_t)$. Instead, optimal decisions are based upon an entire received *sequence* of symbols. The likelihood function to be maximized is thus $f(\mathbf{r}|\mathbf{x})$, where

$$f(\mathbf{r}|\mathbf{x}) = f(\mathbf{r}_0^{L-1}|\mathbf{x}_0^{L-1}) = f(\mathbf{r}_0, \mathbf{r}_1, \ldots, \mathbf{r}_{L-1}|\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_{L-1}) = \prod_{t=0}^{L-1} f(\mathbf{r}_t|\mathbf{x}_t).$$

The fact that the channel is assumed to be memoryless is used to obtain the last equality. It

is convenient to deal with the log likelihood function,

$$\log f(\mathbf{r}|\mathbf{x}) = \sum_{t=0}^{L-1} \log f(\mathbf{r}_t|\mathbf{x}_t).$$

Consider now a sequence $\hat{\mathbf{x}}_0^{t-1} = \{\hat{\mathbf{x}}_0, \hat{\mathbf{x}}_1, \ldots, \hat{\mathbf{x}}_{t-1}\}$ which leaves the encoder in state $\Psi_t = p$ at time t. This sequence determines a path — or sequence of states — through the trellis for the code, which we denote (abstractly) as Π_t or $\Pi_t(\hat{\mathbf{x}}_0^{t-1})$. Thus

$$\Pi_t = \{\Psi_0, \Psi_1, \ldots, \Psi_t\}.$$

The log likelihood function for this sequence is

$$\log f(\mathbf{r}_0^{t-1}|\hat{\mathbf{x}}_0^{t-1}) = \sum_{i=0}^{t-1} \log f(\mathbf{r}_i|\hat{\mathbf{x}}_i).$$

Let $M_{t-1}(p) = -\log f(\mathbf{r}_0^{t-1}|\hat{\mathbf{x}}_0^{t-1})$ denote the **path metric** for the path Π_t through the trellis defined by the sequence $\hat{\mathbf{x}}_0^{t-1}$ and terminating in state p. (We could write $M_{t-1}(p; \Pi_t)$ or $M_{t-1}(p; \hat{\mathbf{x}}_0^{t-1})$ to indicate that the metric depends on the path but this leads to an awkward notation.) The negative sign in this definition means that we seek to *minimize* the path metric (to maximize the likelihood).

Now let the sequence $\hat{\mathbf{x}}_0^t = \{\hat{\mathbf{x}}_0, \hat{\mathbf{x}}_1, \ldots, \hat{\mathbf{x}}_t\}$ be obtained by appending the input $\hat{\mathbf{x}}_t$ to $\hat{\mathbf{x}}_0^{t-1}$ and suppose the input $\hat{\mathbf{x}}_t$ is such that the state at time $t+1$ is $\Psi_{t+1} = q$. The path metric for this longer sequence is

$$M_t(q) = -\sum_{i=0}^{t} \log f(\mathbf{r}_i|\hat{\mathbf{x}}_i) = -\sum_{i=0}^{t-1} \log f(\mathbf{r}_i|\hat{\mathbf{x}}_i) - \log f(\mathbf{r}_t|\hat{\mathbf{x}}_t) = M_{t-1}(p) - \log f(\mathbf{r}_t|\hat{\mathbf{x}}_t).$$

Let $\mu_t(\mathbf{r}_t, \hat{\mathbf{x}}_t) = -\log f(\mathbf{r}_t|\hat{\mathbf{x}}_t)$ denote the negative log likelihood for this input. As pointed out in (12.16), we could equivalently use

$$\mu_t(\mathbf{r}_t, \hat{\mathbf{x}}_t) = a[-\log f(\mathbf{r}_t|\hat{\mathbf{x}}^{(p,q)}) - b], \tag{12.17}$$

for any $a > 0$. The quantity $\mu_t(\mathbf{r}_t, \hat{\mathbf{x}}_t)$ is called the **branch metric** for the decoder. Since $\hat{\mathbf{x}}_t$ moves the trellis from state p at time t to state q at time $t+1$, we can write $\mu_t(\mathbf{r}_t, \hat{\mathbf{x}}_t)$ as $\mu_t(\mathbf{r}_t, \hat{\mathbf{x}}^{(p,q)})$. Then

$$M_t(q) = \sum_{i=0}^{t-1} \mu_t(\mathbf{r}_i, \hat{\mathbf{x}}_i) + \mu_t(\mathbf{r}_t, \hat{\mathbf{x}}_t) = M_{t-1}(p) + \mu_t(\mathbf{r}_t, \hat{\mathbf{x}}^{(p,q)}).$$

That is, the path metric along a path to state q at time t is obtained by adding the path metric to the state p at time $t-1$ to the branch metric for an input which moves the encoder from state p to state q. (If there is no such input then the branch metric is ∞.)

With this notation, we now come to the crux of the Viterbi algorithm: What do we do when paths merge? Suppose $M_{t-1}(p_1)$ is the path metric of a path ending at state p_1 at time t and $M_{t-1}(p_2)$ is the path metric of a path ending at state p_2 at time t. Suppose further that both of these states are connected to state q at time $t+1$, as suggested in Figure 12.11. The resulting path metrics to state q are

$$M_{t-1}(p_1) + \mu_t(\mathbf{r}_t, \hat{\mathbf{x}}^{(p_1,q)}) \quad \text{and} \quad M_{t-1}(p_2) + \mu_t(\mathbf{r}_t, \hat{\mathbf{x}}^{(p_2,q)}).$$

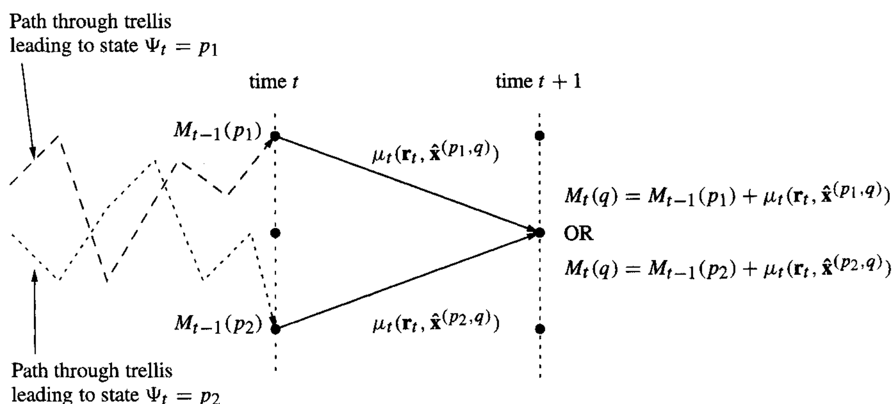


Figure 12.11: The Viterbi step: Select the path with the best metric.

The governing principle of the Viterbi algorithm is this: To obtain the shortest path through the trellis, the path to state q must be the shortest possible. Otherwise, it would be possible to find a shorter path through the trellis by finding a shorter path to state q. (This is Bellman's principle of optimality; see, e.g., [17].) Thus, when the two or more paths merge, the path with the *shortest* path metric is retained and the other path is eliminated from further consideration. That is,

$$M_t(q) = \min\{M_{t-1}(p_1) + \mu_t(\mathbf{r}_t, \hat{\mathbf{x}}^{(p_1,q)}), M_{t-1}(p_2) + \mu_t(\mathbf{r}_t, \hat{\mathbf{x}}^{(p_2,q)})\}$$

and the path with minimal length becomes the path to state q. This is called the **survivor path.**

Since it is not known at time $t < L$ which states the final path passes through, the paths to *each* state are found for each time. The Viterbi algorithm thus maintains the following data:

- A path metric to each state at time t.
- A path to each state at time t.

The Viterbi algorithm is thus summarized as follows:

1. For each state q at time $t+1$, find the path metric for each path to state q by adding the path metric $M_{t-1}(p)$ of each survivor path to state p at time t to the branch metric $\mu_t(\mathbf{r}_t, \hat{\mathbf{x}}^{(p,q)})$.
2. The survivor path to q is selected as that path to state q which has the smallest path metric.
3. Store the path and path metric to each state q.
4. Increment t and repeat until complete.

In the event that the path metrics of merging paths are equal, a random choice can be made with no negative impact on the likelihood.

More formally, there is the description in Algorithm 12.1. In this description, the path to state q is specified by listing for each state its predecessor in the graph. Other descriptions of the path are also possible. The algorithm is initialized reflecting the assumption that the initial state is 0 by setting the path metric at state 0 to 0 and all other path metrics to ∞ (i.e., some large number).

Algorithm 12.1 The Viterbi Algorithm

1 **Input:** A sequence $\mathbf{r}_0, \mathbf{r}_1, \ldots, \mathbf{r}_{L-1}$
2 **Output:** The sequence $\hat{\mathbf{x}}_0, \hat{\mathbf{x}}_1, \ldots, \hat{\mathbf{x}}_{L-1}$ which maximizes the likelihood $f(\mathbf{r}_0^{L-1}|\hat{\mathbf{x}}_0^{L-1})$.
3 **Initialize:** Set $M(0) = 0$ and $M(p) = \infty$ for $p = 1, 2, \ldots, 2^\nu - 1$ (initial path costs)
4 Set $\Pi_p = \emptyset$ for $p = 0, 1, \ldots, 2^\nu - 1$ (initial paths)
5 Set $t = 0$
6 Begin
7 For each state q at time $t + 1$
8 Find the path metric for each path to state q:
9 for each p_i connected to state q corresponding to input $\hat{\mathbf{x}}^{(p_i,q)}$, compute
$m_i = M(p_i) + \mu_t(\mathbf{r}_t, \hat{\mathbf{x}}^{(p_i,q)}))$.
10 Select the smallest metric $M(q) = \min_i m_i$ and the corresponding predecessor state p.
11 Extend the path to state q: $\Pi_q = [\Pi_p \; p]$
12 end (for)
13 $t = t + 1$
14 if $t < L - 1$, goto line 6.
15 Termination:
16 If terminating in a known state (e.g. 0)
Return the sequences of inputs along the path to that known state
17 If terminating in any state
Find final state with minimal metric; Return the sequence of inputs along that path to that state.
18 End

The operations of extending and pruning that constitute the heart of the Viterbi algorithm are summarized as:

$$M_t(q) = \underbrace{\min_p \underbrace{\left[M_{t-1}(p) + \mu_t(\mathbf{r}_t, \hat{\mathbf{x}}^{(p,q)})\right]}_{\text{Extend all paths at time } t \text{ to state } q \ldots}}_{\text{Then choose smallest cost}}. \tag{12.18}$$

Example 12.13 Consider the encoder

$$G(x) = \begin{bmatrix} x^2 + 1 & x^2 + x + 1 \end{bmatrix}$$

of Example 12.1, whose realization and trellis diagram are shown in Figure 12.5, passing the data through a BSC. When the data sequence

$$\begin{aligned} \mathbf{m} &= [1, 1, 0, 0, 1, 0, 1, 0, \ldots] \\ &= [m_0, m_1, m_2, m_3, m_4, m_5, m_6, m_7, \ldots] \end{aligned}$$

is applied to the encoder, the coded output bit sequence is

$$\begin{aligned} \mathbf{c} &= [11, 10, 10, 11, 11, 01, 00, 01, \ldots] \\ &= [\mathbf{c}_0, \mathbf{c}_1, \mathbf{c}_2, \mathbf{c}_3, \mathbf{c}_4, \mathbf{c}_5, \mathbf{c}_6, \mathbf{c}_7, \ldots]. \end{aligned}$$

For the BSC, we take the mapped data the same as the encoder output $\mathbf{a}_t = \mathbf{c}_t$. The output sequence and corresponding states of the encoder are shown here, where $\Psi_0 = 0$ is the initial state.

t	Input m_k	Output $\mathbf{c}_t$	State Ψ_{t+1}
0	1	11	1
1	1	10	3
2	0	10	2
3	0	11	0
4	1	11	1
5	0	01	2
6	1	00	1
7	0	01	2

The sequence of states through the trellis for this encoder is shown in Figure 12.12; the solid line shows the state sequence for this sequence of outputs. The coded output sequence passes through a

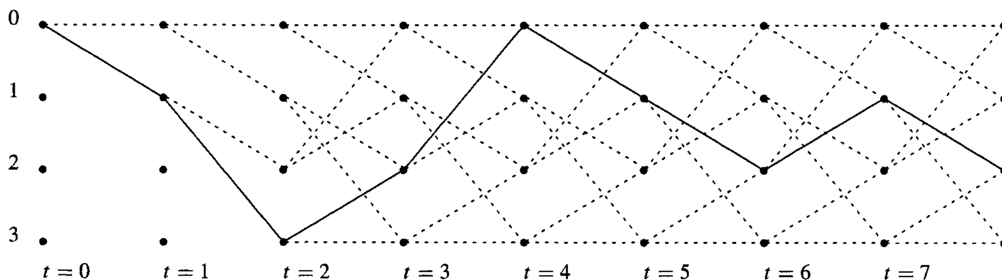


Figure 12.12: Path through trellis corresponding to true sequence.

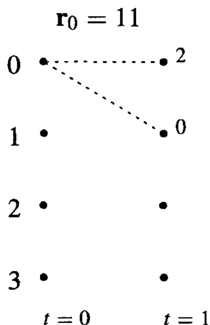
channel, producing the received sequence

$$\mathbf{r} = [11\ 10\ \underline{0}0\ 1\underline{0}\ 11\ 01\ 00\ 01\ \ldots] = [\mathbf{r}_0, \mathbf{r}_1, \mathbf{r}_2, \mathbf{r}_3, \mathbf{r}_4, \mathbf{r}_5, \mathbf{r}_6, \mathbf{r}_7, \ldots].$$

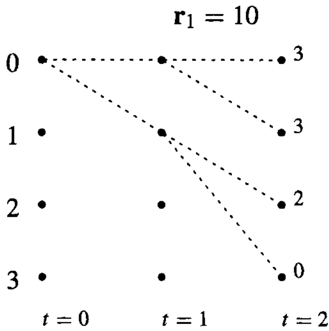
The two underlined bits are flipped by noise in the channel.

The algorithm proceeds as follows:

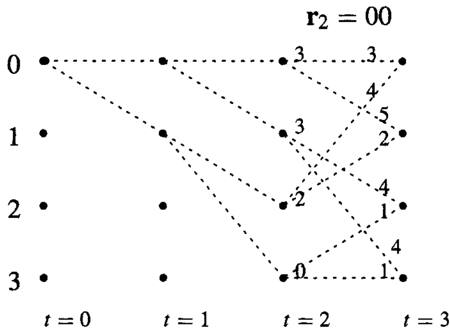
$t = 0$: The received sequence is $\mathbf{r}_0 = 11$. We compute the metric to each state at time $t = 1$ by finding the (Hamming) distance between $\mathbf{r}_0$ and the possible transmitted sequence $\mathbf{c}_0$ along the branches of the first stage of the trellis. Since state 0 was known to be the initial state, we end up with only two paths, with path metrics 2 and 0, as shown here:



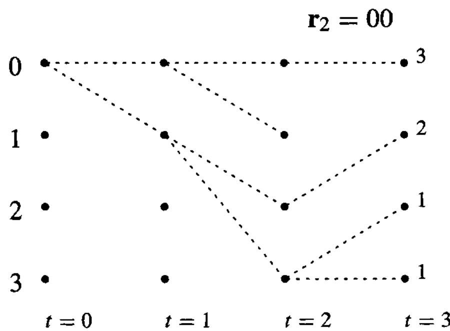
$t = 1$: The received sequence is $\mathbf{r}_1 = 10$. Again, each path at time $t = 1$ is simply extended, adding the path metric to each branch metric:



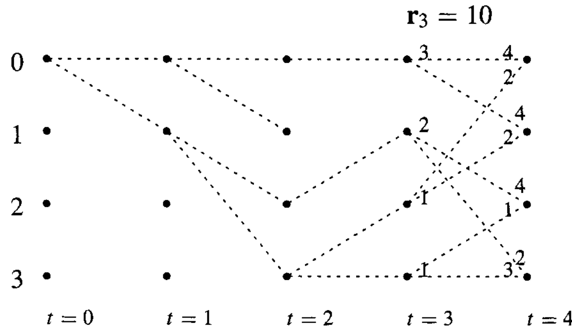
$t = 2$: The received sequence is $\mathbf{r}_2 = 00$. Each path at time $t = 2$ is extended, adding the path metric to each branch metric of each path.



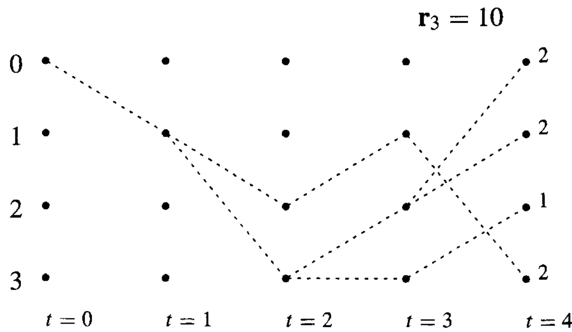
There are now multiple paths to each node at time $t = 3$. We select the path to each node with the best metric and eliminate the other paths. This gives the diagram as follows:



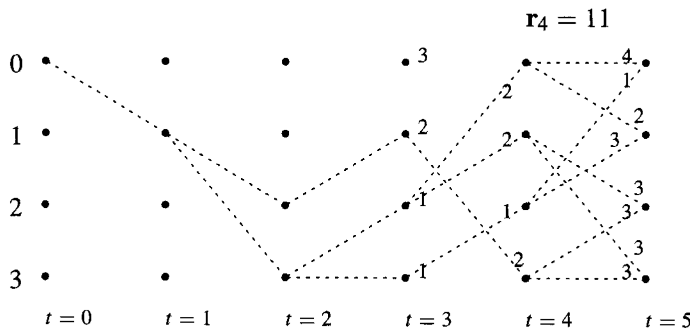
$t = 3$: The received sequence is $\mathbf{r}_3 = 10$. Each path at time $t = 3$ is extended, adding the path metric to each branch metric of each path.



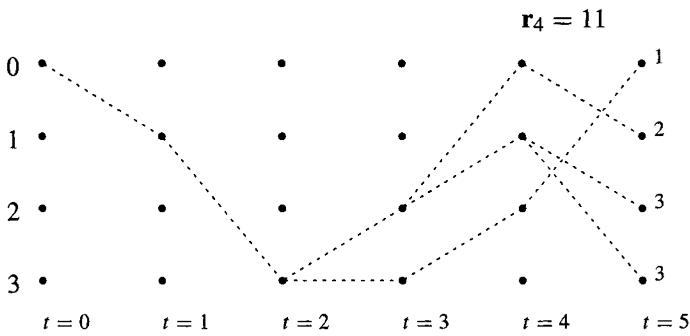
Again, the best path to each state is selected. We note that in selecting the best paths, some of the paths to some states at earlier times have no successors; these orphan paths are deleted now in our portrayal:



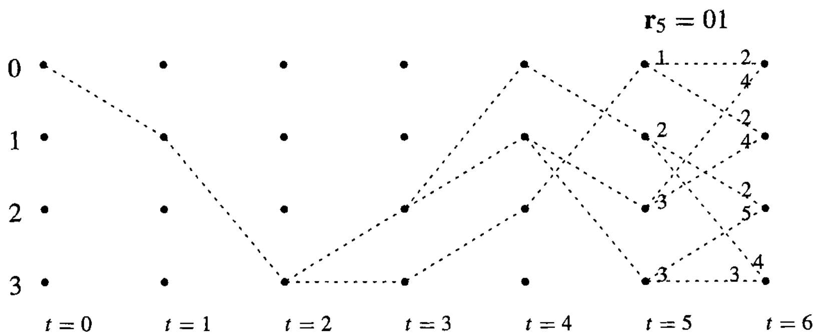
$t = 4$: The received sequence is $\mathbf{r}_4 = 11$. Each path at time $t = 4$ is extended, adding the path metric to each branch metric of each path.



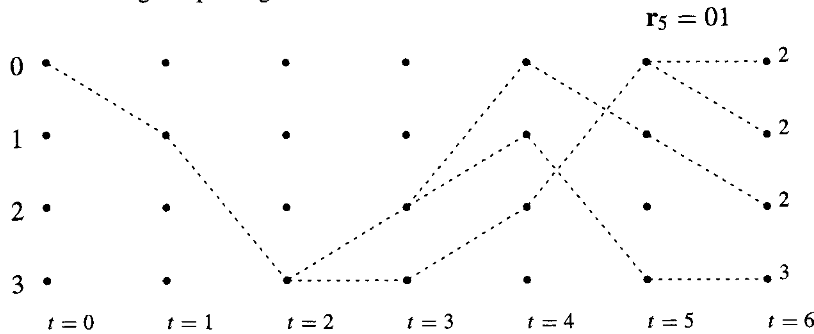
In this case, we note that there are multiple paths into state 3 which both have the same path metric; also there are multiple paths into state 2 with the same path metric. Since one of the paths must be selected, the choice can be made arbitrarily (e.g., at random). After selecting and pruning of orphan paths we obtain:



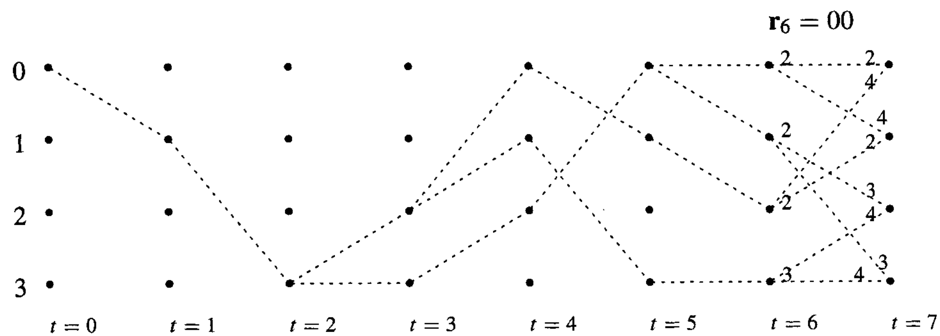
$t = 5$**:** The received sequence is $\mathbf{r}_5 = 01$.



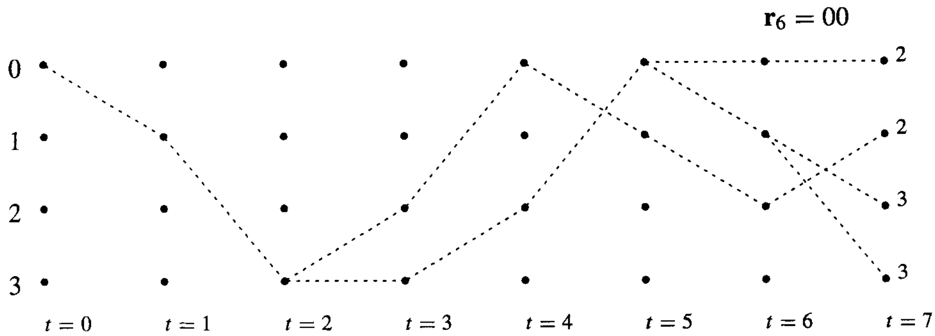
After selecting and pruning:



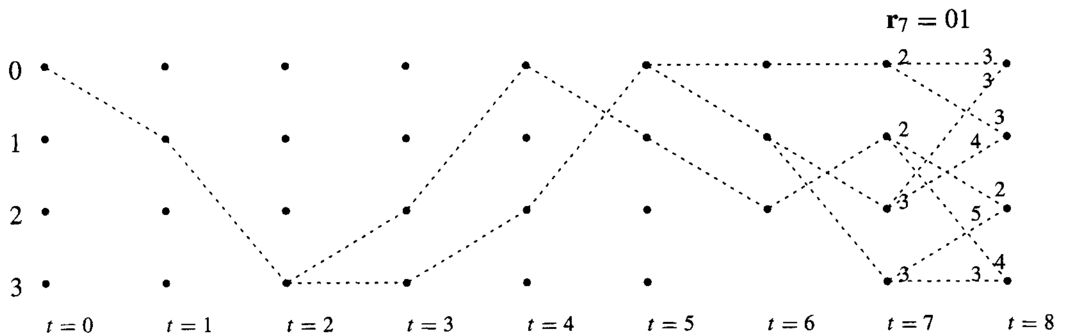
$t = 6$**:** The received sequence is $\mathbf{r}_6 = 00$.



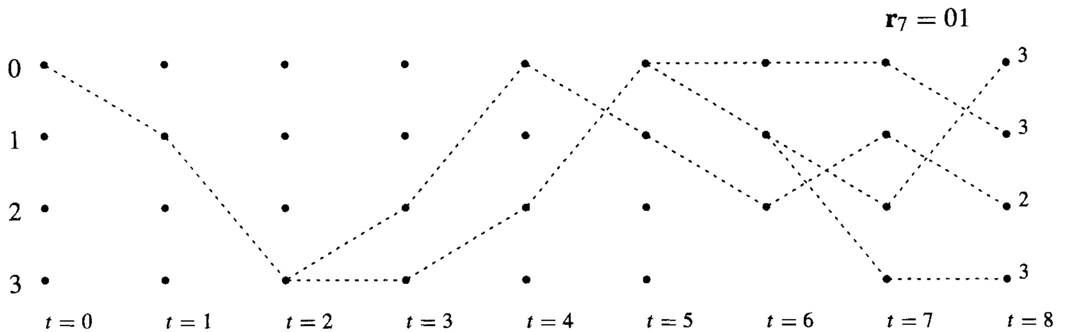
After selecting and pruning:



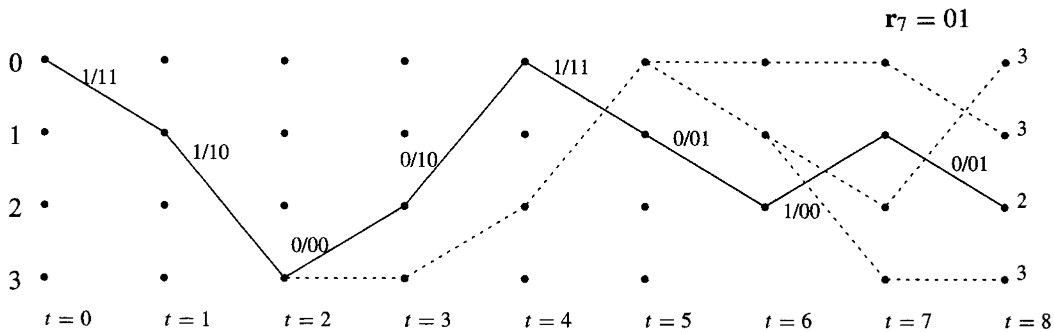
$t = 7$: The received sequence is $\mathbf{r}_7 = 01$.



After selecting and pruning:



The decoding is finalized at the end of the transmission (the 16 received data bits) by selecting the state at the last stage having the lowest cost, traversing backward along the path so indicated to the beginning of the trellis, then traversing forward again along the best path, reading the input bits and decoded output bits along the path. This is shown with the solid line below; input/output pairs are indicated on each branch.



Note that the path through the trellis is the same as in Figure 12.12 and that the recovered input bit sequence is the same as the original bit sequence. Thus, out of this sequence of 16 bits, two bit errors have been corrected. □

12.3.3 Some Implementation Issues

The Basic Operation: Add-Compare-Select

The basic operation of the Viterbi algorithm is Add-Compare-Select (ACS): Add the branch metric to the path metric for each path leading to a state; compare the resulting path metrics at that state; and select the better metric. A schematic of this idea appears in Figure 12.13. High-speed operation can be obtained in hardware by using a bank of 2^ν such ACS units in parallel. A variation on this theme, the compare-select-add (CSA) operation, is capable of somewhat improving the speed for some encoders. The algorithm employing CSA is called the *differential Viterbi algorithm*; see [105] for a description.

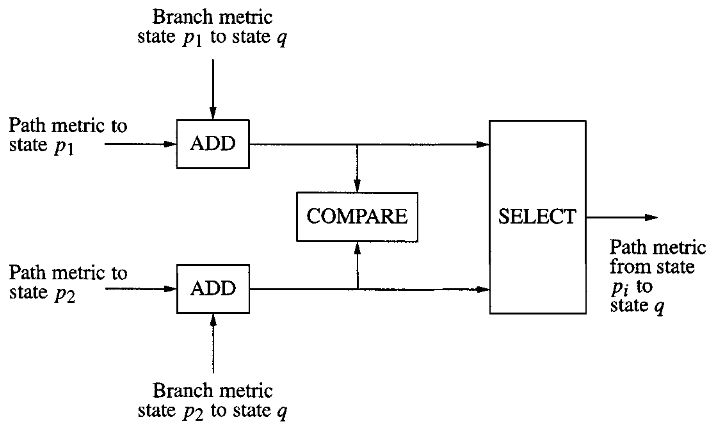


Figure 12.13: Add-compare-select Operation.

Decoding Streams of Data: Windows on the Trellis

In convolutional codes, data are typically encoded in a stream. Once the encoding starts, it may continue indefinitely, for example until the end of a file or until the end of a data transmission session. If such a data stream is decoded using the Viterbi algorithm as described above, the paths through the trellis would have to have as many stages as the code is long.

For a long data stream, this could amount to an extraordinary amount of data to be stored, since the decoder would have to store 2^ν paths whose lengths grow longer with each stage. Furthermore, this would result in a large decoding latency: strictly speaking it would not be possible to output *any* decoded values until the maximum likelihood path is selected at the end of the file.

Fortunately, it is not necessary to wait until the end of transmission. Consider the paths in Example 12.13. In this example, by $t = 4$, there is a single surviving path in the first two stages of the trellis. Regardless of how the Viterbi algorithm operates on the paths as it continues through the trellis, those first two stages could be unambiguously decoded.

In general, with very high probability there is a single surviving path some number of stages back from the "current" stage of the trellis. The initial stages of the survivor paths tend to merge if a sufficient decoding delay is allowed. Thus, it is only necessary to keep a "window" on the trellis consisting of the current stage and some number of previous stages. The number of stages back that the decoding looks to make its decision is called the **decoding depth**, denoted by Γ. At time t the decoder outputs a decision on the code bits $\mathbf{c}_{t-\Gamma}$. While it is possible to make an incorrect decoding decision on a finite decoding depth, this error, called the **truncation error**, is typically very small if the decoding depth is sufficiently large. It has been found (see, e.g., [90, 149]) that if a decoding depth of about five to ten constraint lengths is employed, then there is very little loss of performance compared to using the full length due to truncation error.

It is effective to implement the decoding window using a circular queue of length Γ to hold the current window. As the window is "shifted," it is only necessary to adjust the pointers to the beginning and end of the window.

As the algorithm proceeds through the stream of data, the path metrics continue to accumulate. Overflow is easily avoided by periodically subtracting from all path metrics an equal amount (for example, the smallest path metric). The path metrics then show the differential qualities of each path rather than the absolute metrics (or their approximations), but this is sufficient for decoding purposes.

Output Decisions

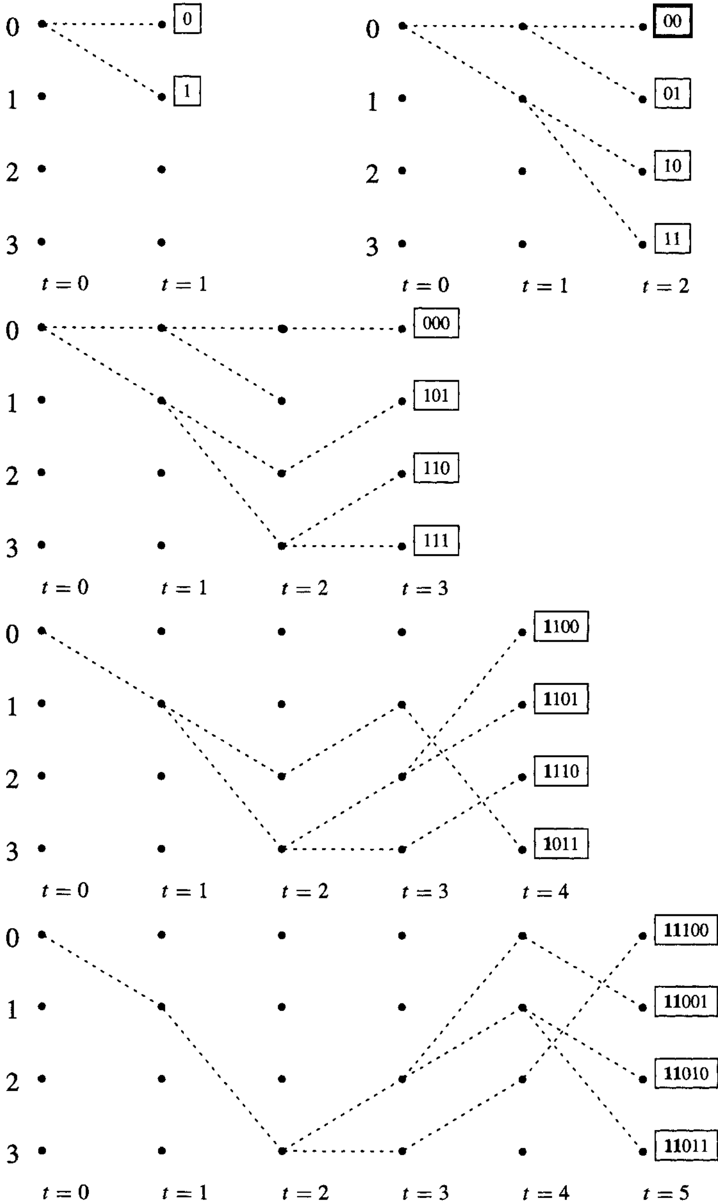
When a decision about the output $\mathbf{c}_{t-\Gamma}$ is to be made at time t, there are a few ways that this can be accomplished [373]: Output $\mathbf{c}_{t-\Gamma}$ on a randomly selected survivor path; Output $\mathbf{c}_{t-\Gamma}$ on the survivor path with the best metric; Output $\mathbf{c}_{t-\Gamma}$ that occurs most often among all the survivor paths; Output $\mathbf{c}_{t-\Gamma}$ on any path. In reality, if Γ is sufficiently large that all the survivor paths have merged Γ decoding stages back, then the performance difference among these alternatives is very small.

When the decision is to be output, a survivor path is selected (using one of the methods just mentioned). Then it is necessary to determine the $\mathbf{c}_{t-\Gamma}$. There are a couple of ways of accomplishing this, the register exchange and the traceback.

In the *register exchange* implementation, an input register at each state contains the sequence of input bits associated with the surviving path that terminates at that state. A register capable of storing the $k\Gamma$ bits is necessary for each state. As the decoding algorithm proceeds, for the path selected from state p to state q, the input register at state q is obtained by copying the input register for state p and appending the k input bits resulting in that state transition. (Double buffering of the data may be necessary, so that the input registers are not lost in copying the information over.) When an output is necessary, the first k bits of the register for the terminating state of the selected path can be read immediately from its

input register.

Example 12.14 For the decoding sequence of Example 12.13, the registers for the register exchange algorithm are shown here (boxed) for the first five steps of the algorithm.



□

The number of initial bits all the registers have in common is the number of branches of the path that are shared by all paths. These common input bits are shown in bold above. In some implementations, it may be of interest to output only those bits corresponding to path branches which have merged.

In the *traceback* method, the path is represented by storing the *predecessor* to each state. This sequence of predecessors is traced back Γ stages. The state transition $\Psi_{t-\Gamma}$ to $\Psi_{t-\Gamma+1}$ determines the output $\mathbf{c}_{t-\Gamma}$ and its corresponding input bits $\mathbf{x}_{t-\Gamma}$.

Example 12.15 The table

t:	1	2	3	4	5	6	7	8
State	Previous State/Input							
0:	0/0	0/0	0/0	**2/0**	2/0	0/0	0/0	2/0
1:	**0/1**	0/1	2/1	2/1	**0/1**	0/1	**2/1**	0/1
2:	-	1/0	**3/0**	3/0	1/0	**1/0**	1/0	**1/0**
3:	-	**1/1**	3/1	2/1	1/1	3/1	1/1	3/1

shows the previous state traceback table which would be built up by the decoding of Example 12.13. For example, at time $t = 8$, the predecessor of state 0 is 2, the predecessor of state 1 is 0, and so forth. Starting from state 2 (having the lowest path cost), the sequence of states can be read off in reverse order from this table (the bold entries):

$$2 \to 1 \to 2 \to 1 \to 0 \to 2 \to 3 \to 1 \to 0.$$

Thus the first state transition is from state 0 to state 1 and the input at that time is a 1.

The inputs for the entire sequence can also be read off, starting at the right, 11001010. □

In the traceback method, it is necessary to trace backward through the trellis once for each output. (As a variation on the theme, the predecessor to a state could be represented by the input bits that lead to that state.)

In comparing the requirements for these two methods, we note that the register exchange method requires shuffling registers among all the states at each time. In contrast, the traceback method requires no such shuffling, but it does require working back through the trellis to obtain a decision. Which is more efficient depends on the particular hardware available to perform the tasks. Typically, the traceback is regarded as faster but more complicated.

Hard and Soft Decoding; Quantization

Example 12.13 presents an instance of *hard-decision decoding*. If the outputs of a Gaussian channel had been used with the Euclidean distance as the branch metric, then *soft-decision* decoding could have been obtained. Comparing soft decision decoding using BPSK over an AWGN with hard decision decoding over a BSC, in which received values are converted to binary values with a probability of error of $p_c = Q(\sqrt{2E_b/N_0})$ (see Section 1.5.6), it has been determined that soft-decision decoding provides 2 to 3 dB of gain over hard-decision decoding.

For a hard-decision metric, $\mu_t(\mathbf{r}_t, \mathbf{x}^{(p,q)})$ can be computed and stored in advance. For example, for an $n = 2$ binary code, there are four possible received values, 00, 01, 10, 11, and four possible transmitted values. The metric could be stored in a 4×4 array, such as the following.

$\mu_t(\mathbf{r}_t, \hat{\mathbf{x}}^{(p,q)})$	$\mathbf{r}_t = 00$	01	10	11
$\mathbf{x}^{(p,q)} = 00$	0	1	1	2
$\mathbf{x}^{(p,q)} = 01$	1	0	2	1
$\mathbf{x}^{(p,q)} = 10$	1	2	0	1
$\mathbf{x}^{(p,q)} = 11$	2	1	1	0

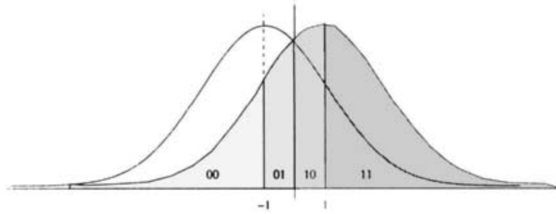


Figure 12.14: A two-bit quantization of the soft-decision metric.

Soft-decision decoding typically requires more expensive computation than hard-decision decoding. Furthermore, soft-decision decoding cannot exactly precompute these values to reduce the ongoing decoding complexity, since $\mathbf{r}_t$ takes on a continuum of values. Despite these disadvantages, it is frequently desirable to use soft-decision decoding because of its superior performance. A computational compromise is to *quantize* the received value to a reasonably small set of values, then precompute the metrics for each of these values. By converting these metrics to small integer quantities, it is possible to efficiently accumulate the metrics. It has been found [148] that quantizing each $r_t^{(i)}$ into 3 bits (eight quantization levels) results in a loss in coding gain of around only 0.25 dB. It is possible to trade metric computation complexity for performance, using more bits of quantization to reduce the loss.

As noted above, if a branch metric μ is modified by $\tilde{\mu} = a\mu + b$ for any $a > 0$ and any real b, an equivalent decoding algorithm is obtained; this simply scales and shifts the resulting path metrics. In quantizing, it may be convenient to find scale factors which make the arithmetic easier.

A widely used quantizer is presented in Section 12.4.

Example 12.16 A two-bit quantizer. In a BPSK-modulated system the transmitted signal amplitudes are $a = 1$ or $a = -1$. The received signal r_t is quantized by a quantization function $Q[\cdot]$ to obtain quantized values

$$q_t = Q[r_t]$$

using quantization thresholds at ± 1 and 0, as shown in Figure 12.14, where the quantized values are denoted as 00, 01, 10 and 11. These thresholds determine the regions $\mathcal{R}_q$. That is,

$$q_t = Q[r_t] = \begin{cases} 00 & \text{if } r_t \in \mathcal{R}_{00} = (-\infty, -1] \\ 01 & \text{if } r_t \in \mathcal{R}_{01} = (-1, 0] \\ 10 & \text{if } r_t \in \mathcal{R}_{10} = (0, 1] \\ 11 & \text{if } r_t \in \mathcal{R}_{11} = (1, \infty). \end{cases}$$

(This is not an optimal quantizer, merely convenient.) For each quantization bin we can compute the likelihood that r_t falls in that region, given a particular input, as

$$P(q_t|a) = \int_{\mathcal{R}_{q_t}} f_R(r|a)\,dr.$$

For example,

$$P(00|a = -1) = \int_{-\infty}^{-1} f_R(r|-1)\,dr = \frac{1}{2}.$$

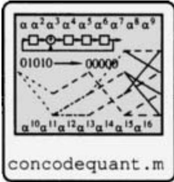
Suppose the likelihoods for all quantized points are computed as follows.

$P(q_t\|a)$	$q_t =$	00	01	10	11
$a = 1$		0.02	0.14	0.34	0.5
$a = -1$		0.5	0.34	0.14	0.02

The – log probabilities are

$-\log(P(q_t\|a))$	$q_t =$	00	01	10	11
$a = 1$		3.5066	2.0402	1.0788	0.6931
$a = -1$		0.6931	1.0788	2.0402	3.5066

The logarithms of these probabilities can be approximated as integer values by computing



$$-1.619(\log P(q_t|a - \log P(00|1))$$

(the factor $a = 1.619$ was found by a simple computer search and b was chosen to make the smallest value 0), resulting in the following metrics:

$a(-\log(P(q_t\|a) - b)$	$q_t =$	00	01	10	11
$a = 1$		5.0028	2.109	0.618	0
$a = -1$		0	0.618	2.109	5.0028

which can be rounded to

$\lfloor a(-\log(P(q_t\|a) - b)\rfloor$	$q_t =$	00	01	10	11
$a = 1$		5	2	1	0
$a = -1$		0	1	2	5

Although the signal is quantized into two bits, the metric requires three bits to represent it. With additional loss of coding gain, this could be reduced to two bits of metric (reducing the hardware required to accumulate the path metrics). For example, the first row of the metric table could be approximated as 3, 2, 1, 0.

Note that, by the symmetry of the pdf and constellation, both rows of the table have the same values, so that in reality only a single row would need to be saved in an efficient hardware implementation. □

Synchronization Issues

The decoder must be synchronized with the stream of incoming data. If the decoder does not know which of the n symbols in a block initiates a branch of the trellis, then the data will be decoded with a very large number of errors. Fortunately, the decoding algorithm can detect this. If the data are correctly aligned with the trellis transitions, then with high probability, one (or possibly two) of the path metrics are significantly smaller than the other path metrics within a few stages of decoding. If this does not occur, the data can be shifted relative to the decoder and the decoding re-initialized. With at most n tries, the decoder can obtain symbol synchronization.

Many carrier tracking devices employed in communication systems experience a phase ambiguity. For a BPSK system, it is common that the phase is determined only up to $\pm\pi$, resulting in a sign change. For QPSK or QAM systems, the phase is often known only up to a multiple of $\pi/2$. The decoding algorithm can possibly help determine the absolute phase. For example, in a BPSK system if the all ones sequence is not a codeword, then for a given code sequence $\mathbf{c}$, $1 + \mathbf{c}$ cannot be a codeword. In this case, if decoding seems to indicate that no path is being decoded correctly (i.e., no path seems to emerge as a strong candidate compared to the other paths), then the receiver can complement all of its zeros and ones and decode again. If this decodes correctly the receiver knows that it has the phase off by π. For a QPSK system, four different phase shifts could be examined to see when correct decoding behavior emerges.

12.4 Some Performance Results

Bit error rate characterization of convolutional codes is frequently accomplished by simulation and approximation. In this section, we present performance as a function of quantization, constraint length, window length, and codeword size. These results generally follow [148], but have been recomputed here.

Quantization of the metric was discussed in the previous section. In the results here, a simpler quantized metric is used. Assume that BPSK modulation is employed and that the transmitted signal amplitude a is normalized to ± 1. The received signal r is quantized to m bits, resulting in $M = 2^m$ different quantization levels, using uniformly spaced quantization thresholds. The distance between quantization thresholds is Δ. Figure 12.15 shows 4-level quantization with $\Delta = 1$ and 8-level quantization using $\Delta = 0.5$ and $\Delta = \frac{1}{3}$. Rather than compute the log likelihood of the probability of falling in a decision region, in this approach the quantized q value itself is used as the branch metric if the signal amplitude 1 is sent, or the complement $M - q - 1$ is used if -1 is sent. The resulting integer branch metric is computed as shown in Table 12.1. This branch metric is clearly suboptimal, not being an affine transformation of the log likelihood. However, simulations have shown that it performs very close to optimal and it is widely used. Obviously, the performance depends upon the quantization threshold Δ employed. For the 8-level quantizer, the value $\Delta = 0.4$ is employed. For the 16-level quantizer, $\Delta = 0.25$ is used, and for the 4-level quantizer, $\Delta = 1$ is used. We demonstrate below the dependence of the bit-error rate upon Δ.

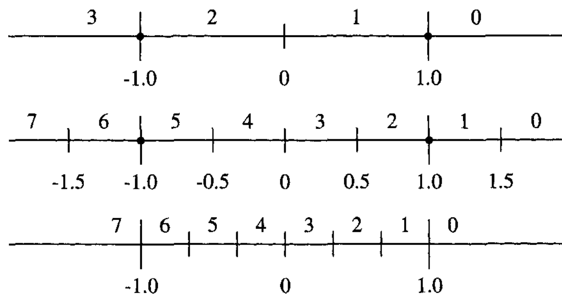
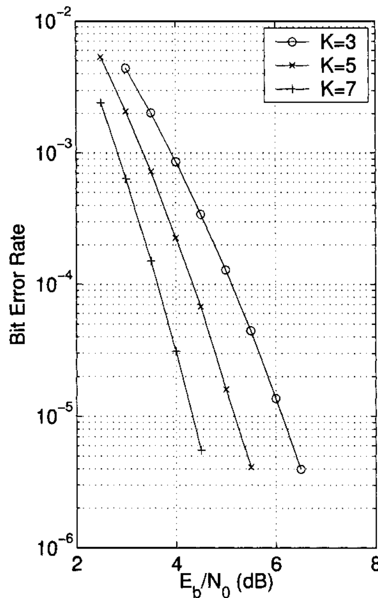


Figure 12.15: Quantization thresholds for 4- and 8-level quantization.

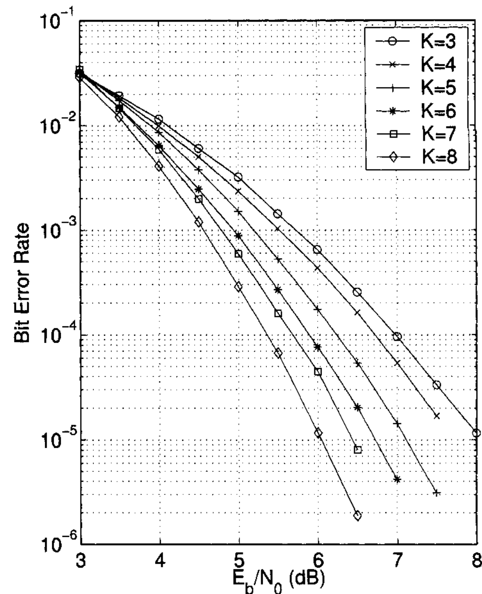
Table 12.1: Quantized Branch Metrics Using Linear Quantization

Signal Amplitude	Quantization Level 0	1	2	3	4	5	6	7
	Branch Metric μ							
-1	7	6	5	4	3	2	1	0
1	0	1	2	3	4	5	6	7

Figure 12.16(a) shows the bit error rate as a function of SNR for codes with constraint lengths (here employing $K = 1 + \max \deg(g_j)$ as the constraint length) of $K = 3$, $K = 5$ and $K = 7$ using 8-level uniform quantization with $\Delta = 0.42$. The generators employed



(a) 8-level quantization, $K = 3, 5, 7$.



(b) 1-bit (hard) quantization, $K = 3$ through 8.

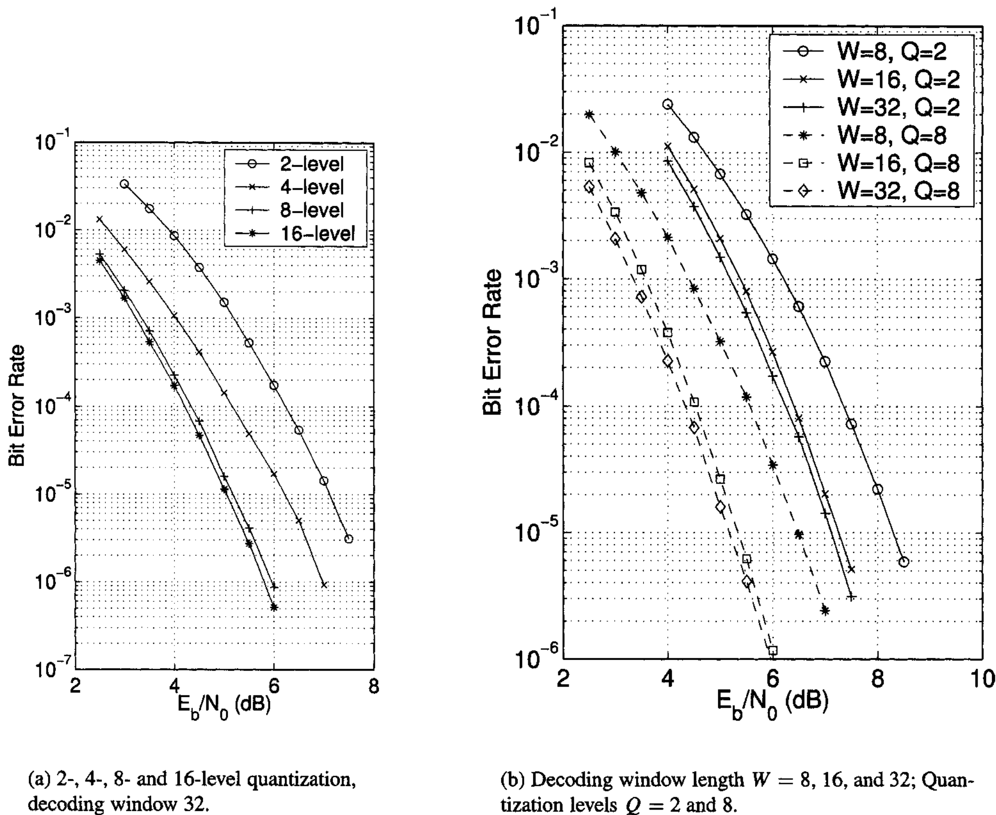
Figure 12.16: Bit error rate as a function of E_b/N_0 of $R = 1/2$ convolutional codes with 32 bit window decoding (following [148]).

in this and the other simulations are the following:

K	$g_1(x)$	$g_2(x)$	d_{free}
3	$1 + x^2$	$1 + x + x^2$	5
4	$1 + x + x^3$	$1 + x + x^2 + x^3$	6
5	$1 + x^3 + x^4$	$1 + x + x^2 + x^4$	7
6	$1 + x^2 + x^4 + x^5$	$1 + x + x^2 + x^3 + x^5$	8
7	$1 + x^2 + x^3 + x^5 + x^6$	$1 + x + x^2 + x^3 + x^6$	10
8	$1 + x^2 + x^5 + x^6 + x^7$	$1 + x + x^2 + x^3 + x^4 + x^7$	10

The Viterbi decoder uses a window of 32 bits. As this figure demonstrates, the performance improves with the constraint length. Figure 12.16(b) shows 1-bit (hard) quantization for $K = 3$ through 8.

Comparisons of the effect of the number of quantization levels and the decoding window are shown in Figure 12.17. In part (a), the performance of a code with $K = 5$ is shown with 2, 4, 8, and 16 quantization levels. As the figure demonstrates, there is very little improvement from 8 to 16 quantization levels; 8 is frequently chosen as an adequate performance/complexity tradeoff. In part (b), again a $K = 5$ code is characterized. In this case, the effect of the length of the decoding window is shown for two different quantization levels. With a decoding window of length 32, most of the achievable performance is attained.



(a) 2-, 4-, 8- and 16-level quantization, decoding window 32.

(b) Decoding window length $W = 8, 16$, and 32; Quantization levels $Q = 2$ and 8.

Figure 12.17: Bit error rate as a function of E_b/N_0 of $R = 1/2$, $K = 5$ convolutional code with different quantization levels and decoding window lengths (following [148]).

Figure 12.18 shows the effect of the quantizer threshold spacing Δ on the performance for an 8-level quantizer with a $K = 5$, $R = 1/2$ code and a $K = 5$, $R = 1/4$ code. The plots are at SNRs of 3.3 dB, 3.5 dB, and 3.7 dB (reading from top to bottom) for the $R = 1/2$ code and 2.75 dB, 2.98 dB, and 3.19 dB (reading from top to bottom) for the $R = 1/4$ code. (These latter SNRs were selected to provide roughly comparable bit error rate for the two codes.) These were obtained by simulating, counting 20,000 bit errors at each point of data.

A convolutional code can be employed as a block code by simply truncating the sequence at a block length N (see Section 12.9). This truncation results in the last few bits in the codeword not having the same level of protection as the rest of the bits, a problem referred to as unequal error protection. The shorter the block length N, the higher the fraction of unequally protected bits, resulting in a higher bit error rate. Figure 12.19 shows BER for maximum likelihood decoding of convolutional codes truncated to blocks of length $N = 200$, $N = 2000$, as well as the "conventional" mode in which the codeword simply streams.

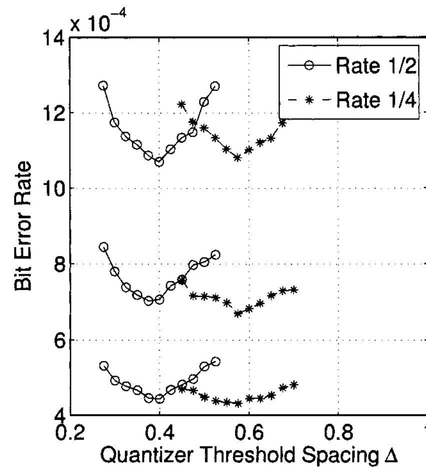


Figure 12.18: Viterbi algorithm bit error rate performance as a function of quantizer threshold spacing.

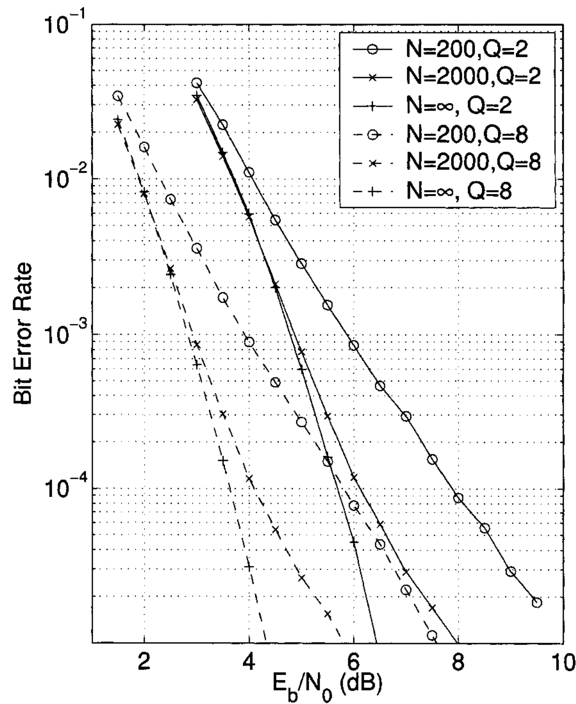


Figure 12.19: BER performance as a function of truncation block length, $N = 200$ and $N = 2000$, for 2- and 8-level quantization.

12.5 Error Analysis for Convolutional Codes

While for block codes it is conventional to determine (or estimate) the probability of decoding a block incorrectly, the performance of convolutional codes is largely determined by the rate and the constraint length. It is not very meaningful to determine the probability of a block in error, since the block may be very long. It is more useful to explore the *probability of bit error*, or the *bit error rate*, which is the average number of message bits in error in a given sequence of bits divided by the total number of message bits in the sequence. We shall denote the bit error rate by P_b. In this section we develop an upper bound for P_b [357].

Consider how errors can occur in the decoding process. The decision mechanism of the Viterbi algorithm operates when two paths join together. If two paths join together and the path with the lower (better) metric is actually the incorrect path, then an incorrect decision is made at that point. We call such an error a *node error* and say that the error event occurs at the place where the paths first diverged. We denote the probability of a node error as P_e. A node error, in turn, could lead to a number of input bits being decoded incorrectly.

Since the code is linear, it suffices to assume that the all-zero codeword is sent: With $d_H(\mathbf{r}, \mathbf{c})$ the Hamming distance between $\mathbf{c}$ and $\mathbf{r}$, we have $d_H(\mathbf{r}, \mathbf{c}) = d_H(\mathbf{r} + \mathbf{c}, \mathbf{c} + \mathbf{c}) = d_H(\mathbf{r} + \mathbf{c}, \mathbf{0})$. Consider the error events portrayed in Figure 12.20. The horizontal line across the top represents the all-zero path through the trellis. Suppose the path diverging from the all-zero path at a has a lower (better) metric when the paths merge at a'. This gives rise to an error event at a. Suppose that there are error events also at b and d. Now consider the path diverging at c: even if the metric is lower (better) at c', the diverging path from c may not ultimately be selected if its metric is worse than the path emerging at b. Similarly the path emerging at d may not necessarily be selected, since the path merging at e may take precedence. This overlapping of decision paths makes the exact analysis of the bit error rate difficult. We must be content with bounds and approximations.

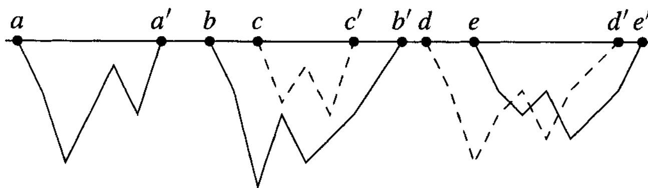


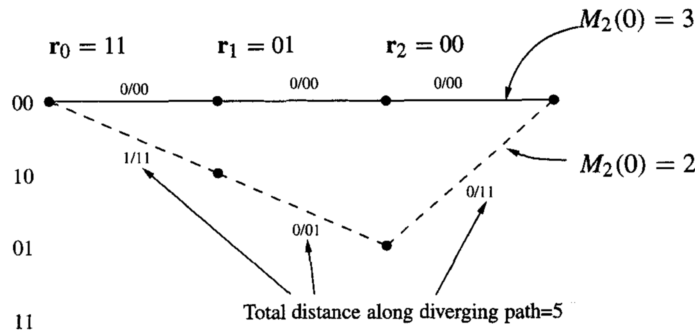
Figure 12.20: Error events due to merging paths.

The following example illustrates some of these issues.

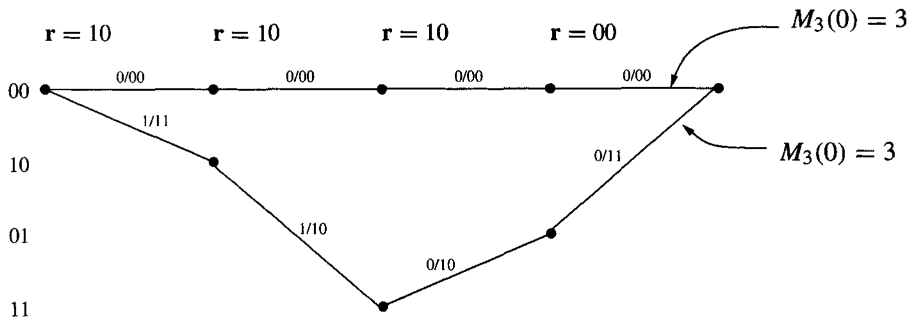
Example 12.17 Consider again the convolutional code from Example 12.1, with

$$G(x) = \left[1 + x^2, 1 + x + x^2\right].$$

Suppose that the input sequence is $\mathbf{x} = [0, 0, 0, 0, \ldots]$ with the resulting transmitted sequence $\mathbf{c} = [00, 00, 00, 00, \ldots]$, but that the received sequence after transmission through a BSC is $\mathbf{r} = [11, 01, 00, \ldots]$. A portion of the decoding trellis for this code is shown in Figure 12.21. After three stages of the trellis when the paths merge, the metric for the lower path (shown as a dashed line) is lower than the metric for the all-zero path (the solid line). Accordingly, the Viterbi algorithm selects the erroneous path, resulting in a node error at the first node. However, while the decision results



(a) Diverging path of distance 5.



(b) Diverging path of distance 6.

Figure 12.21: Two decoding examples.

in three incorrect branches on the path through the trellis, the input sequence corresponding to this selected path is [1, 0, 0], so that only one bit is incorrectly decoded due to this decision.

As the figure shows, there is a path of metric 5 which deviates from the all-zero path. The probability of incorrectly selecting this path is denoted as P_5. This error occurs when the received sequence has three or more errors (1s) in it. In general, we denote

$$P_d = \text{Probability of a decoding error on a path of metric } d.$$

For a deviating path of odd weight, there will be an error if more than half of the bits are in error. The probability of this event for a BSC with crossover probability p_c is

$$P_d = \sum_{i=(d+1)/2}^{d} \binom{d}{i} p_c^i (1 - p_c)^{d-i} \quad \text{(with } d \text{ odd).} \tag{12.19}$$

Suppose now the received signal is $\mathbf{r} = [10, 10, 10, 00]$. Then the trellis appears as in Figure 12.21(b). In this case, the path metrics are equal; one-half of the time the decoder chooses the wrong path. If the incorrect path is chosen, the decoded input bits would be [1, 1, 0, 0], with two bits incorrectly decoded. The probability of the event of choosing the incorrect path in this case is P_6,

where

$$P_d = \frac{1}{2}\binom{d}{d/2} p_c^{d/2}(1-p_c)^{d/2} + \sum_{i=d/2+1}^{d} \binom{d}{i} p_c^i (1-p_c)^{d-i} \quad \text{(with } d \text{ even)} \tag{12.20}$$

is the probability that more than half of the bits are in error, plus $\frac{1}{2}$ times the probability that exactly half the bits are in error. □

We can glean the following information from this example:

- Error events can occur in the decoding algorithm when paths merge together. If the erroneous path has lower (better) path metric than the correct path, the algorithm selects it.
- Merging paths may be of different lengths (number of stages).
- This trellis has a shortest path of metric 5 (three stages long) which diverges from the all-zero path then remerges. We say there is an error path of metric 5. There is also an error path of metric 6 (four stages long) which deviates then remerges.
- When an error path is selected, the number of *input* bits that are erroneously decoded depends on the particular path.
- The probability of a particular error event can be calculated and is denoted as P_d.
- The error path of metric 5 was not disjoint of the error path of metric 6, since they both share a branch.

In the following sections, we first describe how to enumerate the paths through the trellis. Then bounds on the probability of node error and the bit error rate are obtained by invoking the union bound.

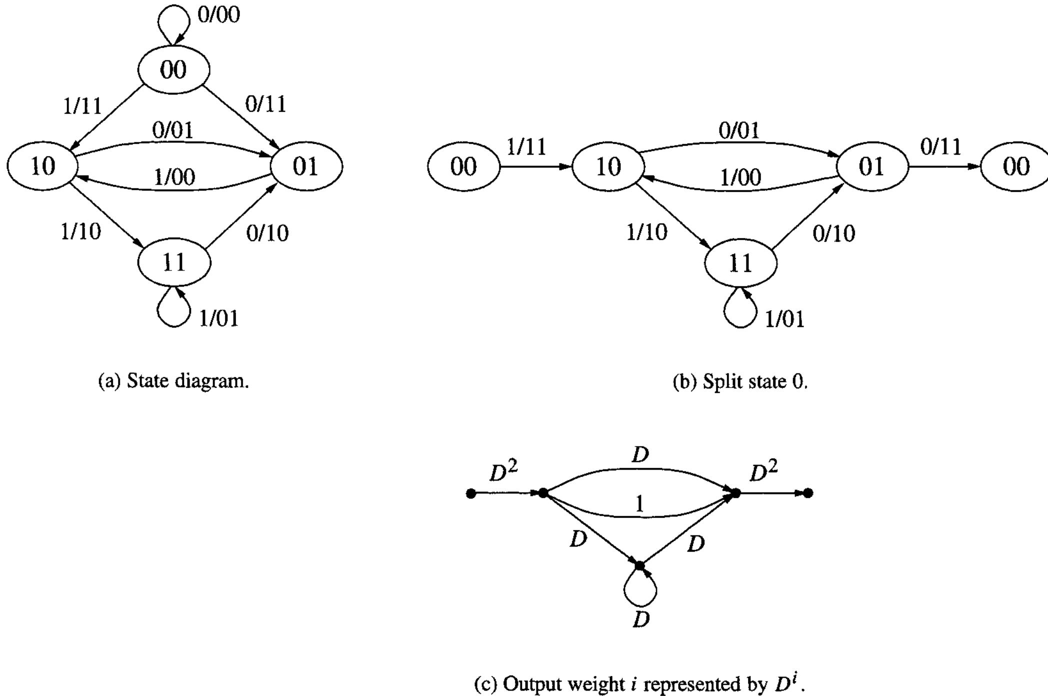
12.5.1 Enumerating Paths Through the Trellis

In computing (or bounding) the overall probability of decoder error, it is expedient to have a method of enumerating all the paths through the trellis. This initially daunting task is aided somewhat by the observation that, for the purposes of computing the probability of error, since the convolutional code is linear it is sufficient to consider only those paths which diverge from the all-zero path then remerge.

We develop a transfer function method which enumerates all the paths that diverge from the all-zero path then remerge. This transfer function is called the **path enumerator**. We demonstrate the technique for the particular code we have been examining.

Example 12.18 Figure 12.22(a) shows the state diagram for the encoder of Example 12.1. In Figure 12.22(b), the 00 state has been "split," or duplicated. Furthermore, the transition from state 00 to state 00 has been omitted, since we are interested only in paths which diverge from the 00 state. Any path through the graph in Figure 12.22(b) from the 00 node on the left to the 00 node on the right represents a path which diverges from the all-zero path then remerges. In Figure 12.22(c), the output codeword of weight i along each edge is represented using D^i. (For example, an output of 11 is represented by D^2; an output of 10 is represented by $D^1 = D$ and an output of 00 is represented by $D^0 = 1$.) For convenience the state labels have been removed.

The labels on the edges in the graph are to be thought of as transfer functions. We now employ the conventional rules for flow graph simplification as summarized in Figure 12.23



(a) State diagram.

(b) Split state 0.

(c) Output weight i represented by D^i.

Figure 12.22: The state diagram and graph for diverging/remerging paths.

- Blocks in series multiply the transfer functions.
- Blocks in parallel add the transfer functions.
- Blocks in feedback configuration employ the rule "forward gain over 1 minus loop gain."

(For a thorough discussion on more complicated flow graphs, see [221].) For the state diagram of Figure 12.22, we take each node as a summing node. The sequence of steps by successively applying the simplification rules is shown in Figure 12.24. Simplifying the final diagram, we find

$$T(D) = D^2 \frac{\frac{D}{1-D}}{1 - \frac{D}{1-D}} D^2 = \frac{D^5}{1-2D}.$$

To interpret this, we use the formal series expansion[4] (check by long division)

$$\boxed{\frac{1}{1-D} = 1 + D + D^2 + D^3 + \cdots .}$$

Expanding $T(D)$ we find

$$T(D) = D^5(1 + 2D + (2D)^2 + (2D)^3 + \cdots) = D^5 + 2D^6 + 4D^7 + \cdots + 2^k D^{k+5} + \cdots .$$

Interpreting this, we see that we have:

[4]A formal series is an infinite series that is obtained by symbolic manipulation, without particular regard to convergence.

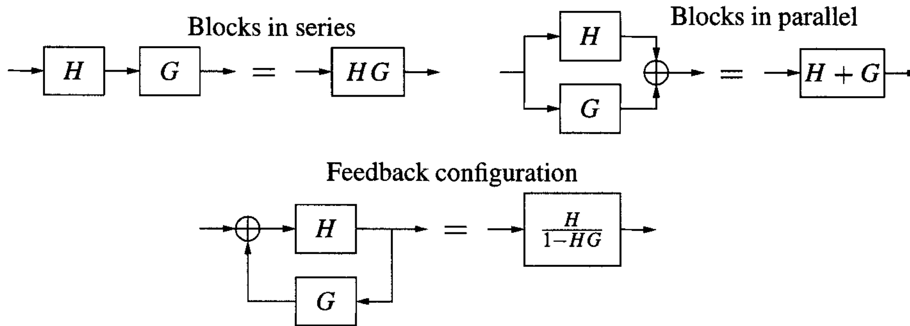


Figure 12.23: Rules for simplification of flow graphs.

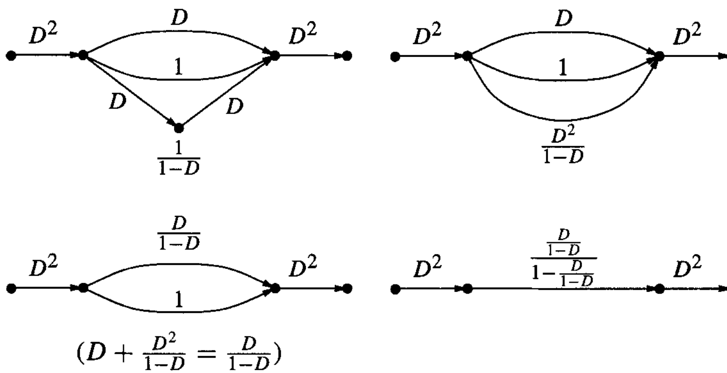


Figure 12.24: Steps simplifying the flow graph for a convolutional code.

- One diverging/remerging error path at metric 5 from the all-zero path;
- 2 error paths of metric 6;
- 4 error paths of metric 7, etc.

Furthermore, the shortest error path has metric 5. □

Definition 12.8 The minimum metric of a path diverging from then remerging to the all-zero path is called the **free distance** of the convolutional code, and is denoted as d_{free}. The number of paths at that metric is denoted as N_{free}. □

In general, we write

$$T(D) = \sum_{d=d_{\text{free}}}^{\infty} a(d)D^d,$$

where $a(d_{\text{free}}) = N_{\text{free}}$.

Additional information about the paths in the trellis can be obtained with a more expressive transfer function. We label each path with three variables: D^i, where i is the output code weight; N^i, where i is the input weight; and L, to account for the length of the branch.

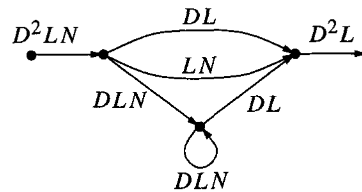


Figure 12.25: State diagram labeled with output weight, input weight, and branch length.

Example 12.19 Returning to the state diagram of Figure 12.22, we obtain the labeled diagram in Figure 12.25. Using the same rules for block diagram simplification as previously, the transfer function for this diagram is

$$T(D, N, L) = \frac{D^5L^3N}{1 - DLN(1 + L)}. \tag{12.21}$$

Expanding this as a formal series we have

$$T(D, N, L) = D^5L^3N + D^6L^4(1 + L)N^2 + D^7L^5(1 + L)^2N^3 + \cdots, \tag{12.22}$$

which has the following interpretation:

- There is one error path of metric 5 which is three branches long (from L^3) and one input bit is 1 (from N^1) along that path.
- There are two error paths of metric 6: one of them is four branches long and the other is five branches long. Along both of them, there are two input bits that are 1.
- There are four error paths of metric 7. One of them is five branches long; two of them are six branches long; and one is seven branches long. Along each of these, there are three input bits that are 1.
- Etc.

□

Clearly, we can obtain the simpler transfer function by $T(D) = T(D, N, L)|_{N=1,L=1}$. When we don't care to keep track of the number of branches, we write

$$T(D, N) = T(D, N, L)|_{L=1}.$$

Enumerating on More Complicated Graphs: Mason's Rule

Some graphs are more complicated than the three rules introduced above can accommodate. A more general approach is Mason's rule [221]. Its generality leads to a rather complicated notation, which we summarize here and illustrate by example (not by proof) [373]. We will enumerate all paths from the 0 state to the 0 state in the state diagram shown in Figure 12.26.

A **loop** is a sequence of states which starts and ends in the same state, but otherwise does not enter any state more than once. We will say that a **forward loop** is a loop that starts and stops in state 0. A set of loops is **nontouching** if they have no vertices in common. Thus $\{0, 1, 2, 4, 0\}$ is a forward loop. The loop $\{3, 6, 5, 3\}$ is a loop that does not touch this forward loop. The set of all forward loops in the graph is denoted as $L = \{L_1, L_2, \ldots\}$. The corresponding set of path gains is denoted as $F = \{F_1, F_2, \ldots\}$. Let $\mathcal{L} = \{\mathcal{L}_1, \mathcal{L}_2, \ldots\}$ denote the set of loops in the graph that does *not* contain the vertex 0. Let $\mathcal{F} = \{\mathcal{F}_1, \mathcal{F}_2, \ldots\}$ be the set of corresponding path gains. (Determining

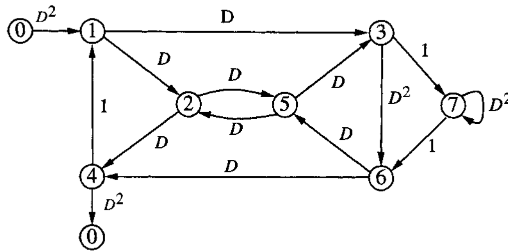


Figure 12.26: A state diagram to be enumerated.

all the loops requires some care.) For the graph in Figure 12.26, the forward loops and their gains are

$$
\begin{array}{lll}
L_1: & \{0,1,3,7,6,5,2,4,0\} & F_1 = D^8 \\
L_2: & \{0,1,3,7,6,4,0\} & F_2 = D^6 \\
L_3: & \{0,1,3,6,5,2,4,0\} & F_3 = D^{10} \\
L_4: & \{0,1,3,6,4,0\} & F_4 = D^8 \\
L_5: & \{0,1,2,5,3,7,6,4,0\} & F_5 = D^8 \\
L_6: & \{0,1,2,5,3,6,4,0\} & F_6 = D^{10} \\
L_7: & \{0,1,2,4,0\} & F_7 = D^6
\end{array}
$$

and the other loops and their gains are

$$
\begin{array}{lll}
\mathcal{L}_1: & \{1,3,7,6,5,2,4,1\} & \mathcal{F}_1 = D^4 \\
\mathcal{L}_2: & \{1,3,7,6,4,1\} & \mathcal{F}_2 = D^2 \\
\mathcal{L}_3: & \{1,3,6,5,2,4,1\} & \mathcal{F}_3 = D^6 \\
\mathcal{L}_4: & \{1,3,6,4,1\} & \mathcal{F}_4 = D^4 \\
\mathcal{L}_5: & \{1,2,5,3,7,6,4,1\} & \mathcal{F}_5 = D^4 \\
\mathcal{L}_6: & \{1,2,5,3,6,4,1\} & \mathcal{F}_6 = D^6 \\
\mathcal{L}_7: & \{1,2,4,1\} & \mathcal{F}_6 = D^2 \\
\mathcal{L}_8: & \{2,5,2\} & \mathcal{F}_8 = D^2 \\
\mathcal{L}_9: & \{3,7,6,5,3\} & \mathcal{F}_9 = D^2 \\
\mathcal{L}_{10}: & \{3,6,5,3\} & \mathcal{F}_{10} = D^4 \\
\mathcal{L}_{11}: & \{7,7\} & \mathcal{F}_{11} = D^2
\end{array}
$$

We also need to identify the *pairs* of nontouching loops in $\mathcal{L}$, the triples of nontouching loops in $\mathcal{L}$, etc., and their corresponding product gains. There are ten pairs of nontouching loops in $\mathcal{L}$:

$$
\begin{array}{llll}
(\mathcal{L}_2, \mathcal{L}_8) & \mathcal{F}_2\mathcal{F}_8 = D^4 & (\mathcal{L}_3, \mathcal{L}_{11}) & \mathcal{F}_3\mathcal{F}_{11} = D^8 \\
(\mathcal{L}_4, \mathcal{L}_8) & \mathcal{F}_4\mathcal{F}_8 = D^6 & (\mathcal{L}_4, \mathcal{L}_{11}) & \mathcal{F}_4\mathcal{F}_{11} = D^6 \\
(\mathcal{L}_6, \mathcal{L}_{11}) & \mathcal{F}_6\mathcal{F}_{11} = D^8 & (\mathcal{L}_7, \mathcal{L}_9) & \mathcal{F}_7\mathcal{F}_9 = D^4 \\
(\mathcal{L}_7, \mathcal{L}_{10}) & \mathcal{F}_7\mathcal{F}_{10} = D^6 & (\mathcal{L}_7, \mathcal{L}_{11}) & \mathcal{F}_7\mathcal{F}_{11} = D^4 \\
(\mathcal{L}_8, \mathcal{L}_{11}) & \mathcal{F}_8\mathcal{F}_{11} = D^4 & (\mathcal{L}_{10}, \mathcal{L}_{11}) & \mathcal{F}_{10}\mathcal{F}_{11} = D^6
\end{array}
$$

There are two **triplets** of nontouching loops in $\mathcal{L}$,

$$
(\mathcal{L}_4, \mathcal{L}_8, \mathcal{L}_{11}) \quad \mathcal{F}_4\mathcal{F}_8\mathcal{F}_{11} = D^8 \qquad (\mathcal{L}_7, \mathcal{L}_{10}, \mathcal{L}_{11}) \quad \mathcal{F}_7\mathcal{F}_{10}\mathcal{F}_{11} = D^8
$$

but no sets of four or more nontouching loops in $\mathcal{L}$.

With these sets collected, we define the **graph determinant** Δ as

$$
\Delta = 1 - \sum_{\mathcal{L}_i} \mathcal{F}_i + \sum_{(\mathcal{L}_i, \mathcal{L}_j)} \mathcal{F}_i\mathcal{F}_j - \sum_{(\mathcal{L}_i, \mathcal{L}_j, \mathcal{L}_k)} \mathcal{F}_i\mathcal{F}_j\mathcal{F}_k + \cdots
$$

where the first sum is over all the loops in $\mathcal{L}$, the second sum is over all pairs of nontouching loops in $\mathcal{L}$, the third sum is over all triplets of nontouching loops in $\mathcal{L}$, and so forth.

We also define the **graph cofactor of the forward path** L_i, denoted as Δ_i, which is similar to the graph determinant except that all loops touching L_i are removed from the summations. This can be written as

$$\Delta_i = 1 - \sum_{(L_i,\mathcal{L}_j)} \mathcal{F}_j + \sum_{(L_i,\mathcal{L}_j,\mathcal{L}_k)} \mathcal{F}_j\mathcal{F}_k - \sum_{(L_i,\mathcal{L}_j,\mathcal{L}_k,\mathcal{L}_l)} \mathcal{F}_j\mathcal{F}_k\mathcal{F}_l + \cdots$$

With this notation we finally can express **Mason's rule** for the transfer function through the graph:

$$T(D) = \frac{\sum_l F_l \Delta_l}{\Delta}, \tag{12.23}$$

where the sum is over all forward paths.

For our example, we have

$$\Delta_1 = 1 \qquad \Delta_5 = 1$$

(since there are no loops that do not contain vertices in the forward paths L_1 and L_5),

$$\Delta_3 = \Delta_6 = 1 - \mathcal{F}_{11} = 1 - D^2$$

(since L_3 and L_6 do not cross vertex 7 and so do not touch loop $\mathcal{L}_{11}$),

$$\Delta_2 = 1 - \mathcal{F}_8 = 1 - D^2$$

(since L_2 does not touch $\mathcal{L}_8$ but it does touch all other loops), and

$$\Delta_4 = 1 - (\mathcal{F}_8 + \mathcal{F}_{11}) + \mathcal{F}_8\mathcal{F}_{11} = 1 - 2D^2 + D^4$$

$$\Delta_7 = 1 - (\mathcal{F}_9 + \mathcal{F}_{10} + \mathcal{F}_{11}) + (\mathcal{F}_{10}\mathcal{F}_{11}) = 1 - 2D^2 - D^4 + D^6.$$

The graph determinant is

$$\begin{aligned}\Delta &= 1 - (D^4 + D^2 + D^6 + D^4 + D^4 + D^6 + D^2 + D^2 + D^2 + D^4 + D^2) \\ &\quad + (D^4 + D^8 + D^6 + D^6 + D^8 + D^4 + D^6 + D^4 + D^4 + D^6) - (D^8 + D^8) \\ &= 1 - 5D^2 + 2D^6.\end{aligned}$$

Finally, using (12.23) we obtain

$$T(D) = \frac{2D^6 + D^{10}}{1 - 5D^2 + 2D^6}.$$

12.5.2 Characterizing the Node Error Probability P_e and the Bit Error Rate P_b

We now return to the question of the probability of error for convolutional codes. Let $\mathcal{P}_j$ denote the set of all error paths that diverge from node j of the all-zero path in the trellis then remerge and let $\mathbf{p}_{i,j} \in \mathcal{P}_j$ be one of these paths. Let $\Delta M(\mathbf{p}_{i,j}, 0)$ denote the difference between the metric accumulated along path $\mathbf{p}_{i,j}$ and the all-zero path. An error event at node j occurs due to path $\mathbf{p}_{i,j}$ if $\Delta M(\mathbf{p}_{i,j}, 0) < 0$. Letting $P_e(j)$ denote the probability of an error event at node j, we have

$$P_e(j) \le \Pr\left[\bigcup_i \{\Delta M(\mathbf{p}_{i,j}, 0) \le 0\}\right], \tag{12.24}$$

where the inequality follows since an error might not occur when $\Delta M(\mathbf{p}_{i,j}, 0) = 0$. The paths $\mathbf{p}_{i,j} \in \mathcal{P}$ are not all disjoint since they may share branches, so the events $\{\Delta M(\mathbf{p}_{i,j}, 0) \leq 0\}$ are not disjoint. This makes (12.24) very difficult to compute exactly. However, it can be upper bounded by the union bound (see Box 1.1 in chapter 1) as

$$P_e(j) \leq \sum_i \Pr(\Delta M(\mathbf{p}_{i,j}, 0) \leq 0). \tag{12.25}$$

Each term in this summation is now a *pairwise* event between the paths $\mathbf{p}_{i,j}$ and the all-zero path.

We here develop expressions or bounds on the pairwise events in (12.25) for the case that the channel is memoryless. (For example, we have already seen that for the BSC, the probability P_d developed in (12.19) and (12.20) is the probability of the pairwise events in question.) For a memoryless channel, $\Delta M(\mathbf{p}_{i,j}, 0)$ depends only on those branches for which $\mathbf{p}_{i,j}$ is nonzero. Let d be the Hamming weight of $\mathbf{p}_{i,j}$ and let P_d be the probability of the event that this path has a lower (better) metric than the all-zero path. Let $a(d)$ be the number of paths at a distance d from the all-zero path. The probability of a node error event can now be written as follows:

$$\begin{aligned} P_e(j) &\leq \sum_{d=d_{\text{free}}}^{\infty} \Pr(\text{error caused by any of the } a(d) \text{ incorrect paths at distance } d) \\ &= \sum_{d=d_{\text{free}}}^{\infty} a(d) P_d. \end{aligned} \tag{12.26}$$

Any further specification on $P_e(j)$ requires characterization of P_d. We show below that bounds on P_d can be written in the form

$$P_d < Z^d \tag{12.27}$$

for some channel-dependent function Z and develop explicit expressions for Z for the BSC and AWGN channel. For now, we simply express the results in terms of Z. With this bound we can write

$$P_e(j) < \sum_{d=d_{\text{free}}}^{\infty} a(d) Z^d.$$

Recalling that the path enumerator for the encoder is $T(D) = \sum_{d=d_{\text{free}}}^{\infty} a(d) D^d$, we obtain a closed-form expression for the bound:

$$P_e(j) < T(D)|_{D=Z}. \tag{12.28}$$

The bound (12.28) is a bound on the probability of a node error. From this, a bound on the bit error rate can be obtained by enumerating the number of bits in error for each node error. The derivative

$$\frac{\partial}{\partial N} T(D, N)$$

brings exponents of N down as multipliers for each term in the series.

Example 12.20 For the weight enumerator of Example 12.18,

$$T(D, N) = D^5 N + 2D^6 N^2 + 4D^7 N^3 + \cdots,$$

we have

$$\frac{\partial}{\partial N}T(D,N) = (1)D^5 + (2)2D^6N + (3)4D^7N^2 + \cdots$$

so that a node error on the error path of metric 5 contributes one bit of error; a node error on either of the error paths of metric 6 contributes two bits of error, and so forth. □

The average number of bits in error along the branches of the trellis is

$$\left.\frac{\partial}{\partial N}T(D,N)\right|_{N=1,D=Z}.$$

For a rate $R = k/n$ code, each branch corresponds to k message bits, so that

$$P_b < \left.\frac{1}{k}\frac{\partial}{\partial N}T(D,N)\right|_{N=1,D=Z}. \tag{12.29}$$

An approximation on the probability of bit error can be obtained by retaining only the first term of the series in (12.29) and using the bound $P_d < Z^d$. To retain only the first term of the series, write

$$T(D,N) = D^{d_{\text{free}}}(N^{n_1} + N^{n_2} + \cdots) + \cdots.$$

Then

$$\left.\frac{\partial}{\partial N}T(D,N)\right|_{N=1} = D^{d_{\text{free}}}(n_1 + n_2 + \cdots) + \cdots.$$

The number $n_1 + n_2 + \cdots$ is the number of nonzero message bits associated with codewords of weight d_{free}. Let us denote this number as $b_{\text{dfree}} = n_1 + n_2 + \cdots$

Example 12.21 Suppose

$$T(D,N) = D^6N + D^6N^3 + 3D^8N + 5D^8N^4 + \cdots$$

Then there are two codewords of weight 6: one corresponding to a message of weight 1 and one corresponding to a message of weight 3. We could write

$$T(D,N) = D^6(N + N^3) + 3D^8N + 5D^8N^4 + \cdots.$$

Then $b_6 = 1 + 3 = 4$. □

Then the approximation is

$$P_b \approx \left.\frac{1}{k}b_{d_{\text{free}}}D^{d_{\text{free}}}\right|_{D=Z}. \tag{12.30}$$

A lower bound can be found as

$$P_b > \frac{1}{k}b_{d_{\text{free}}}P_{d_{\text{free}}}, \tag{12.31}$$

where $P_{d_{\text{dfree}}}$ is P_d at $d = d_{\text{free}}$.

Example 12.22 For

$$T(D,N) = \frac{D^5}{1-2DN} = D^5N + 2D^6N^2 + 4D^7N^3 + \cdots$$

the derivative is

$$\frac{\partial}{\partial N}T(D,N) = D^5 + 4D^6N + 12D^7N^2 + \cdots$$

so the probability of error is approximated by

$$P_b \approx Z^5$$

or lower bounded by

$$P_b > P_5.$$

□

12.5.3 A Bound on P_d for Discrete Channels

In this section we develop a bound on P_d for discrete channels such as the BSC [373, Section 12.3.1]. Let $p(r_i|1)$ denote the likelihood of the received signal r_i, given that the symbol corresponding to 1 was sent through the channel; similarly $p(r_i|0)$. Then

$$\begin{aligned} P_d &= P(\Delta M(\mathbf{p}_{i,j}, 0) \leq 0 \text{ and } d_H(\mathbf{p}_{i,j}, \mathbf{0}) = d) \\ &= P\Big[\sum_{i=1}^{d} \log P(r_i|1) - \sum_{i=1}^{d} \log P(r_i|0) \geq 0\Big] \end{aligned}$$

where $\{r_1, r_2, \ldots, r_d\}$ are the received signals at the d coordinates where $\mathbf{p}_{i,j}$ is nonzero. Continuing,

$$P_d = P\Big[\sum_{i=1}^{d} \log \frac{p(r_i|1)}{p(r_i|0)} \geq 0\Big] = P\Big[\prod_{i=1}^{d} \frac{p(r_i|1)}{p(r_i|0)} \geq 1\Big].$$

Let R' be the set of vectors of elements $\mathbf{r} = (r_1, r_2, \ldots, r_d)$ such that

$$\prod_{i=1}^{d} \frac{p(r_i|1)}{p(r_i|0)} \geq 1. \tag{12.32}$$

(For example, for $d = 5$ over the BSC, R' is the set of vectors for which 3 or 4 or 5 of the r_i are equal to 1.) The probability of any one of these elements is $\prod_{i=1}^{d} p(r_i|0)$, since we are assuming that all zeros are sent. Thus, the probability of any vector in R' is $\prod_{i=1}^{d} p(r_i|0)$. The probability P_d can be obtained by summing over all the vectors in R':

$$P_d = \sum_{\mathbf{r}\in R'} \prod_{i=1}^{d} p(r_i|0).$$

Since the left-hand side of (12.32) is ≥ 1, we have

$$P_d \leq \sum_{\mathbf{r}\in R'} \prod_{i=1}^{d} p(r_i|0) \left[\frac{p(r_i|1)}{p(r_i|0)}\right]^s$$

for any s such that $0 \leq s < 1$. The tightest bound is obtained by minimizing with respect to s:

$$P_d \leq \min_{0\leq s<1} \sum_{\mathbf{r}\in R'} \prod_{i=1}^{d} p(r_i|0) \left[\frac{p(r_i|1)}{p(r_i|0)}\right]^s = \min_{0\leq s<1} \sum_{\mathbf{r}\in R'} \prod_{i=1}^{d} p(r_i|0)^{1-s} p(r_i|1)^s.$$

This is made more tractable (and larger) by summing over the set R of *all* sequences $(r_1, r_2, \ldots, r_d)$:

$$P_d < \min_{0\leq s<1} \sum_{\mathbf{r}\in R} \prod_{i=1}^{d} p(r_i|0)^{1-s} p(r_i|1)^s.$$

The order of summation and product can be reversed, resulting in

$$P_d < \min_{0 \le s < 1} \prod_{i=1}^{d} \sum_{r_i} p(r_i|0)^{1-s} p(r_i|1)^s.$$

This is known as the **Chernoff bound**. Let

$$Z = \min_{0 \le s < 1} \sum_{r_i} p(r_i|0)^{1-s} p(r_i|1)^s. \tag{12.33}$$

Then $P_d < Z^d$.

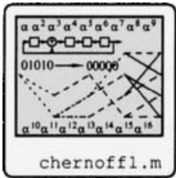
If the channel is symmetric, then by symmetry arguments the minimum must occur when $s = 1/2$. Then $Z = \sqrt{p(r_i|0)p(r_i|1)}$ and

$$P_d < \prod_{i=1}^{d} \sum_{r_i} \sqrt{p(r_i|0)p(r_i|1)}. \tag{12.34}$$

This bound is known as the **Bhattacharya bound**.

Example 12.23 Suppose the encoder with $T(D, N) = \frac{D^5 N}{1-2DN}$ is used in conjunction with an asymmetric channel having the following transition probabilities:

$P(r\|a)$	$a = 0$	$a = 1$
$r = 0$	0.98	0.003
$r = 1$	0.02	0.997.



Then

$$\begin{aligned} Z &= \min_{0 \le s < 1} \sum_{r_i} p(r_i|0)^{1-s} p(r_i|1)^s \\ &= p(0|0)^{1-s} p(0|1)^s + p(1|0)^{1-s} p(1|1)^s \\ &= (0.98)^{1-s}(0.003)^s + (0.02)^{1-s}(0.997)^s. \end{aligned}$$

The minimum value can be found numerically as $Z = 0.1884$, which occurs when $s = 0.442$.

The node error probability can be bounded using (12.28) as

$$P_e(j) < \left. \frac{D^5 N}{1 - 2DN} \right|_{N=1, D=0.1884} = 3.8 \times 10^{-4}$$

and the bit error rate is bounded using (12.29) as

$$P_b < \left. \frac{1}{k} \frac{\partial}{\partial N} \frac{D^5 N}{1 - 2DN} \right|_{N=1, D=0.1884} = 6.1 \times 10^{-4}$$

The approximate bit error rate from (12.30) is

$$P_b \approx \left. \frac{1}{k} b_{d_{\text{free}}} D^{d_{\text{free}}} \right|_{D=0.1884} = 2.4 \times 10^{-4},$$

where $d_{\text{free}} = 5$ and $b_{d_{\text{free}}} = 1$. □

Performance Bound on the BSC

For the BSC with crossover probability p_c, (12.34) can be written as follows:

$$P_d < \prod_{i=1}^{d} \sum_{r_i} \sqrt{p(r_i|0)p(r_i|1)} = \prod_{i=1}^{d} (\sqrt{p(0|0)p(0|1)} + \sqrt{P(1|0)p(1|1)})$$
$$= \prod_{i=1}^{d} 2\sqrt{p_c(1-p_c)} = [4p_c(1-p_c)]^{d/2} = \sqrt{4p_c(1-p_c)}^d.$$

The expression Z in (12.27) is thus $Z = [4p_c(1-p_c)]^{1/2}$.

Let us now return to $P_e(j)$. Inserting this bound on P_d in (12.26) we obtain

$$P_e(j) < \sum_{d=d_{\text{free}}}^{\infty} a(d)P_d < \sum_{d=d_{\text{free}}}^{\infty} a(d)\sqrt{4p_c(1-p_c)}^d.$$

The closed-form expression for the bound on the probability of error is

$$P_e(j) < T(D)|_{D=\sqrt{4p_c(1-p_c)}}. \tag{12.35}$$

The bound on the bit error rate of (12.29) is

$$P_b < \frac{1}{k}\frac{\partial}{\partial N}T(D,N)\bigg|_{N=1, D=\sqrt{4p_c(1-p_c)}}. \tag{12.36}$$

A lower bound can be obtained using (12.31)

$$P_b > \frac{1}{k} b_{d_{\text{free}}} P_{d_{\text{free}}}, \tag{12.37}$$

where $P_{d_{\text{free}}}$ can be computed using (12.19) and (12.20).

12.5.4 A Bound on P_d for BPSK Signaling Over the AWGN Channel

Suppose that the coded bits $c_t^{(i)}$ are mapped to a BPSK signal constellation by $a_t^{(i)} = \sqrt{E_c}(2c_t^{(i)} - 1)$, where E_c is the coded signal energy, with $E_c = RE_b$, and E_b is the energy per message bit. If the all-zero sequence is sent, then the sequence of amplitudes $(-\sqrt{E_c}, -\sqrt{E_c}, -\sqrt{E_c}, \ldots)$ is sent. A sequence which deviates from this path in d locations is at a squared distance $2dE_c$ from it. Then P_d is the probability that a d-symbol sequence is decoded incorrectly, compared to the sequence for all-zero transmission. That is, it is the problem of distinguishing $\mathbf{p}_1 = (-\sqrt{E_c}, -\sqrt{E_c}, -\sqrt{E_c}, \ldots, -\sqrt{E_c})$ from $\mathbf{p}_2 = (\sqrt{E_c}, \sqrt{E_c}, \sqrt{E_c}, \ldots, \sqrt{E_c})$, where these vectors each have d elements. The Euclidean distance between these two points is

$$d_{\text{Euclidean}}(\mathbf{p}_1, \mathbf{p}_2) = 2[dE_c]^{1/2}.$$

The probability of a detection error is (see Section 1.5.4)

$$P_d = Q\left(\sqrt{\frac{2dE_c}{N_0}}\right).$$

To express this in the form Z^d (for use in the bound (12.28)) use the bound $Q(x) < \frac{1}{2}e^{-x^2/2}$ (see Exercise 1).12. We thus obtain

$$P_d < \frac{1}{2}e^{-dE_c/N_0}.$$

Then (12.28) and (12.29) give

$$P_e(j) < \frac{1}{2}T(D)\Big|_{D=e^{-E_c/N_0}} \qquad P_b < \frac{1}{2}\frac{1}{k}\frac{\partial}{\partial N}T(D,N)\Big|_{N=1,D=e^{-E_c/N_0}}. \tag{12.38}$$

Another bound on the Q function is [359]

$$Q(\sqrt{x+y}) \leq Q(\sqrt{x})e^{-y/2}, \qquad x \geq 0, y \geq 0. \tag{12.39}$$

Then

$$P_d = Q\left(\sqrt{\frac{2dE_c}{N_0}}\right) = Q\left(\sqrt{\frac{2d_{\text{free}}E_c}{N_0} + \frac{2(d-d_{\text{free}})E_c}{N_0}}\right)$$
$$\leq Q\left(\sqrt{\frac{2d_{\text{free}}E_c}{N_0}}\right)e^{d_{\text{free}}E_c/N_0}e^{-dE_c/N_0} = Q\left(\sqrt{\frac{2d_{\text{free}}E_c}{N_0}}\right)e^{d_{\text{free}}E_c/N_0}(e^{-E_c/N_0})^d.$$

Then

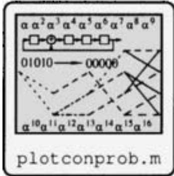
$$P_e(j) \leq e^{d_{\text{free}}E_c/N_0}Q\left(\sqrt{\frac{2d_{\text{free}}E_c}{N_0}}\right)T(D)\Big|_{D=e^{-E_c/N_0}}$$

and

$$P_b \leq \frac{1}{k}e^{d_{\text{free}}E_c/N_0}Q\left(\sqrt{\frac{2d_{\text{free}}E_c}{N_0}}\right)\frac{\partial}{\partial N}T(D,N)\Big|_{N=1,D=e^{-E_c/N_0}}. \tag{12.40}$$

A lower bound can be obtained using (12.31)

$$P_b > \frac{1}{k}b_{d_{\text{free}}}Q\left(\sqrt{\frac{2d_{\text{free}}E_c}{N_0}}\right). \tag{12.41}$$



Example 12.24 For the $R = 1/2$ code of Example 12.1 with $d_{\text{free}} = 5$, Figure 12.27 shows the bounds on the probability of bit error of the code for both hard- and soft-decision decoders compared with uncoded performance. For soft decoding, the lower bound and the upper bound of (12.40) approach each other for high signal to noise ratios, so the bounds are asymptotically tight. (The bound of (12.38) is looser.) Gains of approximately 4 dB at high SNR are evident for soft decision decoding.

The hard-decision decoding bounds are clearly not as tight. Also, there is approximately 3 dB less coding gain for the hard-decision decoder. □

12.5.5 Asymptotic Coding Gain

The lower bound for the probability of bit error for the coded signal (using soft-decision decoding)

$$P_b > \frac{1}{k}a(d_{d_{\text{free}}})n_{d_{\text{free}}}Q\left(\sqrt{\frac{2d_{\text{free}}E_c}{N_0}}\right)$$

can be approximated using the bound $Q(x) < \frac{1}{2}e^{-x^2/2}$ as

$$P_b \approx \frac{1}{k}\frac{1}{2}a(d_{d_{\text{free}}})n_{d_{\text{free}}}e^{-d_{\text{free}}E_c/N_0}. \tag{12.42}$$

The probability of bit error for uncoded transmission is

$$P_b = Q\left(\sqrt{\frac{2E_b}{N_0}}\right) < \frac{1}{2}e^{-E_b/N_0}. \tag{12.43}$$

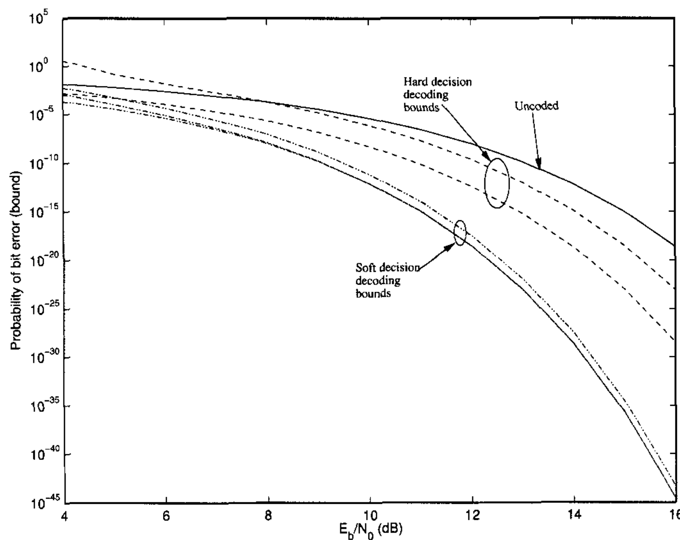


Figure 12.27: Performance of a (3, 1) convolutional code with $d_{\text{free}} = 5$.

The dominant factor in (12.42) and (12.43) for large values of signal-to-noise ratio is determined by the exponents. Comparing the exponents in these two using $E_c = RE_b$ we see that the exponent in the probability of bit error for the coded case is a factor of Rd_{free} larger than the exponent for the uncoded case. The quantity

$$\gamma_{\text{soft}} = 10 \log_{10} Rd_{\text{free}}$$

is called the *asymptotic coding gain* of the code. For sufficiently large SNR, performance essentially equivalent to uncoded performance can be obtained with γ dB less SNR when coding is employed. A similar argument can be made to show that the asymptotic coding gain for hard decision decoding is

$$\gamma_{\text{hard}} = 10 \log_{10} \frac{Rd_{\text{free}}}{2}.$$

This shows that asymptotically, soft-decision decoding is 3 dB better than hard-decision decoding.

As the SNR increases, the dominant term in computing the bit error rate is the first term in $T(x, N)$. As a result the free distance has a very strong bearing on the performance of the code.

12.6 Tables of Good Codes

Unlike block codes, where many good codes have been found by exploiting the algebraic structure of the codes, good convolutional codes have been found mostly by computer search. As a result, good codes are known only for relatively short constraint lengths. The following tables [251, 254, 197, 65] provide the best known polynomial codes. It may be observed that all of these codes are nonsystematic.

There are separate tables for different rates. Within each table, different memory lengths L, are used, where

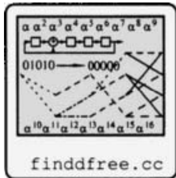
$$L = \nu + 1,$$

where $\nu = \max_{i,j} \deg(g_{ij}(x))$ is the degree of the highest polynomial. (This quantity is called in many sources the constraint length.) For the rate k/n codes with $k > 1$, L represents the largest degree and ν represents the total memory.

In these tables, the coefficients are represented using octal digits with the least significant bit on the right. Thus, $0 \to 000$, $1 \to 001$, $2 \to 010$, $3 \to 011$, and so forth. There may be trailing zeros on the right. For example, for the rate 1/4 code, the entry with $L = 5$ has generators $g_0 = 52$, $g_1 = 56$, $g_2 = 66$ and $g_3 = 76$. The corresponding bit values are

$$\mathbf{g}_0 = (101\,010) \quad \mathbf{g}_1 = (101\,110) \quad \mathbf{g}_2 = (110\,110) \quad \mathbf{g}_3 = (111\,110).$$

The first coefficient (on the left) is the first coefficient in the encoder.[5] Thus the coded output streams are



$$c_t^{(1)} = m_t + m_{t-2} + m_{t-4} \qquad c_t^{(2)} = m_t + m_{t-2} + m_{t-3} + m_{t-4}$$
$$c_t^{(3)} = m_t + m_{t-1} + m_{t-3} + m_{t-4} \quad c_t^{(4)} = m_t + m_{t-1} + m_{t-2} + m_{t-3} + m_{t-4}$$

The program `finddfree` finds d_{free} for a given set of connection coefficients. It has been used to check these results. (Currently implemented only for $k = 1$ codes.)

$R = 1/2$ [251, 197]			
L	$g^{(1)}$	$g^{(2)}$	d_{free}
3	5	7	5
4	64	74	6
5	46	72	7
6	65	57	8
7	554	744	10
8	712	476	10
9	561	753	12
10	4734	6624	12
11	4762	7542	14
12	4335	5723	15
13	42554	77304	16
14	43572	56246	16
15	56721	61713	18
16	447254	627324	19
17	716502	514576	20

$R = 1/3$ [251, 197]				
L	$g^{(1)}$	$g^{(2)}$	$g^{(3)}$	d_{free}
3	5	7	7	8
4	54	64	74	10
5	52	66	76	12
6	47	53	75	13
7	554	624	764	15
8	452	662	756	16
9	557	663	711	18
10	4474	5724	7154	20
11	4726	5562	6372	22
12	4767	5723	6265	24
13	42554	43364	77304	24
14	43512	73542	76266	26

[5]To use the class `BinConvFIR`, the left bit must be interpreted as the LSB of a binary number. The function `octconv` returns an integer value that can be used directly in `BinConvFIR`.

$R = 1/4$ [251, 197]					
L	$g^{(1)}$	$g^{(2)}$	$g^{(3)}$	$g^{(4)}$	d_{free}
3	5	7	7	7	10
4	54	64	64	74	13
5	52	56	66	76	16
6	53	67	71	75	18
7	564	564	634	714	20
8	472	572	626	736	22
9	463	535	733	745	24
10	4474	5724	7154	7254	27
11	4656	4726	5562	6372	29
12	4767	5723	6265	7455	32
13	44624	52374	66754	73534	33
14	42226	46372	73256	73276	36

$R = 2/3$ [254, 172]					
		$g^{(1,1)}$	$g^{(1,2)}$	$g^{(1,3)}$	
L	ν	$g^{(2,1)}$	$g^{(2,2)}$	$g^{(2,3)}$	d_{free}
2	2	6	2	6	3
		2	4	8	
3	3	5	2	6	4
		1	4	7	
3	4	7	1	4	5
		2	5	7	
4	5	60	30	70	6
		14	40	74	
4	6	64	30	64	7
		30	64	74	
5	7	60	34	54	8
		16	46	74	
5	8	64	12	52	8
		26	66	44	
6	9	52	06	74	9
		05	70	53	
6	10	63	15	46	10
		32	65	61	

$R = 3/4$ [254, 172]						
		$g^{(1,1)}$	$g^{(1,2)}$	$g^{(1,3)}$	$g^{(1,4)}$	
		$g^{(2,1)}$	$g^{(2,2)}$	$g^{(2,3)}$	$g^{(2,4)}$	
L	ν	$g^{(3,1)}$	$g^{(3,2)}$	$g^{(3,3)}$	$g^{(3,4)}$	d_{free}
2	3	4	4	4	4	4
		0	6	2	4	
		0	2	5	5	
3	5	6	2	2	6	5
		1	6	0	7	
		0	2	5	5	
3	6	6	1	0	7	6
		3	4	1	6	
		2	3	7	4	
4	8	70	30	20	40	7
		14	50	00	54	
		04	10	74	40	
4	9	40	14	34	60	8
		04	64	20	70	
		34	00	60	64	

Table 12.2 presents a comparison of d_{free} for systematic and nonsystematic codes (with polynomial generators), showing that nonsystematic codes have generally better distance properties. Results are even more pronounced for longer constraint lengths.

12.7 Puncturing

In Section 3.9, puncturing was introduced as a modification to block codes, in which one of the parity symbols is removed. In the context of convolutional codes, **puncturing** is accomplished by periodically removing bits from one or more of the encoder output streams [40]. This has the effect of increasing the rate of the code.

Example 12.25 Let the coded output sequence of a rate $R = 1/2$ code be

$$\mathbf{c} = (c_0^{(1)}, c_0^{(2)},\ c_1^{(1)}, c_1^{(2)},\ c_2^{(1)}, c_2^{(2)},\ c_3^{(1)}, c_3^{(2)},\ c_4^{(1)}, c_4^{(2)}, \ldots)$$

Table 12.2: Comparison of Free Distance as a Function of Constraint Length for Systematic and Nonsystematic Codes

$R = 1/3$ [251]		
	Systematic	Nonsystematic
L	d_{free}	d_{free}
2	3	3
3	4	5
4	4	6
5	5	7
6	6	8
7	6	8
8	7	10

$R = 1/2$ [251]		
	Systematic	Nonsystematic
L	d_{free}	d_{free}
2	5	5
3	6	8
4	8	10
5	9	12
6	10	13
7	12	15
8	12	16

When the code is punctured by removing every fourth coded symbol, the punctured sequence is

$$\tilde{\mathbf{c}} = (c_0^{(1)}, c_0^{(2)}, c_1^{(1)}, -, c_2^{(1)}, c_2^{(2)}, c_3^{(1)}, -, c_4^{(1)}, c_4^{(2)}, \ldots).$$

The $-$ symbols merely indicate where the puncturing takes place; they are not transmitted. The punctured sequence thus produces three coded symbols for every two input symbols, resulting in a rate $R = 2/3$ code. □

Decoding of a punctured code can be accomplished using the same trellis as the unpunctured code, but simply not accumulating any branch metric for the punctured symbols. One way this can be accomplished is by inserting symbols into the received symbol stream whose branch metric computation would be 0, then using conventional decoding.

The pattern of puncturing is often described by means of a **puncturing matrix** P. For a rate k/n code, the puncture matrix has n rows. The number of columns is the number of symbols over which the puncture pattern repeats. For example, for the puncturing of the previous example,

$$P = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}.$$

The element P_{ij} is 1 if the ith symbol is sent in the jth epoch of the puncturing period.

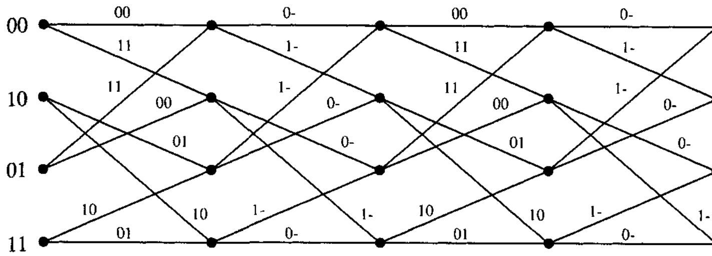
While the punctured code can be encoded as initially described — by encoding with the lower rate code then puncturing — this is wasteful, since computations are made which are promptly ignored. However, since the code obtained is still a convolutional code, it has its own trellis, which does not require any explicit puncturing.

Example 12.26 We demonstrate puncturing for the code which has been a *leitmotif* for this chapter, with generators $g^{(1)}(x) = 1 + x^2$ and $g^{(2)}(x) = 1 + x + x^2$. Puncturing is accomplished by deleting every other bit of the second output stream (as above). Four stages of the trellis for this punctured code are shown in Figure 12.28(a).

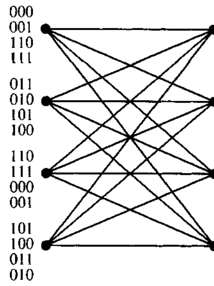
Now draw the trellis for the resulting $R = 2/3$ code by taking the input bits two at a time, or two stages in the original trellis, and think of this as representing a *single* transition of the new code. The resulting trellis is shown in Figure 12.28(b).

□

Besides being used to increase the rate of the code, puncturing can sometimes be used to reduce decoding complexity. In decoding, each state must be extended to 2^k states at the



(a) Trellis for initial punctured code.



(b) Trellis by collapsing two stages of the initial trellis into a single stage.

Figure 12.28: Trellises for a punctured code.

next time. Thus, the decoding complexity scales exponentially with k. Given the trellis for an unpunctured code with rate $R = k/n$ with $k > 1$, if a trellis for an equivalent punctured code having $k' < k$ input bits can be found, then the decoding complexity can be decreased.

Suppose, for example, that the encoder having the trellis in Figure 12.28(b) is used. In decoding, four successor states must be examined for each state, so that the best of four paths to a state must be selected. However, we know that this encoder also has the trellis representation in Figure 12.28(a). Decoding on this trellis only has two successor states for each state. This results in only a two-way comparison, which can be done using a conventional add/compare/select circuit.

Of course, puncturing changes the distance properties of the code: a good rate $R = 2/3$ code is not necessarily obtained by puncturing a good $R = 1/2$ code. Tables of the best $R = 3/4$ and $R = 2/3$ codes obtainable by puncturing are presented in [40].

12.7.1 Puncturing to Achieve Variable Rate

Puncturing can also be used to generate codes of various rates using the same encoder. Such flexibility might be used, for example, to match the code to the channel in a situation in which the channel characteristics might change. Suppose that a rate $R = 1/2$ encoder is used as the "basic" code. As mentioned above, puncturing 1 bit out of every 4 results in a

$R = 2/3$ code. Puncturing 3 out of every 8 bits results in a $R = 4/5$ code.

If the puncturing is done in such a way that bits punctured to obtain the $R = 2/3$ code are included among those punctured to obtain the $R = 4/5$ code, then the $R = 4/5$ codewords are embedded in the $R = 2/3$ codewords. These codewords are, in turn, embedded in the original $R = 1/2$ codewords. Such codes are said to be **rate compatible punctured convolutional** (RCPC) codes. Assuming that all the RCPC codes have the same period (the same width of the P matrix), then the P matrix of a higher rate code is obtained simply by changing one or more of the 1s to 0s. An RCPC code system can be designed so that the encoder and the decoder have the same structure for all the different rates. Extensive tables of codes and puncturing schedules which produce rate compatible codes appear in [130]. Abbreviated tables appear in Table 12.3.

Table 12.3: Best Known $R = 2/3$ and $R = 3/4$ Convolutional Codes Obtained by Puncturing a $R = 1/2$ Code [198]

Initial Code			Punctured Code			Initial Code			Punctured Code		
ν	$g^{(0)}$	$g^{(1)}$	P	d_{free}	$N_{d_{\text{free}}}$	ν	$g^{(0)}$	$g^{(1)}$	P	d_{free}	$N_{d_{\text{free}}}$
2	5	7	$\begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$	3	1	2	5	7	$\begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}$	3	6
3	13	17	$\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$	4	3	3	13	17	$\begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$	4	29
4	31	27	$\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$	4	1	4	31	27	$\begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}$	4	1
5	65	57	$\begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$	6	19	5	65	57	$\begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$	4	1
6	155	117	$\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$	6	1	6	155	117	$\begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$	5	8

12.8 Suboptimal Decoding Algorithms for Convolutional Codes

While the Viterbi algorithm is an optimal decoding algorithm, its complexity grows as 2^{ν}, exponentially with the number of states. The probability bound presented in (1.49) suggests that better performance is obtained by codes with longer memory (constraint length). These two facts conflict: it may not be possible to build a decoder with a sufficiently long memory to achieve some desired level of performance.

The Viterbi algorithm also has fixed decoding costs, regardless of the amount of noise. It would be desirable to have an algorithm which is able to perform fewer computations when there is less noise, adjusting the amount of effort required to decode to the severity of the need.

In this section we present two algorithms which address these problems. These algorithms have decoding complexity which is essentially *constant* as a function of constraint length. Furthermore, the less noisy the channel, the less work the decoders have to do, on average. This makes them typically very fast decoders. These positive attributes are obtained, however, at some price. These are *suboptimal* decoding algorithms: they do not always provide the maximum-likelihood decision. Furthermore, the decoding time and decoder memory required are a random variables, depending on the particular received sequence.

In recent years, the availability of high-speed hardware has led to almost universal use of

Viterbi decoding. However, there are still occasions where very long constraint lengths may be of interest, so these algorithms are still of value. Viterbi algorithms can be practically used on codes with constraint lengths up to about 10, while the sequential algorithms discussed here can be used with constraint lengths up to 50 or more.

The first algorithm is known as the **stack algorithm**, or the ZJ algorithm, after Zigangirov (1966) [388] and Jelinek (1969) [165]. The second algorithm presented is the **Fano algorithm** (1963) [80]. These are both instances of **sequential decoding** algorithms [378].

12.8.1 Tree Representations

While the Viterbi algorithm is based on a trellis representation of the code, the sequential algorithms are best understood using a tree representation. Figure 12.29 shows the tree for the convolutional code with generator $G(x) = \left[1 + x^2, 1 + x + x^2\right]$ whose state diagram and trellis are shown in Figure 12.5. At each instant of time, the input bit selects either the upper branch (input bit = 0) or the lower branch (input bit = 1). The output bits for the code are shown along the branches of the tree. By recognizing common states, it is possible to "fold" the tree back into a trellis diagram.

The tree shown in figure 12.29 is for an input sequence of length 4 in the "branching portion" of the tree, followed by a sequence of zeros which drives the tree back to the all-zero state in the "nonbranching" portion of the tree. The length of the codeword is L branches. Each path of length L from the root node to a leaf node of the tree corresponds to a unique convolutional codeword.

Since the size of the tree grows exponentially with the code length, it is not feasible to search the whole tree. Instead, a partial search of the tree is done, searching those portions of the tree that appear to have the best possibility of succeeding. The sequential decoding algorithms which perform this partial search can be described heuristically as follows: Start at the root node and follow the branches that appear to best match the noisy received data. If, after some decisions, the received word and the branch labels are not matching well, back up and try a different route.

12.8.2 The Fano Metric

As a general rule, paths of differing lengths are compared as the algorithm moves around the tree these in sequential decoding algorithms. A path of length five branches through the tree might be compared with a path of length twenty branches. A first step, therefore, is to determine an appropriate metric for comparing different paths. The log likelihood function used as the branch metric for the Viterbi algorithm is *not* appropriate to use for the sequential algorithm. This is because log likelihood functions are biased against long paths.

Example 12.27 Suppose the transmitted sequence of a rate $R = 1/2$ code is

$$\mathbf{a} = [11, 10, 10, 11, 11, 01, 00, 01]$$

and the received sequence is

$$\mathbf{r} = [01, 10, 00, 11, 11, 01, 00, 01].$$

Using the Hamming distance as the path metric, this is to be compared with a partial path of one branch, $\tilde{\mathbf{a}}^{(1)} = [00]$ and a partial path of six branches, $\mathbf{a}^{(2)} = [11, 10, 10, 11, 11, 01]$. Letting $[\mathbf{r}]_i$ denote i branches of received data, we have

$$d_H([\mathbf{r}]_1, \mathbf{a}^{(1)}) = 1 \qquad d_H([\mathbf{r}]_6, \mathbf{a}^{(2)}) = 2.$$

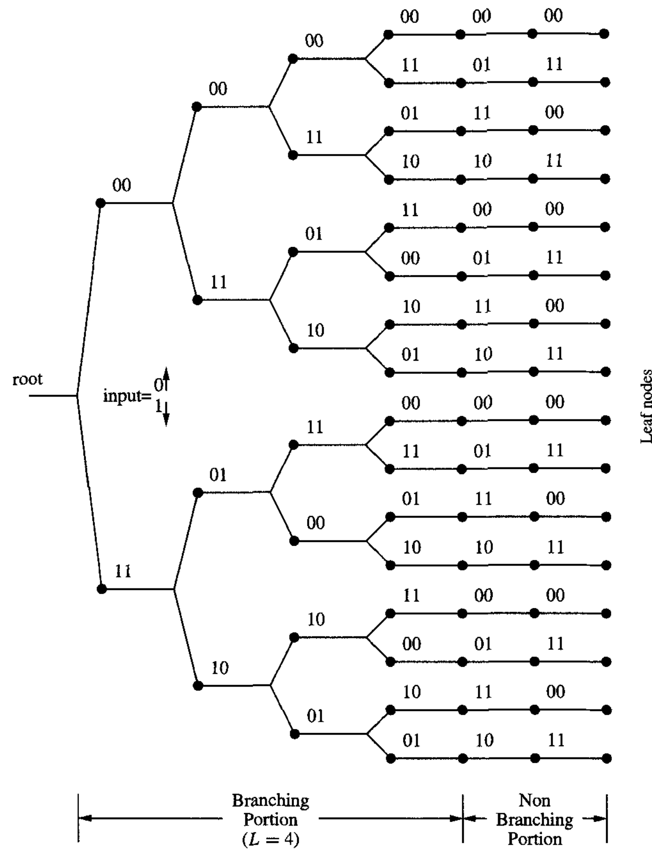


Figure 12.29: A tree representation for a rate $R = 1/2$ code.

By not taking into account the fact that branches of different length are being compared, $\mathbf{a}_1$ appears to be better, since the Hamming distance is smaller. But intuitively, it seems that 2 errors out of 12 bits should be superior to 1 error out of 2 bits. □

The Fano metric is designed to take into account paths of different lengths. Let

$$\tilde{\mathbf{a}}^{(i)} = (\mathbf{a}_0^{(i)}, \mathbf{a}_1^{(i)}, \ldots, \mathbf{a}_{n_i-1}^{(i)})$$

be a partial input sequence of length n_i corresponding to a particular path through the tree, where each $\mathbf{a}_j^{(i)}$ consists of n bit symbols. Accordingly, let us write this as a vector of nn_i bits,

$$\tilde{\mathbf{a}}^{(i)} = (\mathbf{a}_0^{(i)}, \mathbf{a}_1^{(i)}, \ldots, \mathbf{a}_{n_i-1}^{(i)}) = (a_0^{(i)}, a_1^{(i)}, \ldots, a_{n_i n-1}^{(i)}).$$

Assuming that each encoded bit occurs with equal probability, each sequence $\tilde{\mathbf{a}}^{(i)}$ occurs with probability

$$P(\tilde{\mathbf{a}}^{(i)}) = (2^{-k})^{n_i} = 2^{-Rnn_i}. \tag{12.44}$$

Suppose that there are M partial sequences to be compared, represented as elements of the set $\mathcal{X}$,

$$\mathcal{X} = \{\tilde{\mathbf{a}}^{(1)}, \tilde{\mathbf{a}}^{(2)}, \ldots, \tilde{\mathbf{a}}^{(M)}\}.$$

Let $n_{\max}$ be the longest of the partial sequences,

$$n_{\max} = \max(n_1, n_2, \ldots, n_M).$$

Let $\mathbf{r} = (\mathbf{r}_0, \mathbf{r}_1, \ldots, \mathbf{r}_{L-1})$ be a received sequence corresponding to a codeword and let

$$\tilde{\mathbf{r}} = (\mathbf{r}_0, \mathbf{r}_1, \ldots, \mathbf{r}_{n_{\max}-1})$$

be a "partial" sequence, starting at the beginning of $\mathbf{r}$, but extending only through $n_{\max}$ branches. (The tilde is used to represent partial sequences.) Each $\mathbf{r}_i$ consists of n symbols, so we can also write this as a vector of $nn_{\max}$ elements

$$\tilde{\mathbf{r}} = (r_0, r_1, \ldots, r_{nn_{\max}-1}).$$

From among the sequences in $\mathcal{X}$, the optimal receiver chooses the $\tilde{\mathbf{a}}^{(i)}$ which maximizes

$$P(\tilde{\mathbf{a}}^{(i)}|\tilde{\mathbf{r}}) = \frac{P[\tilde{\mathbf{a}}^{(i)}]p(\tilde{\mathbf{r}}|\tilde{\mathbf{a}}^{(i)})}{p(\tilde{\mathbf{r}})}.$$

Assuming (as is typical) that the channel is memoryless, this can be written as

$$P(\tilde{\mathbf{a}}^{(i)}|\tilde{\mathbf{r}}) = \frac{P[\tilde{\mathbf{a}}^{(i)}]\prod_{j=0}^{n_i-1} p(\mathbf{r}_j|\mathbf{a}_j^{(i)})\prod_{j=n_i}^{n_{\max}-1} p(\mathbf{r}_j)}{p(\tilde{\mathbf{r}})}, \tag{12.45}$$

where the second product arises since there are no known data associated with the sequence $\tilde{\mathbf{a}}^{(i)}$ for $j \geq n_i$. Canceling common terms in the numerator and denominator of (12.45) we obtain

$$P(\tilde{\mathbf{a}}^{(i)}|\tilde{\mathbf{r}}) = P[\tilde{\mathbf{a}}^{(i)}]\prod_{j=0}^{n_i-1}\frac{p(\mathbf{r}_j|a_j^{(i)})}{p(\mathbf{r}_j)}.$$

Taking the logarithm of both sides and using (12.44), we have

$$\log_2 P(\tilde{\mathbf{a}}^{(i)}|\tilde{\mathbf{r}}) = \sum_{j=0}^{n_i-1}\left[p(\mathbf{r}_j|\mathbf{a}_j^{(i)}) - p(\mathbf{r}_j)\right] - \log_2 Rnn_i.$$

Each $\mathbf{r}_j$ and $\mathbf{a}_j^{(i)}$ consists of n symbols, so we can write this as

$$\log_2 P(\tilde{\mathbf{a}}^{(i)}|\tilde{\mathbf{r}}) = \sum_{j=0}^{nn_i-1}\left[p(r_j|a_j^{(i)}) - p(r_j)\right] - \log_2 Rnn_i.$$

We use this as the *path metric* and denote it as

$$M(\tilde{\mathbf{a}}^{(i)}, \mathbf{r}) = \log_2 P(\tilde{\mathbf{a}}^{(i)}|\tilde{\mathbf{r}}).$$

The corresponding *branch metric* which is accumulated for each new symbol is

$$\boxed{\mu(r_j, a_j^{(i)}) = \underbrace{\log_2 P(r_j|a_j^{(i)})}_{\text{ML metric}}\underbrace{- \log_2 P(r_j) - R}_{\text{path length bias}}.}$$

This metric is called the **Fano metric**. As indicated, the Fano metric consists of two parts. The first part is the same maximum likelihood metric used for conventional Viterbi decoding. The second part consists of a bias term which accounts for different path lengths. Thus, when

comparing paths of different lengths, the path with the largest Fano metric is considered the best path, most likely to be part of the maximum likelihood path. If all paths are of the same length, then the path length bias becomes the same for all paths and may be neglected.

If transmission takes place over a BSC with transition probability p_c, then $P(r_j = 0) = P(r_j = 1) = \frac{1}{2}$. The branch length bias is

$$\log_2 \frac{1}{P(r_j)} - R = \log_2 2 - R = 1 - R,$$

which is > 0 for all codes of rate $R < 1$. The cumulative path length bias for a path of nn_i bits is $nn_i(1 - R)$: the path length bias increases linearly with the path length. For the BSC, the branch metric is

$$\mu(r_j, a_j) = \begin{cases} \log_2(1 - p_c) - \log_2 \frac{1}{2} - R = \log_2 2(1 - p_c) - R & \text{if } r_j = a_j \\ \log_2 p_c - \log_2 \frac{1}{2} - R = \log_2 2p_c - R & \text{if } r_j \neq a_j. \end{cases} \tag{12.46}$$

Example 12.28 Let us contrast the Fano metric with the ML metric for the data in Example 12.27, assuming that $p_c = 0.1$. Using the Fano metric, we have

$$M([00], \mathbf{r}) = \log_2(1 - p_c) + \log_2 p_c + 2(1 - 1/2) = -2.479$$

$$M([11, 10, 10, 11, 11, 01], \mathbf{r}) = 10 \log_2(1 - p_c) + 2 \log_2 p_c + 12(1 - 1/2) = -2.164.$$

Thus the longer path is has a better (higher) Fano metric than the shorter path. □

Example 12.29 Suppose that $R = 1/2$ and $p_c = 0.1$. Then from (12.46),

$$\mu(r_j, a_j) = \begin{cases} 0.348 & r_j = a_j \\ -2.82 & r_j \neq a_j. \end{cases}$$

It is common to scale the metrics by a constant so that they can be closely approximated by integers. Scaling the metric by $1/0.348$ results in the metric

$$\mu(r_j, a_j) = \begin{cases} 1 & r_j = a_j \\ -8 & r_j \neq a_j. \end{cases}$$

Thus, each bit a_j that agrees with r_j results in a +1 added to the metric. Each bit a_j that disagrees with r_j results in −8 added to the metric. □

A path with only a few errors (the correct path) tends to have a slowly increasing metric, while an incorrect path tends to have a rapidly decreasing metric. Because the metric decreases so rapidly, incorrect paths are not extended far before being effectively rejected.

For BPSK transmission through an AWGN , the branch metric is

$$\mu(r_j, a_j) = \log_2 p(r_j|a_j) - \log_2 p(r_j) - R,$$

where $p(r_j|a_j)$ is the PDF of a Gaussian r.v. with mean a_j and variance $\sigma_2 = N_0/2$ and

$$p(r_j) = \frac{p(r_j|a_j = 1) + p(r_j|a_j = -1)}{2}.$$

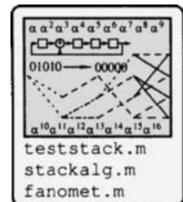
12.8.3 The Stack Algorithm

Let $\tilde{\mathbf{a}}^{(i)}$ represent a path through the tree and let $M(\tilde{\mathbf{a}}^{(i)}, \mathbf{r})$ represent the Fano metric between $\tilde{\mathbf{a}}^{(i)}$ and the received sequence $\mathbf{r}$. These are stored together as a pair $(M(\tilde{\mathbf{a}}^{(i)}, \mathbf{r}), \tilde{\mathbf{a}}^{(i)})$ called a stack entry.

In the stack algorithm, an ordered list of stack entries is maintained which represents all the partial paths which have been examined so far. The list is ordered with the path with the largest (best) metric on top, with decreasing metrics beneath. Each decoding step consists of pulling the top stack entry off the stack, computing the 2^k successor paths and their path metrics to that partial path, then rearranging the stack in order of decreasing metrics. When the top partial path consists of a path from the root node to a leaf node of the tree, then the algorithm is finished.

Algorithm 12.2 The Stack Algorithm

1 **Input:** A sequence $\mathbf{r}_0, \mathbf{r}_1, \ldots, \mathbf{r}_{L-1}$
2 **Output:** The sequence $\mathbf{a}_0, \mathbf{a}_1, \ldots, \mathbf{a}_{L-1}$
3 **Initialize:** Load the stack with the empty path with Fano path metric 0: $S = (\emptyset, 0)$.
4 Compute the metrics of the successors of the top path in the stack
5 Delete the top path from the stack
6 Insert the paths computed in step 4 into the stack, and
  rearrange the stack in order of decreasing metric values.
7 If the top path in the stack terminates at a leaf node of the tree, Stop.
  Otherwise, goto step 4.



Example 12.30 The encoder and received data of Example 12.13 are used in the stack algorithm. We have

$$\mathbf{r} = [11\ 10\ \underline{0}0\ 1\underline{0}\ 11\ 01\ 00\ 01\ \ldots].$$

Figure 12.30 shows the contents of the stack as the algorithm progresses. After 14 steps of the algorithm, the algorithm terminates with the correct input sequence on top. (The metrics here are not scaled to integers.) □

A major part of the expense of the stack algorithm is the need to sort the metrics at every iteration of the algorithm. A variation on this algorithm due to Jelinek [165] known as the *stack bucket algorithm* avoids some of this complication. In the stack bucket algorithm, the range of possible metric values (e.g., for the data in Figure 12.30, the range is from 0.7 to −9.2) is partitioned into fixed intervals, where each interval is allocated a certain number of storage locations called a *bucket*. When a path is extended, it is deleted from its bucket and a new path is inserted as the top item in the bucket containing the metric interval for the new metric. Paths within buckets are *not* reordered. The top path in the nonempty bucket with the highest metric interval is chosen as the path to be extended. Instead of sorting, it only becomes necessary to determine which bucket new paths should be placed in. Unfortunately, the bucket approach does not always choose the best path, but only a "very good" path, to extend. Nevertheless, if there are enough buckets that the quantization into metric intervals is not too coarse, and if the received signal is not too noisy, then the top bucket contains only the best path. Any degradation from optimal is minor.

Another practical problem is that the size of the stack must necessarily be limited. For long codewords, there is always the probability that the stack fills up before the correct

Step 1	
0.7	[1]
−5.6	[0]

Step 2	
1.4	[11]
−4.9	[10]
−5.6	[0]

Step 3	
−1.1	[111]
−1.1	[110]
−4.9	[10]
−5.6	[0]

Step 4	
−0.39	[1110]
−1.1	[110]
−4.9	[10]
−5.6	[0]
−6.7	[1111]

Step 5	
0.31	[11100]
−1.1	[110]
−4.9	[10]
−5.6	[0]
−6	[11101]
−6.7	[1111]

Step 6	
−1.1	[110]
−2.2	[111001]
−2.2	[111000]
−4.9	[10]
−5.6	[0]
−6	[11101]
−6.7	[1111]

Step 7	
−2.2	[111000]
−2.2	[111001]
−3.6	[1101]
−3.6	[1100]
−4.9	[10]
−5.6	[0]
−6	[11101]
−6.7	[1111]

Step 8	
−1.5	[1110000]
−2.2	[111001]
−3.6	[1100]
−3.6	[1101]
−4.9	[10]
−5.6	[0]
−6	[11101]
−6.7	[1111]
−7.8	[1110001]

Step 9	
−2.2	[111001]
−3.6	[1101]
−3.6	[1100]
−3.9	[11100001]
−3.9	[11100000]
−4.9	[10]
−5.6	[0]
−6	[11101]
−6.7	[1111]
−7.8	[1110001]

Step 10	
−3.6	[1100]
−3.6	[1101]
−3.9	[11100000]
−3.9	[11100001]
−4.6	[1110011]
−4.6	[1110010]
−4.9	[10]
−5.6	[0]
−6	[11101]
−6.7	[1111]
−7.8	[1110001]

Step 11	
−2.9	[11001]
−3.6	[1101]
−3.9	[11100001]
−3.9	[11100000]
−4.6	[1110010]
−4.6	[1110011]
−4.9	[10]
−5.6	[0]
−6	[11101]
−6.7	[1111]
−7.8	[1110001]
−9.2	[11000]

Step 12	
−2.2	[110010]
−3.6	[1101]
−3.9	[11100000]
−3.9	[11100001]
−4.6	[1110011]
−4.6	[1110010]
−4.9	[10]
−5.6	[0]
−6	[11101]
−6.7	[1111]
−7.8	[1110001]
−8.5	[110011]
−9.2	[11000]

Step 13	
−1.5	[1100101]
−3.6	[1101]
−3.9	[11100001]
−3.9	[11100000]
−4.6	[1110010]
−4.6	[1110011]
−4.9	[10]
−5.6	[0]
−6	[11101]
−6.7	[1111]
−7.8	[1110001]
−7.8	[1100100]
−8.5	[110011]
−9.2	[11000]

Step 14	
−0.77	[11001010]
−3.6	[1101]
−3.9	[11100000]
−3.9	[11100001]
−4.6	[1110011]
−4.6	[1110010]
−4.9	[10]
−5.6	[0]
−6	[11101]
−6.7	[1111]
−7.1	[11001011]
−7.8	[1110001]
−7.8	[1100100]
−8.5	[110011]
−9.2	[11000]

Figure 12.30: Stack contents for stack algorithm decoding example: metric, [input list].

codeword is found. This is handled by simply throwing away the paths at the bottom of the stack. Of course, if the path that would ultimately become the best path is thrown away at an earlier stage of the algorithm, the best path can never be found. However, if the stack is sufficiently large the probability that the correct path will be thrown away is negligible. Another possibility is to simply throw out the block where the frame overflow occurs and declare an erasure.

12.8.4 The Fano Algorithm

While the stack algorithm moves around the decoding tree, and must therefore save information about each path still under consideration, the Fano algorithm retains only one path and moves through the tree only along the edges of the tree. As a result, it does not require as much memory as the stack algorithm. However, some of the nodes are visited more than once, requiring recomputation of the metric values.

The general outline of the decoder is as follows. A threshold value T is maintained by the algorithm. The Fano decoder moves forward through the tree as long as the path metric at the next node (the "forward" node), denoted by M_F, exceeds the threshold and the path metric continues to increase (improve). The algorithm is thus a depth-first search. When the path metric would drop below the threshold if a forward move were made, the decoder examines the preceding node (the "backward" node, with path metric M_B). If the path metric at the backward node does not exceed the current threshold, then the threshold is reduced by Δ, and the decoder examines the next forward node. If the path metric at the previous node does exceed the threshold, then the decoder backs up and begins to examine other paths from that node. (This process of moving forward, then backward, then adjusting the threshold and moving forward again is why nodes may be visited many times.) If all nodes forward of that point have already been examined, then the decoder once again considers backing up. Otherwise, the decoder moves forward on one of the remaining nodes.

Each time a node is visited for the first time (and if it is not at the end of the tree) the decoder "tightens" the threshold by the largest multiple of Δ such that the adjusted threshold does not exceed the current metric. (As an alternative, tightening is accomplished in some algorithms by simply setting $T = M_F$.)

Since the Fano algorithm does backtracking, the algorithm needs to keep the following information at each node along the path that it is examining: the path metric at the previous node, M_B; which of the 2^k branches it has taken; the input at that node; and also the state of the encoder, so that next branches in the tree can be computed. A forward move consists of adding this information to the end of the path list. A backward move consists of popping this information from the end of the path list. The path metric at the root node is set at $M_B = -\infty$; when the decoder backs up to that node, the threshold is always reduced and the algorithm moves forward again.

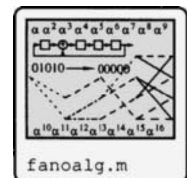

fanoalg.m

Figure 12.31 shows the flowchart for the Fano algorithm. The number i indicates the length of the current path. At each node, all possible branches might be taken. The metric to each next node is stored in sorted order in the array P. At each node along the path, the number t_i indicates which branch number has been taken. When $t_i = 0$, the branch with the best metric is chosen, when $t_i = 1$ the next best metric is chosen, and so forth. The information stored about each node along the path includes the input, the state at that node, and t_i. (Recomputing the metric when backtracking could be avoided by also storing P at each node.)

The threshold is adjusted by the quantity Δ. In general, the larger Δ is, the fewer the

number of computations are required. Ultimately, Δ must be below the likelihood of the maximum likelihood path, and so must be lowered to that point. If Δ is too small, then many iterations might be required to get to that point. On the other hand, if Δ is lowered in steps that are too big, then the threshold might be set low enough that other paths which are not the maximum likelihood path also exceed the threshold and can be considered by the decoder. Based on simulation experience [203], Δ should be in the range of (2,8) if unscaled metrics are used. If scaled metrics are used, then Δ should be scaled accordingly. The value of Δ employed should be explored by thorough computer simulation to ensure that it gives adequate performance.

Example 12.31 Suppose the same code and input sequence as in Example 12.13 is used. The following traces the execution of the algorithm when the scaled (integer) metric of Example 12.29 is used with $\Delta = 10$. The step number n is printed every time the algorithm passes through point A in the flow chart.

$n = 1$: $T = 0$ $P = [2 - 16]$ $t_0 = 0$
Look forward: $M_F = 2$
$M_F \geq T$. Move forward
First visit: Tighten T
$M_F = 2$ $M_B = 0$ Node=[1]
$M = 2$ $T = 0$

$n = 2$: $T = 0$ $P = [4 - 14]$ $t_1 = 0$
Look forward: $M_F = 4$
$M_F \geq T$. Move forward
First visit: Tighten T
$M_F = 4$ $M_B = 2$ Node=[11]
$M = 4$ $T = 0$

$n = 3$: $T = 0$ $P = [-3 - 3]$ $t_2 = 0$
Look forward: $M_F = -3$
$M_F < T$: Look back
$M_B = 2$
$M_B \geq T$: Move back
All forward nodes not yet tested. $t_1 = 1$
$M_F = -3$ $M_B = 2$ Node=[1]
$M = 2$ $T = 0$

$n = 4$: $T = 0$ $P = [4 - 14]$ $t_1 = 1$
Look forward: $M_F = -14$
$M_F < T$: Look back
$M_B = 0$
$M_B \geq T$: Move back
All forward nodes not yet tested. $t_0 = 1$
$M_F = -14$ $M_B = 0$ Node=[]
$M = 0$ $T = 0$

$n = 5$: $T = 0$ $P = [2 - 16]$ $t_0 = 1$
Look forward: $M_F = -16$
$M_F < T$: Look back
$M_B = -\infty$
$M_B < T$: $T = T - \Delta$
$M_F = -16$ $M_B = -\infty$ Node=[]
$M = 0$ $T = -10$

$n = 6$: $T = -10$ $P = [2 - 16]$ $t_0 = 0$
Look forward: $M_F = 2$
$M_F \geq T$. Move forward
$M_F = 2$ $M_B = 0$ Node=[1]
$M = 2$ $T = -10$

$n = 7$: $T = -10$ $P = [4 - 14]$ $t_1 = 0$
Look forward: $M_F = 4$
$M_F \geq T$. Move forward
$M_F = 4$ $M_B = 2$ Node=[11]
$M = 4$ $T = -10$

$n = 8$: $T = -10$ $P = [-3 - 3]$ $t_2 = 0$
Look forward: $M_F = -3$
$M_F \geq T$. Move forward
First visit: Tighten T
$M_F = -3$ $M_B = 4$ Node=[111]
$M = -3$ $T = -10$

$n = 9$: $T = -10$ $P = [-1 - 19]$ $t_3 = 0$
Look forward: $M_F = -1$
$M_F \geq T$. Move forward
First visit: Tighten T
$M_F = -1$ $M_B = -3$ Node=[1110]
$M = -1$ $T = -10$

$n = 10$: $T = -10$ $P = [1 - 17]$ $t_4 = 0$
Look forward: $M_F = 1$
$M_F \geq T$. Move forward
First visit: Tighten T
$M_F = 1$ $M_B = -1$ Node=[11100]
$M = 1$ $T = 0$

$n = 11$: $T = 0$ $P = [-6 - 6]$ $t_5 = 0$
Look forward: $M_F = -6$
$M_F < T$: Look back
$M_B = -1$
$M_B < T$: $T = T - \Delta$
$M_F = -6$ $M_B = -1$ Node=[11100]
$M = 1$ $T = -10$

$n = 12$: $T = -10$ $P = [-6 - 6]$ $t_5 = 0$
Look forward: $M_F = -6$
$M_F \geq T$. Move forward
First visit: Tighten T
$M_F = -6$ $M_B = 1$ Node=[111001]
$M = -6$ $T = -10$

$n = 13$: $T = -10$ $P = [-13 - 13]$
$t_6 = 0$
Look forward: $M_F = -13$
$M_F < T$: Look back
$M_B = 1$
$M_B \geq T$: Move back
All forward nodes not yet tested. $t_5 = 1$
$M_F = -13$ $M_B = 1$ Node=[11100]
$M = 1$ $T = -10$

$n = 14$: $T = -10$ $P = [-6 - 6]$ $t_5 = 1$
Look forward: $M_F = -6$
$M_F \geq T$. Move forward
First visit: Tighten T
$M_F = -6$ $M_B = 1$ Node=[111000]
$M = -6$ $T = -10$

$n = 15$: $T = -10$ $P = [-4 - 22]$ $t_6 = 0$
Look forward: $M_F = -4$
$M_F \geq T$. Move forward
First visit: Tighten T
$M_F = -4$ $M_B = -6$ Node=[1110000]
$M = -4$ $T = -10$

$n = 16$: $T = -10$ $P = [-11 - 11]$
$t_7 = 0$
Look forward: $M_F = -11$
$M_F < T$: Look back
$M_B = -6$
$M_B \geq T$: Move back
All forward nodes not yet tested. $t_6 = 1$
$M_F = -11$ $M_B = -6$ Node=[111000]
$M = -6$ $T = -10$

$n = 17$: $T = -10$ $P = [-4 - 22]$ $t_6 = 1$
Look forward: $M_F = -22$
$M_F < T$: Look back
$M_B = 1$
$M_B \geq T$: Move back
No more forward nodes
$M_B = -1$
$M_B \geq T$: Move back
All forward nodes not yet tested. $t_4 = 1$
$M_F = -22$ $M_B = -1$ Node=[1110]
$M = -1$ $T = -10$

$n = 18$: $T = -10$ $P = [1 - 17]$ $t_4 = 1$
Look forward: $M_F = -17$
$M_F < T$: Look back
$M_B = -3$
$M_B \geq T$: Move back
All forward nodes not yet tested. $t_3 = 1$
$M_F = -17$ $M_B = -3$ Node=[111]
$M = -3$ $T = -10$

$n = 19$: $T = -10$ $P = [-1 - 19]$ $t_3 = 1$
Look forward: $M_F = -19$
$M_F < T$: Look back
$M_B = 4$
$M_B \geq T$: Move back
All forward nodes not yet tested. $t_2 = 1$
$M_F = -19$ $M_B = 4$ Node=[11]
$M = 4$ $T = -10$

$n = 20$: $T = -10$ $P = [-3 - 3]$ $t_2 = 1$
Look forward: $M_F = -3$
$M_F \geq T$. Move forward
First visit: Tighten T
$M_F = -3$ $M_B = 4$ Node=[110]
$M = -3$ $T = -10$

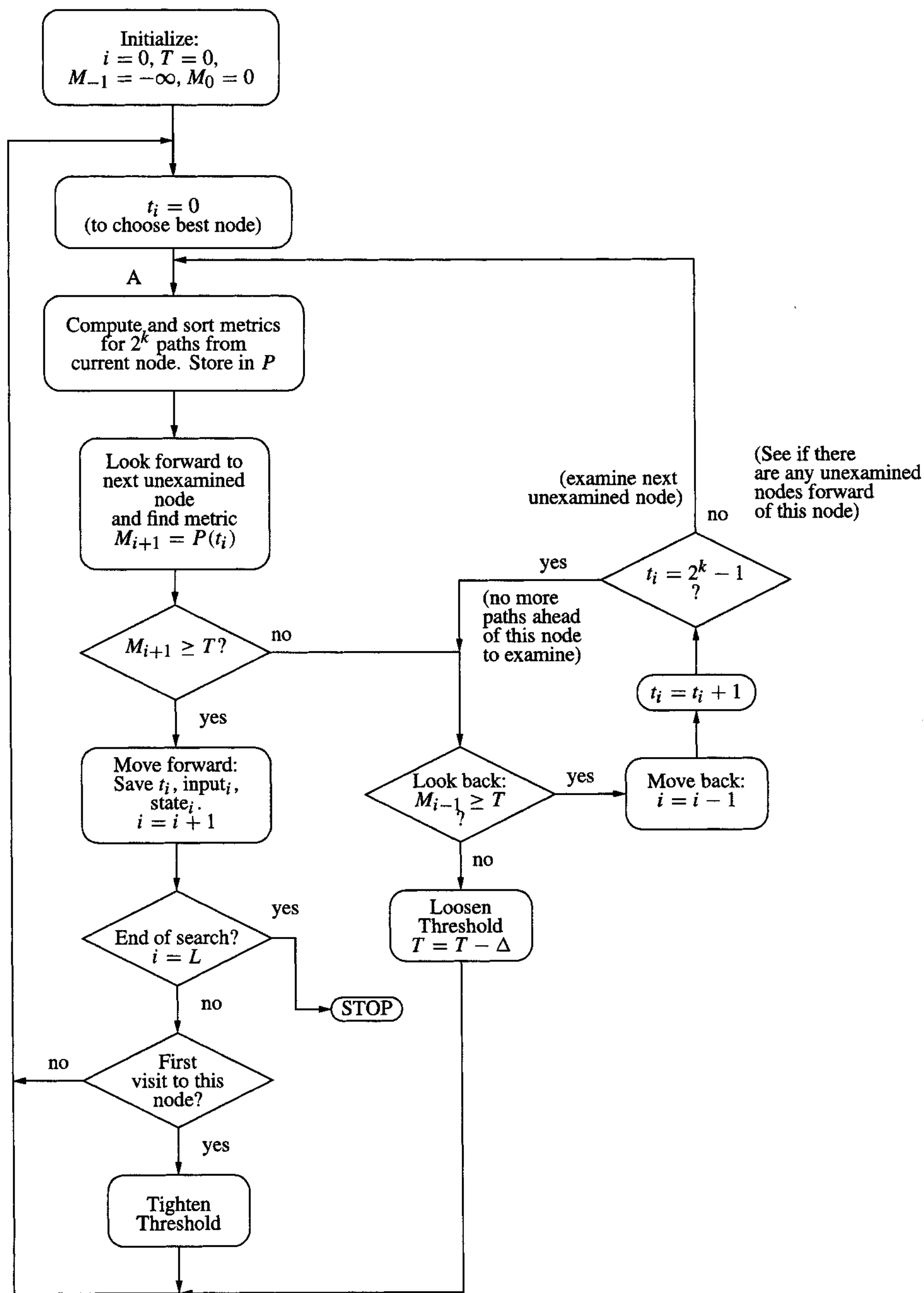


Figure 12.31: Flowchart for the Fano algorithm.

$n = 21$: $T = -10$ $P = [-10 - 10]$
$t_3 = 0$
Look forward: $M_F = -10$
$M_F \geq T$. Move forward
First visit: Tighten T
$M_F = -10$ $M_B = -3$ Node=[1101]
$M = -10$ $T = -10$

$n = 22$: $T = -10$ $P = [-17 - 17]$
$t_4 = 0$
Look forward: $M_F = -17$
$M_F < T$: Look back
$M_B = -3$
$M_B \geq T$: Move back
All forward nodes not yet tested. $t_3 = 1$
$M_F = -17$ $M_B = -3$ Node=[110]
$M = -3$ $T = -10$

$n = 23$: $T = -10$ $P = [-10 - 10]$
$t_3 = 1$
Look forward: $M_F = -10$
$M_F \geq T$. Move forward
First visit: Tighten T
$M_F = -10$ $M_B = -3$ Node=[1100]
$M = -10$ $T = -10$

$n = 24$: $T = -10$ $P = [-8 - 26]$ $t_4 = 0$
Look forward: $M_F = -8$
$M_F \geq T$. Move forward
First visit: Tighten T
$M_F = -8$ $M_B = -10$ Node=[11001]
$M = -8$ $T = -10$

$n = 25$: $T = -10$ $P = [-6 - 24]$ $t_5 = 0$
Look forward: $M_F = -6$
$M_F \geq T$. Move forward
First visit: Tighten T
$M_F = -6$ $M_B = -8$ Node=[110010]
$M = -6$ $T = -10$

$n = 26$: $T = -10$ $P = [-4 - 22]$ $t_6 = 0$
Look forward: $M_F = -4$
$M_F \geq T$. Move forward
First visit: Tighten T
$M_F = -4$ $M_B = -6$ Node=[1100101]
$M = -4$ $T = -10$

$n = 27$: $T = -10$ $P = [-2 - 20]$ $t_7 = 0$
Look forward: $M_F = -2$
$M_F \geq T$. Move forward
First visit: Tighten T
$M_F = -2$ $M_B = -4$ Node=[11001010]
$M = -2$ $T = -10$

For this particular set of data, the value of Δ has a tremendous impact both on the number of steps the algorithm takes and whether it decodes correctly. Table 12.4 shows that the number of decoding steps decreases typically as Δ gets larger, but that the decoding might be incorrect for some values of Δ.

Table 12.4: Performance of Fano Algorithm on a Particular Sequence as a Function of Δ

Δ	Number of Decoding Steps	Correct Decoding	Δ	Number of Decoding Steps	Correct Decoding
1	158	yes	7	31	no
2	86	yes	8	31	no
3	63	no	9	31	no
4	47	no	10	27	yes
5	40	yes	11	16	no
6	33	no	12	16	no

□

In comparing the stack algorithm and the Fano algorithm, we note the following.

- The stack algorithm visits each node only once, but the Fano algorithm may revisit nodes.
- The Fano algorithm does not have to manage the stack (e.g., resort the metrics).

Despite its complexity, when the noise is low the Fano algorithm tends to decode faster than the stack algorithm. However, as the noise increases more backtracking might be required and the stack algorithm has the advantage. Overall, the Fano algorithm is usually selected when sequential decoding is employed.

12.8.5 Other Issues for Sequential Decoding

We briefly introduce some issues related to sequential decoding, although space precludes a thorough treatment. References are provided for interested readers.

Computational complexity The computational complexity is a random variable, and so is described by a probability distribution. Discussions of the performance of the decoder appear in [302, 161, 164, 90].

Code design The performance of a code decoded using the Viterbi algorithm is governed largely by the free distance d_{free}. For sequential decoding however, the codewords must have a distance that increases as rapidly as possible over the first few symbols of the codeword (i.e., the code must have a good **column distance function**) so that the decoding algorithm can make good decisions as early as possible. A large free distance and a small number of nearest neighbors are also important. A code having an optimum distance profile is one in which the column distance function over the first constraint length is better than all other codes of the same rate and constraint length. Tables of codes having optimum distance profiles are provided in [174]. Further discussion and description of the algorithms for finding these codes appear in [169, 170, 171, 172, 43, 173].

Variations on sequential decoding algorithms In the interest of reducing the statistical variability of the decoding, or improving the decoder performance, variations on the decoding algorithms have been developed. In [48], a multiple stack algorithm is presented. This operates like the stack algorithm, except that the stack size is limited to a fixed number of entries. If the stack fills up before decoding is complete, the top paths are transferred to a second stack and decoding proceeds using these best paths. If this stack also fills up before decoding is complete, a third stack is created using the best paths, and so forth. In [79] and [166], hybrid algebraic/sequential decoding was introduced in which algebraic constraints are imposed across frames of sequentially decoded data. In [129], a generalized stack algorithm was proposed, in which more than one path in the stack can be extended at one time (as in the Viterbi algorithm) and paths merging together are selected as in the Viterbi algorithm. Compared to the stack algorithm, the generalized stack algorithm does not have buffer size variations as large and the error probability is closer to that of the Viterbi algorithm.

12.8.6 A Variation on the Viterbi Algorithm: The M Algorithm

For a trellis with a large number of states at each time instant, the Viterbi algorithm can be very complex. Furthermore, since there is only one correct path, most of the computations are expended in propagating paths that are not be used, but must be maintained to ensure the optimality of the decoding procedure. The M algorithm (see, e.g., [270]) is a suboptimal, breadth-first decoding algorithm whose complexity is parametric, allowing for more complexity in the decoding algorithm while decoding generally closer to the optimum.

A list of M paths is maintained. At each time step, these M paths are extended to $M2^k$ paths (where k is the number of input bits), and the path metric along each of these paths is computed just as for the Viterbi algorithm. The path metrics are sorted, then the best M paths are retained in preparation for the next step. At the end of the decoding cycle, the path with the best metric is used as the decoded value. While the underlying graphical structure for the Viterbi algorithm is a trellis, in which paths merge together, the underlying graphical structure for the M algorithm is a tree: the merging of paths is not explicitly represented, but better paths are retained by virtue of the sorting operation. The M-algorithm is thus a cross between the stack algorithm (but using all paths of the same length) and the Viterbi

algorithm. If M is equal to the number of states, the M algorithm is nearly equivalent to the Viterbi algorithm. However, it is possible for M to be significantly less than the number of states with only modest loss of performance.

Another variation is the T-algorithm. It starts just like the M algorithm. However, instead of retaining only the best M paths, in the T algorithm all paths which are within a threshold T of the best path at that stage are retained.

12.9 Convolutional Codes as Block Codes

In this section, we use $m = \max_i v_i$ as the maximum amount of memory in any of the elements of the transfer function matrix.

A block code takes a fixed-length block of k symbols and maps it to a block of n symbols. Convolutional codes, on the other hand, can operate on an entire "stream" of data: a stream of data is simply passed through the filtering system of the convolutional coder. As a result, the "block length" of the code is not usually referred to in the context of convolutional codes.

However, convolutional encoders can be used as encoders for block codes. In fact, this perspective allows bounds on block codes to provide useful bounds for convolutional codes (see, e.g., [197]). Here are some natural ways that block codes can be obtained from convolutional codes. (This discussion applies to polynomial transfer function matrices. Some modifications are necessary for creation of the transfer function matrices.)

Truncation A sequence of L blocks of k-bits can be input to the decoder. This results in an (nL, nk) decoder. This has the disadvantage that there is little (if any) error protection afforded to the last digits input into the encoder [213], resulting in what is called unequal error protection. The effect of unequal error protection is shown in Figure 12.19. Decoding takes place using the Viterbi algorithm starting in state 0 and ending after L stages at any state.

Zero tail Following L k-bit blocks of bits, a sequence of m k-bit blocks of zeros is input to the encoder, driving the state of the encoder to the zero state. The resultant code is an $((L+m)n, kL)$ decoder, with rate $R = kL/(L+m)n$. There is thus a loss of rate, but for large block lengths the rate reduction is negligible. Decoding is accomplished using the Viterbi algorithm starting in state 0 and ending in state 0.

Tail biting In a tail-biting codeword, no additional bits are appended to drive the encoder to the zero state. Instead, the encoder ends in whatever state it happens to end in after the input bits are encoded. There is thus no loss of rate due to the zero-state forcing sequence. The encoder is modified to avoid the problem of unequal error protection by allowing it to start in *any state* and not just the 0-state. Then the initial state is determined by the terminal bits in the sequence. Then valid codewords are those that *start and end in the same state.*

For feedforward encoders, the state is determined by the most recent ν input bits. The final state is thus determined by the last ν input bits. Since the initial and final state must match, the initial state is also determined by the last ν input bits. This allows one to view the trellis as a circular trellis: the final state of the trellis wraps around to become the initial state of the trellis. (This circular trellis structure initially gave rise to the term tail-biting.)

Decoding a tail-biting code more complicated: the Viterbi algorithm should find a path which starts and ends in the same state. In principle, this could require running

the decoder 2^k times, starting in each possible state, then checking that the best path terminates in the same state as the starting state. This is computationally infeasible for many codes. Variations on tail biting codes and their decoding algorithms, are described in [213]. One variation runs through two passes. In the first pass, decoding starts at an arbitrary state (such as the 0 state) and finds the terminal state with the best path metric. Then the Viterbi algorithm is run again, starting with the initial state as that terminal state. Other alternatives are in [4].

12.10 Trellis Representations of Block and Cyclic Codes

In this section we take a dual perspective to that of the previous section: we describe how linear block codes can be represented in terms of a trellis. Besides theoretical insight, the trellis representation can also be used to provide a means of soft-decision decoding that does not depend upon any particular algebraic structure of the code. These decoding algorithms can make block codes "more competitive with convolutional codes" [205, p. 3].

12.10.1 Block Codes

We demonstrate the trellis idea with a (7, 4, 3) binary Hamming code. Let

$$H = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \mathbf{h}_1 & \mathbf{h}_2 & \mathbf{h}_3 & \mathbf{h}_4 & \mathbf{h}_5 & \mathbf{h}_6 & \mathbf{h}_7 \end{bmatrix} \qquad (12.47)$$

be the parity check matrix for the code. Then a column vector $\mathbf{x}$ is a codeword if and only if $\mathbf{s} = H\mathbf{x} = \mathbf{0}$; that is, the syndrome $\mathbf{s}$ must satisfy

$$\mathbf{s} = \begin{bmatrix} \mathbf{h}_1 & \mathbf{h}_2 & \mathbf{h}_3 & \mathbf{h}_4 & \mathbf{h}_5 & \mathbf{h}_6 & \mathbf{h}_7 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix} = \sum_{i=1}^{7} \mathbf{h}_i x_i = \mathbf{0}.$$

We define the *partial syndrome* by

$$\mathbf{s}_{r+1} = \sum_{i=1}^{r} \mathbf{h}_i x_i = \mathbf{s}_r + \mathbf{h}_r x_r,$$

with $\mathbf{s}_1 = \mathbf{0}$. Then the $\mathbf{s}_{n+1} = \mathbf{s}$.

A trellis representation of a code is obtained by using $\mathbf{s}_r$ as the state, with an edge between a state $\mathbf{s}_r$ and $\mathbf{s}_{r+1}$ if $\mathbf{s}_{r+1} = \mathbf{s}_r$ (corresponding to $x_r = 0$) or if $\mathbf{s}_{r+1} = \mathbf{s}_r + \mathbf{h}_r$ (corresponding to $x_r = 1$). Furthermore, the trellis is terminated at the state $\mathbf{s}_{n+1} = \mathbf{0}$, corresponding to the fact that a valid codeword has a syndrome of zero. The trellis has at most 2^{n-k} states in it.

Figure 12.32 shows the trellis for the parity check matrix of (12.47). Horizontal transitions correspond to $x_i = 0$ and diagonal transitions correspond to $x_i = 1$. Only those paths which end up at $\mathbf{s}_8 = \mathbf{0}$ are retained.

As may be observed, the trellis for a block code is "time-varying" — it has different connections for each section of the trellis. The number of states "active" at each section of the trellis also varies.

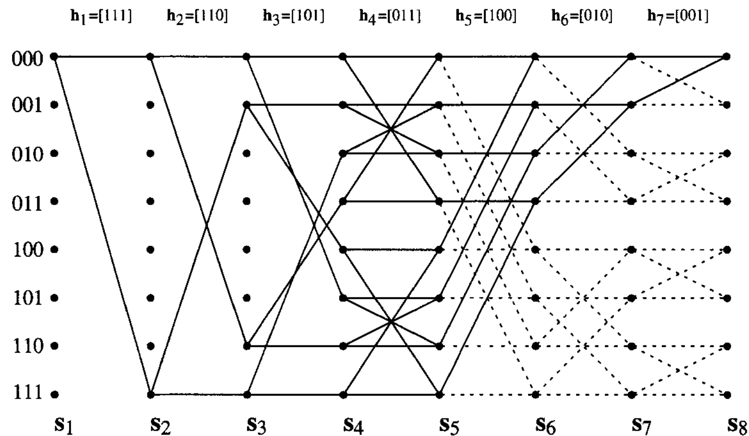


Figure 12.32: The trellis of a (7, 4) Hamming code.

For a general code, the trellis structure is sufficiently complicated that it may be difficult to efficiently represent in hardware. There has been recent work, however, on families of codes whose trellises have a much more regular structure. These are frequently obtained by recursive constructions (e.g., based on Reed-Muller codes). Interested readers can consult [205].

12.10.2 Cyclic Codes

An alternative formulation of a trellis is available for a cyclic code. Recall that a cyclic code can be encoded using a linear feedback shift register as a syndrome computer. The sequence of possible states in this encoder determines a trellis structure which can be used for decoding. We demonstrate the idea again using a (7, 4, 3) Hamming decoder, this time represented as a cyclic code with generator polynomial $g(x) = x^3 + x + 1$.

Figure 12.33 shows a systematic encoder. For the first $k = 4$ clock instants, switch 1 is closed (enabling feedback) and switch 2 is in position 'a'. After the systematic part of the data has been clocked through, switch 1 is opened and switch 2 is moved to position 'b'. The state contents then shift out as the coefficients of the remainder polynomial. Figure

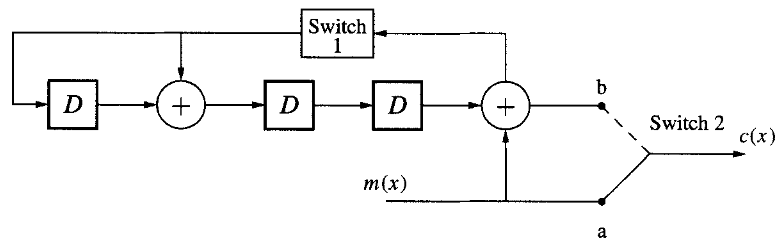


Figure 12.33: A systematic encoder for a (7, 4, 3) Hamming code.

12.34 shows the trellis associated with this encoder. For the first $k = 4$ bits, the trellis state depends upon the input bit. The coded output bit is equal to the input bit. For the last

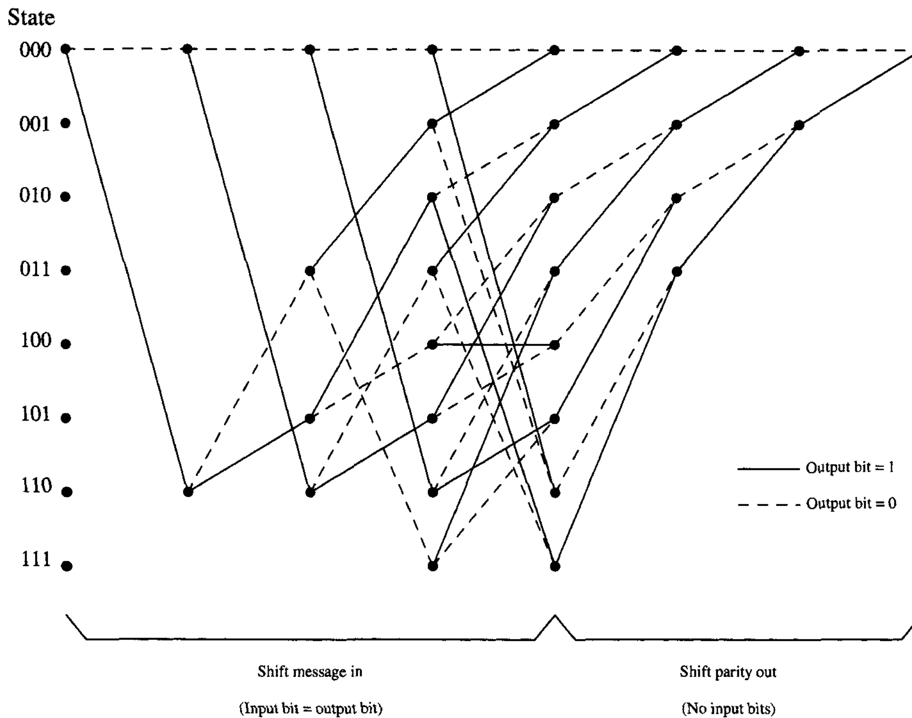


Figure 12.34: A trellis for a cyclically encoded (7,4,3) Hamming code.

$n - k = 3$ bits, the next state is determined simply by shifting the current state. There are no input bits so the output is equal to the bit that is shifted out of the registers.

12.10.3 Trellis Decoding of Block Codes

Once a trellis for a code is established by either of the methods described above, the code can be decoded with a Viterbi algorithm. The time-varying structure of the trellis makes the indexing in the Viterbi algorithm perhaps somewhat more complicated, but the principles are the same. For example, if BPSK modulation is employed, so that the transmitted symbols are $a_i = 2c_i - 1 \in \{\pm 1\}$, and that the channel is AWGN, the branch metric for a path taken with input x_i is $(r_i - (2x_i - 1))^2$. Such soft decision decoding can be shown to provide up to 2 dB of gain compared to hard decision decoding (see, for example [303, pp. 222–223]). However, this improvement does not come without a cost: for codes of any appreciable size, the number of states 2^{n-k} can be so large that trellis-based decoding is infeasible.

Programming Laboratory 9:

Programming Convolutional Encoders

Objective

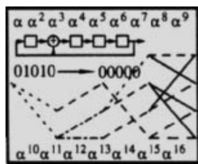
In this lab you are to create a program structure to implement both polynomial and systematic rational convolutional encoders.

Background

Reading: Sections 12.1, 12.2.

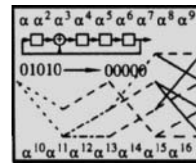
Since both polynomial and systematic ration encoders are "convolutional encoders" and they share many attributes. Furthermore, when we get to the decoding operations, it is convenient to employ one decoder which operates on data from either kind of encoder. As a result, it is structurally convenient to create a base class `BinConv`, then create two derived classes, `BinConvFIR` and `BinConvIIR`. Since the details of the encoding operation and the way the state is determined differ, each of these classes employs its own encoder function. To achieve this, a virtual function `encode` is declared in the base class, which is then realized separately in each derived class.[6] Also, virtual member functions `getstate` and `setstate` are used for reading and setting the state of the encoder. These can be used for testing purposes; they are also used to build information tables that the decoder uses.

The declaration for the `BinConv.h` base class is shown here.



Algorithm 12.3 Base Class for Binary Convolutional Encoder
File: `BinConv.h`

The derived classes `BinConvFIR` and `BinConvIIR` are outlined here.



Algorithm 12.4 Derived classes for FIR and IIR Encoders
File: `BinConvFIR.h`
`BinConvIIR.h`
`BinConvFIR.cc`
`BinConvIIR.cc`

Programming Part

1) Write a class `BinConvFIR` that implements convolutional encoding for a general polynomial encoder. That is, the generator matrix is of the form in (12.1), where each $g^{(i,j)}(x)$ is a polynomial. The class should have an appropriate constructor and destructor. The class should implement the virtual functions `encode`, `getstate`, and `setstate`, as outlined above.

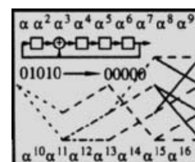
Test your encoder as follows:

a) Using the encoder with transfer function

$$G(x) = \begin{bmatrix} 1+x^2 & 1+x+x^2 \end{bmatrix},$$

verify that the impulse response is correct, that the `getstate` and `nextstate` functions work as expected, and that the state/nextstate table is correct. Use Figure 12.5.

The program `testconvenc.cc` may be helpful.



Algorithm 12.5 Test program for convolutional encoders
File: `testconvenc.cc`

b) The polynomial transfer function

$$G(x) = \begin{bmatrix} x^2+x+1 & x^2 & 1+x \\ x & 1 & 0 \end{bmatrix} \tag{12.48}$$

has the state diagram and trellis shown in Figure 12.35. Verify that for this encoder, the impulse response is correct, that the `getstate` and `nextstate` functions work as expected, and that the state/nextstate table is correct.

2) Write a class `BinConvIIR` that implements a systematic encoder (possibly employing IIR filters). The generator

[6] This is a tradeoff between flexibility and speed. In operation, the virtual functions are called via a pointer, so there is a pointer-lookup overhead associated with them. This also means that virtually called functions cannot be inline, even if they are very small. However, most of the computational complexity associated with these codes is associated with the decoding operation, which takes advantage of precomputed operations. So for our purposes, the virtual function overhead is not too significant.

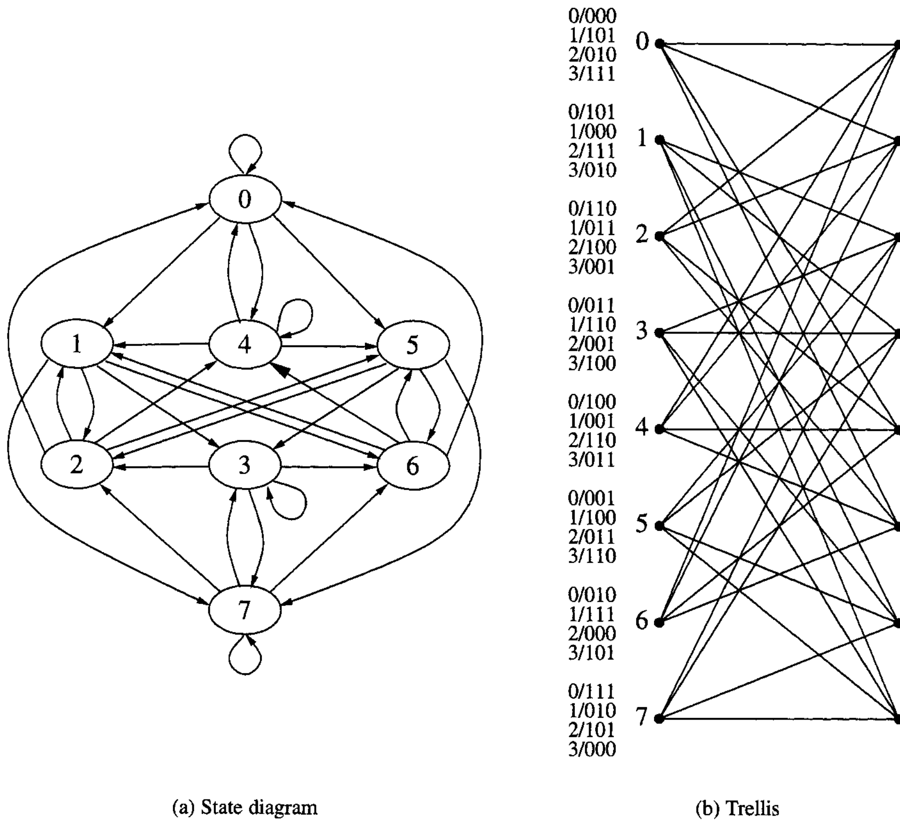


Figure 12.35: State diagram and trellis for the encoder in (12.48)

matrix is of the form

$$G(x) = \begin{bmatrix} I & \begin{matrix} \frac{p_1(x)}{q_1(x)} \\ \frac{p_2(x)}{q_2(x)} \\ \vdots \\ \frac{p_k(x)}{q_k(x)} \end{matrix} \end{bmatrix}$$

for polynomials $p_i(x)$ and $q_i(x)$.

Test your class using the recursive systematic encoder of (12.2), checking as for the first case. (You may find it convenient to find the samples of the impulse by long division.)

Programming Laboratory 10: Convolutional Decoders: The Viterbi Algorithm

Objective

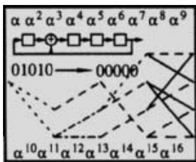
You are to write a convolutional decoder class that decodes using both hard and soft metrics with the Viterbi algorithm.

Background

Reading: Section 12.3

While there are a variety of ways that the Viterbi algorithm can be structured in C++, we recommend using a base class `Convdec.h` that implements that actual Viterbi algorithm and using a virtual function `metric` to compute the metric. This is used by derived classes `BinConvdec01` (for binary 0-1 data) and `BinConvBPSK` (for BPSK modulated data), where each derived class has its own metric function.

The base class `Convdec` This class is a base class for all of the Viterbi-decoded objects.

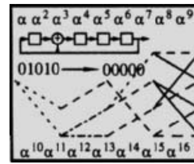


Algorithm 12.6 The Base Decoder Class Declarations
File: `Convdec.h`
`Convdec.cc`

In this class, an object of type `BinConv` (which could be either an FIR or IIR convolutional encoder, if you have used the class specification in lab 9) is passed in. The constructor builds appropriate data arrays for the Viterbi algorithm, placing them in the variables `prevstate` and `inputfrom`. A virtual member function `metric` is used by derived classes to compute the branch metric. The core of the algorithm is used in the member function `viterbi`, which is called by the derived classes. Some other functions are declared:

- `showpaths` — You may find it helpful while debugging to dump out information about the paths. This function (which you write) should do this for you.
- `getinpnow` — This function decodes the last available branch in the set of paths stored, based on the best most recent metric. If `adv` is asserted, the pointer to the end of the branches is incremented. This can be used for dumping out the decisions when the end of the input stream is reached.
- `buildprev` builds the state/previous state array, which indicates the connections between states of the trellis.

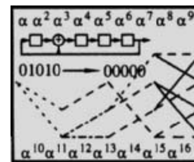
The derived class `BinConvdec01` The first derived class is `BinConvdec01.h`, for binary 0-1 decoding using the Hamming distance as the branch metric.



Algorithm 12.7 Convolutional decoder for binary (0,1) data
File: `BinConvdec01.h`
`BinConvdec01.cc`

This class provides member data `outputmat`, which can be used for direct lookup of the output array given the state and the input. Since the output is, in general, a vector quantity, this is a three-dimensional array. It is recommended that space be allocated using `CALLOCTENSOR` defined in `matalloc.h`. The member variable `data` is used by the `metric` function, as shown. The class description is complete as shown here, except for the function `buildoutputmat`, which is part of the programming assignment.

The derived class `BinConvdecBPSK` The next derived class is `BinConvdecBPSK.h`, for decoding BPSK-modulated convolutionally coded data using the Euclidean distance as the branch metric.



Algorithm 12.8 Convolutional decoder for BPSK data
File: `BinConvdecBPSK.h`
`BinConvdecBPSK.cc`

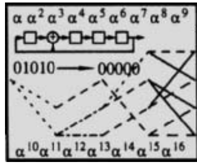
As for the other derived class, space is provided for `outputmat` and `data`; the class is complete as presented here except for the function `buildoutputmat`.

Programming Part

1) Finish the functions in `Convdec.cc`.

2) Test the binary (0,1) `BinConvdec01` decoder for the encoder $G(x) = [1 + x^2, 1 + x + x^2]$ by reproducing the results in Example

12.13. The program `testconvdec` can help.



Algorithm 12.9 Test the convolutional decoder
File: `testconvdec.cc`

3) Test the convolutional decoder `BinConvdecBPSK` by modulating the $\{0, 1\}$ data. Again, `testconvdec` can help.
4) Determine the performance of the encoder $G(x) = \begin{bmatrix} 1+x^2 & 1+x+x^2 \end{bmatrix}$ by producing an error curve on the AWGN channel using BPSK modulation with a soft metric. Compare the soft metric performance with hard metric performance, where the BSC is modeled as having crossover probability $p_c = Q(\sqrt{2E_c/N_0})$. Compare the two kinds coded performances with uncoded BPSK modulation. Also, plot the bound (12.40) and the approximation (12.42) on the same graph.

How much coding gain is achieved? How do the simulation results compare with the theoretical bounds/approximations? How do the simulations compare with the theoretically predicted asymptotic coding gain? How much better is the soft-decoding than the hard-decoding?
5) Repeat the testing, but use the catastrophic code with encoder $G(x) = [1 + x, 1 + x^2]$. How do the results for the noncatastrophic encoder compare with the results for the catastrophic encoder?

12.11 Exercises

12.1 For the $R = 1/2$ convolutional encoder with

$$G(x) = \begin{bmatrix} 1+x^2+x^3 & 1+x+x^3 \end{bmatrix} \tag{12.49}$$

(a) Draw a hardware realization of the encoder.
(b) Determine the convolutional generator matrix G.
(c) For the input sequence $\mathbf{m} = [1, 0, 1, 1, 0, 1, 1]$ determine the coded output sequence.
(d) Draw the state diagram. Label the branches of the state diagram with input/output values.
(e) Draw the trellis.
(f) What is the constraint length of the code?
(g) Determine the State/Next State table.
(h) Determine the State/Previous State table.
(i) Is this a catastrophic realization? Justify your answer.
(j) Determine the weight enumerator $T(D, N)$.
(k) What is d_{free}?
(l) Determine upper and lower bounds on P_b for a BSC using (12.36) and (12.37) and an approximation using (12.30). Plot as a function of the signal-to-noise ratio, where $p_c = Q(\sqrt{2E_c/N_0})$. Compare the bounds to uncoded performance.
(m) Determine upper and lower bounds on P_b for an AWGN channel using (12.40) and (12.41) and plot as a function of the signal-to-noise ratio.
(n) Determine the theoretical asymptotic coding gain for the BSC and AWGN channels. Compare with the results from the plots. Also, comment on the difference (in dB) between the hard and soft metrics.
(o) Express $G(x)$ as a pair of octal numbers using both leading 0 and trailing 0 conventions.
(p) Suppose the output of a BSC is $\mathbf{r} = [11, 11, 00, 01, 00, 00, 10, 10, 10, 11]$. Draw the trellis for the Viterbi decoder and indicate the maximum likelihood path through the trellis. Determine the maximum likelihood estimate of the transmitted codeword and the message bits. According to this estimate, how many bits of $\mathbf{r}$ are in error?

12.2 For the $R = 1/3$ convolutional coder with

$$G(x) = \begin{bmatrix} 1+x & 1+x^2 & 1+x+x^2 \end{bmatrix}$$

(a) Draw a hardware realization of the encoder.
(b) Determine the convolutional generator matrix G.
(c) For the input sequence $\mathbf{m} = [1, 0, 1, 1, 0, 1, 1]$ determine the coded output sequence.
(d) Draw the state diagram. Label the branches of the state diagram with input/output values.
(e) Draw the trellis.
(f) What is the constraint length of the code?
(g) Determine the State/Next State table.
(h) Determine the State/Previous State table.
(i) Is this a catastrophic realization? Justify your answer.
(j) Determine the weight enumerator $T(D, N)$.
(k) What is d_{free}?
(l) Express $G(x)$ as a triplet of octal numbers.

12.3 Find a catastrophic encoder equivalent to $G(x) = \begin{bmatrix}1+x^2 & 1+x+x^2\end{bmatrix}$ and determine an infinite-weight message $m(x)$ that results in a finite-weight codeword for this catastrophic encoder.

12.4 Show that $G_4(x)$ defined in (12.8) is equivalent to $G_2(x)$ of (12.5).

12.5 Let $G(x)$ be the transfer function matrix of a basic convolutional code. Show that $G(x)$ is equivalent to a basic transfer function matrix $G'(x)$ if and only if $G'(x) = T(x)G(x)$, where $T(x)$ is a unimodular matrix.

12.6 Determine a systematic encoder transfer function matrix equivalent to

$$G(x) = \begin{bmatrix} 1+x & x & 1 \\ x^2 & 1 & 1+x+x^2 \end{bmatrix}.$$

12.7 For the transfer function matrices

$$G(x) = \begin{bmatrix} 1+x & x & 1 \\ x^2 & 1 & 1+x+x^2 \end{bmatrix} \quad \text{and} \quad G'(x) = \begin{bmatrix} 1+x & x & 1 \\ 1+x^2+x^3 & 1+x+x^2+x^3 & 0 \end{bmatrix}$$

(a) Show that $G(x)$ is equivalent to $G'(x)$.
(b) Show that $G(x)$ is a minimal basic encoder matrix.
(c) Show that $G'(x)$ is not a minimal basic encoder matrix.
(d) Using the procedure described in association with (12.13), determine a transfer function matrix $G''(x)$ which is a minimal basic encoding matrix equivalent to $G'(x)$, but different from $G(x)$.

12.8 For the code generated by

$$G(x) = \begin{bmatrix} \frac{1}{1+x+x^2} & \frac{x}{1+x^3} & \frac{1}{1+x^3} \\ \frac{x^2}{1+x^3} & \frac{1}{1+x^3} & \frac{1}{1+x^3} \end{bmatrix},$$

use elementary row operations to convert the generator matrix to systematic form. Draw a circuit realization of the systematic encoder.

12.9 Catastrophic codes.

(a) For a rate $R = 1/2$ code, let $g_1(x) = 1+x$, $g_2(x) = x^2+1$. Show that when $m(x) = \frac{1}{1+x}$ that the transmitted sequence has finite weight. Determine GCD($g_1(x), g_2(x)$).
(b) Motivated by this result, prove the following: For a rate $1/n$ code, if

$$\text{GCD}[g_1(x), g_2(x), \ldots, g_n(x)] = 1,$$

then the code is noncatastrophic.

12.10 For the catastrophic code with generators $g_1(x) = 1 + x$, $g_2(x) = 1 + x^2$:

(a) Draw the state diagram.
(b) Determine the weight enumerator $T(D, N)$ for the code.
(c) What is the minimum free distance of the code?
(d) How is the catastrophic nature of the code evidenced in the weight enumerator?

12.11 For the generator $G(x) = \left[1 + x^2, 1 + x + x^2 + x^3\right]$:

(a) Find the GCD of the generator polynomials.
(b) Find an infinite-weight message sequence that generates a codeword of finite weight.

12.12 Prove that d_{free} is independent of the encoder realization, so that it is a property of the *code* and not of a particular *encoder* for the code.

12.13 Show that the formal series expansion

$$\frac{1}{1-x} = 1 + x + x^2 + x^3 + \cdots$$

is correct. Show the formal series expansions of

$$\frac{1}{1-2D} \quad \text{and} \quad \frac{D^5L^3N}{1-DLN(1+L)}.$$

12.14 Show that the expressions for P_d in (12.19) and (12.20) can be bounded by $P_d < [4p_c(1-p_c)]^{d/2}$. *Hint*: Show that $\sum_{i=(d+1)/2}^{d} \binom{d}{i} p_c^i (1-p_c)^{d-i} < \sum_{i=(d+1)/2}^{d} \binom{d}{i} p_c^{d/2} (1-p_c)^{d/2}$.

12.15 For a BSC where $Z = \sqrt{4p_c(1-p_c)}$, show that (12.27) can be replaced by $P_d < Z^{d+1}$ when d is odd. Using this result, show that (12.28) can be replaced by

$$P_e(j) < \frac{1}{2}[(1+Z)T(Z) + (1-Z)T(-Z)].$$

12.16 [147] An upper bound on d_{free}. Let K be the number of outputs determining the output of a rate $R = 1/n$ code (i.e., K is the constraint length). The code can be represented by a matrix such as that in (12.3), in which all rows are obtained by shifting the first row.

(a) Show that for any binary linear code, if the codewords are arranged as the rows of a matrix, then any column is either all zeros or half zeros and half ones.
(b) Consider the set of all sequences of length no greater than L. Show that the code generated by these finite-length sequences has length $(K - 1 + L)n$ symbols. Also show that the average weight of all codewords (excluding the all-zero codeword) is $w_{\text{av}}(L) \leq 2^{L-1}(K - 1 + L)n/(2^L - 1)$.
(c) Argue that the code has a minimum distance between paths of $d_{\text{free}} \leq w_{\text{av}}(L)$.

12.17 Show that (12.21) and (12.22) are correct.

12.18 A code with $k = 1$ has weight enumerator $T(D, N) = \frac{D^5N}{1-2ND}$. The codewords are passed through a BSC with $p_c = 0.01$. Compute upper and lower bounds on the node error probability and the bit-error rate for Viterbi decoding. Repeat this when the code is passed through an AWGN channel with $E_b/N_0 = 6$ dB.

12.19 For a rate $R = 1/2$ code, suppose the output sequence

$$\mathbf{c} = (c_0^{(1)}, c_0^{(2)}, c_1^{(1)}, c_1^{(2)}, c_2^{(1)}, c_2^{(2)}, c_3^{(1)}, c_3^{(2)}, c_4^{(1)}, c_4^{(2)}, c_5^{(1)}, c_5^{(2)}, \ldots)$$

is punctured as

$$\mathbf{c} = (c_0^{(1)}, c_0^{(2)}, -, c_1^{(2)}, c_2^{(1)}, -, c_3^{(1)}, c_3^{(2)}, -, c_4^{(2)}, c_5^{(1)}, -, \ldots).$$

Write down the puncture matrix P.

12.20 A binary-input/binary-output channel with input a and output r has transition probabilities

$$P(r=0|a=0)=0.9 \qquad P(r=0|a=1)=0.3$$
$$P(r=1|a=0)=0.1 \qquad P(r=1|a=1)=0.7.$$

(a) Determine the log likelihoods.

(b) Scale and shift these values to obtain a set of bit metrics that can be reasonably approximated with not more than 3 bits.

12.21 A channel has binary inputs and three outputs, 0 and 1, and E, where E denotes an erasure. When an erasure occurs, the symbol is known to be suspicious and does not influence the decoding process — it is erased. (It is a lot like a punctured bit). This channel is called the *binary erasure channel*. The channel has transition probabilities

$$P(r=0|a=0)=0.6 \qquad P(r=E|a=0)=0.3 \qquad P(r=1|a=0)=0.1$$
$$P(r=0|a=1)=0.2 \qquad P(r=E|a=1)=0.2 \qquad P(r=1|a=1)=0.6$$

(a) Determine the log likelihood ratios.

(b) Scale and shift these values to obtain a set of bit metrics that can be reasonably approximated with not more than 3 bits, making sure that erased symbols do not contribute differentially to the path metric.

12.22 An AWGN with variance $\sigma^2 = 2$ is used with BPSK-modulated data sending signals with amplitudes $a = \pm 1$. The received signal r_t is quantized to four different values $q = Q[r]$ with quantization thresholds at ± 1.5 and 0.

(a) Determine the probabilities $P(q_t|a)$ and the log probabilities $-\log P(q_t|a)$.

(b) Determine a and b so that $a(-log P(q_t|a) - b)$ can be approximated well by integers using at most two bits.

12.23 A binary input/binary output channel with input a and output r has

$$P(r=0|a=0)=0.99999 \qquad P(r=0|a=1)=0.05$$
$$P(r=1|a=0)=0.00001 \qquad P(r=1|a=1)=0.95.$$

(a) Determine Z in the Chernoff bound from (12.33).

(b) The input to this channel is coded using a convolutional code whose path enumerator is given by

$$T(D,N) = \frac{D^5 N}{1-2ND}.$$

Using (12.28), determine an upper bound on the node error probability $P_e(j)$. Using (12.29) and (12.30), determine an upper bound and an approximation on the bit error rate P_b.

12.24 Chernoff bound. Let $X_1, X_2, \ldots, X_n$ be independent random variables with densities $p_i(x)$ and moment generating functions $\phi_i(s) = E[e^{sX_i}]$. Let $Z = \sum_{i=1}^n X_i$, with moment generating function $\phi_Z(s)$. Using the following steps, show that

$$P(Z \geq \gamma) \leq e^{-s\gamma} \prod_{i=1}^{n} \phi_i(s).$$

for all $s \geq 0$ such that $\phi_i(s)$ exists.

(a) Let $\phi_Z(s)$ be the moment generating function for Z. Show that $\phi_Z(s) = \prod_{i=1}^n \phi_i(s)$.

(b) Show that $\int_{-\infty}^{\infty} e^{sZ} f_Z(z)\,dz \geq \int_{\gamma}^{\infty} e^{sZ} f_Z(z)\,dz$.

(c) Finish the proof.

12.25 Show that (12.39) is correct.

12.26 A rate-compatible punctured convolutional code (RCPC) based on a rate $R = 1/4$ convolutional code has puncturing period 8 and puncturing matrices

$$P_1 = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \quad P_2 = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$P_3 = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

(a) Determine the actual rate when using the puncture matrix P_1. Also for P_2 and P_3.

(b) The generators for the convolutional code are $g^{(1)}(x) = 1 + x^3 + x^4$, $g^{(2)}(x) = 1 + x + x^2 + x^4$, $g^{(3)}(x) = 1 + x^2 + x^3 + x^4$, and $g^{(4)}(x) = 1 + x + x^3 + x^4$. Draw a convolutional encoder capable of transmitting at these three different rates.

12.27 Determine the branch Fano metric for binary transmission of a rate $R = 1/3$ code through a BSC with $p_c = 0.1$. Then scale the metric so it has nearly integer values. Repeat for $p_c = 0.05$ and $p_c = 0.001$.

12.28 For the asymmetric channel in Exercise 12.20, determine the Fano metric for a rate $R = 1/2$ code. Then shift and scale the metric so it has nearly integer values.

12.29 For the code with generator

$$G(x) = \begin{bmatrix} 1 + x^2 + x^3 & 1 + x + x^3 \end{bmatrix},$$

the sequence $\mathbf{r} = [11, 11, 11, 01, 11, 00, 00]$ is received through a BSC with $p_c = 0.125$. Using the stack algorithm, determine the transmitted sequence. Repeat using the Fano algorithm. (The Matlab code may prove very helpful.)

12.30 Draw a trellis representing the binary block code with parity check matrix

$$H = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 \end{bmatrix}.$$

12.31 Draw a trellis representation for the cyclic code with generator $g(x) = x^3 + x^2 + 1$ that employs a systematic encoder.

12.12 References

Convolutional codes were introduced in 1955 by Elias [76]. Our presentation overall has benefited greatly from [303]. The discussion of structural properties comes from that source, which, in turn, closely follows [175]. This, in turn, builds on the landmark paper on the algebraic structure of convolutional codes [97]. Catastrophic codes were first discussed in [226]. The criterion for catastrophic codes in terms of the GCD of the generators appears in [226]. Extensive simulation studies of convolutional codes and error curves appears in [148]. The results here were computed by Ojas Chauhan.

The Viterbi algorithm was described in [358]. However, it was not until [89] that the Viterbi algorithm was shown to be a maximum likelihood sequence estimator. This paper also presented the weight enumerator and the graph analysis associated with the performance of convolutional codes.

An important and still very relevant source on convolutional codes is [357]. This book presents random coding performance bounds for convolutional codes and shows that convolutional codes have a higher cutoff rate than block codes. A recent book dedicated to convolutional codes is [174]. Convolutional codes are also presented in most books on coding theory and digital communication theory.

Puncturing appears to have been first explored in [40]. Tail biting was introduced in [213]. Work on short tailbiting codes with many examples of good codes appears in [320]. Basic results regarding the structure of tail-biting trellises appears in [189]. The trellis representation of a block code was presented first in [11] and developed more fully in [377]. It has been the topic of a detailed monograph [205]. Readers interested in fully developed design methodologies should consult that source. A summary of this work is in [303].

The stack algorithm was explored in [388] and [165]. The genre of sequential decoding algorithms was explored early on in [378]. The Fano algorithm appeared first in [80]. The Fano metric received theoretical foundation as a maximum likelihood metric in [223]. A comparison of sequential decoding algorithms appears in [115, 116]. A discussion of the performance of the M algorithm as a function of M and comparison with the Viterbi algorithm is summarized in [303].

Chapter 13

Trellis Coded Modulation

13.1 Adding Redundancy by Adding Signals

The error correction codes studied up to this point in the book have added redundancy by increasing the number of coded symbols. If the channel is bandlimited so that the transmitted symbol rate is fixed, this results in a lower information transmission rate. In the very common case that the high transmission rate is of interest (in contrast to minimizing transmission power), this reduction in effective information rate is unfortunate. Up until the early 1970s it was believed that coding would not greatly benefit channels needing a spectral efficiency — the number of bits transmitted per channel use — exceeding 1.

In 1976, a new method of coding was introduced by Ungerboeck [344, 346, 347, 345] which adds redundancy to the coded signal by increasing the number of symbols in the signal constellation employed in the modulation. If the average signal energy is fixed, having more signals in the signal constellation would tend to decrease the distance between points in the signal constellation. The key, therefore, is to combine the coding and modulation into a single unit which transmits only constrained *sequences* of symbols, and to employ a sequence detector (i.e., Viterbi algorithm) to detect the sequence. The combination of constrained symbol sequences and larger signal constellation gives rise to what is known as *trellis coded modulation*, or TCM.

13.2 Background on Signal Constellations

Because TCM is built upon signal constellations, we briefly review concepts related to signal constellations. For now, we restrict our attention to one- and two-dimensional signal constellations. (A review of the communications concepts in Section 1.4 may prove helpful.)

A *signal constellation* $\mathcal{S}$ is a discrete set of points, typically a subset of the real line $\mathbb{R}$ or the plane $\mathbb{R}^2$ (sometimes regarded as the complex plane). A one-dimensional constellation is used in what is often called *amplitude shift keying* (ASK). A one-dimensional constellation with two points $\pm\sqrt{E_b}$ is more frequently called BPSK (binary phase-shift keying). A two-dimensional constellation with all the points lying on a circle is referred as phase-shift keying (PSK). The constellation is frequently expressed in terms of the number of points, such as 4-PSK, 8-PSK, or 16-PSK. QPSK — quaternary PSK — is a synonym for 4-PSK. Figure 13.1 shows examples of PSK constellations, scaled so that they all have the same signal energy E_s. The minimum distance between signal points is denoted as d_0. A two-dimensional constellation with points on a square grid is frequently referred to as quadrature-amplitude modulation (QAM). Figure 13.2 shows overlays of examples of QAM constellations, where the minimum distance between points is called d_0. (The 32-point and 128-point constellations are referred to as cross constellations, since the points are arranged in a cross; this reduces the average energy compared to rectangular constellations.) Other

Table 13.1: Average Energy Requirements for Some QAM Constellations

Constellation	Spectral Efficiency η (bits/symbol)	E_s	E_b
BPSK	1	$\frac{1}{4}d_0^2$	$\frac{1}{4}d_0^2$
QPSK	2	$\frac{1}{2}d_0^2$	$\frac{1}{4}d_0^2$
16-QAM	4	$\frac{5}{2}d_0^2$	$\frac{5}{8}d_0^2$
32-cross	5	$5d_0^2$	d_0^2
64-QAM	6	$\frac{21}{2}d_0^2$	$\frac{7}{4}d_0^2$
128-cross	7	$\frac{41}{2}d_0^2$	$\frac{41}{14}d_0^2$
256-QAM	8	$\frac{85}{2}d_0^2$	$\frac{85}{16}d_0^2$

arrangements are also possible in two dimensions.

The *spectral efficiency* η of a constellation is the number of bits carried by each symbol. Assuming the bits are identically distributed, the average *symbol energy* E_s is the average of the squared distances of the constellation points from the origin. For example, for the 16-QAM constellation with minimum distance d_0,

$$\begin{aligned} E_s &= \frac{1}{16}\left(4((d_0/2)^2 + (d_0/2)^2) + 8((d_0/2)^2 + (3d_0/2)^2) + 4((3d_0/2)^2 + (3d_0/2)^2)\right) \\ &= \frac{5d_0^2}{2}. \end{aligned}$$

Table 13.1 lists average energies and spectral efficiencies for various QAM constellations. Also shown is the average energy per bit, where $E_b = \frac{1}{\eta}E_s$. It may be computed that for a *square* constellation with M points,

$$E_s = \frac{M-1}{6}d_0^2. \tag{13.1}$$

The elements of a point $(a_1, a_2) \in \mathcal{S} \subset \mathbb{R}^2$ represent the amplitudes of two basis functions, which we denote as $\varphi_1(t)$ and $\varphi_2(t)$, which are assumed to be orthonormal (unit energy and orthogonal)

$$\int_{-\infty}^{\infty} \varphi_i^2(t)\,dt = 1 \qquad \int_{-\infty}^{\infty} \varphi_1(t)\varphi_2(t)\,dt = 0.$$

Furthermore, shifts of the functions by the *symbol period* T are orthogonal,

$$\int_{-\infty}^{\infty} \varphi_i(t)\varphi_i(t-kT)\,dt = 0 \qquad i = 1, 2, \text{for all integer } k \neq 0.$$

The time T is called the *symbol time*, or sometimes the *baud interval*. The number of symbols transmitted per second, $1/T$ is called the *symbol rate* or the *baud rate*.

The transmitted signal $s(t)$ is obtained by juxtaposing a sequence of scaled basis signals, each with their own amplitude representing the transmitted symbol

$$s(t) = \sum_k a_{1,k}\varphi_1(t-kT) + a_{2,k}\varphi_2(t-kT).$$

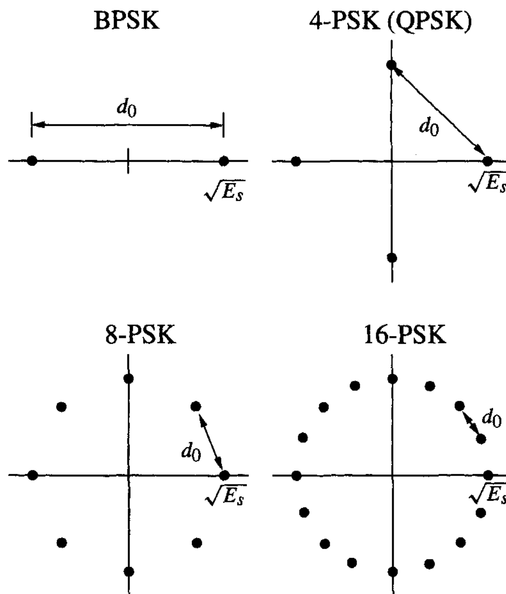


Figure 13.1: PSK signal constellations.

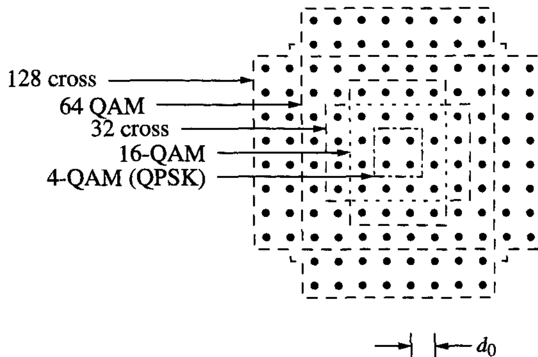
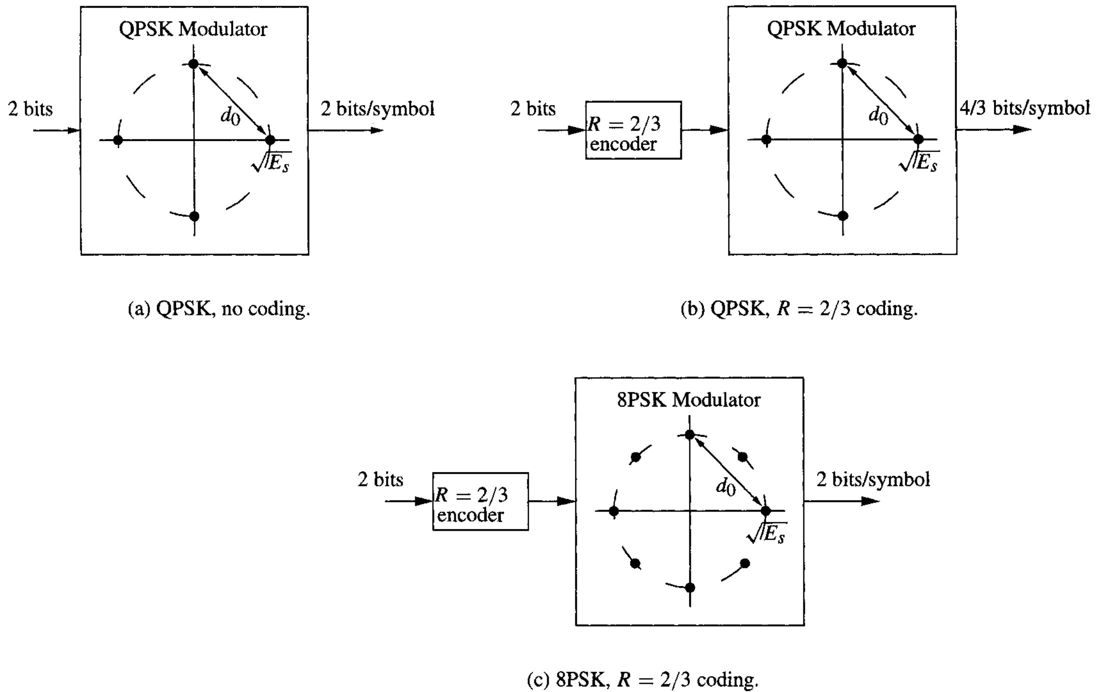


Figure 13.2: QAM Signal constellations (overlaid).

At the receiver the received signal $r(t)$ is again projected back onto the signal constellation plane by matched filtering. Over each symbol interval, a point (r_1, r_2) is received, then the maximum likelihood (ML) detector without coding determines the constellation point nearest to (r_1, r_2).

13.3 TCM Example

With this background on signal constellations, consider the following three scenarios. In the first case, Figure 13.3(a), 2 bits select a single signal point in a QPSK constellation, resulting in $\eta = 2$ bits of information per transmitted symbol. In the second case, $R = 2/3$ coding is used with the same QPSK constellation. The efficiency is reduced to $\eta = 4/3$ bits



(a) QPSK, no coding.

(b) QPSK, $R = 2/3$ coding.

(c) 8PSK, $R = 2/3$ coding.

Figure 13.3: Three communication scenarios.

of information per transmitted symbol. In the third case, 8-PSK modulation is employed on the coded bits, and again there are $\eta = 2$ bits of information per transmitted symbol. Thus the larger signal constellation is able to attain the uncoded data rate. However, if the average signal energy E_s is the same for both the QPSK and 8-PSK, the symbol points are closer in the 8-PSK constellation: approximately 4 dB additional signal energy would be required to make the minimum distance between 8-PSK points the same as the QPSK points. The problem of closer points can be overcome by *combining* the coding and the modulation.

Consider coded modulation with the 8-PSK signal constellation with points labeled as shown in Figure 13.4 [346]. (The rationale for the labeling by this partitioning mechanism is discussed below.) The points on the signal constellation correspond to elements of the sets labeled D_i. The signal point i as a binary number corresponds to the "set" of points D_i, with the least significant bit of i on the right. The minimum distance d_j at the jth partition between points in the constellation increases with j. The constellation is used with the rate $R = 2/3$ binary convolutional code, with the trellis as shown in Figure 13.5. The outputs of the convolutional coder are mapped to points in the signal constellation, resulting in a single 8-PSK symbol transmitted for each pair of input bits. We regard the convolutional encoder simply as a finite-state machine with a given number of states and specified state transitions, used to select points or subsets of the signal constellation. The combination of the convolutional coding followed by the mapping is indicated by the labeling of the trellis, with the sequence of outputs D_i corresponding to the sequence of branches read from top to bottom. Thus, for example, if the coder starts in the first state and the top branch is taken,

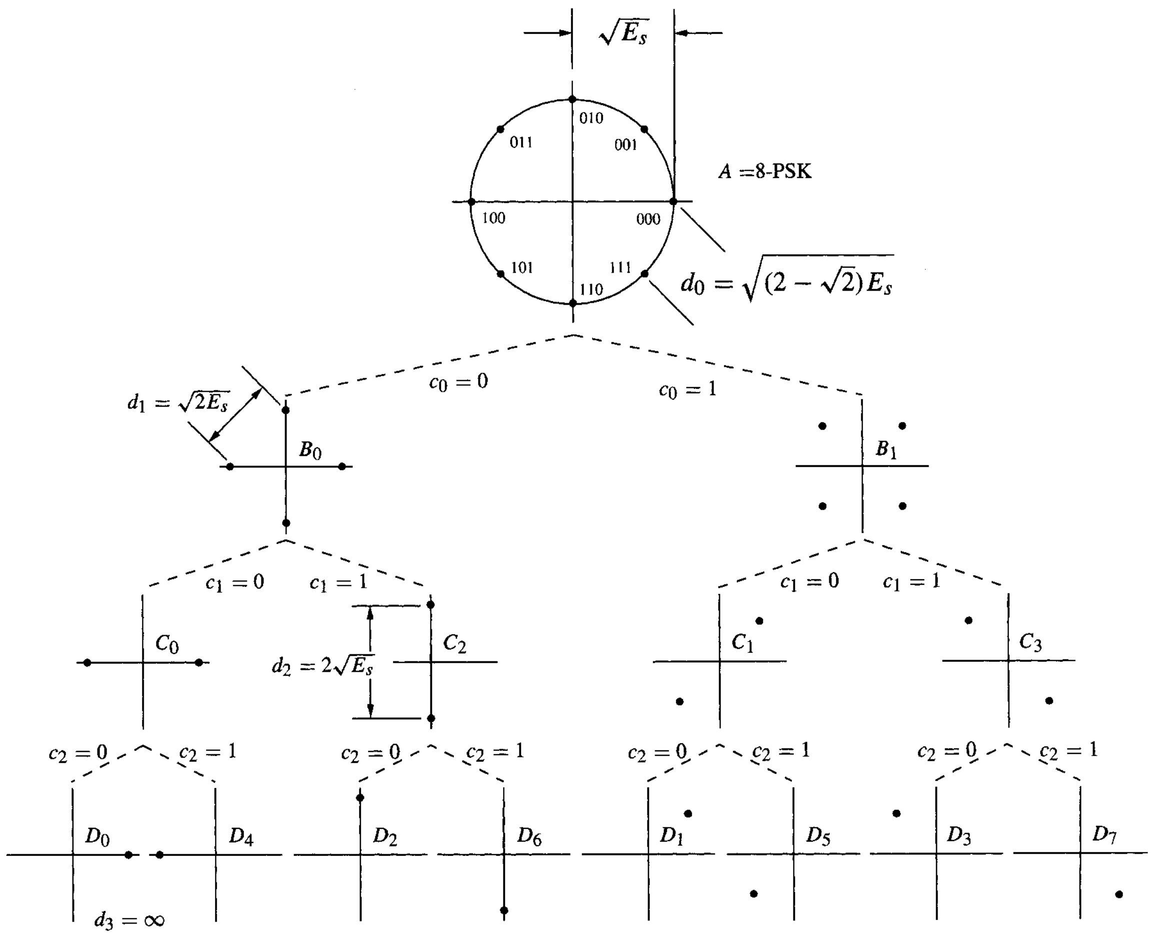


Figure 13.4: Set partitioning of an 8-PSK signal.

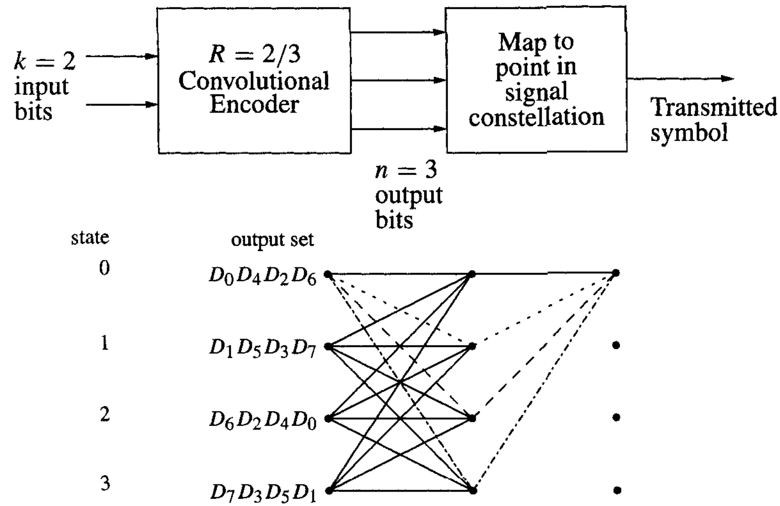


Figure 13.5: $R = 2/3$ trellis coded modulation example.

the point in the set D_0 is transmitted; if the second branch is taken from the first state, the point in the set D_4 is transmitted, and so forth. The trellis structure imposes constraints on the *sequences* of symbols that can be transmitted. For example, starting from state 0, it is impossible to transmit the sequence (D_4, D_2). Thus when determining the performance of the system, distances between *sequences* of symbols much be considered, rather than distances between individual points in the signal constellation.

The optimal decoding algorithm (Viterbi) finds a shortest path through the trellis, that is, a sequence of symbols in the trellis which is closest to the sequence observed at the receiver. Assuming that the channel is AWGN, the branch metric is related to the squared *Euclidean distance* between received signal points and transmitted signal points along a branch. As in the case of convolutional codes, the overall performance of the system is dominated by the shortest distance between two paths which diverge the come back together — errors which lead to the path metric exceeding half of this distance and result in selecting the incorrect path in the Viterbi algorithm.

Accordingly, let us find the shortest distance between two paths which diverge then remerge. One candidate to consider is the distance between the transmitted symbols (D_0, D_0) and the symbols along the path (D_4, D_1). The sequence is indicated in Figure 13.5 with a dotted line, remerging after two branches. The squared distance between the sequences is the sum of the squares of the distances, which can be determined with the help of the diagram in Figure 13.4:

$$\begin{aligned} d^2((D_0, D_0), (D_4, D_1)) &= d^2(D_0, D_4) + d^2(D_0, D_1) = d_2^2 + d_0^2 \\ &= 4E_s + (2 - \sqrt{2})E_s = (6 - \sqrt{2})E_s. \end{aligned}$$

A second path to consider is represented by the sequence (D_2, D_6), shown with dashed lines, with

$$\begin{aligned} d^2((D_0, D_0), (D_2, D_6)) &= d^2(D_0, D_2) + D^2(D_0, D_6) = d_1^2 + d_1^2 \\ &= 2E_s + 2E_s = 4E_s. \end{aligned}$$

A third path (dash-dot line) represented by (D_6, D_1) has

$$d^2((D_0, D_0), (D_6, D_1)) = d^2(D_0, D_6) + d^2(D_0, D_1) = d_1^2 + d_0^2$$
$$= 2E_s + (2 - \sqrt{2})E_s = (4 - \sqrt{2})E_s.$$

This is the minimum distance path between any sequences in the trellis.

The minimum distance between any sequences in the trellis for a code is called the *free Euclidean distance*. It is usually denoted as d_{free}. Thus for this code

$$d_{\text{free}}^2 = (4 - \sqrt{2})E_s.$$

How does the performance of this coded scheme compare with uncoded 4-PSK that transmits information at the same rate? The quantity

$$\gamma = \left(\frac{E_{s,\text{uncoded}}}{E_{s,\text{coded}}}\right)\left(\frac{d_{\text{free,coded}}^2}{d_{\text{free,uncoded}}^2}\right) = \gamma_C \gamma_D$$

is called the **(asymptotic) coding gain** for the code. Here, $d_{\text{free,uncoded}}$ is the minimum distance between points in the original signal constellation and $d_{\text{free,coded}}$ is the free Euclidean distance between nearest sequences of the coded signal. The factor $\gamma_C = E_{s,\text{uncoded}}/E_{s,\text{coded}}$ is called the **constellation expansion factor**; it accounts for the average energy of the constellations — larger average energy in the coded constellation reduces the coding gain. The factor $\gamma_D = d_{\text{free,coded}}^2/d_{\text{free,uncoded}}^2$ is called the **increased distance factor**. In our case, the constellation expansion factor is 1 (the PSK constellations require the same energy per symbol) and we find the coding gain is

$$\gamma = \frac{(4 - \sqrt{2})E_s}{2E_s} = 1.29.$$

This is frequently expressed in dB, $\gamma_{\text{dB}} = 10\log_{10}(\gamma) = 10\log_{10}(1.29) = 1.1$ dB. Asymptotically (for high SNR), the coded 8-PSK scheme requires 1.1 dB less energy for (essentially) the same performance as the uncoded QPSK scheme.

There are other four-state convolutional coding schemes than can provide better coding performance. Consider the coding scheme shown in Figure 13.6. In this figure, there are two input bits. However, only one of them goes into the convolutional encoder, which is a rate $R = 1/2$ encoder. The two coded output bits are used to select one of four *sets* of constellation points, which are the sets denoted C_0, C_1, C_2, and C_3 in Figure 13.4. Each set has two symbols. The other input bit is used to select one of the two points within a selected set. The result is that the pair of input bits can be used to select a single output symbol. The behavior of the convolutional code and the signal mapper is shown by the trellis of Figure 13.6. The sets selected by the output bits are listed to the left of the trellis. For example, from state 0, the output sets C_0 and C_2 can be selected, depending on which branch of the trellis is taken. The fact that C_0 actually consists of two points is shown as *parallel* paths in the trellis between state 0 and state 0. One of the paths corresponds to the point $D_0 \in C_0$; the parallel path corresponds to the point $D_4 \in C_0$. The parallel paths corresponding to C_2 are similarly labeled, and the other (unlabeled) parallel paths of the trellis have their corresponding symbol point assignments.

What is the minimum distance between diverging/remerging paths for this code? Let us consider the path (C_0, C_0, C_0) and the path (C_2, C_1, C_2), starting from state 0, indicated

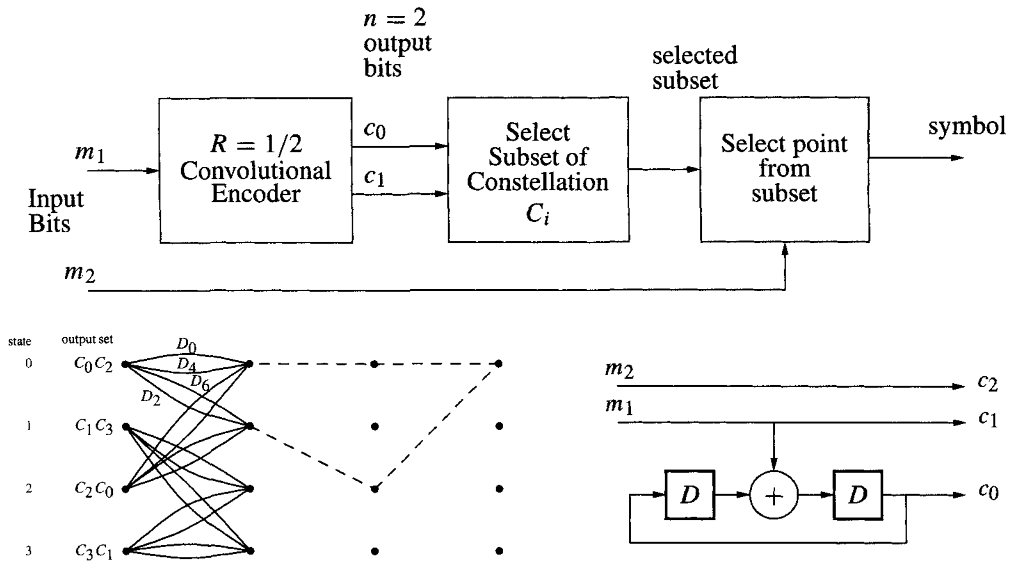


Figure 13.6: A TCM encoder employing subset selection and a four-state trellis.

with dashed lines in Figure 13.6. In comparing distances between sets, the distance between the *nearest* points in the sets must be used. The squared distance is

$$\begin{aligned} d^2((C_0, C_0, C_0), (C_2, C_1, C_2)) &= d^2(C_0, C_2) + d^2(C_0, C_1) + d^2(C_0, C_2) \\ &= d_1^2 + d_0^2 + d_1^2 = (6 - \sqrt{2})E_s. \end{aligned}$$

Is this the smallest distance between diverging/remerging paths? There is, in fact, another way that paths can diverge and remerge — through the parallel paths. Consider the distance between the path C_0 and C_0, where in one case the symbol D_0 is sent, and in the other case the symbol D_4 is sent. Then the distance is

$$d^2(D_0, D_4) = d_2^2 = 4E_s,$$

which is smaller than the last distance found and is, in fact, the smallest distance between diverging/remerging sequences.

The coding gain for this code compared to uncoded QPSK (transmitting information at the same rate) is

$$\gamma = \frac{4E_s}{2E_s} = 2,$$

a coding gain of 3 dB. It can be verified that this is the best possible coding gain for a TCM code having four states.

Let us now consider a convolutional encoder with 8 states, with trellis and encoder as shown in Figure 13.7. The coder selects single subsets (the D_i). The minimum squared distance in this case is

$$d^2((D_0, D_0, D_0), (D_6, D_7, D_6)) = d_1^2 + d_0^2 + d_1^2 = 4.585E_s.$$

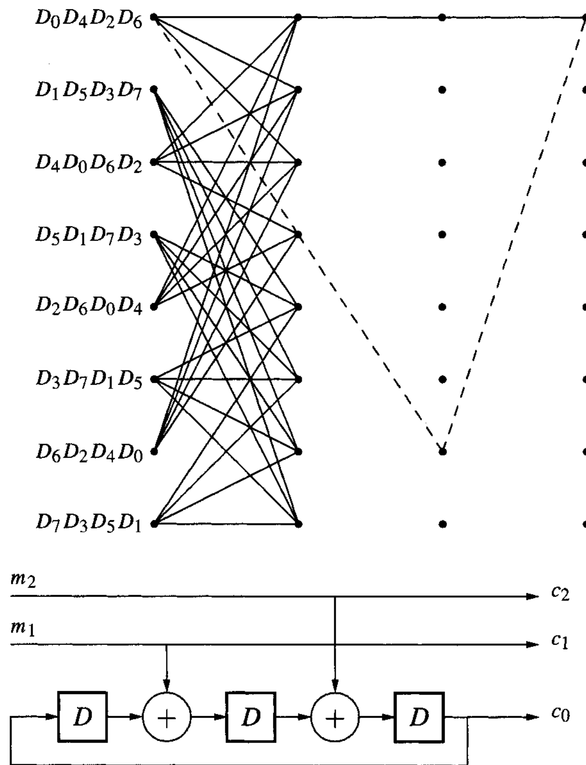


Figure 13.7: An 8-state trellis for 8-PSK TCM.

The coding gain relative to uncoded QPSK is

$$\gamma = \frac{4.585E_s}{2E_s} = 2.29 \qquad \gamma_{\text{dB}} = 10\log_{10}\gamma = 3.6\text{ dB}.$$

From these examples, we may make the following observations.

- TCM relies on signal space enlargement to compensate for coding redundancy, resulting in equivalent data rates for coded data.
- The trellis-coded modulation concept combines convolutional coding with the signal mapping (modulation). Rather than optimizing the coding and modulation separately, TCM code design seeks a jointly optimum solution for coding and modulation.
- The finite-state machine structure imposed by the underlying convolutional code imposes constraints between sequences of symbols. The performance depends upon distances between *sequences* of symbols. By proper design, the reduced distance between symbols in the enlarged signal constellation or the additional average energy in the enlarged constellation can be more than compensated for by effective distance between sequences, resulting in net coding gain.

  While these examples have used convolutional codes, actually any finite state machine, even a nonlinear state machine, could be used to impose constraints on the sequences of allowed symbols.

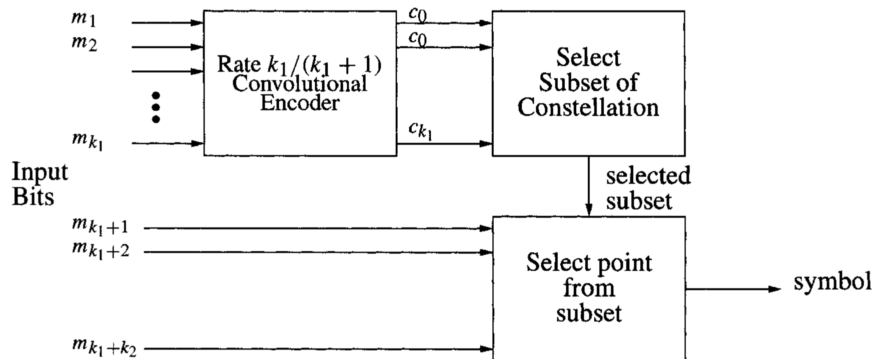


Figure 13.8: Block diagram of a TCM encoder.

- The (asymptotic) performance depends upon the minimum distance between diverging/remerging paths, where sums of squared Euclidean distances are used (in AWGN). This minimum distance is referred to as the free Euclidean distance of the code. The (asymptotic) coding gain in dB is computed as

$$\gamma_{\text{dB}} = 10\log_{10}\left(\frac{E_{s,\text{uncoded}}}{E_{s,\text{coded}}}\right)\left(\frac{d^2_{\text{free,coded}}}{d^2_{\text{free,uncoded}}}\right) = \gamma_{C,\text{dB}} + \gamma_{D,\text{dB}}.$$

- The encoding architecture includes (in general) two stages. The first stage selects *sets* of points, based on the convolutional coder output. The second stage selects a single point for transmission from the set.
- The sets which are used in the TCM code can be obtained from a "set partitioning" process.
- The subset selection may give rise to parallel paths in the trellis; the number of parallel paths is the number of points in the set.
- Finding minimum distance requires consideration of distances between parallel paths, as well as other diverging/remerging paths through the trellis.
- As the number of states in the coder increases, increased coding gain is possible.

13.3.1 The General Ungerboeck Coding Framework

The general trellis coded modulation idea is shown in Figure 13.8. We take $k = k_1 + k_2$ message bits as inputs. The first k_1 bits go into a rate $R = k_1/(k_1+1)$ convolutional encoder. The $k_1 + 1$ coded bits then select a subset of points. The remaining k_2 bits select a point from within the subset. The constellation must therefore have $2^{k_1+k_2+1}$ points in it.

In the first example above, $k_1 = 2$ and $k_2 = 0$ and the coder had four states. That is, we simply employ a rate 2/3 encoder, then use the output to select signal points. In the second example, $k_1 = 1$, $k_2 = 1$ and the encoder had four states. In the third example, $k_1 = 2$, $k_2 = 0$ and the encoder had eight states.

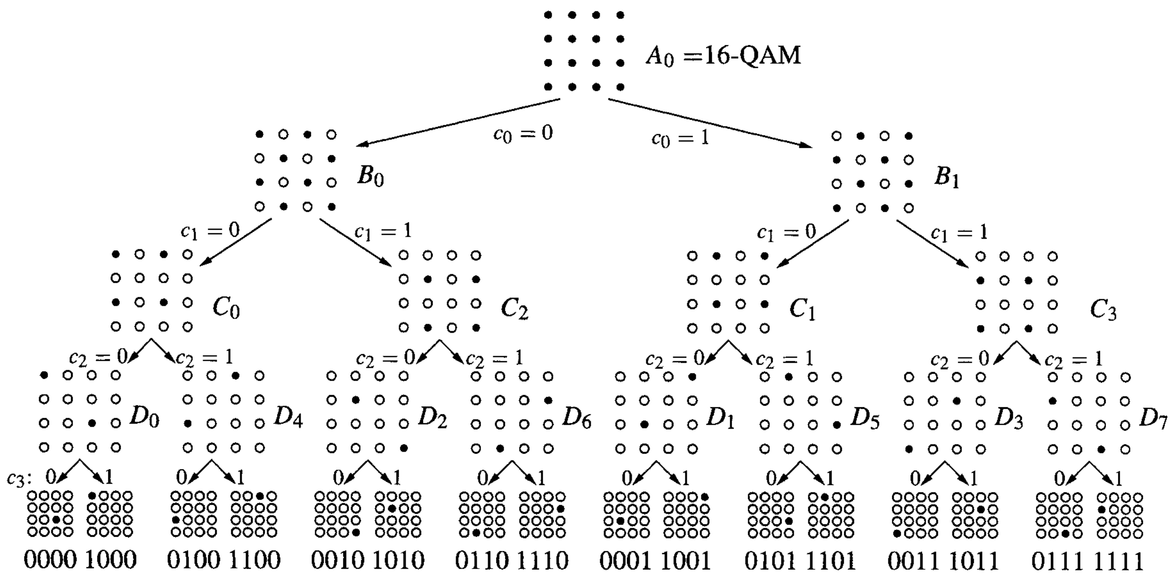


Figure 13.9: Set partitioning on a 16-QAM constellation.

13.3.2 The Set Partitioning Idea

The problem now is how to determine the subsets of the signal constellation. An effective answer was developed by Ungerboeck, using what he called *set partitioning*. We recursively divide a constellation into subsets with *increasing* intraset distance. The Ungerboeck set partitioning rules [347] are summarized as follows:

- Signals in the lowest partition of the partition tree are assigned parallel transitions.

  This rule maximizes the distance between symbols assigned to parallel transitions in the trellis.

- State transitions that begin and end in the same state should be assigned subsets separated by the largest Euclidean distance.

  This ensures that the total distance is at least the sum of the minimum distances between signals in these subsets.

  For example, in the 8-PSK example, the 8-PSK constellation was partitioned into two 4-PSK constellations (the sets B_0 and B_1 in Figure 13.4).

- The signal points should be used equally often.

Furthermore, the partitioned constellation should produce subsets that have a higher minimum distance than the sets above it.

Figure 13.9 provides an example of set partitioning for a 16-QAM signal constellation. Figure 13.10 shows a partition for an amplitude shift-keyed system, 8-ASK, a one-dimensional constellation.

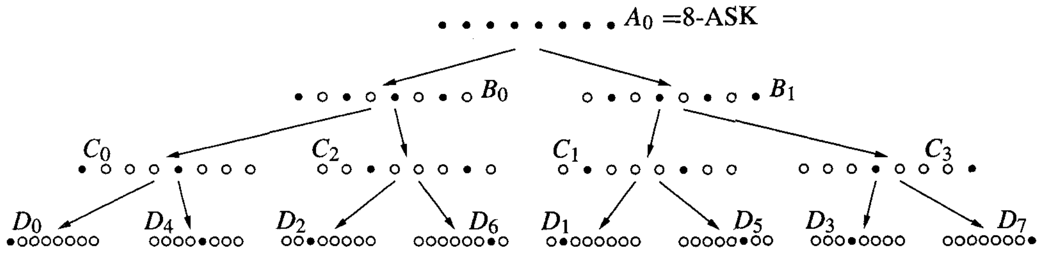


Figure 13.10: Partition for 8-ASK signaling.

13.4 Some Error Analysis for TCM Codes

13.4.1 General Considerations

The probability of error analysis for TCM codes starts out very similar to that of convolutional codes: We employ the union bound to express the probability of node error and bit error rate in terms of binary error probabilities, then develop expressions for those error probabilities. The problem is complicated, however, by the fact that TCM codes are not, in general, linear, even when the underlying state machine is a linear convolutional coder. Additional effort to bound the probability of error is therefore needed.

Denote the "correct" path through the trellis by $\mathbf{c}$. Let $\mathcal{P}_j$ denote the set of all paths that diverge from node j in the trellis and let $\mathbf{p}_{i,j} \in \mathcal{P}_j$ be an incorrect path that diverges at node j, then remerges. Let $e_{i,j}$ be the event that $\mathbf{p}_{i,j}$ is chosen by the decoding (Viterbi) algorithm. The probability of a node error (i.e., the Viterbi algorithm chooses an incorrect path) at any node when $\mathbf{c}$ is the correct path is

$$P_{|\mathbf{c}} = \Pr\left(\bigcup_j \bigcup_i e_{i,j}|\mathbf{c}\right).$$

The average probability of error $\overline{P}_e$ is obtained by averaging over all correct paths,

$$\overline{P}_e = \sum_{\mathbf{c}} P(\mathbf{c}) P_{|\mathbf{c}} = \sum_{\mathbf{c}} P(\mathbf{c}) \Pr\left(\bigcup_j \bigcup_i e_{i,j}|\mathbf{c}\right),$$

where $P(\mathbf{c})$ is the probability of the path $\mathbf{c}$. Since the paths are not disjoint, the probability is difficult to compute, so the union bound is employed to obtain a somewhat simpler expression,

$$\overline{P}_e \leq \sum_{\mathbf{c}} P(\mathbf{c}) \sum_j \Pr\left(\bigcup_i e_{i,j}|\mathbf{c}\right). \tag{13.2}$$

If the length l of the encoded sequence is very long, then it is probable that a node error eventually occurs. In fact, $\overline{P}_e \to 1$ as $l \to \infty$. A more interesting measure is the rate at which node errors occur. We denote

$$\overline{P} = \lim_{l \to \infty} \frac{1}{l} \overline{P}_e.$$

Averaged over an infinite trellis, every node has the same characteristics, so the dependence on an individual node j can be removed to write

$$\overline{P} \leq \sum_{\mathbf{c}} P(\mathbf{c}) \Pr\left(\bigcup_i e_i | \mathbf{c}\right),$$

where e_i is the event that an error event starts at an arbitrary time unit.

We now employ the union bound again to write

$$\overline{P} \leq \sum_{\mathbf{c}} P(\mathbf{c}) \sum_{e_i} \Pr(e_i | \mathbf{c}).$$

The probability $\Pr(e_i|\mathbf{c})$ is the probability of the error event e_i when $\mathbf{c}$ is sent. This is the probability of error for a *binary detection* problem. We denote this probability as $P_{\mathbf{c} \to e_i}$.

Now let d_{ci} denote the distance (metric) between the correct path $\mathbf{c}$ and the incorrect path corresponding to the error event e_i. The probability of the error event $P_{\mathbf{c} \to e_i}$ is a function of the distance d_{ci} between the correct path $\mathbf{c}$ and the error path e_i. We write the functional dependence in general as $P_{\mathbf{c} \to e_i} = P_{d_{ci}}$. The particular functional form depends on the particular channel. For example, for the AWGN channel,

$$P_{\mathbf{c} \to e_i} = P_{d_{ci}} = Q\left(\sqrt{\frac{d_{ci}^2 R E_b}{2N_0}}\right). \tag{13.3}$$

We thus have

$$\overline{P} \leq \sum_{\mathbf{c}} P(\mathbf{c}) \sum_{e_i} P_{d_{ci}}.$$

This sum can be rearranged as

$$\overline{P} \leq \sum_{d_{ci}} A_{d_{ci}} P_{d_{ci}}, \tag{13.4}$$

where $A_{d_{ci}}$ is the average number of paths $\mathbf{p}_i$ that are at a distance d_{ci} from $\mathbf{c}$, and where the sum is over all the distances. The set of pairs $(d_{ci}, A_{d_{ci}})$ is known as the **distance spectrum** of the code [303, p. 124]. The smallest distance d_{ci} is the free distance of the code.

A lower bound on the probability of node error can be obtained by keeping only the first term of (13.4),

$$\overline{P} \geq A_{d_{\text{free}}} P_{d_{\text{free}}},$$

where d_{free} is the minimum of the distances between any correct sequence $\mathbf{c}$ and an incorrect sequence.

The discussion above applies to probability of a node error. Each node error causes a certain number of bit errors in the decoded message bits. Let $B_{d_{ci}}$ denote the average number of bit errors on error paths with distance d_{ci}. Since the trellis code encodes k bits per symbol, the average bit error rate is bounded by

$$\overline{P}_b \leq \sum_{d_{ci}} \frac{1}{k} B_{d_{ci}} P(d_{ci}).$$

For an AWGN we have, using (13.3),

$$\overline{P} \leq \sum_{d_{ci}} A_{d_{ci}} Q\left(\sqrt{\frac{d_{ci}^2 R E_b}{2N_0}}\right) \qquad \overline{P} \geq A_{d_{\text{free}}} Q\left(\sqrt{\frac{d_{\text{free}}^2 R E_b}{2N_0}}\right)$$

$$\overline{P}_b \leq \sum_{d_{ci}} \frac{1}{k} B_{d_{ci}} Q\left(\sqrt{\frac{d_{ci}^2 R E_b}{2N_0}}\right) \qquad \overline{P}_b \geq \frac{1}{k} B_{d_{\text{free}}} Q\left(\sqrt{\frac{d_{free}^2 R E_b}{2N_0}}\right).$$

13.4.2 A Description of the Error Events

For the error analysis of convolutional codes (Section 12.5), it was not necessary to average over the set of correct code sequences **c**, since it suffices to consider only the all zero codeword as the correct codeword. However, TCM is not necessarily a linear code. It may be necessary to consider average behavior over all correct paths. In this section we introduce some notation to describe how this is done.

Consider the case illustrated in Figure 13.11, where the correct path **c** passes through the states $p = p_0 \rightarrow p_1 \rightarrow p_2 \rightarrow \cdots \rightarrow p_{L-1} \rightarrow p_L$ and the incorrect path $\mathbf{e}_i$ consists of the states $q = q_0 \rightarrow q_1 \rightarrow q_2 \rightarrow \cdots \rightarrow q_{L-1} \rightarrow q_L$, where $q_0 = p_0$ and $q_L = p_L$. To

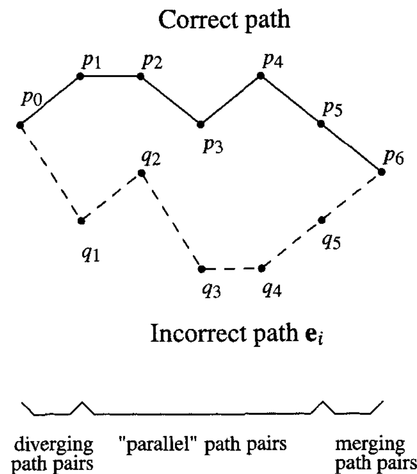


Figure 13.11: A correct path and an error path.

describe the error events corresponding to all error paths, we consider all paths $\mathbf{e}_i$ that deviate from the correct path. The two-tuple sequence $(p_0, q_0) \rightarrow (p_1, q_1) \rightarrow \cdots \rightarrow (p_L, q_L)$ denotes the pair of paths

$$p_0 \rightarrow p_1 \rightarrow p_2 \rightarrow \cdots \rightarrow p_{L-1} \rightarrow p_L \text{ and } q_0 \rightarrow q_1 \rightarrow q_2 \rightarrow \cdots \rightarrow q_{L-1} \rightarrow q_L.$$

Let $\delta((p, q) \rightarrow (p_1, q_1))$ denote the squared distance accrued (the branch metric) when the correct path transitions from state p to state p_1 while the incorrect path transitions from state q to state q_1. If there is no transition $(p, q) \rightarrow (p_1, q_1)$, then $\delta((p, q) \rightarrow (p_1, q_1))$ is defined to be ∞. The cumulative squared distance along this path is

$$\sum_{l=1}^{L} \delta((p_{l-1}, q_{l-1}) \rightarrow (p_l, q_l)) \stackrel{\Delta}{=} d_{ci}^2,$$

where d_{ci}^2 is the squared inter-path distance between the correct path **c** and the incorrect path $\mathbf{e}_i$.

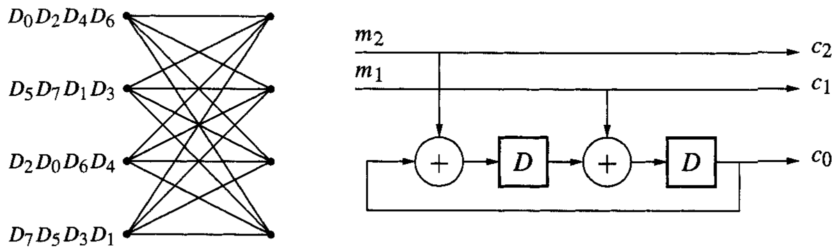


Figure 13.12: Example trellis for four-state code.

We develop an algebraic expression for the set of interpath distances using a power-series-like notation. Let x be a "dummy" variable. The squared distance $\delta((p, q) \rightarrow (p_1, q_1))$ is represented as the monomial $x^{\delta((p,q)\rightarrow(p_1,q_1))}$. Using this notation, products of monomials accumulate distances in the exponent. Thus

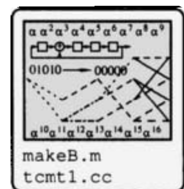
$$\prod_{l=1}^{L} x^{\delta((p_{l-1},q_{l-1})\rightarrow(p_l,q_l))} = x^{\sum_{l=1}^{L}\delta((p_{l-1},q_{l-1})\rightarrow(p_l,q_l))} = x^{d_{ci}^2}.$$

We assume that each transition $p \rightarrow p_1$ occurs with probability $1/2^k$ for a k-input TCM, which is the probability of the correct branch c.

We now define the **output transition matrix** associated with the encoder and decoder by

$$B_{(p,q),(p_1,q_1)} = [b_{(p,q),(p_1,q_1)}] = \frac{1}{2^k}[x^{\delta((p,q)\rightarrow(p_1,q_1))}] = \frac{1}{2^k}x^{d^2((p,q),(p_1,q_1))}.$$

The matrix B is indexed by all possible pairs of "from" states (p, q) and all possible pairs of "to" states (p_1, q_1). The elements of the matrix are monomials whose exponent is the squared branch metric.



Example 13.1 The trellis and encoder of Figure 13.12, having connection coefficients $g_0 = 5$, $g_1 = 4$ and $g_2 = 2$, are used with the 8-PSK partition shown in Figure 13.4. The following are some branch costs for this coder, assuming that the constellation is normalized so that $E_s = 1$.

$$\delta((0,0) \rightarrow (0,1)) = d^2(D_0, D_2) = 2 \qquad \delta((0,0) \rightarrow (0,2)) = d^2(D_0, D_4) = 4$$
$$\delta((0,1) \rightarrow (0,0)) = d^2(D_0, D_5) = 3.4 \qquad \delta((0,1) \rightarrow (0,1)) = d^2(D_0, D_7) = 0.6$$

The corresponding output transition matrix B is

$$B(x) = \frac{1}{4}\begin{array}{c} \\ 00 \\ 01 \\ 02 \\ 03 \\ 10 \\ 11 \\ 12 \\ 13 \\ 20 \\ 21 \\ 22 \\ 23 \\ 30 \\ 31 \\ 32 \\ 33 \end{array}\begin{array}{c} \begin{array}{cccccccccccccccc} 00 & 01 & 02 & 03 & 10 & 11 & 12 & 13 & 20 & 21 & 22 & 23 & 30 & 31 & 32 & 33 \end{array} \\ \left[\begin{array}{cccccccccccccccc}
1 & x^2 & x^4 & x^2 & x^2 & 1 & x^2 & x^4 & x^4 & x^2 & 1 & x^2 & x^2 & x^4 & x^2 & 1 \\
x^{3.4} & x^{0.6} & x^{0.6} & x^{3.4} & x^{3.4} & x^{3.4} & x^{0.6} & x^{0.6} & x^{0.6} & x^{3.4} & x^{3.4} & x^{0.6} & x^{0.6} & x^{0.6} & x^{3.4} & x^{3.4} \\
x^2 & 1 & x^2 & x^4 & 1 & x^2 & x^4 & x^2 & x^2 & x^4 & x^2 & 1 & x^4 & x^2 & 1 & x^2 \\
x^{0.6} & x^{3.4} & x^{3.4} & x^{0.6} & x^{3.4} & x^{3.4} & x^{0.6} & x^{0.6} & x^{3.4} & x^{0.6} & x^{0.6} & x^{3.4} & x^{0.6} & x^{0.6} & x^{3.4} & x^{3.4} \\
x^{3.4} & x^{3.4} & x^{0.6} & x^{0.6} & x^{0.6} & x^{3.4} & x^{3.4} & x^{0.6} & x^{0.6} & x^{0.6} & x^{3.4} & x^{3.4} & x^{3.4} & x^{0.6} & x^{0.6} & x^{3.4} \\
1 & x^2 & x^4 & x^2 & x^2 & 1 & x^2 & x^4 & x^4 & x^2 & 1 & x^2 & x^2 & x^4 & x^2 & 1 \\
x^{3.4} & x^{3.4} & x^{0.6} & x^{0.6} & x^{3.4} & x^{0.6} & x^{0.6} & x^{3.4} & x^{0.6} & x^{0.6} & x^{3.4} & x^{3.4} & x^{0.6} & x^{3.4} & x^{3.4} & x^{0.6} \\
x^2 & 1 & x^2 & x^4 & 1 & x^2 & x^4 & x^2 & x^2 & x^4 & x^2 & 1 & x^4 & x^2 & 1 & x^2 \\
x^2 & 1 & x^2 & x^4 & 1 & x^2 & x^4 & x^2 & x^2 & x^4 & x^2 & 1 & x^4 & x^2 & 1 & x^2 \\
x^{3.4} & x^{3.4} & x^{0.6} & x^{0.6} & x^{3.4} & x^{0.6} & x^{0.6} & x^{3.4} & x^{0.6} & x^{0.6} & x^{3.4} & x^{3.4} & x^{0.6} & x^{3.4} & x^{3.4} & x^{0.6} \\
1 & x^2 & x^4 & x^2 & x^2 & 1 & x^2 & x^4 & x^4 & x^2 & 1 & x^2 & x^2 & x^4 & x^2 & 1 \\
x^{3.4} & x^{3.4} & x^{0.6} & x^{0.6} & x^{0.6} & x^{3.4} & x^{3.4} & x^{0.6} & x^{0.6} & x^{0.6} & x^{3.4} & x^{3.4} & x^{3.4} & x^{0.6} & x^{0.6} & x^{3.4} \\
x^{0.6} & x^{3.4} & x^{3.4} & x^{0.6} & x^{3.4} & x^{3.4} & x^{0.6} & x^{0.6} & x^{3.4} & x^{0.6} & x^{0.6} & x^{3.4} & x^{0.6} & x^{0.6} & x^{3.4} & x^{3.4} \\
x^2 & 1 & x^2 & x^4 & 1 & x^2 & x^4 & x^2 & x^2 & x^4 & x^2 & 1 & x^4 & x^2 & 1 & x^2 \\
x^{3.4} & x^{0.6} & x^{0.6} & x^{3.4} & x^{3.4} & x^{3.4} & x^{0.6} & x^{0.6} & x^{0.6} & x^{3.4} & x^{3.4} & x^{0.6} & x^{0.6} & x^{0.6} & x^{3.4} & x^{3.4} \\
1 & x^2 & x^4 & x^2 & x^2 & 1 & x^2 & x^4 & x^4 & x^2 & 1 & x^2 & x^2 & x^4 & x^2 & 1
\end{array}\right] \end{array}.$$

□

For the output transition matrix, the $((p, p), (q, q))$ entry of the $B(x)^L$ is a polynomial in x whose exponents are all the distances between path pairs originating at (p, p) and terminating at (q, q); the coefficients of the polynomials are the average multiplicities of these distances. Thus matrix multiplication can be used to keep track of the distances between paths. The $B(x)$ matrix can thus be used to compute the distance spectrum for a given encoder.

We now split $B(x)$ into matrices corresponding to branches which diverge from a common node, branches which are on a "parallel" path, and branches which merge to a common node, denoting these as $D(x)$, $P(x)$, and $M(x)$, respectively. Thus the rows of $D(x)$ are indexed by values of (p, q) which are the same, the rows and columns of $P(x)$ are indexed by values of (p, q), (p_1, q_1) which are pairwise distinct, and the columns of $M(x)$ are indexed by values of (p_1, q_1) which are the same.

Example 13.2 For the matrix $B(x)$ of Example 13.1, we have

$$D(x) = \frac{1}{4}\begin{array}{c} \\ 00 \\ 11 \\ 22 \\ 33 \end{array}\begin{array}{c} \begin{array}{cccccccccccc} 01 & 02 & 03 & 10 & 12 & 13 & 20 & 21 & 23 & 30 & 31 & 32 \end{array} \\ \left[\begin{array}{cccccccccccc}
x^2 & x^4 & x^2 & x^2 & x^2 & x^4 & x^4 & x^2 & x^2 & x^2 & x^4 & x^2 \\
x^2 & x^4 & x^2 & x^2 & x^2 & x^4 & x^4 & x^2 & x^2 & x^2 & x^4 & x^2 \\
x^2 & x^4 & x^2 & x^2 & x^2 & x^4 & x^4 & x^2 & x^2 & x^2 & x^4 & x^2 \\
x^2 & x^4 & x^2 & x^2 & x^2 & x^4 & x^4 & x^2 & x^2 & x^2 & x^4 & x^2
\end{array}\right] \end{array}$$

$$P(x) = \frac{1}{4}\begin{array}{c} \\ 01 \\ 02 \\ 03 \\ 10 \\ 12 \\ 13 \\ 20 \\ 21 \\ 23 \\ 30 \\ 31 \\ 32 \end{array}\begin{array}{c} \begin{array}{cccccccccccc} 01 & 02 & 03 & 10 & 12 & 13 & 20 & 21 & 23 & 30 & 31 & 32 \end{array} \\ \left[\begin{array}{cccccccccccc}
x^{0.6} & x^{0.6} & x^{3.4} & x^{3.4} & x^{0.6} & x^{0.6} & x^{0.6} & x^{3.4} & x^{0.6} & x^{0.6} & x^{0.6} & x^{3.4} \\
1 & x^2 & x^4 & 1 & x^4 & x^2 & x^2 & x^4 & 1 & x^4 & x^2 & 1 \\
x^{3.4} & x^{3.4} & x^{0.6} & x^{3.4} & x^{0.6} & x^{0.6} & x^{3.4} & x^{0.6} & x^{3.4} & x^{0.6} & x^{0.6} & x^{3.4} \\
x^{3.4} & x^{0.6} & x^{0.6} & x^{0.6} & x^{3.4} & x^{0.6} & x^{0.6} & x^{0.6} & x^{3.4} & x^{3.4} & x^{0.6} & x^{0.6} \\
x^{3.4} & x^{0.6} & x^{0.6} & x^{3.4} & x^{0.6} & x^{3.4} & x^{0.6} & x^{0.6} & x^{3.4} & x^{0.6} & x^{3.4} & x^{3.4} \\
1 & x^2 & x^4 & 1 & x^4 & x^2 & x^2 & x^4 & 1 & x^4 & x^2 & 1 \\
1 & x^2 & x^4 & 1 & x^4 & x^2 & x^2 & x^4 & 1 & x^4 & x^2 & 1 \\
x^{3.4} & x^{0.6} & x^{0.6} & x^{3.4} & x^{0.6} & x^{3.4} & x^{0.6} & x^{0.6} & x^{3.4} & x^{0.6} & x^{3.4} & x^{3.4} \\
x^{3.4} & x^{0.6} & x^{0.6} & x^{0.6} & x^{3.4} & x^{0.6} & x^{0.6} & x^{0.6} & x^{3.4} & x^{3.4} & x^{0.6} & x^{0.6} \\
x^{3.4} & x^{3.4} & x^{0.6} & x^{3.4} & x^{0.6} & x^{0.6} & x^{3.4} & x^{0.6} & x^{3.4} & x^{0.6} & x^{0.6} & x^{3.4} \\
1 & x^2 & x^4 & 1 & x^4 & x^2 & x^2 & x^4 & 1 & x^4 & x^2 & 1 \\
x^{0.6} & x^{0.6} & x^{3.4} & x^{3.4} & x^{0.6} & x^{0.6} & x^{0.6} & x^{3.4} & x^{0.6} & x^{0.6} & x^{0.6} & x^{3.4}
\end{array}\right] \end{array}$$

$$M(x) = \frac{1}{4} \begin{array}{c} \\ 01 \\ 02 \\ 03 \\ 10 \\ 12 \\ 13 \\ 20 \\ 21 \\ 23 \\ 30 \\ 31 \\ 32 \end{array} \begin{array}{c} \begin{array}{cccc} 00 & 11 & 22 & 33 \end{array} \\ \begin{bmatrix} x^{3.4} & x^{3.4} & x^{3.4} & x^{3.4} \\ x^2 & x^2 & x^2 & x^2 \\ x^{0.6} & x^{3.4} & x^{0.6} & x^{3.4} \\ x^{3.4} & x^{3.4} & x^{3.4} & x^{3.4} \\ x^{3.4} & x^{0.6} & x^{3.4} & x^{0.6} \\ x^2 & x^2 & x^2 & x^2 \\ x^2 & x^2 & x^2 & x^2 \\ x^{3.4} & x^{0.6} & x^{3.4} & x^{0.6} \\ x^{3.4} & x^{3.4} & x^{3.4} & x^{3.4} \\ x^{0.6} & x^{3.4} & x^{0.6} & x^{3.4} \\ x^2 & x^2 & x^2 & x^2 \\ x^{3.4} & x^{3.4} & x^{3.4} & x^{3.4} \end{bmatrix} \end{array}.$$

□

With these matrices we can now describe the set of all error events. An error path diverges from a node and only remerges at the end of the error event; in between the error path and the correct path are never in the same state at the same time step. The set of all metrics of error events of exactly L branches is computed by

$$G_L(x) = D(x)P(x)^{L-2}M(x) \qquad L \geq 2.$$

This expression can be used to compute the distance spectrum for the code, although it becomes computationally infeasible for codes of even moderate numbers of states, due to the size of the matrices involved. (Algorithms based on the Viterbi algorithm are generally more efficient ways of actually computing the distance spectrum.) The rows and columns of the $G_L(x)$ matrix are indexed with (p, p) or (q, q) pairs. The $((p, p), (q, q))$ entry of G_L is an enumerator (or table) of all weighted distances between paths that start at the state p and end at the state q and have L branches. Note that

$$1^T G_L(x) 1,$$

where 1 is a vector of 2ν 1s, is the sum of all the elements of the matrix, which contains all paths of length L from any state to any state.

Returning to (13.4), let us use the bound

$$Q\left(\sqrt{\frac{d_{ci}^2 R E_b}{2N_0}}\right) \leq \frac{1}{2} \exp\left(-\frac{d_{ci}^2 R E_b}{4N_0}\right)$$

(see Exercise 1.12) so that

$$\overline{P} \leq \frac{1}{2} \sum_{d_{ci}} A_{d_{ci}} \exp\left(-\frac{d_{ci}^2 R E_b}{4N_0}\right).$$

Since the elements of G_L tabulate all the distances between path segments of length L, this sum can be written as

$$\overline{P} \leq \frac{1}{2}\frac{1}{2^\nu} \sum_{L=2}^{\infty} 1^T D(x)P(x)^{L-2}M(x)1 \Bigg|_{x=\exp(-RE_b/4N_0)}.$$

This can be manipulated as

$$\begin{aligned}\overline{P} &\leq \frac{1}{2}\frac{1}{2^\nu}1^T D(x)\sum_{i=0}^{\infty} P(x)^i M(x)1\Bigg|_{x=\exp(-RE_b/4N_0)} \\ &= \frac{1}{2}\frac{1}{2^\nu}1^T D(x)(I-P(x))^{-1}M(x)1\Bigg|_{x=\exp(-RE_b/4N_0)},\end{aligned} \tag{13.5}$$

where we have used the matrix identity $(I-P)^{-1} = \sum_{i=0}^{\infty} P^i$, analogous to the identity for scalars $\frac{1}{1-p} = 1 + p + p^2 + p^3 + \cdots$. The identity holds when $\lambda_{\max} < 1$, the largest eigenvalue of $P(x)$. When the code is noncatastrophic and the SNR is sufficiently large, this is the case. The bound (13.5) is referred to as the *transfer function bound.* Computationally, actually computing this bound could be difficult, since it requires computing the inverse of a $(N^2 - N) \times (N^2 - N)$ matrix, where $N = 2^\nu$.

A tighter bound can be obtained (see, e.g., [303, p.131]) by using a tight union bound for path differences up to a certain length, then employing the transfer function bound for the tail.

13.4.3 Known Good TCM Codes

Tables 13.2 through 13.5 describe TCM encoders which have been found by computer search [345, 347, 268]. The numbers g^i are connection polynomials in octal format. For example, the number $g = 23$ represents 10011, with the LSB g_0 on the right. The connections are used with the systematic convolutional encoder circuit shown in Figure 13.13. The mapping

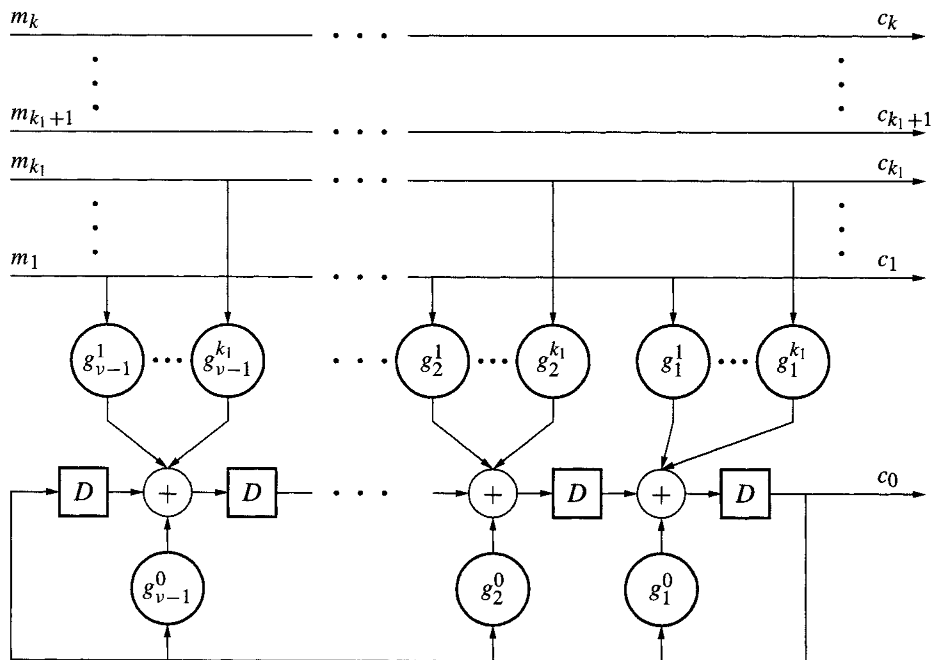


Figure 13.13: Trellis coder circuit.

from the outputs c_0, c_1, c_2 to the signal constellation is that of Figure 13.4, that is, with the

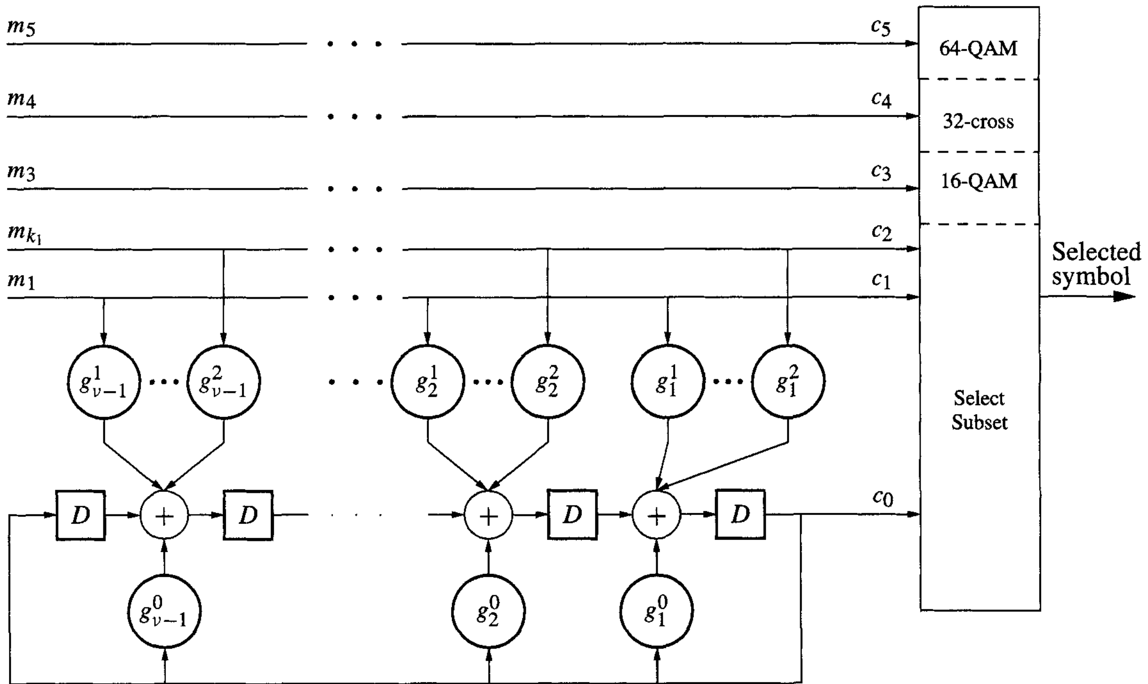


Figure 13.14: TCM encoder for QAM constellations.

points numbered consecutively around the circle. The asymptotic coding gain is with respect to QPSK, with minimum squared distance between signals of $2E_s$. The column $A_{d_{\text{free}}}$ is the average number of paths at distance d_{free}. The column $B_{d_{\text{free}}}$ is the average number of bit errors on those paths. The probability of a node error (selecting the wrong path) can be approximated by computing the probability of an error due to the shortest path (at distance d_{free} from the path corresponding to sending the all-zero sequence), scaled by the number of such paths:

$$P_e \approx A_{d_{\text{free}}} Q\left(\sqrt{\frac{d^2_{\text{free}} R E_b}{2N_0}}\right).$$

The probability of bit error is approximately

$$P_b \approx \frac{1}{k} B_{d_{\text{free}}} Q\left(\sqrt{\frac{d^2_{\text{free}} R E_b}{2N_0}}\right).$$

Another computer search [385] has yielded the improved 8-PSK designs also shown in Table 13.2. While the free distance is the same, the multiplicities $A_{d_{\text{free}}}$ and $B_{d_{\text{free}}}$ values are smaller, resulting in smaller error probabilities. This uses the 8-PSK with the points around the constellation labeled in a different order, as noted in the table.

A table of good codes for 16-QAM and larger constellations is shown in Table 13.5. The columns labeled Asymptotic Coding Gain show both the coded constellation and the constellation used for comparison. The corresponding circuit diagram for this code is

Table 13.2: Maximum Free-Distance Trellis Codes for 8-PSK Constellation [347, 268, 303, 385]

$\Delta_0 = 2\sin(\pi/8)$

Number of States	g_0	g_1	g_2	$d^2_{\text{free}}/\Delta^2_0$	$A_{d_{\text{free}}}$	$B_{d_{\text{free}}}$	Asymptotic Coding Gain (dB) 8-PSK/4-PSK
4	5	2	–	4.00	1	1	3.0
8	11	2	4	4.59	2	7	3.6
8†	17	2	6	4.59	2	5	3.6
16	23	4	15	5.17	2.25	11.5	4.1
16†	27	4	12	5.17	2.25	7.5	4.1
32	45	16	34	5.76	4	22.25	4.6
32†	43	4	24	5.76	2.375	7.375	4.6
64	103	30	66	6.34	5.25	31.125	5.0
64†	147	12	66	6.34	3.25	14.8755	5.0
128	277	54	122	6.59	0.5	2.5	5.2
128†	277	54	176	6.59	0.5	2	5.2
256	435	72	130	7.52	1.5	12.25	5.8
256†	435	72	142	7.52	1.5	7.813	5.8
512	1525	462	360	7.52	0.313	2.75	5.8
512†	1377	304	350	7.52	0.0313	0.25	5.8
1024	2701	1216	574	8.10	1.32	10.563	6.1
1024†	2077	630	1132	8.10	0.2813	1.688	6.1
2048	4041	1212	330	8.34	3.875	21.25	6.2
4096	15201	6306	4112	8.68	1.406	11.758	6.4
8192	20201	12746	304	8.68	0.617	2.711	6.4
32768	143373	70002	47674	9.51	0.25	2.5	6.8
131072	616273	340602	237374	9.85			6.9

† Use point labeling (000), (001), (010), (011), (110), (111), (100), (101).

shown in Figure 13.14. An interesting feature about this structure is that it can be employed with larger signal constellations by using more uncoded bits. The 32-cross and 64-QAM constellations are shown in Figure 13.2. The set partition and assignment for most of these follow the pattern set in Figure 13.9. Also shown in Table 13.5 are connectors for another set of codes with generally lower $A_{d_{\text{free}}}$ and $B_{d_{\text{free}}}$ due to [385]. These are indicated with †. These use the labels shown below the table.

13.5 Decoding TCM Codes

Optimal decoding is accomplished using a Viterbi algorithm. The general outline is the same as for convolutional codes. However, in computing the branch metric associated with a received signal $\mathbf{r}_t$, the *nearest* point in the subset for that branch is used. For branches with parallel transitions (that is, whose subsets contain more than one point), it is necessary to compute the distance between $\mathbf{r}$ and every point in the subset.

In the second step the signal point selected from each subset (in step 1) is used to determine a branch cost for a Viterbi algorithm using a squared distance measurement. The optimal sequence is that which has the minimum sum of squared distances along the trellis.

Table 13.3: Maximum Free-Distance Trellis Codes for 16-PSK Constellation [347]
$\Delta_0 = 2\sin(\pi/16)$

Number of States	g_0	g_1	g_2	$d^2_{\text{free}}/\Delta_0^2$	$A_{d_{\text{free}}}$	Asymptotic Coding Gain (dB) 16-PSK/8-PSK
4	5	2	–	1.324	4	3.54
8	13	4	–	1.476	4	4.01
16	23	4	–	1.628	8	4.44
32	45	10	–	1.910	8	5.13
64	103	24	–	2.000	2	5.33
128	203	24	–	2.000	2	5.33
256	427	374	176	2.085	8	5.51

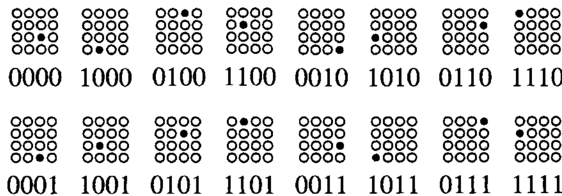
Table 13.4: Maximum Free-Distance Trellis Codes for Amplitude Modulated (One-Dimensional) Constellations [347]

Number of States	g_0	g_1	$d^2_{\text{free}}/\Delta_0^2$	$A_{d_{\text{free}}}$	Asympt. Gain (dB) (coded/uncoded)	
					4-AM/ 2-AM	8-AM/4-AM
4	5	2	9.0	4	2.55	3.31
8	13	4	10.0	4	3.01	3.77
16	23	4	11.0	8	3.42	4.18
32	45	10	13.0	12	4.15	4.91
64	103	24	14.0	36	4.47	5.23
128	235	126	16.0	66	5.05	5.81
256	515	362	16.0	2	–	5.81

Table 13.5: Encoder Connections and Coding Gains for Maximum Free-Distance QAM Trellis Codes [347][385]†

Number of States	g_0	g_1	g_2	g_3	d^2_{free}	$A_{d_{\text{free}}}$	$B_{d_{\text{free}}}$	Asympt. Gain (dB) (coded/uncoded)		
								16-QAM/ 8-PSK	32-cross/ 16-QAM	64-QAM/ 32-cross
4	5	2	–	–	4.0			4.4	3.0	2.8
8	11	2	4	–	5.0	3.656	18.313	5.3	4.0	3.8
8†	13	4	2	6	5.0	3.656	12.344			
16	23	4	16	–	6.0	9.156	53.5	6.1	4.8	4.6
16†	25	12	6	14	6.0	9.156	37.594			
32	41	6	10	–	6.0	2.641	16.063	6.1	4.8	4.6
32†	47	22	16	34	6.0	2	6			
64	101	16	64	–	7.0	8.422	55.688	6.8	5.4	5.2
64†	117	26	74	52	7.0	5.078	21.688			
128	203	14	42	–	8.0	36.36	277.367	7.4	6.0	5.8
128†	313	176	154	22	8.0	20.328	100.031			
256	401	56	304	–	8.0	7.613	51.953	7.4	6.0	5.8
256†	417	266	40	226	8.0	3.273	16.391			
512	1001	346	510	–	8.0			7.4	6.0	5.8

† Use the labeling shown here.



13.6 Rotational Invariance

A real digital receiver must typically estimate the phase of the received signal. For QAM signals, methods exist which can estimate the phase, but only up to a phase uncertainty of a multiple of $\pi/2$ radians. This introduces a $p\pi/2$ *phase ambiguity* (p an integer) which must be accommodated in the receiver. When TCM is employed, it may be possible to identify if the receiver has the correct decoding phase by examining the likelihoods computed by the Viterbi algorithm. If no path emerges as having significantly better likelihood than the others, than it is likely that the wrong phase has been selected. The receiver can adjust the phase by $\pi/2$ and try again. This procedure, however, takes additional synchronization time. Another approach is to transmit the information in such a way that it can be accurately recovered regardless of the $p\pi/2$ ambiguity. This can be accomplished by (1) using a TCM code which is invariant with respect to rotation; and (2) employing differential encoding of some of the bits.

Let $\mathbf{a} = (\mathbf{a}_0, \mathbf{a}_1, \mathbf{a}_2, \ldots)$ be a sequence of complex coded symbols from a two-dimensional signal constellation. Let $\mathbf{a}^\phi$ be the sequence obtained by rotating each $\mathbf{a}_t$ by a fixed angle ϕ: $\mathbf{a}_t^\phi = e^{j\phi}\mathbf{a}_t$. We have the following:

Definition 13.1 A TCM code is **rotationally invariant** with respect to a rotation by ϕ if $\mathbf{a}^\phi$ is also a valid coded symbol sequence for every valid coded symbol sequence $\mathbf{a}$. □

Rotational invariance in TCM can be related to the trellis as follows. Let $\mathcal{S}$ denote the signal constellation and let $\mathcal{S}_i$ denote the set of subsets, at some level of signal partitioning, which are transmitted along the branches of the trellis. Assume that the partitioning is done such that, for each possible phase rotation, each subset $\mathcal{S}_i$ rotates into another $\mathcal{S}_j$. Then the set of subsets is invariant under phase rotation. It turns out that this invariance holds automatically for one- and two-dimensional signal constellations [362]. (For example, consider the set partitions in Figures 13.4 and 13.9.) For such a rotational invariant set of subsets, we have the following.

Theorem 13.1 *[362, 366, 303] For each transition on the trellis from state i to state j associated with a subset A, let B denote the subset obtained when A is rotated by ϕ, as shown in Figure 13.15.*

Then a TCM code is rotationally invariant with respect to a rotation by an angle ϕ if there exists a bijective function $f_\phi : \mathcal{S} \to \mathcal{S}$ with the property that B is the subset associated with the transition from $f_\phi(i)$ to state $f_\phi(j)$ (and so $f_\phi(i) \to f_\phi(j)$ is a valid state transition) when A is the subset associated with the transition from state i to state j.

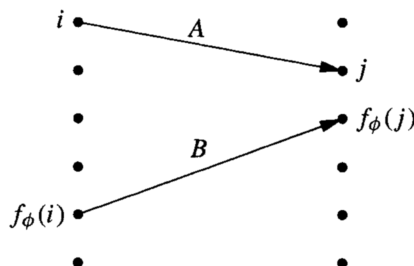


Figure 13.15: Mapping of edge (i, j) to edge $(f_\phi(i), f_\phi(j))$.

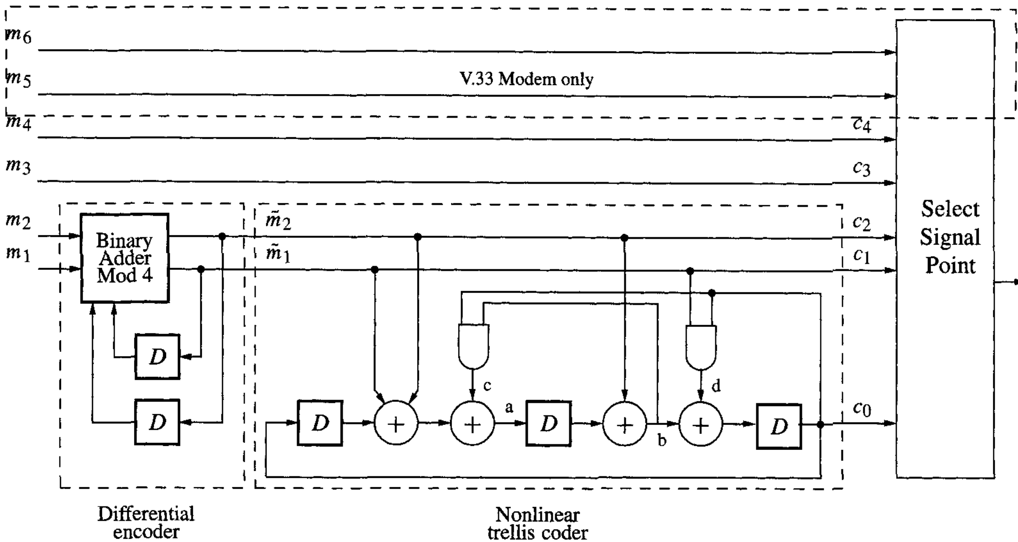
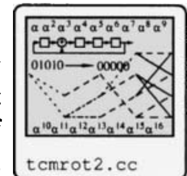


Figure 13.16: Encoder circuit for rotationally invariant TCM code.

Proof Let $i_1, i_2, i_3, \ldots$ denote a sequence of states on a valid path through the trellis. If the conditions of the theorem are satisfied, then $f_\phi(i_1), f_\phi(i_2), f_\phi(i_3), \ldots$ are also on a valid path, each branch selecting a rotated symbol. □

It has been found that using a linear convolutional code as the state machine underlying the TCM cannot achieve rotational invariance [263]. Nonlinear trellis codes, however, have been found which can achieve rotational invariance [363, 364]. We present examples of such codes which are widely used in V.32 and V.33 industry standards, referring the interested reader to the literature [363, 364] for design methodologies. The V.32 standard [32] operates at bit rates of 9600 bits/second using a symbol rate of 2400 symbols/second (suitable for use on a standard telephone line) by achieving up to 4 bits per symbols. To do this, it uses a coded signal constellation with 32 points in it (the 32-cross constellation). The V.33 standard [32] provides for data rates of up to 14,400 bits/second using 2400 symbols/second by carrying six bits per symbol using a 128-point coded signal constellation. This code provides up to 4 dB of coding gain.

The encoder of Figure 13.16 is a nonlinear trellis encoder whose trellis is shown in Figure 13.17. In this figure, the input that gives rise to output subset D_i can be found by taking the two most significant bits of i. Thus a branch transmitting D_5 is due to an input of $(\tilde{m}_2, \tilde{m}_1) = (1, 0)$, since $5 = \mathbf{10}1_2$. A branch transmitting D_7 is due to an input of $(\tilde{m}_2, \tilde{m}_1) = (1, 1)$, etc. It can be shown that the code represented by this trellis is invariant, in the sense of Theorem 13.1. The corresponding labeled signals and the constellation partition are shown in Figure 13.18. We observe that the first two bits (labeled with the light font) are invariant with respect to $\pi/2$ rotations. However, the last three label bits *do* change with rotation. The code uses differential coding techniques to achieve invariance to the changes in the last three bits. The first stage of the encoder takes two input bits and differentially encodes them. The differentially encoded bits are used by the trellis coder.



The overall framework of invariance works as follows. If a transmitted signal point is rotated by some multiple of $\pi/2$, then the corresponding received signal point is identical

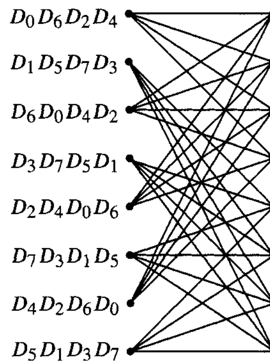


Figure 13.17: Trellis for the rotationally invariant code of Figure 13.16

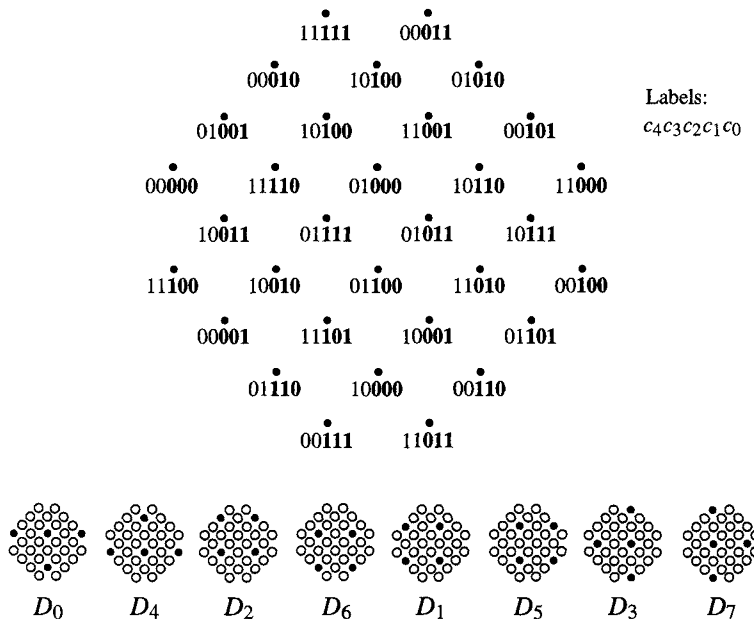


Figure 13.18: 32-cross constellation for rotationally invariant TCM code.

in the first two bits. The last three bits differ because of the rotation. However, because the code is rotationally invariant, there is still a valid path through the trellis which can be used to decode this rotated sequence of symbols. The input bits for the rotated signals can be decoded. Then, since the bits are differentially encoded, so that the sequence of *differences* does not change when the signal sequence is rotated, the original bits can be recovered.

Differential Encoding

The **differential encoder** operates as follows. The input bits $m_{1,t}$ and $m_{2,t}$ are converted to an integer, $m_t = 2m_{2,t} + m_{1,t}$. The encoder keeps the previous outputs $\tilde{m}_{1,t-1}, \tilde{m}_{2,t-1}$,

represented as an integer by $\tilde{m}_{t-1} = 2\tilde{m}_{2,t-1} + \tilde{m}_{1,t-1}$. Then the differential encoder computes

$$\tilde{m}_t = m_t + \tilde{m}_{t-1} \pmod 4. \tag{13.6}$$

The initial memory of the differential encoder is assumed to be set at 0.

Given a received sequence of differentially encoded data $\tilde{m}_{1,t}, \tilde{m}_{2,t}$, the original data can be recovered as

$$m_t = \tilde{m}_t - \tilde{m}_{t-1} \pmod 4. \tag{13.7}$$

Example 13.3 Suppose the sequence of input data $(m_{2,t}, m_{1,t})$ is

$$(01)(10)(11)(01)(10)(11)\ldots$$

The differential encoding proceeds as in the following table.

$(m_{2,t}, m_{1,t})$	m_t	$(\tilde{m}_{2,t-1}, \tilde{m}_{1,t-1})$	$\tilde{m}_{t-1}$	$\tilde{m}_t = m_t + \tilde{m}_{t-1} \pmod 4$	$(\tilde{m}_{2,t}, \tilde{m}_{1,t})$
					(0,0)
(0,1)	1	(0,0)	0	1	(0,1)
(1,0)	2	(0,1)	1	3	(1,1)
(1,1)	3	(1,1)	3	2	(1,0)
(0,1)	1	(1,0)	2	3	(1,1)
(1,0)	2	(1,1)	3	1	(0,1)
(1,1)	3	(0,1)	1	0	(0,0)

□

Constellation Labels and Partitions

The sets D_i consist of points having the label i in the last three digits in binary notation (in bold font in Figure 13.18). Examination of the subsets in figure 13.18 reveals that under rotation of $\pi/2$, the subset D_0 maps to D_7, and D_7 maps to D_4, and so forth. The sets map under $\pi/2$ rotations as

$$\begin{gathered} D_0 \rightarrow D_7 \rightarrow D_4 \rightarrow D_3 \rightarrow D_0 \\ D_1 \rightarrow D_6 \rightarrow D_5 \rightarrow D_2 \rightarrow D_1. \end{gathered} \tag{13.8}$$

Example 13.4 The sequence of bits

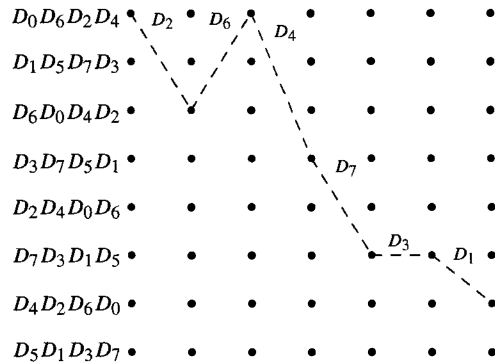
$$(11\,01)(01\,10)(11\,11)(10\,01)(00\,10)(01\,11)$$

is to be transmitted, where the 4-tuples represent $(m_{4,t}, m_{3,t}, m_{2,t}, m_{1,t})$. After differential encoding the second pair of bits (see the previous example), the sequence of bits $(m_4, m_3, \tilde{m}_2, \tilde{m}_1)$ is

$$(11\,01)(01\,11)(11\,10)(10\,11)(00\,01)(01\,00).$$

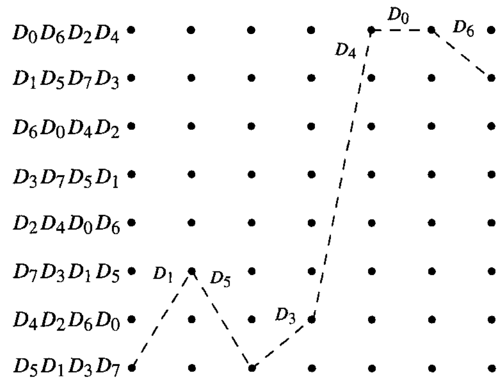
This sequence is presented to the nonlinear trellis coder (starting from state 0) resulting in the following output and path through the trellis.

state	input	output	subset	next state
000	1101	11**010**	D_2	010
010	0111	01**110**	D_6	000
000	1110	11**100**	D_4	011
011	1011	10**111**	D_7	101
101	0001	00**011**	D_3	111
111	0100	01**001**	D_1	111



Now suppose that at the receiver the sequence is received with a $\pi/2$ rotation, so that the points of the signal constellation correspond to the following bit patterns:

received bits	subset
11001	D_1
01101	D_5
11011	D_3
10100	D_4
00000	D_0
01110	D_6



Of course, the initial state in the trellis is not known initially, but would be discovered by the Viterbi algorithm.

We make the following observations about these bits:

- The first two bits are unchanged by the rotation. This occurs because the symbol labels were created so that $p\pi/2$ rotations do not affect the first two bits.
- The subsets represented by the last three bits are obtained by rotating the transmitted subsets according to the cyclic translations of (13.8).

As shown, a valid path through the trellis can be found. However, it does not necessarily start with state 0. The Viterbi decoding algorithm must be prepared to start with any state. Furthermore, if the initial state is not zero, the differential decoder must be initialized with the data corresponding to the rotation which moves to the initial state. Since the path starts at state 7, the differential decoder is initialized with $(m_{2,-1}, m_{1,-1}) = (1, 1)$.

The sequence of input bits corresponding to this path is

$$(11\,00)(01\,10)(11\,01)(10\,10)(00\,00)(01\,11),$$

where the last two bits of the 4-tuple are differentially encoded. Using (13.7) to undo the effect of the differential decoding on those bits, we obtain the following:

$(\tilde{m}_{2,t}, m_{1,t})$	$\tilde{m}_t$	$(\tilde{m}_{2,t-1}, \tilde{m}_{1,t-1})$	$\tilde{m}_{t-1}$	$m_t = \tilde{m}_t - \tilde{m}_{t-1}$ (mod 4)	$(m_{2,t}, m_{1,t})$
		(1,1)			
(0,0)	0	(1,1)	3	1	(0,1)
(1,0)	2	(0,0)	0	2	(1,0)
(0,1)	1	(1,0)	2	3	(1,1)
(1,0)	2	(0,1)	1	1	(0,1)
(0,0)	0	(1,0)	2	2	(1,0)
(1,1)	3	(0,0)	0	3	(1,1)

The decoded sequence is thus

$$(11\,01)(01\,10)(11\,11)(10\,01)(00\,10)(01\,11),$$

the same as transmitted originally. Thus even though the signal constellation was rotated due to phase ambiguity, the decoder was invariant to such rotations. □

13.7 Multidimensional TCM

The TCM described up to this point has employed one- or two-dimensional signal constellations. However, there are several compelling reasons for dealing with constellations in more than two dimensions. After presenting some of these reasons, we present one of several possible frameworks for mathematical descriptions of signal constellations and their partitions in multiple dimensions using lattices and their cosets. These rather general descriptions are followed by an extended example, the code used in the V.34 (also known as V.fast) modem protocol.

We begin, however, with a discussion of how to obtain multiple dimensions using digital signaling. We detail the notation only with even-numbers of dimensions; modification to odd-numbers of dimensions is straightforward.

The $2L$-dimensional signal point $\mathbf{a} = (a_1, a_2, \ldots, a_{2L})$ can be transmitted by sending a sequence of L two-dimensional points over L signaling intervals

$$(a_1, a_2), (a_3, a_4), \ldots, (a_{2L-1}, a_{2L}).$$

If the uncoded multi-dimensional constellation is employed with an overall spectral efficiency of η bits/symbol, then there must be $2^{\eta L}$ symbols in the multidimensional constellation.

Example 13.5 Suppose that a signal is to be transmitted with $\eta = 4$ bits/symbol using a 4-dimensional constellation. Then the two symbols required to carry the 4 coordinates must represent $2 \times 4 = 8$ bits, so that the constellation must have 2^8 points in it.

Now suppose that a TCM is used with a 4-dimensional constellation with a $k/(k+1)$ convolutional encoder. There must be 2^9 points in the signal constellation. □

Multidimensional TCM is similar to one- or two-dimensional TCM: k_1 out of k input bits are input into a rate $k_1/(k_1 + 1)$ trellis encoder. These bits are used to selected one of 2^{k_1+1} subsets of a $2L$-dimensional signal constellation. The remaining $k_2 = k - k_1$ input bits then select a single point out of the subset. This signal point is then transmitted by a sequence of L two-dimensional points. A difference from one- or two-dimensional TCM is that the trellis encoder circuit is used only every L symbol times.

13.7.1 Some Advantages of Multidimensional TCM

Energy expansion advantage In one- or two-dimensional TCM, the rate $k_1/(k_1 + 1)$ encoder requires that the number of points in the signal constellation be doubled to preserve rate so that there must be $2^{\eta L+1}$ symbols in the constellation to transmit with a spectral efficiency of η bits/symbol. This roughly doubles the average signal energy, since the extra redundancy must be accommodated over a single symbol interval. This results in approximately a 3 dB penalty in γ_C.

However, in multiple dimensions there is only one redundant bit spread over L symbol times, so the energy penalty is reduced. The extra energy required to represent this larger signal constellation is shared among L transmitted symbols. For a 4-dimensional signal constellation, the penalty in γ_C is 1.5 dB.

Sphere-packing advantages To obtain the smallest average signal energy, it is desirable to pack the points of the signal constellation as closely as possible while maintaining minimum inter-symbol distance requirements. The problem of placing points in a signal constellation is thus an instance of the "sphere packing problem." This can be expressed in familiar terms in three dimensions as the problem of packing as many identical spherical oranges (maintaining at least a minimum distance between centers) as possible into a crate of given dimensions. It can be shown that if the oranges are stacked in layers with one orange resting over the interstices formed by the oranges in the layer below, then more oranges can be packed into the crate than if the oranges are stacked in "$\mathbb{Z}^3$" way, in a square lattice with the center of each orange over the center of the orange below it.

In higher dimensions it may be possible to stack points in such a way that the density is higher than simply stacking them on a multidimensional rectangular grid. This results in a lower average signal energy compared to the rectangular lattice $\mathbb{Z}^n$.

Spectral efficiency If the channel has bandwidth to support η bits/symbol but not $\eta + 1$ bits/symbol, it may be possible to squeeze a little more out by using $\eta + \delta$ bits/symbol, for some rational number $0 < \delta < 1$. Using multidimensional constellations, it is possible to design transmission systems with such fractional spectral efficiencies.

Rotational invariance For two-dimensional constellations, nonlinear trellis coders must be employed to obtain rotational invariance. However, linear encoders can be used in higher dimensional TCM.

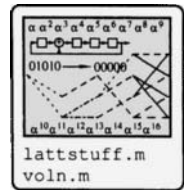
Signal shape Signal shape [94, 196] slightly reduces the average energy requirements even further by selectively using points of smaller energy. This is used in the V.34 modem, as described below.

Peak-to-average power ratio Multidimensional constellations can be designed which have a lower peak-to-average power ratio.

Decoding speed The first step in decoding is to determine the closest point in a subset to the received data. For many lattices, efficient algorithms exist for doing this (see, e.g., [57]). Furthermore, the state of the trellis must be advanced only every L received signals. These factors allow for higher speed decoding.

13.7.2 Lattices and Sublattices

While there are many ways of constructing multidimensional signal constellations, one very important way employs lattices and sublattices. We briefly introduce lattices here; extensive detail is presented in [56].



Basic Definitions

A lattice Λ is an (infinite) discrete periodic arrangement of points in $\mathbb{R}^m$. A signal constellation based on a lattice is obtained by selecting a finite number of points from the lattice, possibly with a translation, with the points usually selected in such a way as to minimize the average energy in the constellation.

A lattice may be described by a *generator matrix*[1] M, where

$$M = \begin{bmatrix} v_{11} & v_{12} & \cdots v_{1m} \\ v_{21} & v_{22} & \cdots v_{2m} \\ \vdots & & \\ v_{n1} & v_{n2} & \cdots v_{nm} \end{bmatrix}$$

with $m \geq n$, where, following convention, each *row* is a basis vector. Then the lattice is the set

$$\Lambda = \{\boldsymbol{\xi} M : \boldsymbol{\xi} \in \mathbb{Z}^n\};$$

that is, *integer* linear combinations of the basis vectors. Note that a lattice forms a group under addition.

It should be obvious that the generator is not unique. Two generator matrices M and $\tilde{M}$ define *equivalent* lattices if $\tilde{M} = cUMB$, where c is a nonzero constant, U is a matrix with integer entries and $\det(U) = \pm 1$ (that is, U is unimodular) and B is orthogonal, $BB^T = I$. Equivalent lattices are essentially just rotated and/or scaled versions of each other.

Example 13.6 A portion of the lattice $\mathbb{Z}^2$, consisting of points (n_1, n_2) for $n_i \in \mathbb{Z}$, is shown in Figure 13.19(a). It has the generator $M = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$. A 16-QAM constellation can be obtained, for example, by selecting 16 points of $\Lambda + (1/2, 1/2)$. The n-dimensional extensions of this lattice, denoted by $\mathbb{Z}^n$, are generated by the $n \times n$ identity matrix. □

Example 13.7 The hexagonal lattice, known as the A_2 lattice, is shown in Figure 13.20(a). It can be generated by

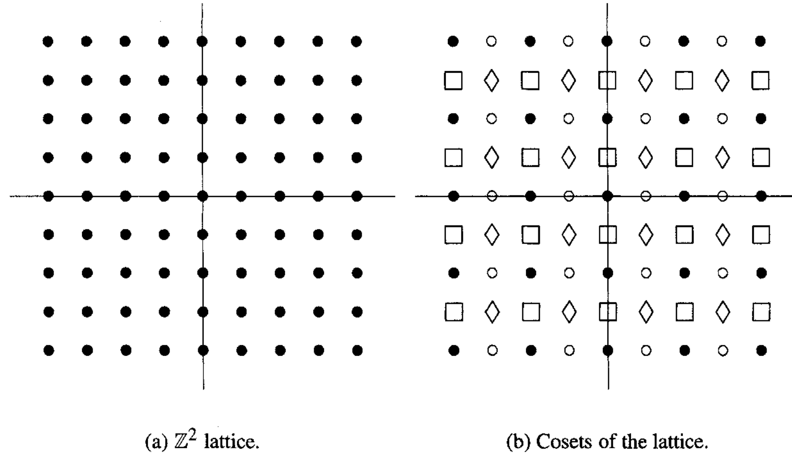
$$M = \begin{bmatrix} 1 & 0 \\ \frac{1}{2} & \frac{\sqrt{3}}{2} \end{bmatrix}.$$

Not so obviously, the hexagonal lattice can also be generated by

$$M = \begin{bmatrix} 1 & -1 & 0 \\ 0 & 1 & -1 \end{bmatrix},$$

giving a two-dimensional lattice embedded in three dimensions. □

[1] Not to be confused with the generator matrix for a linear block code.



(a) $\mathbb{Z}^2$ lattice. (b) Cosets of the lattice.

Figure 13.19: A portion of the lattice $\mathbb{Z}^2$ and its cosets.

Figure 13.20(b) shows that around each lattice point a circle (or, in general, a sphere) can be drawn which does not intersect with identical spheres around the other lattice points; we denote the radius of the largest such sphere by ρ. Also, around each point is a region known as the *fundamental parallelotope* (shown shaded). Associated with the generator is the *Gram matrix* A,

$$A = MM^T.$$

The determinant of the lattice generated by M is defined to be the determinant of the Gram matrix, $\det \Lambda = \det A$. The volume of the fundamental region of the lattice, or the *fundamental volume*, denoted by $V(\Lambda)$, is

$$V(\Lambda) = |\det(\Lambda)|^{1/2} = |\det(A)|^{1/2}.$$

Example 13.8 For the lattice $\Lambda = \mathbb{Z}^{2n}$ (even-numbered dimensions), the volume of the fundamental parallelotope is

$$V(\Lambda) = |\det(\Lambda)|^{1/2} = |\det(I)|^{1/2} = 1.$$

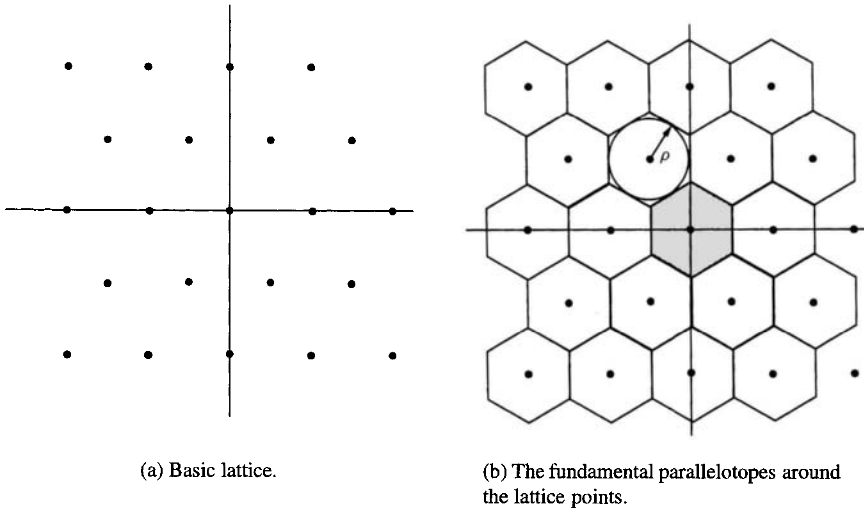
□

Example 13.9 For the A_2 lattice with minimum distance between points equal to 1, the volume of the fundamental parallelotope is

$$V(\Lambda) = |\det(\Lambda)|^{1/2} = \left[\det \begin{bmatrix} 1 & 0 \\ \frac{1}{2} & \frac{\sqrt{3}}{2} \end{bmatrix} \begin{bmatrix} 1 & \frac{1}{2} \\ 0 & \frac{\sqrt{3}}{2} \end{bmatrix}\right]^{1/2} = \sqrt{\frac{3}{4}}.$$

The hexagonal lattice has a smaller fundamental volume than the $\mathbb{Z}^2$ lattice with the same minimum distance. It thus packs points more efficiently into space. □

Another relevant attribute of lattices is the *kissing number*, usually denoted by τ, which is the number of nearest neighbors a lattice point has. This has bearing in code design, since the asymptotic performance is governed by the number of nearest neighbors a point has. For the A_2 lattice the kissing number is $\tau = 6$. For the lattice $\mathbb{Z}^n$ the kissing number is $\tau = 2n$.



(a) Basic lattice.

(b) The fundamental parallelotopes around the lattice points.

Figure 13.20: Hexagonal lattice.

Common Lattices

Table 13.6 summarizes the attributes of the lattices described here.

Table 13.6: Attributes of Some Lattices

Name	Dimension	Kissing Number τ	Fundamental Volume $V(\Lambda)$	Lattice Coding gain γ_{cg}	γ_{cg} (dB)
$\mathbb{Z}^n$	n	$2n$	1	1	0
A_2 (hexagonal)	2	6	$\sqrt{3}/2$	1.15	0.63
A_3 (face-centered cubic)	3	12	$\sqrt{2}/2$	1.26	1.00
D_4	4	24	0.5	$\sqrt{2}$	1.51
D_n	n (even)	$2n(n-1)$	$2^{(1-n/2)}$	$2^{(1-2/n)}$	$3.01(1-2/n)$
E_6	6	72	0.2165	1.6654	2.21
E_8	8	240	1/16	2	3.01
Λ_{16} (Barnes-Wall)	16	4320	2.33×10^{-4}	2.83	4.51
Λ_{24} (Leech)	24	196560	5.96×10^{-8}	4	6.02

The fundamental volume is computed for a lattice normalized so the minimum distance between points equal to 1.

D_4, also known as the checkerboard lattice, is the densest lattice in four dimensions [56, p. 9]. The lattice points are (u_1, u_2, u_3, u_4) where the u_i are integers and $u_1 + u_2 + u_3 + u_4$ is an even integer. The center (0,0,0,0) has the points $(\pm 1, \pm 1, 0, 0)$ and their permutations as nearest neighbors, so that the kissing number is $\tau = 24$. Any two distinct points must differ by at least 1 in at least two coordinates, or by 2 in at least one coordinate, so the minimal distance between centers is $\sqrt{2}$, and $\rho = \sqrt{2}/2$. A generator matrix for this and other lattices mentioned here is provided in `lattstuff.m`.

E_8 provides the densest lattice packing in eight dimensions [56, p. 120]. The lattice can be described as follows: The set of points

$$\{(u_1, u_2, \ldots, u_8) : \text{ all } u_i \in \mathbb{Z} \text{ or all } u_i \in \mathbb{Z} + \frac{1}{2}, \text{ and } \sum x_i \text{ is even}\}.$$

E_6 is the densest lattice in 6 dimensions [56, p. 125]. Points in this lattice are vectors in E_8 which are perpendicular to any A_2 sublattice V in E_8:

$$E_6 = \{x \in E_8 : x \cdot v = 0 \text{ for all } v \in V\};$$

another description is

$$E_6 = \{(x_1, \ldots, x_8) \in E_8 : x_1 + x_8 = x_2 + \cdots + x_7 = 0\}.$$

Another description for E_6 is over the Eisenstein integers, the set $\mathcal{E} = \{a + \omega b : a, b \in \mathbb{Z}, \omega = (-1 + i\sqrt{3})/2\}$. This uses the generator

$$M = \begin{bmatrix} \theta & 0 & 0 \\ 1 & -1 & 0 \\ 1 & 0 & -1 \end{bmatrix}.$$

where $\theta = \sqrt{-3}$.

Λ_{16} is the Barnes-Wall lattice [56, p. 129]. This lattice has strong connections with Reed-Muller codes of length 16.

Λ_{24} is the Leech lattice [56, p. 131]. What makes it remarkable is that it can be constructed in many ways, with many connections to block error correcting codes. We mention only one. The lattice can be generated by all vectors of the form $\frac{1}{\sqrt{8}}(\mp 3, \pm 1^{23})$ (that is, 23 ones), where the ∓ 3 may be in any position, and the upper signs are taken on the set of coordinates where the binary Golay (24,12) code is 1.

Sublattices and Cosets

A **sublattice** of a lattice is a lattice Λ' all of whose points lie in the lattice Λ. The sublattice is generated by a matrix M'.

As a subgroup of a group, there are cosets associated with a sublattice. A coset of a lattice Λ' is a translation $\Lambda' + \mathbf{p}$ of all points in Λ' by $\mathbf{p}$. The set of cosets of Λ produced by Λ' is denoted Λ/Λ'; since the lattice is an Abelian group, Λ/Λ' is a group. We can write the partition of Λ into cosets as

$$\Lambda = \Lambda' \cup \{\mathbf{p}_1 + \Lambda'\} \cup \{\mathbf{p}_2 + \Lambda'\} \cup \cdots \cup \{\mathbf{p}_{N-1} + \Lambda'\}$$

for some number N which is the number of cosets.

Example 13.10 Let $\Lambda = \mathbb{Z}^2$ and let $\Lambda' = 2\mathbb{Z}^2$. That is, the generator is

$$M' = 2M = 2\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

There are four cosets in this lattice, with the following shape designations in Figure 13.19(b):

$S_0 = \Lambda'$ (denoted by $\bullet$) $\quad S_1 = (1, 0) + \Lambda'$ (denoted by $\circ$)

$S_2 = (0, 1) + \Lambda'$ (denoted by $\square$) $\quad S_3 = (1, 1) + \Lambda'$ (denoted by $\Diamond$)

The set of lattices $\mathbb{Z}^2/2\mathbb{Z}^2$ is isomorphic to the group $\mathbb{Z}_2 \times \mathbb{Z}_2$. $\square$

A **partition chain** of a lattice, denoted $\Lambda/\Lambda'/\Lambda''$ is the set obtained by partitioning Λ' and each of its cosets by Λ'', where Λ'' is a sublattice of Λ'.

A commonly used transformation is obtained by stacking 2×2 blocks of the form

$$R = \begin{bmatrix} 1 & -1 \\ 1 & 1 \end{bmatrix}.$$

which represents a rotation of the lattice by 45° and a scaling by $\sqrt{2}$. The sublattice of Λ formed by this transformation is denoted $R\Lambda$.

Example 13.11 Figure 13.21(a) shows $\Lambda = \mathbb{Z}^2$. Figure 13.21(b) shows the cosets in the partition Λ/Λ', where $\Lambda = \mathbb{Z}^2$ and $\Lambda' = R\mathbb{Z}^2$, where the points in the cosets are designated as

$$\begin{array}{ll} \Lambda' = R\mathbb{Z}^2 & (\square) \\ \Lambda' + (1,0) & (\bullet). \end{array}$$

Figure 13.21(c) shows the cosets in the partition chain $\Lambda/\Lambda'/\Lambda''$, where $\Lambda'' = R^2\Lambda = 2\mathbb{Z}^2$, where the points in the four cosets are designated as

$$\begin{array}{ll} \Lambda'' = R^2\mathbb{Z}^2 & (\square) \\ \Lambda'' + (1,0) & (\bullet) \\ \Lambda'' + (0,1) & (\circ) \\ \Lambda'' + (1,1) & (\diamond). \end{array}$$

This four-way partition creates the same partition as that in Figure 13.9. □

The Lattice Code Idea

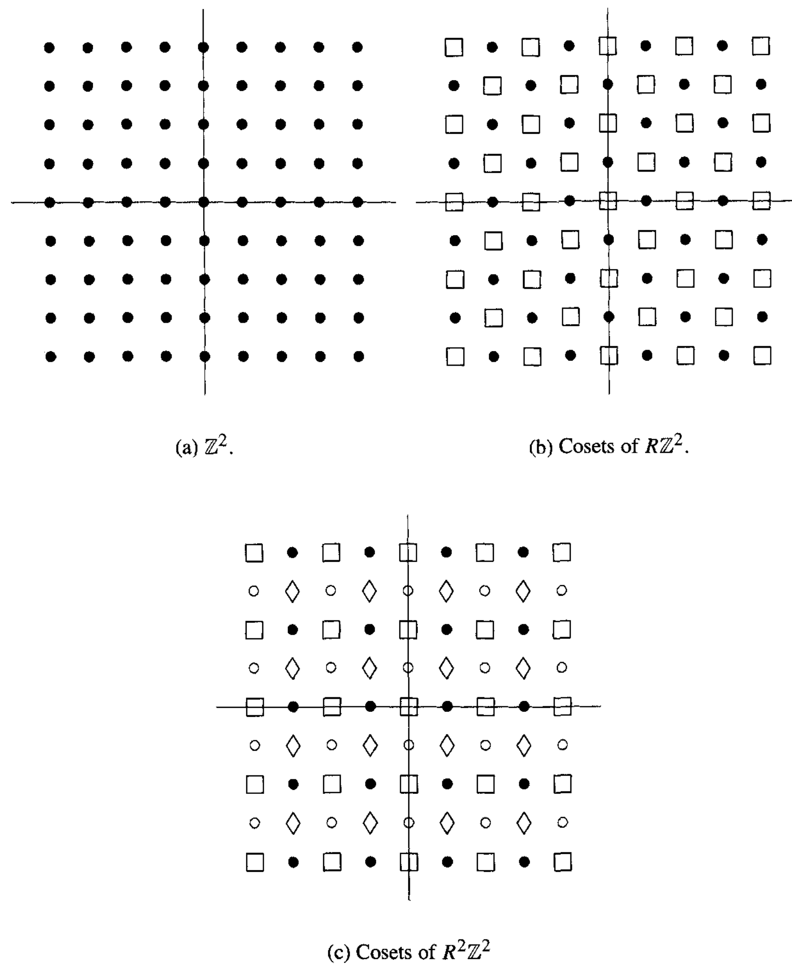
Figure 13.22 shows the idea behind TCM on lattice cosets. It is very similar to TCM in general: a set of coded bits selects a coset (as a subset of the constellation), and a set of uncoded bits selects a point within the subset. The performance of the code is determined by the minimum distance between points in the coset (corresponding to parallel transitions in the trellis) and the minimum distance between diverging paths in the trellis of the encoder.

Sources of Coding Gain in Lattice Codes

In addition to gains due to to the distances between sequences obtained using trellis coding, the very shape of the lattice constellation contributes gains. Two sources of coding gain can be attributed to the use of lattices. The first is referred to as the *lattice coding gain*. The lattice coding gain for an n-dimensional lattice Λ is a measure of how much more effectively points are packed into Λ compared to the lattice $\mathbb{Z}^n$. Let Λ be a lattice that is normalized so that the minimum distance between points is equal to 1, and let the fundamental volume of Λ be $V(\Lambda)$. A rectangular lattice, a multiple of $\mathbb{Z}^n$, with this volume would have a minimum distance of $V(\Lambda)^{1/n}$. There is thus a gain in energy equal to the ratio of the square of the minimum distance of the lattice (which is 1) divided by the square of the minimum distance an equal-volume rectangular lattice would have. This is called the lattice coding gain, and is denoted by γ_{cg}:

$$\gamma_{cg} = \frac{1}{V(\Lambda)^{2/n}}.$$

Table 13.6 lists coding gains for the lattices described in the previous section.

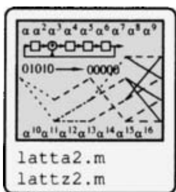


(a) $\mathbb{Z}^2$.

(b) Cosets of $R\mathbb{Z}^2$.

(c) Cosets of $R^2\mathbb{Z}^2$

Figure 13.21: $\mathbb{Z}^2$ and its partition chain and cosets.



The other source of coding gain provided by multidimensional constellations in general is called the *shape gain*, γ_s, which can be obtained by employing a nearly-circular boundary on the constellation, instead of natural rectangular or cross boundaries such as shown in Figure 13.2. Figure 13.23(a) shows circular boundaries for constellations obtained using $\mathbb{Z}^2$; part (b) shows similar boundaries for constellations built from the A_2 lattice. Table 13.7 shows the comparison of the average energies for these constellations with the average energies for the constellations from Table 13.1.(Possible minor reductions in energy could also be obtained by slightly shifting the constellations, but this was not done.) As the table shows, there is efficiency gained by employing a constellation spherical with a boundary instead of a square or cross boundary. (There is also somewhat higher complexity in the decoder.) Gains of about 0.18 dB are possible compared with the square constellation. (The gain is not as large for the 32-point constellation, since the cross form is already an

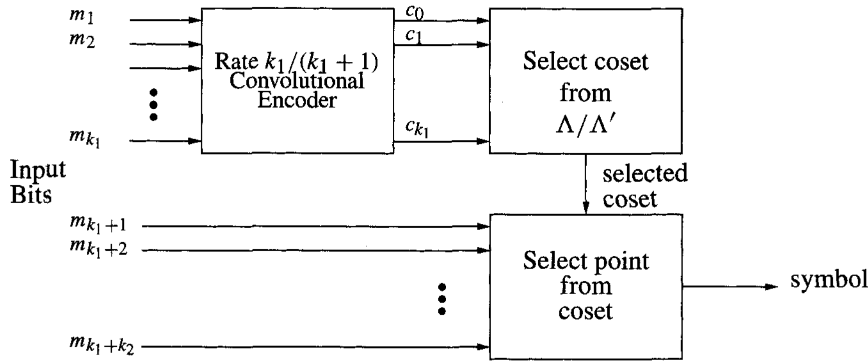


Figure 13.22: Block diagram for a trellis lattice coder.

Table 13.7: Comparison of Average Signal Energy for Circular Boundary $\mathbb{Z}^2$ and A_2 Constellations with Regular QAM

M	E_s, Rect. bdy., Rect. QAM (Table 13.1)	E_s, Circ. Bdy. Rect. QAM	Gain γ_s (dB)	E_s, Circ. Bdy. A_2 Lattice	Gain $\gamma_s\gamma_{cg}$ (dB)
64	10.5	10.19	0.13	8.85	0.74
128	20.5(CR)	20.41	0.02	17.68	0.64
256	42.5	40.79	0.18	35.26	0.81

approximation to the circular constellation.) Also shown in Table 13.7 are the average signal energies and gains when an A_2 lattice with circular boundary is employed. The shape gain is independent of the lattice gain, so the overall gain is additive (on a dB scale) $(\gamma)_{\text{dB}} = (\gamma_s)_{\text{dB}} + (\gamma_{cg})_{\text{dB}}$.

The shape gain is for an N-dimensional constellation is

$$\gamma_s = \frac{(N/2)\text{Average energy for circular 2-D lattice}}{\text{Average energy for square lattice}}.$$

Let M_c denote the size of the 2-dimensional signal constellation, and let $M_{c,N} = M_c^{N/2}$ denote the size of the N-dimensional constellation.

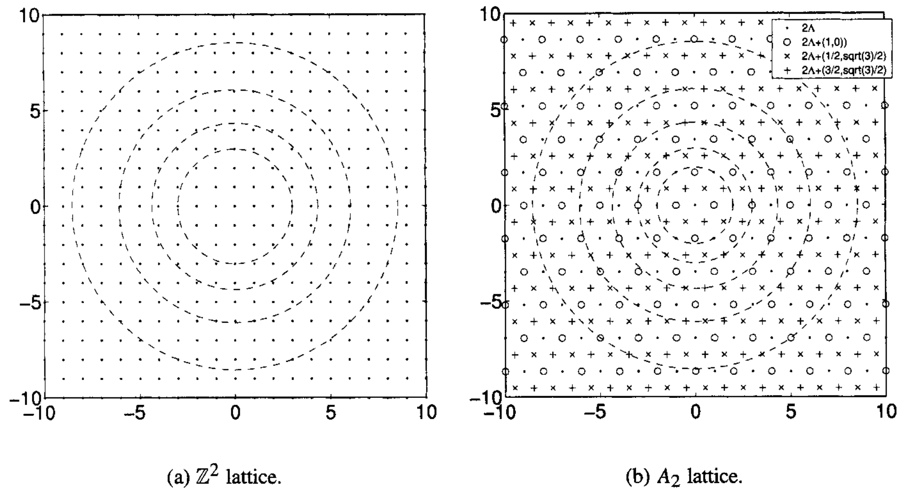
Assuming the minimum distance between points is $d_0 = 1$, the average energy for a circular $\mathbb{Z}^N$ (N even) constellation[2] $\mathcal{C}$ of is

$$E_{s,\text{circ}} = \frac{1}{M_{c,N}} \sum_{\mathbf{w}_i \in \mathcal{C}} \|\mathbf{w}_i\|^2 \approx \frac{1}{M_{c,N}} \int_{\mathcal{V}} \|\mathbf{v}\|^2 d\mathbf{v}.$$

where the summation is approximated by an integral and $\mathcal{V}$ is the volume of the spherical region containing the signal constellation. The number of points in the region can be approximated as

$$M_{c,N} \approx \int_{\mathcal{V}} d\mathbf{v}.$$

[2]The results here hold even for other lattices; the fundamental volume cancels out of the ratio.



(a) $\mathbb{Z}^2$ lattice.

(b) A_2 lattice.

Figure 13.23: Lattice and circular boundaries for 16, 32, 64, 128, and 256-point constellations.

The N-dimensional volume element increment in this integral can be expressed in "polar" form as [30, pp. 242, 246]

$$d\mathbf{v} = \frac{N\pi^{N/2} r^{N-1}}{(N/2)!} dr$$

so that, with the radius of $\mathcal{V}$ equal to ρ,

$$M_{c,N} \approx \int_0^\rho \frac{N\pi^{N/2} r^{N-1}}{(N/2)!} dr = \frac{\pi^{N/2}\rho^N}{(N/2)!}. \tag{13.9}$$

Using the same volume element increment,

$$\int_{\mathcal{V}} \|\mathbf{v}\|^2 \, d\mathbf{v} = \int_0^\rho r^2 \frac{N\pi^{N/2} r^{N-1}}{(N/2)!} dr = \frac{\pi^{N+2} N \rho^{N+2}}{(N+2)(N/2)!}. \tag{13.10}$$

The average energy for the square lattice can be found using (13.1) as

$$E_{s,\text{square}} = \frac{N}{2}\frac{M_c - 1}{6} = \frac{N}{2}\frac{(M_{c,N})^{N/2}}{6} \approx \frac{M_{c,N}^{N/2}}{6} \approx \left[\int_{\mathcal{V}} d\mathbf{v}\right]^{N/2}.$$

Using (13.9) and (13.10), the shape gain is

$$\gamma_s \approx \frac{N/2\left[\int_{\mathcal{V}} d\mathbf{v}\right]^{N/2}}{6\int_{\mathcal{V}} \|\mathbf{v}\|^2 \, d\mathbf{v} / \int_{\mathcal{V}} d\mathbf{v}} = \frac{\pi(1 + N/2)}{6[(N/2)!]^{2/N}}.$$

When $N = 2$, $\gamma = \pi/3 = 1.0472 = 0.2$ dB. This is apparent in Table 13.7 for $N = 256$. Stirling's approximation to $n!$ tells us[3]

$$n! \approx n^n e^{-n}\sqrt{2\pi n}.$$

Using Stirling's approximation, it can be shown that asymptotically, $\gamma_s \rightarrow \pi e/6 = 1.53$ dB.

[3]A very clear derivation of this appears in [136].

Some Good Lattice Codes

Table 13.8 [303] lists some good codes that have been developed in the literature.

Table 13.8: Some Good Multidimensional TCM Codes [303]

Partition Λ/Λ'	$d^2_{\min}$	Number of States	Asymptotic Coding Gain (dB)	N_D	Source
Four dimensions: Add 0.35 dB of shape gain					
$\mathbb{Z}^4/RD_4$	4	8	4.52	44	[365]
$\mathbb{Z}^4/RD_4$	4	16	4.52	12	[365]
$\mathbb{Z}^4/2\mathbb{Z}^4$	4	32	4.52	4	[365]
$\mathbb{Z}^4/2D_4$	5	64	6.28	72	[365]
$\mathbb{Z}^4/2D_4$	6	128	6.28	728	[347]
$D_4/2D_4$	6	16	4.77	152	[42]
$D_4/2D_4$	8	64	5.27	828	[42]
Eight dimensions: Add 0.76 dB of shape gain					
$\mathbb{Z}^8/E_8$	4	16	5.27	316	[365]
$\mathbb{Z}^8/E_8$	4	32	5.27	124	[365]
$\mathbb{Z}^8/E_8$	4	32	5.27	60	[365]
$\mathbb{Z}^8/RD_8$	4	128	5.27	28	[347]
RD_8/RE_8	8	32	6.02	> 500	[365]
RD_8/RE_8	8	64	6.02	316	[365]
RD_8/RE_8	8	128	6.02	124	[365]
E_8/RE_8	8	8	5.27	764	[42]
E_8/RE_8	8	16	5.27	316	[42]
E_8/RE_8	8	32	5.27	124	[42]
E_8/RE_8	8	64	5.27	60	[42]

13.8 Multidimensional TCM Example: The V.34 Modem Standard

In this section we discuss the error correction coding which is used for the V.34 modem standard. This modem is capable of transmitting up to 33.6 kb/second over the standard telephone system (on some lines). There are many technical aspects to this modem; space permits detailing only those related to the error correction coding. A survey and pointers to the literature appears in [98]. However, we briefly summarize some of the aspects of the modem:

- The modem is adaptive in the symbol rate it employs and the size of the constellation. It is capable of sending at symbol rates of 2400, 2743, 2800, 3000, 3200, or 3429 symbols/second. (These peculiar-looking choices are rational multiples of the basic rate of 2400 symbols/second.) The symbol rate is selected by a line probing sequence employed during initialization which determines the available bandwidth.
- The modem is capable of transmitting variable numbers of bits per symbol. At the highest rate, 8.4 bits/symbols are carried. Rate is established at link initialization, and rate adjustments can occur during data transmission.

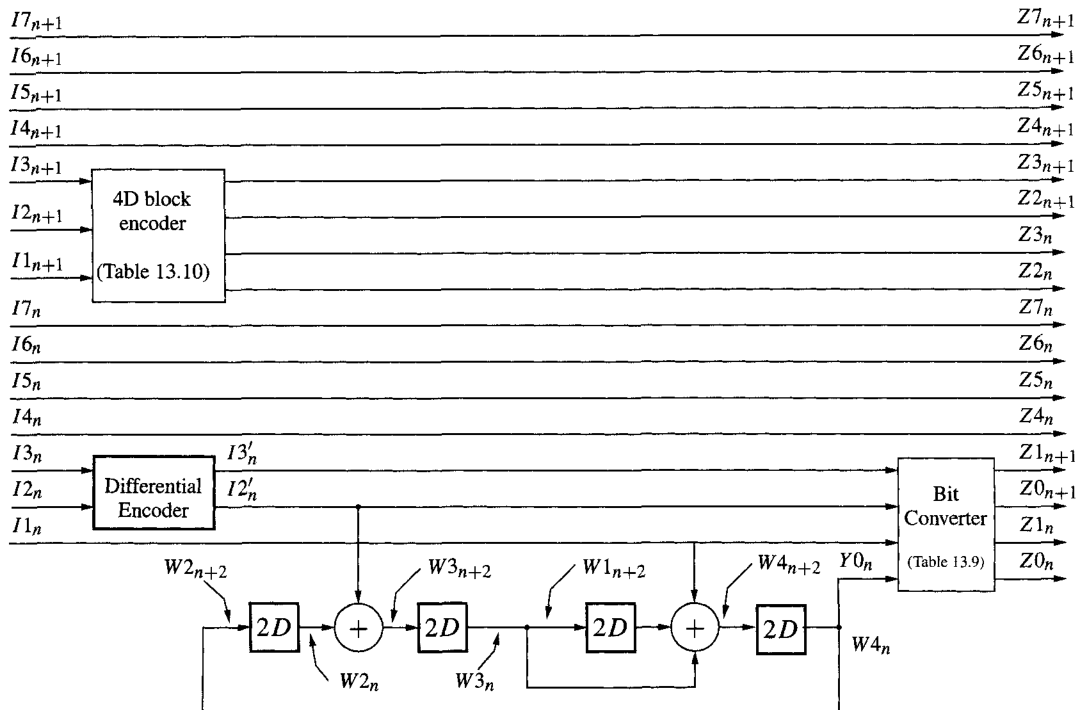


Figure 13.24: 16-state trellis encoder for use with V.34 standard [365].

- Adaptive precoding [339, 142, 99, 78] and decision feedback equalization is also employed to compensate for channel dispersion.
- Shaping via shell mapping is employed, which provides modest gains in addition to the coding gains.
- Adaptive trellis coding is employed. The main code is a 16-state four-dimensional trellis code (to be described below). This code provides 4.66 dB of gain and is rotationally invariant. However, two other trellis codes are also included: a 32-state four-dimensional code with 4.5 dB of gain and a 64-state four dimensional code providing 4.7 dB of gain. These codes are not described here.

Given the complexity of the modem, it is a technological marvel that they are so readily affordable and effective!

The trellis encoder for the modem is shown in Figure 13.24, with the corresponding trellis diagram in Figure 13.25.

The bit converter in the encoder supports the rotational invariance and is outlined in Table 13.9. The 4D block converter supports the shell shaping, controlling the selection of "inner" and "outer" constellation points. The operation is detailed below.

To transmit η information bits per signaling interval using $2N = 4$-dimensional modulation, $N = 2$ signaling intervals are required. For uncoded transmission, $2^{\eta N}$ points in the signal constellation are necessary. For coded transmission, $2^{\eta N+1}$ points are necessary. The V.34 standard carries $\eta = 7$ bits per symbol, so 2^{15} points in the signal constellation

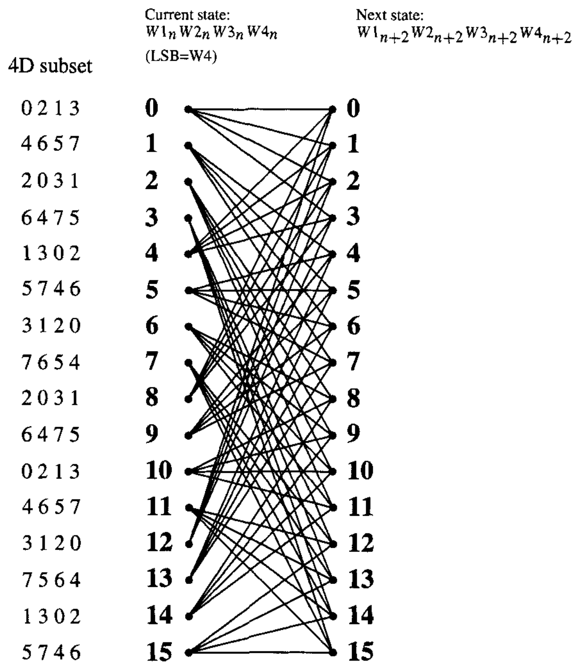


Figure 13.25: Trellis diagram of V.34 encoder [365].

are necessary. The V.34 standard does this using an interesting constellation. The four-dimensional constellation consists of the cross product of the 192-point constellation shown in Figure 13.26. The 192-point constellation contains a 128-point cross constellation in the *inner points*, plus an additional 64 *outer points*. The inner points can be used to transmit 7 uncoded bits per symbol. The outer points are selected as close to the origin as possible (outside of the inner constellation) to minimize energy. Each of the A, B, C and D sets has the same number of points. Also, a rotation of an outer point yields another outer point.

The 2^{15} points in the constellation are obtained by concatenating a pair of 192-point constellations (which would result in a 192^2 point constellation, but $192^2 > 2^{15}$), excluding those 4D points whose corresponding pair of two-dimensional points are both outer points. There are thus

$$192^2 - 64^2 = 2^{15}$$

points in this constellation. The inner points are used three-fourths of the time. By using the inner constellation more often, the average power is reduced compared to other (more straightforward) constellations. The average power of the constellation can be shown to be $28.0625d_0^2$, where d_0^2 is the minimum squared Euclidean distance (MSED) of the constellation. The peak power (which is also the peak power of the inner constellation) is $60.5d_0^2$.

The partition of the constellation proceeds through a sequence of steps which are illustrated in Figure 13.27.

- Each constituent two-dimensional rectangular lattice is partitioned into two *families*

Table 13.9: Bit Converter: Sublattice Partition of 4D Rectangular Lattice [365]

4D Sublattice (subset)	$Y0_n$	$I1_n$	$I2'_n$	$I3'_n$	4D Types	$Z0_n$	$Z1_n$	$Z0_{n+1}$	$Z1_{n+1}$
0	0	0	0	0	(A, A)	0	0	0	0
	0	0	0	1	(B, B)	0	1	0	1
1	0	0	1	0	(C, C)	1	0	1	0
	0	0	1	1	(D, D)	1	1	1	1
2	0	1	0	0	(A, B)	0	0	0	1
	0	1	0	1	(B, A)	0	1	0	0
3	0	1	1	0	(C, D)	1	0	1	1
	0	1	1	1	(D, C)	1	1	1	0
4	1	0	0	0	(A, C)	0	0	1	0
	1	0	0	1	(B, D)	0	1	1	1
5	1	0	1	0	(C, B)	1	0	0	1
	1	0	1	1	(D, A)	1	1	0	0
6	1	1	0	0	(A, D)	0	0	1	1
	1	1	0	1	(B, C)	0	1	1	0
7	1	1	1	0	(C, A)	1	0	0	0
	1	1	1	1	(D, B)	1	1	0	1

Table 13.10: 4D Block Encoder [365]

$I1_{n+1}$	$I2_{n+1}$	$I3_{n+1}$	$Z2_n$	$Z3_n$	$Z2_{n+1}$	$Z3_{n+1}$
0	0	0	0	0	0	0
0	0	1	0	0	0	1
0	1	0	0	0	1	0
0	1	1	0	1	1	0
1	0	0	1	0	0	0
1	0	1	1	0	0	1
1	1	0	0	1	0	0
1	1	1	0	1	0	1

$A \cup B$ and $C \cup D$, where the sublattice A is composed of those points in the constellation of Figure 13.26 labeled with the letter 'a,' and similarly for B, C, and D. The MSED between these two families is $2d_0^2$.

- The two-dimensional families are further partitioned into four sublattices A, B, C, and D, with MSED $4d_0^2$. The sublattices have the property that under a 90° counter-clockwise rotation, sublattice A rotates to sublattice D. Collectively the sublattices rotate as

$$A \to D \to B \to C \to A. \tag{13.11}$$

- Sixteen four-dimensional *types* are defined by concatenating all pairs of two-dimensional sublattices. These types are $(A, A), (A, B), \ldots, (D, D)$. The MSED between the types is $4d_0^2$, the same as for the sublattices, since two of the same sublattices can be used in a type.

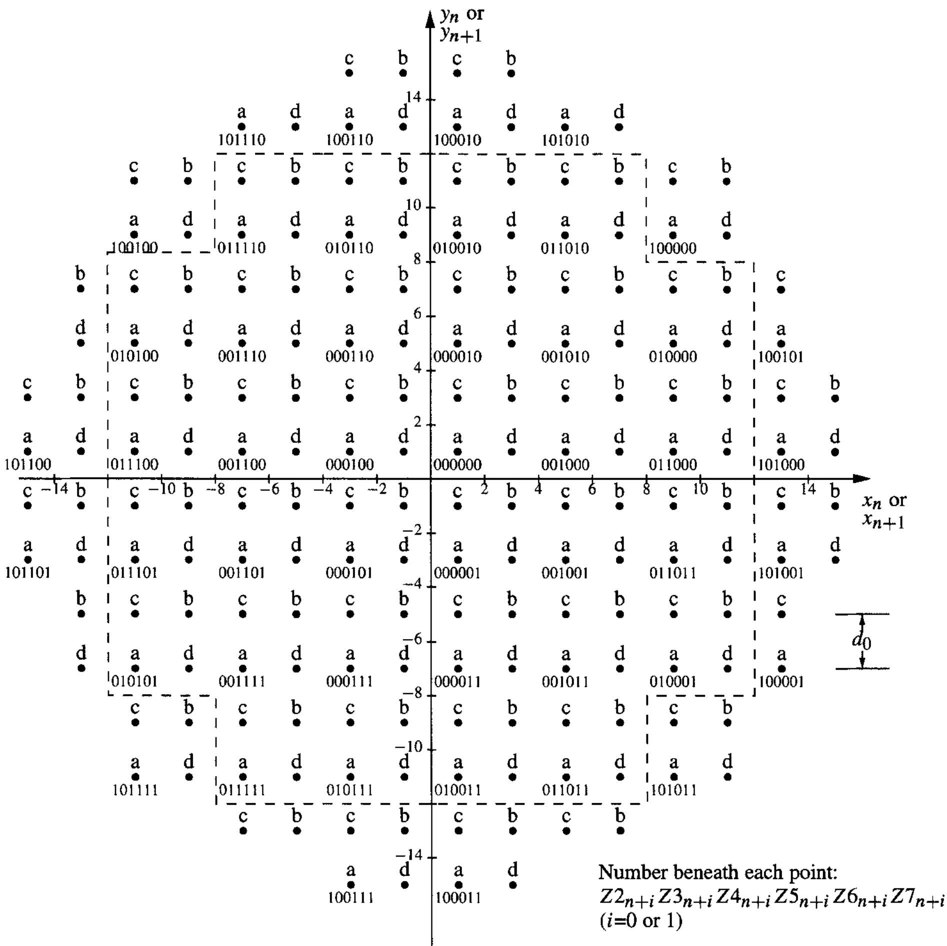


Figure 13.26: The 192-point two-dimensional constellation employed in the V.34 standard.

- The 16 types are grouped into eight four-dimensional *sublattices*, denoted by 0, $1, \ldots, 7$, as denoted in Figure 13.27 and Table 13.9. The MSED between these sublattices is still $4d_0^2$, which may be verified as follows. The two first constituent two-dimensional sublattices in each four-dimensional sublattice are in $A \cup B$ or $C \cup D$, and likewise for the second two two-dimensional sublattices. Each of these thus have the minimum squared distance of the two-dimensional families, $2d_0^2$. Since there are two independent two-dimensional components, the MSED is $2d_0^2 + 2d_0^2$. It can be verified that the four-dimensional sublattices are invariant under 180° rotation.
- The eight sublattices are further grouped into two four-dimensional *families* $\bigcup_{i=0}^{3} i$ and $\bigcup_{i=4}^{7} i$, with MSED $2d_0^2$.

Combining the trellis of Figure 13.25 with the decomposition of Figure 13.27, the assignments of Table 13.9, satisfy the following requirements:

- The 4D sublattices associated with a transition from a state or to a state are all different,

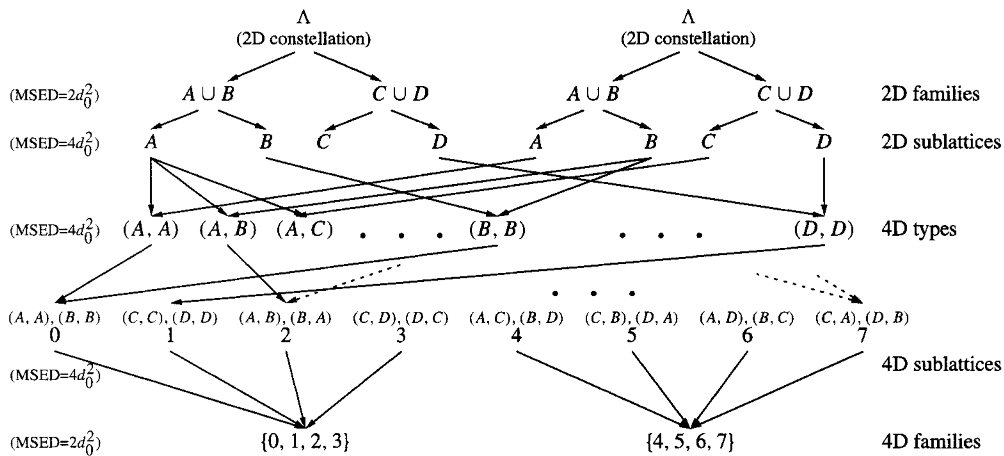


Figure 13.27: Partition steps for the V.34 signal constellation.

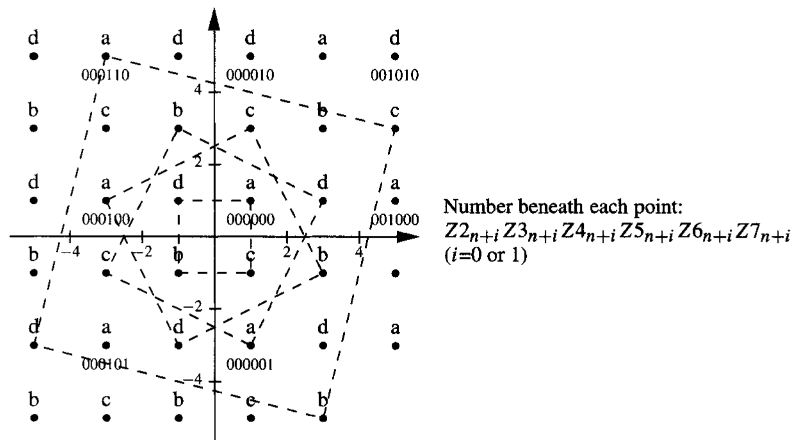


Figure 13.28: Orbits of some of the points under rotation: all points in an orbit are assigned the same bit pattern.

but all belong to the same family $\bigcup_{i=0}^{3} i$ or $\bigcup_{i=4}^{7} i$.

- The MSED between any two allowed sequences in the trellis is greater than $4d_0^2$. In combination with the first requirement, this means that the free distance of the code is established by the MSED of each 4D sublattice, $4d_0^2$. Compared to uncoded 128-point cross signal constellation with average energy $20.5d_0^2$, the asymptotic coding gain is

$$10\log_{10}\left(\frac{4d_0^2}{28.0625d_0^2}/\frac{d_0^2}{20.5d_0^2}\right) = 4.66\text{d}B.$$

- The assignment makes a rotationally invariant code (using just a linear trellis encoder). For a valid transition in the trellis from a state i to a state j, let X be the four-dimensional subset associated with the transition. Let Y be the four-dimensional

subset that is obtained by rotating X by 90°. Then Y is associated with the valid transition from the state $F(i)$ to the next state $F(j)$ for some function F. For this particular code, the function is:

$$F : W1_pW2_pW3_pW4_p \mapsto \overline{W1_p}W2_p\overline{W3_p}W4_p,$$

where the overbar denotes binary complementation. In combination with the fact that the sublattices are invariant with respect to 180° rotations, this makes the code rotationally invariant with respect to multiples of 90° rotations.

The encoding operation is now summarized. Fourteen bits, representing the information for two symbol periods, are presented to the encoder. These fourteen input bits are denoted by $(I1_n, \ldots, I7_n)$ and $(I1_{n+1}, \ldots, I7_{n+1})$, where the subscript n denotes bits or symbols associated with even-numbered symbols and $n+1$ denotes bits or symbols associated with odd-numbered symbols. We refer to the pair of symbol intervals at time n and time $n+1$ as a coding **epoch**. From these fourteen bits, two symbols in the coding epoch are selected according to the following steps:

- The encoded bits $Y0_n$, $I1_n$ and $I2'_n$ select one of the eight four-dimensional sublattices. Then the nontrellis-encoded information bit $I3'_n$ selects one of the two four-dimensional types within the sublattice. This is done in such a way that the system is transparent to phase ambiguities of multiples of 90°. To see this, consider a bit pattern $Y0_nI1_nI2'_nI3'_n$ and let X denote the associated 4D type from Table 13.9. Let $S2_1S3_1$, $S2_2S3_2$, $S2_2S3_2$ denote the bit pairs obtained when the bit pair $I2'_nI3'_n$ is advanced in the circular sequence

$$00 \to 11 \to 10 \to 00. \qquad (13.12)$$

  Let X_1, X_2, and X_3 denote the types obtained by rotating X *counterclockwise* by successive multiples of 90°. Then the 4D types associated with the bit patterns $Y0_nI1_nS2_1S3_1$, $Y0_nI1_nS2_1S3_1$, and $Y0_nI1_nS2_1S3_1$ are X_1, X_2, and X_3, respectively.

  Example 13.12 Let $Y0_nI1_nI2'_nI3'_n = 0010$. Then (from Table 13.9) the 4D type transmitted is $X = (C, C)$. When this is rotated 90°, the type is (see (13.11)) (A, A), corresponding to a transmitted sequence $Y0_nI1_nI2'_nI3'_n = 0000$. Note that the last two bits correspond to the succession of (13.12). □

  To obtain rotational invariance, the bits $I2_nI3'_n$ are obtained as the output of a differential encoder, just as for the V.32 and V.33 standards presented in Section 13.6. The pair $(I3_n, I2_n)$ is converted to a number modulo 4 ($I2_n$ is the LSB), and the differential representation (13.6) is employed.

- The 4D block encoder serves the purpose of selecting points in the inner and outer constellation, ensuring that the outer constellation is not used for both the symbols in the coding epoch. The 4D block encoder takes the input bits $I1_{n+1}$, $I2_{n+1}$ and $I3_{n+1}$ and generates two pairs of output bits $(Z2_n, Z3_n)$ and $(Z2_{n+1}, Z3_{n+1})$ according to Table 13.10. The outputs (00) and (01) correspond to points in the inner constellation and the output 10 corresponds to points in the outer constellation. (See the labeling in Figure 13.26.) Each bit pair can be 00, 01, or 10, but they cannot both be 10.

- There are 16 points in the outer group of a 2D lattice (such as A) or in either half of the inner part of a 2D subset. These sixteen points are indexed by the bits $Z4_pZ5_pZ6_pZ7_p$, where $p = n$ or $n + 1$.
- The bits $Z2_pZ3_pZ4_pZ5_pZ6_pZ7_p$ ($p = n$ or $n + 1$) select from a set of four points in the signal constellation. To ensure rotational invariance, the four rotations of a point are all assigned the same set of bits. Figure 13.28 shows the "orbits" of some of the points under rotation. The a, b, c and d points in the orbit are all assigned the same label $Z2_pZ3_pZ4_pZ5_pZ6_pZ7_p$; then one of the points in the orbit is selected by $Z1_pZ0_p$. Since the bits $Z2_pZ3_pZ4_pZ5_pZ6_pZ7_p$ are rotationally invariant by labeling, and the bits $Z1_pZ0_p$ are invariant by differential encoding, the overall code is rotationally invariant.

The bits $Z0_pZ1_p \ldots Z7_p$ ($p = n$ or $n + 1$) are used to select two points in the signal constellation, corresponding to the two symbols sent in the coding epoch.

Programming Laboratory 11: Trellis-Coded Modulation Encoding and Decoding

Objective

In this laboratory, you will create an encoder and decoder for a particular TCM code.

Background

Reading: Section 13.3.

Programming Part

1) Construct an encoder to implement the trellis code for a four-state trellis with an 8-PSK signal constellation. Verify that it works as expected.

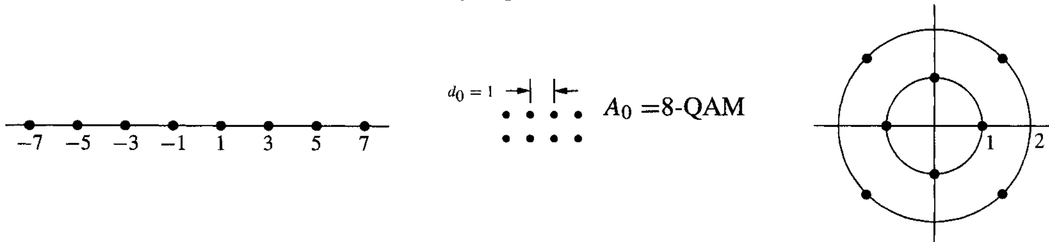
2) Construct a Viterbi decoder for the trellis code. Verify that it works as expected.

3) Make a plot of $P(e)$ as a function of SNR for the code. Compare $P(e)$ with $P(e)$ for uncoded 4-PSK. Plot the theoretical $P(e)$. Is the theoretical coding gain actually achieved?

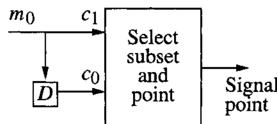
13.9 Exercises

13.1 Verify the energy per symbol E_s and the energy per bit E_b for BPSK signaling from Table 13.1. Repeat for 16-QAM and 32-cross signaling.

13.2 For each of the following signal constellations, determine a signal partition. Compute the minimum distance between signal points at each level of the tree.



13.3 [373] The simple TCM encoder shown here



is used with the 8-AM signal constellation shown in exercise 2.

(a) Determine the trellis for the encoder.

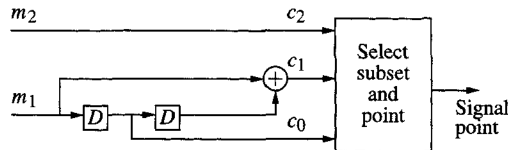
(b) Determine a signal partitioning scheme which transmits 2 bits/symbol.

(c) Determine the squared minimum free distance for the coded system.

(d) Compute the asymptotic coding gain in dB for this system compared with an uncoded 4-AM system.

(e) Determine the output transition matrix $B(x)$ for this code and determine the components $D(x)$, $P(x)$, and $M(x)$.

13.4 For the encoder shown here



employed with an 8-PSK constellation partitioned as in Figure 13.4:

(a) Draw the trellis for the convolutional encoder.

(b) Draw the trellis for the trellis coder, labeling the state transitions with the subsets from the constellation.

(c) Determine the minimum free distance between paths which deviate from the all-zero path, both for non-parallel paths and for parallel paths. Determine the minimum free distance for the code. Assume $E_s = 1$.

(d) Determine the coding gain of the system, compared with 4-PSK transmission.

13.5 The sequence of data $(m_{2,t}, m_{1,t})$ consisting of the pairs $(0,0)$, $(1,0)$, $(1,1)$, $(1,1)$, $(0,0)$, $(0,1)$, $(0,1)$ is applied to the encoder of Figure 13.16.

(a) Determine the differentially encoded sequence $(\tilde{m}_{2,t}, \tilde{m}_{1,t})$. Assume that the differential encoder starts with previous input 0.

(b) The sequence of inputs $(m_{4,t}, m_{3,t}, m_{2,t}, m_{1,t})$ consisting of the 4-tuples $(0,1,0,0)$, $(1,1,1,0)$, $(0,1,1,1)$, $(1,0,1,1)$, $(0,0,0,0)$, $(1,1,0,1)$, $(1,0,0,1)$ is presented to the encoder of Figure 13.16. Determine the sequence of output subsets and plot the corresponding path on the trellis, as in Example 13.4. Assume the encoder starts in state 0.

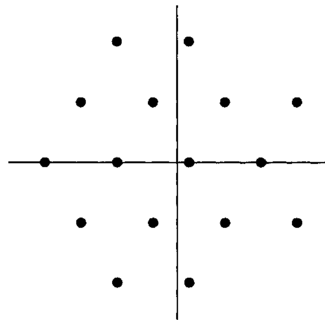
(c) Now take this sequence of output signals and rotate them by $\pi/2$. Determine the sequence of received signal points.

(d) Determine the state sequence decoded at the receiver. Assume the decoder is able to determine that state 7 is the starting state.

(e) Determine the sequence of input bits corresponding to this decoded sequence.

(f) Run the input bits through a differential decoder and verify that the decoded bits match the original sequence of transmitted bits.

13.6 The signal constellation below has 16 points, with the points on a hexagonal lattice with minimum distance between points equal to 1. Adjust the center location and the lattice points so that the constellation has minimum average signal energy. Compute the average signal energy E_s. Compare this average signal energy with a 16-QAM constellation having equal minimum distance. How much energy advantage is there for the hexagonal lattice (in dB)? What practical disadvantages might the hexagonal constellation have?



13.10 References

The idea of combining coding and modulation can be traced at least back to 1974 [225]. It was developed into a mature technique by Ungerböck [344, 346, 347, 345]. Important theoretical foundations were later laid by Forney [100, 91, 92, 93]. Rotationally invariant codes are described in [363, 364]. Additional work in this area appears in [263, 264, 366]. It was also mentioned in [347]. See also rotational invariance appears in [340, 18]

Our bound on the performance in Section 13.4 follows [303] very closely. See also [383]. A random coding bound is also presented there. Other analyses of the code performance appear in [29]. A thorough treatment of TCM appears in [204]. Theoretical foundations of coset codes appear in [91, 92].

TCM using lattices is described in [42]. An example of multidimensional TCM using an eight dimensional lattice is described in [41]. Issues related to packing points in higher dimensional spaces and codes on lattices are addressed in [54, 55, 53, 58, 57]. The definitive reference related to sphere packings is [56]. The sphere-packing advantage as applied to data compression (vector quantization) is described in [210]. A trellis code in six dimensions based on the E_6 lattice is described in [242]. Lattices also have use in some computer algebra and cryptographic systems; in this context an important problem is finding the shortest vector in the lattice. For discussions and references, see [360, Chapter 16]. A concise but effective summary of lattice coding is presented in [5]. Extensive design results appear in [264, 265, 204].

Part IV

Iteratively Decoded Codes

Chapter 14

Turbo Codes

> "Tell me how you decode and I'll be able to understand the code." When you have no particular gift for algebra, ... then think about the decoding side before the encoding one. Indeed, for those who are more comfortable with physics than with mathematics, decoding algorithms are more accessible than coding constructions, and help to understand them.
> — Claude Berrou [26]

14.1 Introduction

Shannon's channel coding theorem implies strong coding behavior for random codes as the code block length increases, but increasing block length typically implies an exponentially increasing decoding complexity. Sequences of codes with sufficient structure to be easily decoded as the length increases were, until fairly recently, not sufficiently strong to approach the limits implied by Shannon's theorem. However, in 1993, an approach to error correction coding was introduced which provided for very long codewords with only (relatively) modest decoding complexity. These codes were termed *turbo codes* by their inventors [27, 28]. They have also been termed parallel concatenated codes [146, 303]. Because the decoding complexity is relatively small for the dimension of the code, very long codes are possible, so that the bounds of Shannon's channel coding theorem become, for all practical purposes, achievable. Codes which can operate within a fraction of a dB of channel capacity are now possible. Since their announcement, turbo codes have generated considerable research enthusiasm leading to a variety of variations, such as turbo decoding of block codes and combined turbo decoding and equalization, which are introduced in this chapter. Actually, the turbo coding idea goes back somewhat earlier than the original turbo code announcement; the work of [207] and [208] also present the idea of parallel concatenated coding and iterative decoding algorithms.

The turbo code encoder consists of two (or more) systematic block codes which share message data via interleavers. In its most conventional realization, the codes are obtained from recursive systematic convolutional (RSC) codes — but other codes can be used as well. A key development in turbo codes is the *iterative* decoding algorithm. In the iterative decoding algorithm, decoders for each constituent encoder take turns operating on the received data. Each decoder produces an estimate of the probabilities of the transmitted symbols. The decoders are thus *soft* output decoders. Probabilities of the symbols from one encoder known as *extrinsic probabilities* are passed to the other decoder (in the symbol order appropriate for the encoder), where they are used as *prior* probabilities for the other decoder. The decoder thus passes probabilities back and forth between the decoders, with each decoder combining the evidence it receives from the incoming prior probabilities with the parity information provided by the code. After some number of iterations, the decoder converges to an estimate of the transmitted codeword. Since the output of one decoder is fed to the input of the next decoder, the decoding algorithm is called a turbo decoder: it is

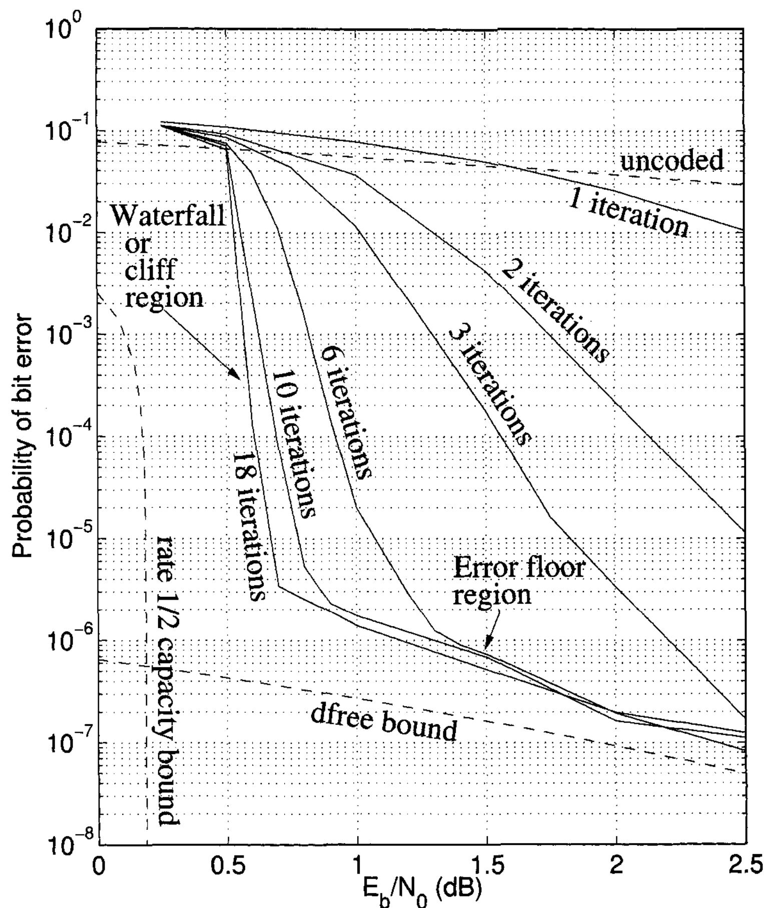


Figure 14.1: Decoding results for a (37,21,65536) code.

reminiscent of turbo charging an automobile engine using engine-heated air at the air intake. Thus it is not really the *code* which is "turbo," but rather the decoding algorithm which is "turbo."

As an example of what turbo codes can achieve, Figure 14.1 shows the performance of a turbo code employing two recursive systematic convolutional (RSC) encoders with parity-producing transfer functions

$$G(x) = \frac{1 + x^4}{1 + x + x^2 + x^3 + x^4} \tag{14.1}$$

in a rate $R = 1/2$ turbo code (i.e., it is punctured) with block length $N = 65536$ and a random interleaver. (The numerator and denominator polynomials are represented using the octal numbers 21 = 10 001 and 37 = 11 111, respectively, so this code is sometimes referred to as a (37, 21, 65536) code.) The decoding performance for up to 18 decoding iterations is shown. Beyond 18 iterations, little additional coding gain is achieved. (These results were obtained by counting up to 100 bits in error.) We note that with 18 iterations of decoding,

performance within about 0.5 dB of the capacity limit is achieved by this code, at least for SNRs up to about 0.6 dB. However, an interesting phenomenon is observed at higher SNRs: while the decoding is still good, it fails to improve as dramatically as a function of SNR. At a certain SNR, the error curves nearly level off, so the improvement with increasing SNR is very modest. This phenomenon is referred to as the *error floor* and is discussed in Section 14.4. Briefly, it is due to the presence of low-weight codewords in the code. A bound due to the free distance of the convolutional coders is also shown in the plot, which indicates the slope of the error floor. The portion of the plot where the error plot drops steeply down as a function of SNR is referred to as the waterfall or cliff region.

In this chapter we discuss the structure of the encoder, present various algorithms for decoding, and provide some indication of the structure of the codes that leads to their good performance and the error floor. We also introduce the idea of turbo equalization and the concept of EXIT analysis for the study of the convergence of the decoding algorithm.

14.2 Encoding Parallel Concatenated Codes

The conventional arrangement for the (unpunctured) turbo encoder is shown in Figure 14.2. It consists of two transfer functions representing the non-systematic components of recursive systematic convolutional (RSC) encoders called the *constituent encoders*, and an *interleaver*, which permutes the input symbols prior to input to the second constituent encoder. (It is also possible to use more than two encoder blocks [70], but the principles remain the same, so for the sake of specific notation we restrict attention here to only two constituent encoders.) As discussed in chapter 12, systematic convolutional codes typically work best when the encoder is a feedback (IIR) encoder, so the transfer function of each convolutional encoder is the rational function

$$G(x) = \frac{h(x)}{g(x)}.$$

Strictly speaking, there is no reason that both constituent transfer functions must be the same. However, it is conventional to use the same transfer function in each branch; research to date has not provided any reason to do otherwise.

A block of input symbols $\mathbf{x} = \{x_0, x_1, \ldots, x_{N-1}\}$ is presented to the encoder, where each x_i is in some alphabet $\mathcal{A}$ with $|\mathcal{A}|$ elements in it. These input symbols may include an appended zero-state forcing sequence, as in Figure 14.6, or it may simply be a message sequence, $\mathbf{x} = \mathbf{m} = \{m_0, m_1, \ldots, m_{N-1}\}$. In the encoder, the input sequence $\mathbf{x}$ is used three ways. First, it is copied directly to the output to produce the systematic output sequence $v_t^{(0)} = x_t$, $t = 0, 1, \ldots, N-1$. Second, the input sequence runs through the first RSC encoder with transfer function $G(x)$, resulting in a parity sequence $\{v_0^{(1)}, v_2^{(1)}, \ldots, v_{N-1}^{(1)}\}$. The combination of the sequence $\{v_t^{(0)}\}$ and the sequence $\{v_t^{(1)}\}$ results in a rate $R = 1/2$ (neglecting the length of the zero-forcing tail, if any) systematically encoded convolutionally encoded sequence. Third, the sequence $\mathbf{x}$ is also passed through an *interleaver* or *permuter* of length N, denoted by Π, which produces the permuted output sequence $\mathbf{x}' = \Pi(\mathbf{x})$. The sequence $\mathbf{x}'$ is passed through another convolutional encoder with transfer function $G(x)$ which produces the output sequence $\mathbf{v}^{(2)} = \{v_0^{(2)}, v_1^{(2)}, \ldots, v_{N-1}^{(2)}\}$. The three output sequences are multiplexed together to form the output sequence

$$\mathbf{v} = \{(v_0^{(0)}, v_0^{(1)}, v_0^{(2)}), (v_1^{(0)}, v_1^{(1)}, v_1^{(2)}), \ldots, (v_{N-1}^{(0)}, v_{N-1}^{(1)}, v_{N-1}^{(2)})\},$$

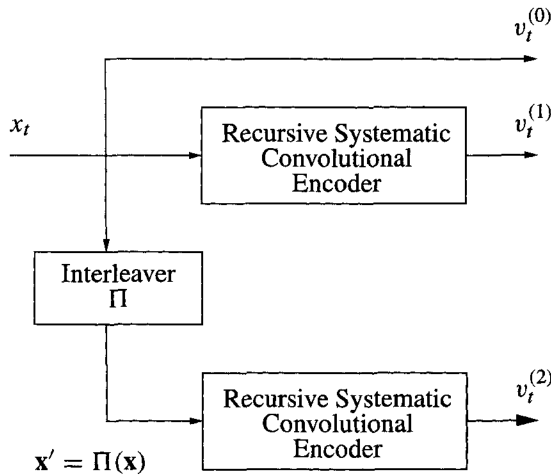


Figure 14.2: Block diagram of a turbo encoder.

resulting in an overall rate $R = 1/3$ linear, systematic, block code. The code has two sets of parity information, $\mathbf{v}^{(1)}$ and $\mathbf{v}^{(2)}$ which, because of the interleaving, are fairly independent. In an ideal setting, the sets of parity bits would be exactly independent.

Frequently, in order to obtain higher rates, the filter outputs are punctured before multiplexing, as shown in Figure 14.3. Puncturing operates only on the parity sequences — the systematic bits are not punctured. The puncturing is frequently represented by a matrix, such as

$$P = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

The first column indicates which bits are output at the even output instants and the second column indicates which bits are output at the odd output instants. For example, this puncture matrix alternately selects the outputs of the encoding filters.

Example 14.1 Consider the transfer function $G(x) = \frac{1}{1+x^2}$ incorporated in the turbo encoder of Figure 14.4(a), with the trellis stage shown in Figure 14.4(b). Let the interleaver be described by

$$\Pi = \{8, 3, 7, 6, 9, 0, 2, 5, 1, 4\}.$$

Then, for example, $x'_0 = x_8$, $x'_1 = x_3$, etc. Let the input sequence be

$$\mathbf{x} = [1, 1, 0, 0, 1, 0, 1, 0, 1, 1] = \mathbf{v}^{(0)}.$$

Then the output of the first encoder is

$$\mathbf{v}^{(1)} = [\underline{1}, 1, \underline{1}, 1, \underline{0}, 1, \underline{1}, 1, \underline{0}, 0], \tag{14.2}$$

and the first encoder happens to be left in state 0 at the end of this sequence. The interleaved bit sequence is

$$\mathbf{x}' = [1, 0, 0, 1, 1, 1, 0, 0, 1, 1]$$

and the output of the second encoder is

$$\mathbf{v}^{(2)} = [1, \underline{0}, 1, \underline{1}, 0, \underline{0}, 0, \underline{0}, 1, \underline{1}]; \tag{14.3}$$

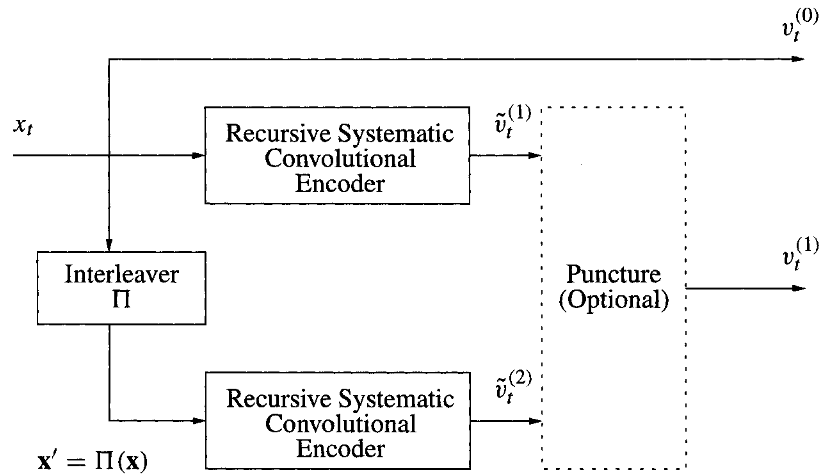


Figure 14.3: Block diagram of a turbo encoder with puncturing.

the second encoder is left in state 3. When the three bit streams are multiplexed together, the bit stream is

$$\mathbf{v} = [1, 1, 1,\ 1, 1, 0,\ 0, 1, 1,\ 0, 1, 1,\ 1, 0, 0,\ 0, 1, 0,\ 1, 1, 0,\ 0, 1, 0,\ 1, 0, 1,\ 1, 0, 1].$$

If the encoded bits are punctured, the underlined parity bits of (14.2) and (14.3) are retained. The resulting rate $R = 1/2$ encoded bit sequence is

$$\mathbf{v} = [1, 1,\ 1, 0,\ 0, 1,\ 0, 1,\ 1, 0,\ 0, 0,\ 1, 1,\ 0, 0,\ 1, 0,\ 1, 1].$$

□

It should be pointed out that there are also *serially* concatenated codes with iterative decoders. One such code is the repeat accumulate (RA) code, which is introduced in Section 15.14.

14.3 Turbo Decoding Algorithms

The multiplexed and encoded data $\mathbf{v}$ are modulated and transmitted through a channel, whose output is the received vector $\mathbf{r}$. The received data vector $\mathbf{r}$ is demultiplexed into the vectors $\mathbf{r}^{(0)}$ (corresponding to $\mathbf{v}^{(0)}$), $\mathbf{r}^{(1)}$ (corresponding to $\mathbf{v}^{(1)}$), and $\mathbf{r}^{(2)}$ (corresponding to $\mathbf{v}^{(2)}$).

The general operation of the turbo decoding algorithm is as follows, as summarized in Figure 14.5. The data $(\mathbf{r}^{(0)}, \mathbf{r}^{(1)})$ associated with the first encoder are fed to Decoder I. This decoder initially uses uniform priors on the transmitted bits and produces probabilities of the bits conditioned on the observed data. These probabilities are called the extrinsic probabilities, as described below. The output probabilities of Decoder I are interleaved and passed to Decoder II, where they are used as "prior" probabilities in the decoder, along with the data associated with the second encoder, which is $\mathbf{r}^{(0)}$ (interleaved) and $\mathbf{r}^{(2)}$. The extrinsic output probabilities of Decoder II are deinterleaved and passed back to become prior probabilities to Decoder I. The process of passing probability information back and

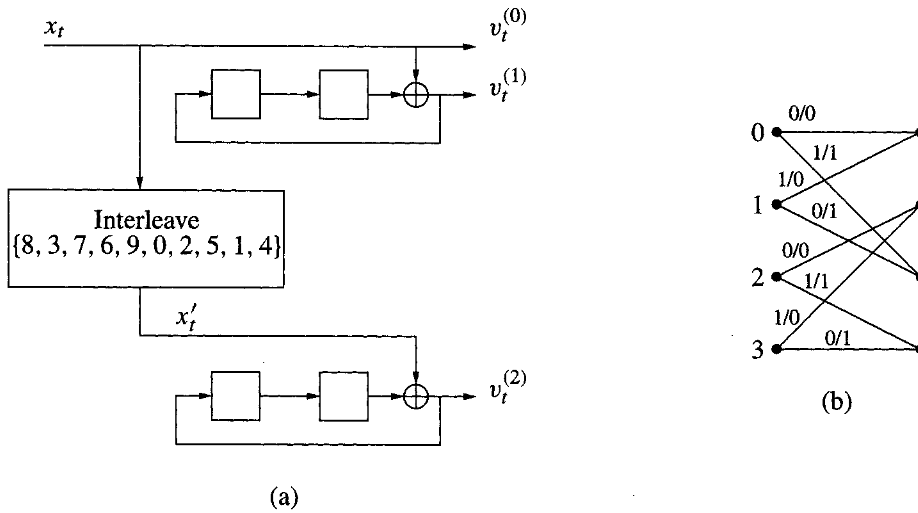


Figure 14.4: Example turbo encoder with $G(x) = 1/1 + x^2$.

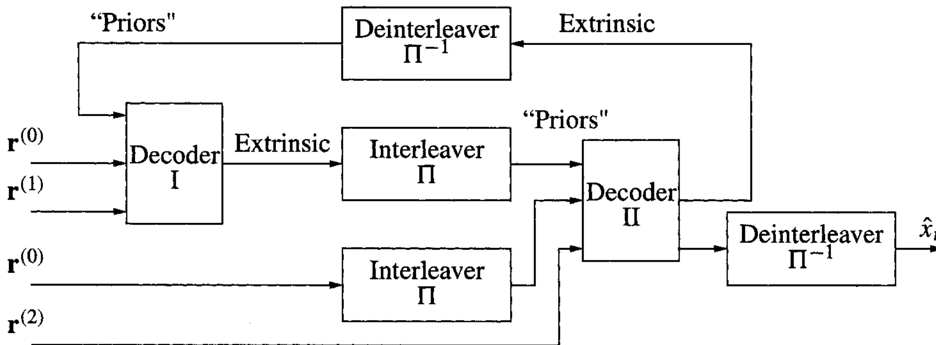


Figure 14.5: Block diagram of a turbo decoder.

forth continues until the decoder determines (somehow) that the process has converged, or until some maximum number of iterations is reached.

The heart of the decoding algorithm is a soft-decision decoding algorithm which provides estimates of the posterior probabilities of each input bit. The algorithm most commonly used for the soft-decision decoding algorithm is the MAP algorithm, also commonly known as the BCJR algorithm. In Section 14.3.1 we describe this algorithm for the case of a general convolutional code. Then in Section 14.3.10 we describe modifications to the algorithm that apply to systematic codes, which sets the stage for the iterative turbo decoding algorithm. The MAP algorithm can also be expressed in a log likelihood setting, as described in Section 14.3.12. A lower-complexity implementation of the MAP algorithm is discussed in Section 14.3.15. Another decoding algorithm, called the soft-output Viterbi algorithm (SOVA) is described in Section 14.3.17, which has even lower computational complexity (but slightly worse performance).

14.3.1 The MAP Decoding Algorithm

The maximum a posteriori (MAP) decoding algorithm suitable for estimating bit and/or state probabilities for a finite-state Markov system is frequently referred to as the BCJR algorithm, after Bahl, Cock, Jelenik, and Raviv who proposed it originally in [11]. The BCJR algorithm computes the posterior probability of symbols from Markov sources transmitted through discrete memoryless channels. Since the output of a convolutional coder passed through a memoryless channel (such as an AWGN channel or a BSC) forms a Markov source, the BCJR algorithm can be used for maximum *a posteriori* probability decoding of convolutional codes. In many respects, the BCJR algorithm is similar to the Viterbi algorithm. However, the Viterbi algorithm computes *hard decisions* — even if it is employing soft branch metrics — since a single path is selected to each state at each time. This result in an overall decision on an entire sequence of bits (or codeword) at the end of the algorithm, and there is no way of determining the reliability of the decoder decisions on the individual bits. Furthermore, the branch metric is based upon log likelihood values; no prior information is incorporated into the decoding process. The BCJR algorithm, on the other hand, computes soft outputs in the form of posterior probabilities for each of the message bits. While the Viterbi algorithm produces the maximum likelihood *message sequence* (or codeword), given the observed data, the BCJR algorithm produces the *a posteriori* most likely sequence of message *bits*, given the observed data. (Interestingly, the sequence of bits produced by the MAP algorithm may not actually correspond to a continuous path through the trellis.) In terms of actual performance on convolutional codes, the distinction between the Viterbi algorithm and the BCJR algorithm is frequently insignificant, since the performance of the BCJR algorithm is usually comparable to that of the Viterbi algorithm and any incremental improvement offered by the BCJR algorithm is offset by its higher computational complexity. However, there are instances where the probabilities produced by the BCJR are important. For example, the probabilities can be used to estimate the *reliability* of a decisions about the bits. This capability is exploited in the decoding algorithms of turbo codes. As a result, the BCJR algorithm lies at the heart of most turbo decoding algorithms.

We first express the decoding algorithm in terms of probabilities then, in Section 14.3.12, we present analogous results for likelihood ratio decoding. The probabilistic description is more general, being applicable to the case of nonbinary alphabets. However, it also requires particular care with normalization. Furthermore, there are approximations that can be made in association with the likelihood ratio formulation that can reduce the computational burden somewhat.

14.3.2 Notation

We present the BCJR algorithm here in the context of a $R = k/n$ convolutional coder. Consider the block diagram of Figure 14.6. The encoder accepts message symbols m_i coming from an alphabet $\mathcal{A}$ — most frequently, $\mathcal{A} = \{0, 1\}$ — which are grouped into k-tuples $\mathbf{m}_i = [m_i^{(0)}, \ldots, m_i^{(k-1)}]$. It is frequently convenient to employ convolutional encoders which terminate in a known state. To accomplish this, Figure 14.6 portrays the input sequence $\mathbf{m} = [\mathbf{m}_0, \mathbf{m}_1, \ldots, \mathbf{m}_{L-1}]$ passing through a system that appends a sequence of $\mathbf{x}_L, \mathbf{x}_{L+1}, \ldots, \mathbf{x}_{L+\nu-1}$, where ν is the constraint length (or memory) of the convolutional coder, which is used to drive the state of the encoder to 0. (For a polynomial encoder, the padding bits would be all zeros, but for the recursive encoder, the padding bits are a function

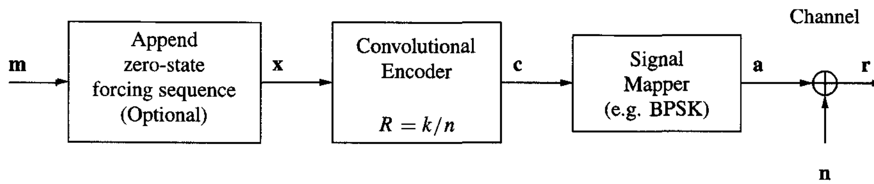


Figure 14.6: Processing stages for BCJR algorithm.

of the state of the encoder after the last message bit m_{L-1} enters the encoder.) The sequence

$$\mathbf{x} = \left[\mathbf{m}, \mathbf{x}_L, \mathbf{x}_{L+1}, \ldots, \mathbf{x}_{L+\nu-1}\right]$$

forms the input to the $R = k/n$ convolutional encoder. We denote the actual length of the input sequence by N, so $N = L + \nu$ if the appended sequence is used, or $N = L$ if not. Each block $\mathbf{x}_i$ is in $\mathcal{A}^k$. The output of the encoder is the sequence of blocks of symbols

$$\mathbf{v} = \left[\mathbf{v}_0, \mathbf{v}_1, \ldots, \mathbf{v}_{N-1}\right],$$

where each block $\mathbf{v}_t$ contains the n output bits of the encoder for the tth input:

$$\mathbf{v}_t = \left[v_t^{(0)}, v_t^{(1)}, \ldots, v_t^{(n-1)}.\right]$$

The encoder symbols $\mathbf{v}_t$ are mapped to a signal constellation (such as BPSK) to produce the output symbols $\mathbf{a}_t$. The dimension of the $\mathbf{a}_t$ depends on the dimension of the signal constellation. For example, if BPSK is employed, we might have $a_t^{(i)} \in \{\pm\sqrt{E_c}\}$, where $RE_b = E_c$, with $\mathbf{a}_t = [a_t^{(0)}, a_t^{(1)}, \ldots, a_t^{(n-1)}]$ and

$$a_t^{(i)} = \sqrt{E_c}(2v_t^{(i)} - 1), \qquad i = 0, 1, \ldots, n-1. \tag{14.4}$$

We also use the notation

$$\tilde{v}_t^{(i)} = 2v_t^{(i)} - 1$$

to indicate the ± 1 modulated signals without the $\sqrt{E_c}$ scaling, so $a_t^{(i)} = \sqrt{E_c}\tilde{v}_t^{(i)}$.

The sequence of output symbols $\mathbf{a} = [\mathbf{a}_0, \mathbf{a}_1, \ldots, \mathbf{a}_{N-1}]$ passes through an additive white Gaussian noise (AWGN) channel to form the received symbol sequence

$$\mathbf{r} = [\mathbf{r}_0, \mathbf{r}_1, \ldots, \mathbf{r}_{N-1}],$$

where

$$\mathbf{r}_t = \mathbf{a}_t + \mathbf{n}_t, \qquad t = 0, 1, \ldots, N-1,$$

and where $\mathbf{n}_i$ is a zero-mean Gaussian noise signal with variance $\sigma^2 = N_0/2$ in each component.

We denote the discrete time index as t. We denote the state of the encoder at time t by Ψ_t. There are $Q = 2^\nu$ possible states, where ν is the constraint length of the encoder, which we denote as integers in the range $0 \leq \Psi_t < 2^\nu$. We assume that the encoder starts in state 0 (the all-zero state) before any message bits are processed, so that $\Psi_t = 0$ for $t \leq 0$. When the zero-forcing sequence is appended, the encoder terminates in state 0, so that $\Psi_N = 0$. Otherwise, it is assumed that the encoder could terminate in any state with equal probability. The sequence of states associated with the input sequence is $\{\Psi_0, \Psi_1, \ldots, \Psi_N\}$.

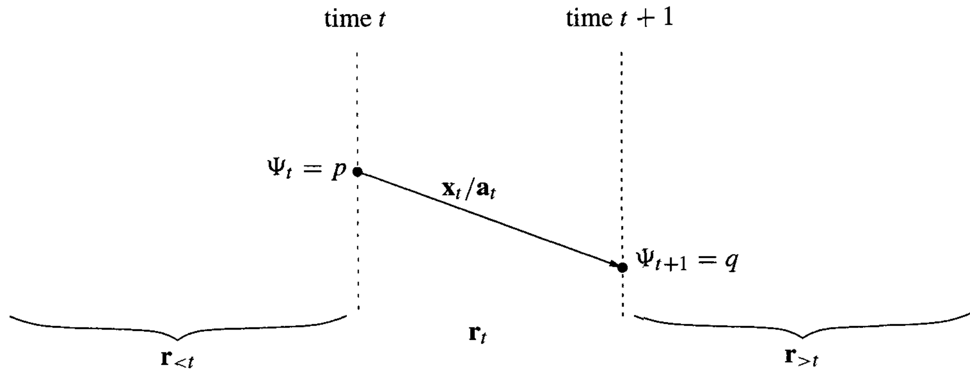


Figure 14.7: One transition of the trellis for the encoder.

A portion of the trellis associated with the encoder is shown in Figure 14.7, portraying a state $\Psi_t = p$ at time t transitioning to a state $\Psi_{t+1} = q$ at time $t+1$. The unique input $\mathbf{x}_t$ which causes this transition is denoted by $\mathbf{x}^{(p,q)}$. The corresponding mapped symbols $\mathbf{a}_t$ produced by this state transition are denoted by $\mathbf{a}^{(p,q)}$, with elements $a^{(0,p,q)}, a^{(1,p,q)}, \ldots, a^{(n-1,p,q)}$.

Notationally, quantities with the time-index subscript $_t$ are often random variables or their realizations, (e.g., $\mathbf{a}_t$, $a_t^{(0)}$, or $\mathbf{x}_t$), whereas quantities without the time-index subscript are usually not random variables.

14.3.3 Posterior Probability

It is clear that the convolutional code introduces dependencies among the symbols $\{\mathbf{a}_t\}$. An optimal decoder should exploit these dependencies, examining the *entire* sequence of data to determine its estimates of the probabilities of the input bits. The goal of the decoder is thus: Determine the *a posteriori* probabilities of the input $P(\mathbf{x}_t = \mathbf{x}|\mathbf{r})$, that is, the probability that the input takes on some value $\mathbf{x}$ conditioned upon the entire received sequence $\mathbf{r}$. The BCJR algorithm provides an efficient way to compute these probabilities. The first step is to determine the probabilities of *state transitions*; once these are determined finding the probabilities of the bits is straightforward.

The convolutional code introduces a Markov property into the probability structure: Knowledge of the state at time $t+1$ renders irrelevant knowledge of the state at time t or previous times. To exploit this Markovity, we partition the observations into three different sets,

$$\mathbf{r} = \mathbf{r}_{<t} \cup \{\mathbf{r}_t\} \cup \mathbf{r}_{>t},$$

where $\mathbf{r}_{<t} = \{\mathbf{r}_l : l < t\}$ is the set of "prior" observations, $\mathbf{r}_t$ is the "current" observation, and $\mathbf{r}_{>t} = \{\mathbf{r}_l : l > t\}$ is the set of the future observations. Then the posterior probability of the transition $(\Psi_t = p, \Psi_{t+1} = q)$ given the observed sequence $\mathbf{r}$ is

$$\begin{aligned} P(\Psi_t = p, \Psi_{t+1} = q|\mathbf{r}) &= p(\Psi_t = p, \Psi_{t+1} = q, \mathbf{r})/p(\mathbf{r}) \\ &= p(\Psi_t = p, \Psi_{t+1} = q, \mathbf{r}_{<t}, \mathbf{r}_t, \mathbf{r}_{>t})/p(\mathbf{r}), \end{aligned} \tag{14.5}$$

where P denotes a probability mass function and p denotes a probability density function.

We now employ the conditioning factorization

$$\begin{aligned} P(\Psi_t = p, \Psi_{t+1} = q|\mathbf{r}) \\ = p(\Psi_t = p, \Psi_{t+1} = q, \mathbf{r}_{<t}, \mathbf{r}_t)p(\mathbf{r}_{>t}|\Psi_t = p, \Psi_{t+1} = q, \mathbf{r}_{<t}, \mathbf{r}_t)/p(\mathbf{r}). \end{aligned} \tag{14.6}$$

Because of the Markov property, we can rewrite the second probability in (14.6) as

$$p(\mathbf{r}_{>t}|\Psi_t = p, \Psi_{t+1} = q, \mathbf{r}_{<t}, \mathbf{r}_t) = p(\mathbf{r}_{>t}|\Psi_{t+1} = q), \tag{14.7}$$

since knowledge of the state at time $t + 1$ renders irrelevant knowledge about prior states or received data. The first factor in (14.6) can be factored further and the Markovity can exploited again:

$$\begin{aligned} p(\Psi_t = p, \Psi_{t+1} = q, \mathbf{r}_{<t}, \mathbf{r}_t) &= p(\Psi_{t+1} = q, \mathbf{r}_t|\Psi_t = p, \mathbf{r}_{<t})p(\Psi_t = p, \mathbf{r}_{<t}) \\ &= p(\Psi_{t+1} = q, \mathbf{r}_t|\Psi_t = p)p(\Psi_t = p, \mathbf{r}_{<t}). \end{aligned} \tag{14.8}$$

Substituting (14.7) and (14.8) into (14.6) we obtain

$$P(\Psi_t = p, \Psi_{t+1} = q|\mathbf{r}) = p(\Psi_t = p, \mathbf{r}_{<t})p(\Psi_{t+1} = q, \mathbf{r}_t|\Psi_t = p)p(\mathbf{r}_{>t}|\Psi_{t+1} = q)/p(\mathbf{r}).$$

We denote the factors in this probability as follows:

$$\boxed{\alpha_t(p) = p(\Psi_t = p, \mathbf{r}_{<t})}$$

represents the probability of the observations up to time $t - 1$, with the state ending in state p at time t;

$$\boxed{\gamma_t(p, q) = p(\Psi_{t+1} = q, \mathbf{r}_t|\Psi_t = p);}$$

represents the probability of the transition from state p to state q, with the observation at time t; and

$$\boxed{\beta_{t+1}(q) = p(\mathbf{r}_{>t}|\Psi_{t+1} = q)}$$

is the probability of the future observed sequence $\mathbf{r}_{>t}$, given that it starts at state q at time $t + 1$. Thus we have the posterior probability of the state transition

$$\boxed{P(\Psi_t = p, \Psi_{t+1} = q|\mathbf{r}) = \alpha_t(p)\gamma_t(p, q)\beta_{t+1}(q)/p(\mathbf{r}).}$$

We determine recursive techniques for efficiently computing α_t and β_t below.

Given the posterior probability of the state transitions, it is straightforward to determine the posterior probability of a bit $P(\mathbf{x}_t = \mathbf{x}|\mathbf{r})$. For each input value $\mathbf{x}$ in the input alphabet $\mathcal{A}$, let $\mathcal{S}_x$ denote the set of state transitions (p, q) which correspond to the input $\mathbf{x}_t = \mathbf{x}$:

$$\mathcal{S}_x = \{(p, q): \mathbf{x}^{(p,q)} = \mathbf{x}\}.$$

For example, for the trellis of Figure 14.4(b),

$$\mathcal{S}_0 = \{(0, 0), (1, 2), (2, 1), (3, 3)\} \qquad \mathcal{S}_1 = \{(0, 2), (1, 0), (2, 3), (3, 1)\}.$$

(We assume for convenience that the trellis is time-invariant, but decoding on time-varying trellises is also possible.) The posterior probability of $\mathbf{x}_t = \mathbf{x}$ is then obtained by summing over all state transitions for which $\mathbf{x}$ is the input:

$$P(\mathbf{x}_t = \mathbf{x}|\mathbf{r}) = \sum_{(p,q)\in\mathcal{S}_x} P(\Psi_t = p, \Psi_{t+1} = q|\mathbf{r}) = \frac{1}{p(\mathbf{r})} \sum_{(p,q)\in\mathcal{S}_x} \alpha_t(p)\gamma_t(p, q)\beta_{t+1}(q), \tag{14.9}$$

for all $\mathbf{x} \in \mathcal{A}^k$. Up to this point, we have been including the factor $1/p(\mathbf{r})$ in all of our posterior probability computations. However, it is nothing more than a normalizing factor and does not need to be explicitly computed. Since $P(\mathbf{x}_t = \mathbf{x}|\mathbf{r})$ is a probability mass function, we have

$$\sum_{\mathbf{x}\in\mathcal{A}} P(\mathbf{x}_t = \mathbf{x}|\mathbf{r}) = 1.$$

Using this fact, we can compute (14.9) without finding $p(\mathbf{r})$ as follows. Let

$$\tilde{p}(\mathbf{x}_t = \mathbf{x}|\mathbf{r}) = \sum_{(p,q)\in\mathcal{S}_x} \alpha_t(p)\gamma_t(p,q)\beta_{t+1}(q), \quad \mathbf{x} \in \mathcal{A}^k.$$

That is, $\tilde{p}(\mathbf{x}_t = \mathbf{x}|\mathbf{r})$ is the same as in $P(\mathbf{x}_t = \mathbf{x}|\mathbf{r})$, but without the factor $1/p(\mathbf{r})$. Then

$$P(\mathbf{x}_t = \mathbf{x}|\mathbf{r}) = \frac{\tilde{p}(\mathbf{x}_t = \mathbf{x}|\mathbf{r})}{\sum_{\mathbf{l}\in\mathcal{A}^k} \tilde{p}(\mathbf{x}_t = \mathbf{l}|\mathbf{r})}. \tag{14.10}$$

It is convenient to express this normalization using an operator. Define the scaling (or *normalization*) operator N_x by

$$\mathsf{N}_x f = \mathsf{N}_x f(\mathbf{x}) = \frac{f(\mathbf{x})}{\sum_{\mathbf{l}\in\mathcal{A}^k} f(\mathbf{l})}.$$

That is, the normalization of a function $f(\mathbf{x})$ is obtained by dividing $f(\mathbf{x})$ by the sum of $f(\mathbf{x})$, summed over the entire domain set of $\mathbf{x}$. The domain of $\mathbf{x}$ is implicit in this notation. Using the normalization notation, we have

$$P(\mathbf{x}_t = \mathbf{x}|\mathbf{r}) = \mathsf{N}_x \tilde{p}(\mathbf{x}_t = \mathbf{x}|\mathbf{r}) = \mathsf{N}_x \sum_{(p,q)\in\mathcal{S}_x} \alpha_t(p)\gamma_t(p,q)\beta_{t+1}(q). \tag{14.11}$$

14.3.4 Computing α_t and β_t

Given $\alpha_t(p)$ for all states $p \in \{0, \ldots, Q-1\}$, the values of $\alpha_{t+1}(q)$ can be computed as follows:

$$\begin{aligned}
\alpha_{t+1}(q) &= p(\Psi_{t+1} = q, \mathbf{r}_{<t+1}) = p(\Psi_{t+1} = q, \mathbf{r}_t, \mathbf{r}_{<t}) && \text{(definition of } \alpha_{t+1} \text{ and } \mathbf{r}_{<t+1}\text{)} \\
&= \sum_{p=0}^{Q-1} p(\Psi_{t+1} = q, \mathbf{r}_t, \Psi_t = p, \mathbf{r}_{<t}) && \text{(compute marginal from joint)} \\
&= \sum_{p=0}^{Q-1} p(\Psi_t = p, \mathbf{r}_{<t}) p(\Psi_{t+1} = q, \mathbf{r}_t | \Psi_t = p, \mathbf{r}_{<t}) && \text{(conditioning factorization)} \\
&= \sum_{p=0}^{Q-1} p(\Psi_t = p, \mathbf{r}_{<t}) p(\Psi_{t+1} = q, \mathbf{r}_t | \Psi_t = p) && \text{(by Markovity)} \\
&= \sum_{p=0}^{Q-1} \alpha_t(p)\gamma_t(p,q) && \text{(definition of } \alpha \text{ and } \gamma\text{).}
\end{aligned}$$

That is,

$$\boxed{\alpha_{t+1}(q) = \sum_{p=0}^{Q-1} \alpha_t(p)\gamma_t(p,q).} \tag{14.12}$$

A backward recursion can similarly be developed for $\beta_t(p)$:

$$\begin{aligned}
\beta_t(p) &= p(\mathbf{r}_{>t-1}|\Psi_t = p) = p(\mathbf{r}_{>t}, \mathbf{r}_t|\Psi_t = p) && \text{(definition of } \mathbf{r}_{>t-1}) \\
&= \sum_{q=0}^{Q-1} p(\mathbf{r}_{>t}, \mathbf{r}_t, \Psi_{t+1} = q|\Psi_t = p) && \text{(marginal from joint)} \\
&= \sum_{q=0}^{Q-1} p(\mathbf{r}_t, \Psi_{t+1} = q|\Psi_t = p)p(\mathbf{r}_{>t}|\mathbf{r}_t, \Psi_{t+1} = q, \Psi_t = p) \\
& && \text{(conditioning factorization)} \\
&= \sum_{q=0}^{Q-1} p(\mathbf{r}_t, \Psi_{t+1} = q|\Psi_t = p)p(\mathbf{r}_{>t}|\Psi_{t+1} = q) \\
& && \text{(by Markovity)} \\
&= \sum_{q=0}^{Q-1} \gamma_t(p, q)\beta_{t+1}(q) && \text{(definition of } \gamma \text{ and } \beta).
\end{aligned} \tag{14.13}$$

That is,

$$\boxed{\beta_t(p) = \sum_{q=0}^{Q-1} \gamma_t(p, q)\beta_{t+1}(q).} \tag{14.14}$$

The α probabilities are computed starting at the beginning of the trellis with the set $\alpha_0(p)$, $p = 0, 1, \ldots, Q-1$, and working forward through the trellis. This computation is called the *forward pass*. The β probabilities are computed starting at the end of the trellis with the set $\beta_N(p)$, $p = 0, 1, \ldots, Q-1$, and working backward through the trellis. This computation is called the *backward pass*. Because the computation of α and β is such an essential part of the BCJR algorithm, it is sometimes also referred to as the *forward-backward* algorithm.

The recursions (14.12) and (14.14) are initialized as follows. Since the encoder is known to start in state 0, set all of the probability weight in state 0 for α_0:

$$[\alpha_0(0), \alpha_0(1), \ldots, \alpha_0(Q-1)] = [1, 0, \ldots, 0]. \tag{14.15}$$

If it is known that the encoder terminates in state 0, set

$$[\beta_N(0), \beta_N(1), \ldots, \beta_N(Q)] = [1, 0, \ldots, 0]. \tag{14.16}$$

Otherwise, since the encoder can terminate in any state with uniform probability, set

$$[\beta_N(0), \beta_N(1), \ldots, \beta_N(2\nu - 1)] = [1/Q, 1/Q, \ldots, 1/Q]. \tag{14.17}$$

14.3.5 Computing γ_t

The transition probability $\gamma_t(p, q)$, or *branch metric* for the branch from $\Psi_t = p$ to $\Psi_{t+1} = q$, depends upon the particular distribution of the observations. For an AWGN channel, the branch metric can be computed as

$$\gamma_t(p, q) = p(\Psi_{t+1} = q, \mathbf{r}_t|\Psi_t = p) = p(\mathbf{r}_t|\Psi_t = p, \Psi_{t+1} = q)P(\Psi_{t+1} = q|\Psi_t = p). \tag{14.18}$$

Knowing $\Psi_t = p$ and $\Psi_{t+1} = q$, that is, the beginning and ending of a state transition, completely determines and is determined by the output $\mathbf{a}^{(p,q)}$ and the corresponding input $\mathbf{x}^{(p,q)}$. The probability of the state transition (p, q) is thus equivalent to the probability of the input bit associated with it:

$$P(\Psi_{t+1} = q | \Psi_t = p) = P(\mathbf{x}_t = \mathbf{x}^{(p,q)}), \tag{14.19}$$

where $P(\mathbf{x}_t = \mathbf{x}^{(p,q)})$ is the *a priori* probability of the message symbol $\mathbf{x}_t$. In conventional binary coding, $P(\mathbf{x}_t = \mathbf{x})$ is usually equal to $1/2^k$; however, we will see below that it is helpful to use other interpretations.

The probability $p(\mathbf{r}_t | \Psi_t = p, \Psi_{t+1} = q)$ can be written as $p(\mathbf{r}_t | \mathbf{a}^{(p,q)})$. For the n-dimensional AWGN channel, this is simply the Gaussian likelihood,

$$p(\mathbf{r}_t | \mathbf{a}^{(p,q)}) = \frac{1}{(2\pi\sigma^2)^{n/2}} \exp\left[-\frac{1}{2\sigma^2} \|\mathbf{r}_t - \mathbf{a}^{(p,q)}\|^2\right], \tag{14.20}$$

where $\|\cdot\|^2$ is the conventional squared Euclidean metric,

$$\|\mathbf{x}\|^2 = \sum_{i=1}^{n} x_i^2.$$

Substituting (14.19) and (14.20) into (14.18) we obtain, for BPSK modulation,

$$\begin{aligned} \gamma_t(p, q) &= \frac{1}{(2\pi\sigma^2)^{n/2}} \exp\left[-\frac{1}{2\sigma^2} \|\mathbf{r}_t - \mathbf{a}^{(p,q)}\|^2\right] P(\mathbf{x}_t = \mathbf{x}^{(p,q)}) \\ &= \frac{1}{(2\pi\sigma^2)^{n/2}} \exp\left[-\frac{1}{2\sigma^2} \sum_{i=0}^{n-1} (r_t^{(i)} - \sqrt{E_c}\tilde{v}_i^{(p,q)})^2\right] P(\mathbf{x}_t = \mathbf{x}^{(p,q)}). \end{aligned} \tag{14.21}$$

14.3.6 Normalization

Two different kinds of normalization are frequently used in computing the forward-backward algorithm. First, normalization is used to simplify the computation of the transition probability γ. Second, the normalization is used to numerically stabilize the computation of the αs and βs.

For some constant C, let $\gamma_t'(p, q) = C\gamma_t(p, q)$ and let $\alpha_t'(p)$ and $\beta_t'(p)$ be the corresponding forward and backward probabilities, defined by

$$\alpha_{t+1}'(q) = \sum_p \alpha_t'(p)\gamma_t'(p, q)$$

$$\beta_{t+1}'(p) = \sum_q \gamma_t'(p, q)\beta_{t+1}'(q)$$

with the same initialization for α_0' and β_N' as for the unnormalized case. At each stage of the propagation, an additional factor C accumulates, so that

$$\alpha_t'(p) = C^t \alpha_t(p) \qquad \beta_t'(p) = C^{N-t}\beta_t(p).$$

When α' and β' are used in (14.10) or (14.33), the factor C cancels out, resulting in identical probability or likelihood values. Since the normalization constant C has no bearing on the

detection problem, it may be chosen to simplify the computations. For example, using $C = 2^k(2\pi\sigma^2)^{n/2}$ in (14.21) yields the normalized branch metric

$$\gamma_t'(p,q) = \left[\frac{P(\mathbf{x}_t = \mathbf{x}^{(p,q)})}{1/2^k}\right] \exp\left[-\frac{1}{2\sigma^2}\|\mathbf{r}_t - \mathbf{a}^{(p,q)}\|^2\right].$$

If the prior probability $P(\mathbf{x}_t = \mathbf{x}^{(p,q)}) = 1/2^k$ for all t, then the factor in front is simply unity.

The propagation of α_t and β_t involves computing products and sums of small numbers. Without some kind of normalization there is a rapid loss in numerical precision in the forward and backward passes. It is therefore customary to normalize these probabilities. The forward probability $\alpha_t(p)$ and the backward probability $\beta_t(p)$ are replaced by $\alpha_t'(p)$ and $\beta_t'(p)$ which are normalized so that

$$\sum_p \alpha_t'(p) = 1 \qquad \sum_p \beta_t'(p) = 1 \tag{14.22}$$

for each t. These normalized versions are propagated by

$$\alpha_{t+1}'(q) = A_t \sum_{p=0}^{Q-1} \alpha_t'(p)\gamma_t(p,q)$$

$$\beta_t'(p) = B_t \sum_{q=0}^{Q-1} \gamma_t(p,q)\beta_{t+1}'(q),$$

where A_t and B_t are chosen so that (14.22) is satisfied for each t. That is,

$$\alpha_{t+1}'(q) = \frac{\sum_{p=0}^{Q-1} \alpha_t'(p)\gamma_t(p,q)}{\sum_{q=0}^{Q-1}\sum_{p=0}^{Q-1} \alpha_t'(p)\gamma_t(p,q)} = \mathsf{N}_q \sum_{p=0}^{Q-1} \alpha_t'(p)\gamma_t(p,q)$$

and

$$\beta_t'(p) = \mathsf{N}_p \sum_{q=0}^{Q-1} \gamma_t(p,q)\beta_{t+1}'(q).$$

The normalizations are

$$A_t = \frac{1}{\sum_{q=0}^{Q-1}\sum_{p=0}^{Q-1} \alpha_t'(p)\gamma_t(p,q)} \qquad B_t = \frac{1}{\sum_{p=0}^{Q-1}\sum_{q=0}^{Q-1} \gamma_t(p,q)\beta_{t+1}'(q)}.$$

The relationship between the normalized and unnormalized versions is

$$\alpha_t'(p) = \left(\prod_{i=0}^{t} A_i\right)\alpha_t(p) \qquad \beta_t'(p) = \left(\prod_{i=t}^{N-1} B_i\right)\beta_t(p).$$

When using α_t' and β_t' in (14.10) or (14.33) the products of the normalization factors cancel from the numerator and denominator. While the normalization does not affect the posterior probability computation mathematically, it does have a significant impact on the numerical performance of the algorithm.

14.3.7 Summary of the BCJR Algorithm

Algorithm 14.1 The BCJR (MAP) Decoding Algorithm, Probability Form

Initialize: Set α'_0 as in (14.15), and initialize β'_N as in (14.16) or (14.17).
For $t = 0, 1, \ldots, N-2$ propagate α':

$$\alpha'_{t+1}(q) = \mathsf{N}_q \sum_{p=0}^{Q-1} \alpha'_t(p)\gamma'_t(p,q).$$

For $t = N-1, N-2, \ldots, 1$ propagate β':

$$\beta'_t(p) = \mathsf{N}_p \sum_{q=0}^{Q-1} \gamma'_t(p,q)\beta'_{t+1}(q).$$

Compute the posterior probability for $\mathbf{x}_t$:

$$P(\mathbf{x}_t = \mathbf{x}|\mathbf{r}) = \mathsf{N}_x \sum_{(p,q)\in\mathcal{S}_x} \alpha'_t(p)\gamma'_t(p,q)\beta'_{t+1}(q).$$

Example 14.2 Referring to Example 14.1, the sequence $\mathbf{x} = [1, 1, 0, 0, 1, 0, 1, 0, 1, 1]$ is input to one of the convolutional encoders of Figure 14.4. The systematic and parity bits are multiplexed together to produce the coded sequence

$$\mathbf{v} = [1,1,\ 1,1,\ 0,1,\ 0,1,\ 1,0,\ 0,1,\ 1,1,\ 0,1,\ 1,0,\ 1,0].$$

The corresponding sequence of encoder states is

$$\boldsymbol{\Psi} = [0, 2, 3, 3, 3, 1, 2, 3, 3, 1, 0]. \tag{14.23}$$

The sequence $\mathbf{v}$ is BPSK modulated with amplitudes ± 1 and passed through an AWGN channel with $\sigma^2 = 0.45$, resulting in the received data

$$\begin{aligned}\mathbf{r} = [&(2.53008, 0.731636)(-0.523916, 1.93052)(-0.793262, 0.307327)(-1.24029, 0.784426)\\ &(1.83461, -0.968171)(-0.433259, 1.26344)(1.31717, 0.995695)(-1.50301, 2.04413)\\ &(1.60015, -1.15293)(0.108878, -1.57889)].\end{aligned} \tag{14.24}$$

If a decision were made at this point based on the sign of the received signal, the detected bits would be

$$[1,1,\ \underline{0},1,\ 0,1,\ 0,1,\ 1,0,\ 0,1,\ 1,1,\ 0,1,\ 1,0,\ 1,0],$$

where the underlined bit is in error.

The forward and backward passes are shown in Table 14.1. Note that the maximum probability states determined by the αs (shown in bold) correspond with the true state sequence of (14.23). The maximum likelihood sequence of states determined by the βs correspond with the true sequence of states from Ψ_{10} down to Ψ_2, but the maximum likelihood state determined by β_1 is $\hat{\Psi}_1 = 3$, whereas the true state is $\Psi_1 = 2$; the confusion arises because the $\mathbf{r}_1 = (-0.523916, 1.93052)$ decodes incorrectly, and the resulting sequence $(0, 1)$ is a valid output transition on a state leading to $\Psi_2 = 3$.

As may be seen from the trellis in Figure 14.4, the input transition sets $\mathcal{S}_0$ and $\mathcal{S}_1$ are

$$\mathcal{S}_0 = \{(0,0), (1,2), (2,1), (3,3)\} \qquad \mathcal{S}_1 = \{(0,2), (1,0), (2,3), (3,1)\}.$$

Table 14.1: α'_t and β'_t Example Computations

Forward Pass
Direction of Processing $\longrightarrow$

q	α'_0	α'_1	α'_2	α'_3	α'_4	α'_5	α'_6	α'_7	α'_8	α'_9
0	**1**	5.06e-7	9.74e-10	1.43e-5	1.36e-6	1.82e-4	5.31e-4	1.89e-8	7.21e-12	2.03e-7
1	0	0	1.92e-3	7.44e-3	1.81e-4	**0.999**	5.60e-8	3.56e-5	2.03e-7	**0.999**
2	0	**0.999**	5.05e-7	1.919e-3	7.45e-3	1.93e-8	**0.999**	5.31e-4	3.56e-5	1.03e-12
3	0	0	**0.998**	**0.991**	**0.992**	1.05e-4	1.05e-4	**0.999**	**0.999**	5.06e-6

Backward Pass
Direction of Processing $\longleftarrow$

p	β'_1	β'_2	β'_3	β'_4	β'_5	β'_6	β'_7	β'_8	β'_9	β'_{10}
0	2.55e-3	2.41e-4	2.75e-6	4.15e-5	0.127	1.32e-3	1.08e-6	3.11e-4	0.381	**1**
1	2.61e-2	8.19e-3	3.23e-4	0.127	**0.870**	3.80e-6	8.14e-4	0.381	**0.619**	0
2	8.62e-2	2.85e-2	8.43e-3	2.85e-4	3.65e-4	**0.996**	1.32e-3	5.04e-4	0	0
3	**0.885**	**0.963**	**0.991**	**0.872**	2.50e-3	2.87e-3	**0.998**	**0.618**	0	0

Table 14.2: Posterior Input Bit Example Computations

t:	0	1	2	3	4	5	6	7	8	9
$P(x_t = 0\|\mathbf{r})$	1.49e-8	1.64e-5	1	1	2.17e-6	1	3.31e-7	1	2.90e-8	1.25e-7
$P(x_t = 1\|\mathbf{r})$	1	1	2.47e-6	2.58e-5	1	2.73e-5	1	7.55e-7	1	1
$\hat{x}_t$	1	1	0	0	1	0	1	0	1	1

The input bit probabilities can be computed as follows for $t = 0$:

$$\begin{aligned}\tilde{p}(x_0 = 1|\mathbf{r}) &= \alpha_0(0)\gamma_0(0,2)\beta_1(2) + \alpha_0(1)\gamma_0(1,0)\beta_1(0) + \alpha_0(2)\gamma_0(2,3)\beta_1(3)\\ &\quad + \alpha_0(3)\gamma_0(3,1)\beta_1(1)\end{aligned}$$

which results in $\tilde{p}(x_0 = 1|\mathbf{r}) = 0.00295$. Similarly, $\tilde{p}(x_0 = 0|\mathbf{r}) = 4.41 \times 10^{-11}$. After normalizing these, we find $P(x_0 = 0|\mathbf{r}) = 1.49 \times 10^{-8}$, $P(x_0 = 1|\mathbf{r}) = 0.999$. Table 14.2 shows the posterior bit probabilities (to three decimal places, so there is some roundoff in the probabilities near 1) and the posterior estimate of the bit sequence. Note that the estimated bit sequence matches the input sequence.

□

14.3.8 A Matrix/Vector Formulation

For notational purposes it is sometimes convenient to express the BCJR algorithm in a matrix formulation (although we do not use this further in this chapter). Let

$$\boldsymbol{\alpha}_t = \begin{bmatrix} \alpha_t(0) \\ \alpha_t(1) \\ \vdots \\ \alpha_t(Q-1) \end{bmatrix} \quad \text{and} \quad \boldsymbol{\beta}_t = \begin{bmatrix} \beta_t(0) \\ \beta_t(1) \\ \vdots \\ \beta_t(Q-1) \end{bmatrix}$$

be vectors of the forward and backward probabilities. Let G_t be the probability matrix with elements $g_{t,i,j}$ defined by

$$g_{t,i,j} = \gamma_t(i,j).$$

Then the forward update (14.12) can be expressed as

$$\boldsymbol{\alpha}_{t+1} = G_t^T \boldsymbol{\alpha}_t.$$

The backward update (14.14) can be expressed as

$$\beta_t = G_t \beta_{t+1}.$$

To compute (14.9), we need to define a matrix describing transitions in the trellis. Let $T(x)$ be defined with elements $t_{i,j}(x)$ by

$$t_{i,j}(x) = \begin{cases} 1 & \text{if } (i, j) \text{ is a state transition with } x^{(i,j)} = x \\ 0 & \text{otherwise.} \end{cases}$$

For example for the trellis shown in Figure 14.4(b),

$$T(0) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad T(1) = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{bmatrix}.$$

Let $\odot$ denote the element-by-element product of two matrices. Then (14.9) can be expressed as

$$P(x_t = x|\mathbf{r}) = \frac{1}{p(\mathbf{r})} \alpha^T (T(x) \odot P_t) \beta_{t+1}.$$

14.3.9 Comparison of the Viterbi Algorithm and the BCJR Algorithm

It is interesting to contrast this update formula with the formula for updating the path metric in the Viterbi algorithm. In the Viterbi algorithm, the path metric is updated by *adding* the branch metric to the previous path metric. Then the *minimum* of the path metrics at a state is computed. In the BCJR case, the path metric $\alpha_t(p)$ is *multiplied* by the branch metric $\gamma_t(p, q)$, then the branch metrics are *summed* at each state. Mapping the operations

$$\text{min} \leftrightarrow \text{sum} \qquad \text{sum} \leftrightarrow \text{product}$$

we obtain the equivalent algorithm. The Viterbi algorithm is sometimes referred to as a "min-sum" algorithm and the BCJR algorithm is referred to as a "sum-product" algorithm. (See [2] or Chapter 16 for other examples.)

14.3.10 The BCJR Algorithm for Systematic Codes

To finish setting the stage for turbo decoding, we now consider the specialization to the case that the convolutional encoder is a systematic $R = 1/2$ coder, and the signal mapper is BPSK, where, to be specific, we use the BPSK signal mapper in (14.4). The encoder output is now

$$\mathbf{v}_t = [v_t^{(0)}, v_t^{(1)}] = [x_t, v_t^{(1)}]$$

and the mapped signals are

$$\mathbf{a}_t = [a_t^{(0)}, a_t^{(1)}] = \sqrt{E_c}[2v_t^{(0)} - 1, 2v_t^{(1)} - 1] = \sqrt{E_c}[2x_t - 1, 2v_t^{(1)} - 1].$$

To denote the output corresponding to the transition from $\Psi_t = p$ to $\Psi_{t+1} = q$, we write

$$\mathbf{a}^{(p,q)} = [a^{(0,p,q)}, a^{(1,p,q)}] = \sqrt{E_c}[2x^{(p,q)} - 1, 2v^{(1,p,q)} - 1].$$

The received signal vector at time t is

$$\mathbf{r}_t = [r_t^{(0)}, r_t^{(1)}],$$

where

$$r_0^{(0)} = a_t^{(0)} + n_t^{(0)} \qquad r_t^{(1)} = a_t^{(1)} + n_t^{(1)}.$$

The transition probability can be written

$$\begin{aligned}
\gamma_t(p,q) &= p(\Psi_{t+1} = q, \mathbf{r}_t | \Psi_t = p) && \text{(definition)}\\
&= p(\mathbf{r}_t | \Psi_t = p, \Psi_{t+1} = q) P(\Psi_{t+1} = q | \Psi_t = p) && \text{(condition factorization)}\\
&= p(r_t^{(0)}, r_t^{(1)} | \Psi_t = p, \Psi_{t+1} = q) P(\Psi_{t+1} = q | \Psi_t = p) && \text{(definition of } \mathbf{r}_t\text{)}\\
&= p(r_t^{(0)}, r_t^{(1)} | \Psi_t = p, \Psi_{t+1} = q) P(x_t = x^{(p,q)}) && (x^{(p,q)} \text{ determines } \Psi_{t+1}).
\end{aligned} \tag{14.25}$$

The conditioning on the state transition can be equivalently expressed as conditioning on the state and the input, since knowing the state and the input determines the next state unequivocally:

$$p(r_t^{(0)}, r_t^{(1)} | \Psi_t = p, \Psi_{t+1} = q) = p(r_t^{(0)}, r_t^{(1)} | \Psi_t = p, x_t = x^{(p,q)}).$$

But $r_t^{(0)}$ and $r_t^{(1)}$ are conditionally independent, given the input, since $r_t^{(0)}$ *depends on the input data and not on the state.* Thus

$$\begin{aligned}
p(r_t^{(0)}, r_t^{(1)} | \Psi_t = p, x_t) &= p(r_t^{(0)} | x_t) p(r_t^{(1)} | \Psi_t = p, \Psi_{t+1} = q)\\
&= p(r_t^{(0)} | x_t) p(r_t^{(1)} | a_t^{(1)} = a^{(1,p,q)}).
\end{aligned} \tag{14.26}$$

Substituting (14.26) into (14.25) we obtain

$$\gamma_t(p,q) = p(r_t^{(0)} | x_t) p(r_t^{(1)} | a_t^{(1)} = a^{(1,p,q)}) P(x_t = x^{(p,q)}). \tag{14.27}$$

Now substitute (14.27) into (14.11):

$$P(x_t = x | \mathbf{r}) = \mathsf{N}_x \sum_{(p,q) \in \mathcal{S}_x} \alpha_t(p) p(r_t^{(0)} | x_t) p(r_t^{(1)} | a_t^{(1)} = a^{(1,p,q)}) p(x_t = x^{(p,q)}) \beta_{t+1}(q). \tag{14.28}$$

In (14.28), since the sum is over elements in $\mathcal{S}_x$, $P(x_t = x^{(p,q)})$ is constant and can be pulled out of the sum. Also, $p(r_t^{(0)} | x_t)$ does not depend on the state, and so can be pulled out of the sum. Thus

$$\begin{aligned}
P(x_t = x | \mathbf{r}) &= \mathsf{N}_x p(r_t^{(0)} | x_t = x) P(x_t = x) \left[\sum_{(p,q) \in \mathcal{S}_x} \alpha_t(p) p(r_t^{(1)} | a_t^{(1)} = a^{(1,p,q)}) \beta_{t+1}(q) \right]\\
&= \mathsf{N}_x P_{s,t}(x) P_{p,t}(x) P_{e,t}(x).
\end{aligned} \tag{14.29}$$

In (14.29), we refer to

$$P_{s,t}(x) = p(r_t^{(0)} | x_t = x)$$

as the *systematic probability*,

$$P_{p,t}(x) = P(x_t = x)$$

as the *prior probability*, and

$$P_{e,t}(x) = \sum_{(p,q) \in \mathcal{S}_x} \alpha_t(p) p(r_t^{(1)} | a_t^{(1)} = a^{(1,p,q)}) \beta_{t+1}(q) \tag{14.30}$$

as the *extrinsic probability*. The word "extrinsic" means[1] "acting from the outside," or separate.

We now describe what the three different probabilities represent.

Prior The prior $P_{p,t}$ represents the information available about bits *before* any decoding occurs, arising from a source other than the received systematic or parity data. It is sometimes called *intrinsic* information, to distinguish it from the extrinsic information. In the iterative decoder, after the first iteration the "prior" is obtained from the other decoder.

Systematic The quantity $P_{s,t}^{(0)}$ represents the information about x_t explicitly available from the measurement of $r_t^{(0)}$. This is a posterior probability.

Extrinsic The extrinsic information $P_{e,t}$ is the information produced by the decoder based on the received sequence and prior information, but excluding the information from the received systematic bit $r_t^{(0)}$ and the prior information related to the bit x_t. It is thus the information that the code itself provides about the bit x_t.

From (14.12) and (14.14), we note that α_t and β_{t+1} do not depend on x_t, but only on received data at other times. Also note that $p(r_t^{(1)}|a_t^{(1)} = a^{(1,p,q)})$ does not depend on the received systematic information $r_t^{(0)}$. Thus the extrinsic probability is, in fact, separate from the information conveyed by the systematic data about x_t. The extrinsic probability $P_{e,t}(x)$ conveys all the information about $P(x_t = x)$ that is available from the *structure of the code*, separate from information which is obtained from an observation of x_t (via $r_t^{(0)}$) or from prior information. This extrinsic probability is an important part of the turbo decoding algorithm; it is, in fact, the information passed between decoders to represent the "prior" probability.

14.3.11 Turbo Decoding Using the BCJR Algorithm

In the turbo decoding algorithm, the posterior probability computed by a previous stage is used as the prior for the next stage. Let us examine carefully which computed probability is to be used. For the moment, for simplicity of notation we ignore the interleavers and the normalization.

We will show that the appropriate probability to pass between the encoders is the *extrinsic* probability by considering what would happen if the entire posterior probability $P(x_t = x\,|\,\mathbf{r})$ were used as the prior for the next decoding phase. Suppose we were to take the MAP probability estimate $P(x_t = x\,|\,\mathbf{r}) = P_{s,t}(x)P_{p,t}(x)P_{e,t}(x)$ of (14.29) computed by the first decoder and use it as the prior $P(x_t = x)$ for the second decoder. Then in the MAP decoding algorithm for the second decoder, the γ computation of (14.27) would be

$$\gamma_t(p,q) = p(r_t^{(0)}|x_t)p(r_t^{(2)}|a_t^{(2)} = a^{(2,p,q)})P_{s,t}(x^{(p,q)})P_{p,t}(x^{(p,q)})P_{e,t}(x^{(p,q)}).$$

But recalling definition of $P_{s,t}$ we have

$$\gamma_t(p,q) = p(r_t^{(2)}|a_t^{(2)} = a_t^{(2,p,q)})\left(P_{s,t}(x^{(p,q)})\right)^2 P_{p,t}(x^{(p,q)})P_{e,t}(x^{(p,q)}). \tag{14.31}$$

We see that the initial prior information $P_{p,t}$ still appears in γ, even though we have already used whatever information it could provide. Furthermore we see in (14.31) that the

[1] *Webster's New World Dictionary*

probability of the systematic information $P_{s,t}$ appears twice. If we were to continue this iteration between stages m times, $P_{s,t}$ would appear to the mth power, coming to yield an overemphasized influence. Thus $P(x_t = x|\mathbf{r})$ is not the appropriate information to pass between the decoders.

Instead, we use the *extrinsic* probability $P_{e,t}$ as the information to pass between stages as the "prior" probability $P(x_t = x)$. Using this, γ_t is computed in the second decoder as

$$\gamma_t(p, q) = p(r_t^{(0)}|x_t)p(r_t^{(2)}|a_t^{(2)} = a^{(2,p,q)})P_{e,t}(x^{(p,q)}).$$

We now flesh out the details somewhat. Let the output of the encoder at time t be

$$\mathbf{v}_t = [x_t, v_t^{(1)}, v_t^{(2)}], \qquad t = 0, 1, \ldots, N-1,$$

where $v_t^{(m)}$ is the output of encoder m, $m = 1, 2$. Also, let the received sequence be

$$\mathbf{r}_t = [r_t^{(0)}, r_t^{(1)}, r_t^{(2)}], \qquad t = 0, 1, \ldots, N-1.$$

There are two MAP decoders employed in the BCJR algorithm, one for each constituent encoder. The first MAP decoder uses the input symbol sequence $\{(r_t^{(0)}, r_t^{(1)}), t = 0, 1, \ldots, N-1\}$, which we also denote (with some abuse of notation) as $(\mathbf{r}^{(0)}, \mathbf{r}^{(1)})$. The second MAP decoder uses the permuted received sequence $\{\Pi r_t^{(0)}\}$ and the received parity information from the second encoder $r_t^{(2)}$. Denote this information as $(\Pi\mathbf{r}^{(0)}, \mathbf{r}^{(2)})$. Let $P^{(0)}(x_t = x)$ denote the initial prior probabilities used by the first MAP decoder. Initially it is assumed that the symbols are equally likely. For binary signaling, $P^{(0)}(x_t = x) = 1/2$.

Let the extrinsic probability produced by decoder j, $j \in \{1, 2\}$, at the lth iteration be denoted by $P_{e,t}^{(l,j)}(x_t = x)$. Let the probability that is used as the prior probability in decoder j at the lth information be denoted by $P^{(l,j)}(x_t = x)$. Let M denote the number of iterations the decoder is to compute. The turbo decoding algorithm can be outlined as follows:

Algorithm 14.2 The Turbo Decoding Algorithm, Probability Form

1. Let $P^{(0,1)}(x_t = x) = P^{(0)}(x_t = x)$ (use the initial priors as the input to the first decoder).
2. For $l = 1, 2, \ldots, M$:
   (a) Using $P^{(l-1,1)}(x_t = x)$ as the prior $P(x_t = x)$, compute:
      - α and β using (14.12), (14.14), and (14.21) (or their normalized equivalents)
      - $P_{e,t}^{(l,1)}(x_t = x)$ using (14.30)

   (b) Let $P^{(l,2)}(x_t = x) = \Pi\left[P_{e,t}^{(l,1)}(x_t = x)\right]$
   (c) Using $P^{(l,2)}(x_t = x)$ as the prior $P(x_t = x)$ compute:
      - α and β using (14.12), (14.14), and (14.21) (or their normalized equivalents)

   (d) If not the last iteration:
      - Compute $P_{e,t}^{(l,2)}(x_t = x)$ using (14.30)
      - Let $P^{(l+1,1)}(x_t = x) = \Pi^{-1}\left[P_{e,t}^{(l,2)}(x_t = x)\right]$

   (e) Else if the last iteration:
      - Using $P^{(l,2)}(x_t = x)$ as the prior $P(x_t = x)$ compute (the permuted) $P(x_t = x|\mathbf{r})$ using (14.29).
      - Un-permute: $P(x_t = x|\mathbf{r}) = \Pi^{-1}[P(x_t = x|\mathbf{r})]$

In the iterative decoding algorithm, the "a priori" information used by a constituent decoder should be completely independent of the channel outputs used by that decoder. However, once the information is fed back around to the first decoder, it is re-using information obtained from the systematic bits used in the second decoder. Thus the prior probabilities used are not truly independent of the channel outputs. However, since the convolutional codes have relatively short memories, the extrinsic probability for a bit x_t are only significantly affected by the received systematic bits close to x_t. Because of the interleaving employed, probability computations for x_t in the first decoder employ extrinsic probabilities that are, with high probability, widely separated. Thus the dependence on the extrinsic probabilities on the systematic data for any given bit x_t is very weak.

The Terminal State of the Encoders

It is straightforward to append a tail sequence so that the terminal state of the first encoder is 0. However, due to interleaving, the state of the second encoder is not known. Discussions on how to drive the state of both encoders to zero are presented in [168, 13]. However, it has been found experimentally that ignorance of the terminal state of the second encoder leads to insignificant differences in decoder performance; it suffices to initialize β uniformly over the states in the decoder.

14.3.12 Likelihood Ratio Decoding

For encoders with a single binary input, the log likelihood ratio is usually used in the detection problem. We denote log likelihood ratios (or log probability ratios of any sort) using the symbol λ. It is convenient to use the mapping from binary values $\{0, 1\}$ to signed binary values $\{1, -1\}$ defined by

$$\tilde{x}_t = 2x_t - 1.$$

For present purposes, we assume that the $\tilde{x}_t$ data are mapped to modulated signals by

$$a_t = \sqrt{E_c}\tilde{x}_t.$$

Let

$$\lambda(x_t|\mathbf{r}) = \log \frac{P(x_t = 1|\mathbf{r})}{P(x_t = 0|\mathbf{r})}, \tag{14.32}$$

where $\mathbf{r}$ is the entire observed sequence. Using (14.9), and noting that the $1/p(\mathbf{r})$ factor cancels in numerator and denominator, the likelihood ratio can be written

$$\lambda(x_t|\mathbf{r}) = \log \frac{\sum_{(p,q)\in\mathcal{S}_1} \alpha_t(p)\gamma_t(p,q)\beta_{t+1}(q)}{\sum_{(p,q)\in\mathcal{S}_0} \alpha_t(p)\gamma_t(p,q)\beta_{t+1}(q)}. \tag{14.33}$$

Assuming that systematic coding is employed and substituting (14.27) into (14.33) we obtain

$$\begin{aligned}\lambda(x_t|\mathbf{r}) &= \log \frac{\sum_{(p,q)\in\mathcal{S}_1} p(r_t^{(0)}|x_t)\alpha_t(p)\beta_{t+1}(q)}{\sum_{(p,q)\in\mathcal{S}_0} p(r_t^{(0)}|x_t)\alpha_t(p)\beta_{t+1}(q)} \\ &\quad + \log \frac{\sum_{(p,q)\in\mathcal{S}_1} P(\Psi_{t+1}=q|\Psi_t=p)\alpha_t(p)\beta_{t+1}(q)}{\sum_{(p,q)\in\mathcal{S}_0} P(\Psi_{t+1}=q|\Psi_t=p)\alpha_t(p)\beta_{t+1}(q)} + \\ &\quad \log \frac{\sum_{(p,q)\in\mathcal{S}_1} p(r_t^{(1)}|\Psi_t=p,\,\Psi_{t+1}=q)\alpha_t(p)\beta_{t+1}(q)}{\sum_{(p,q)\in\mathcal{S}_0} p(r_t^{(1)}|\Psi_t=p,\,\Psi_{t+1}=q)\alpha_t(p)\beta_{t+1}(q)} \\ &= \log \frac{p(r_t^{(0)}|x_t=1)}{p(r_t^{(0)}|x_t=0)} + \log \frac{P(x_t=1)}{P(x_t=0)} \\ &\quad + \log \frac{\sum_{(p,q)\in\mathcal{S}_1} p(r_t^{(1)}|\Psi_t=p,\,\Psi_{t+1}=q)\alpha_t(p)\beta_{t+1}(q)}{\sum_{(p,q)\in\mathcal{S}_0} p(r_t^{(1)}|\Psi_t=p,\,\Psi_{t+1}=q)\alpha_t(p)\beta_{t+1}(q)} \\ &\triangleq \lambda_{s,t}^{(0)} + \lambda_{p,t} + \lambda_{e,t}. \end{aligned} \tag{14.34}$$

The likelihood ratio is thus expressed as the *sum* of the log of the posterior probabilities for the systematic bits,

$$\lambda_{s,t}^{(0)} = \lambda(r_t^{(0)}|x_t) = \log \frac{p(r_t^{(0)}|x_t=1)}{p(r_t^{(0)}|x_t=0)},$$

plus the log likelihood ratio of the prior probabilities,

$$\lambda_{p,t} = \log \frac{P(x_t=1)}{P(x_t=0)}$$

plus the *extrinsic information* $\lambda_{e,t}$,

$$\begin{aligned}\lambda_{e,t} &= \log \frac{\sum_{(p,q)\in\mathcal{S}_1} p(r_t^{(1)}|\Psi_t=p,\,\Psi_{t+1}=q)\alpha_t(p)\beta_{t+1}(q)}{\sum_{(p,q)\in\mathcal{S}_0} p(r_t^{(1)}|\Psi_t=p,\,\Psi_{t+1}=q)\alpha_t(p)\beta_{t+1}(q)} \\ &= \log \frac{\sum_{(p,q)\in\mathcal{S}_1} p(r_t^{(1)}|a_t^{(1)}=a^{(1,p,q)})\alpha_t(p)\beta_{t+1}(q)}{\sum_{(p,q)\in\mathcal{S}_0} p(r_t^{(1)}|a_t^{(1)}=a^{(1,p,q)})\alpha_t(p)\beta_{t+1}(q)}.\end{aligned}$$

The extrinsic information $\lambda_{e,t}$ is the information that is passed from one decoder to the next as the turbo decoding algorithm progresses. This extrinsic information can be computed from (14.34) by

$$\lambda_{e,t} = \lambda(x_t|\mathbf{r}) - \lambda_{p,t} - \lambda_{s,t}. \tag{14.35}$$

Based on this, Figure 14.8 illustrates one way of implementing the turbo decoding. The conventional MAP output of the decoder (expressed in terms of log likelihood ratio) is computed, from which the extrinsic probabilities are computed from (14.35).

We now examine some manipulations which can simplify the computation of some of these log likelihoods.

Log Prior Ratio $\lambda_{p,t}$

The log ratio of the priors

$$\lambda_{p,t} = \log \frac{P(x_t=1)}{P(x_t=0)}$$

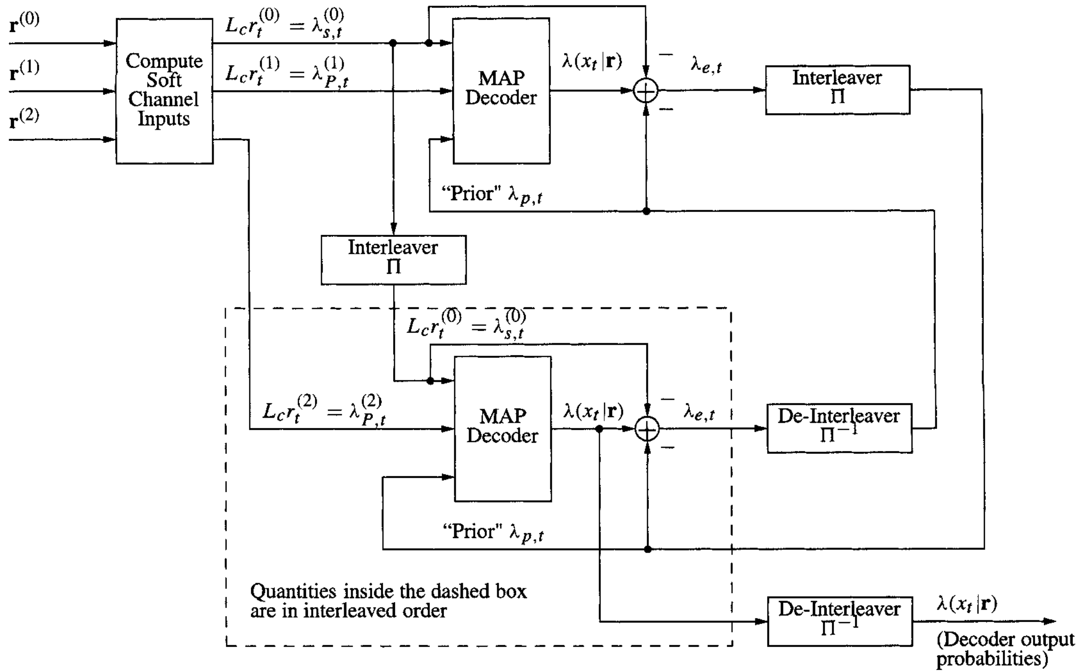


Figure 14.8: A log likelihood turbo decoder.

can be solved for the prior probabilities. Since $P(x_t = 1) + P(x_t = 0) = 1$ and since

$$e^{\lambda_{p,t}} = \frac{P(x_t = 1)}{1 - P(x_t = 1)}$$

it is straightforward to show that

$$P(x_t = 1) = \frac{e^{\lambda_{p,t}}}{1 + e^{\lambda_{p,t}}} = \frac{1}{1 + e^{-\lambda_{p,t}}} \tag{14.36}$$

and

$$P(x_t = 0) = \frac{e^{-\lambda_{p,t}}}{1 + e^{-\lambda_{p,t}}} = \frac{1}{1 + e^{\lambda_{p,t}}}. \tag{14.37}$$

For $x \in \{0, 1\}$ and $\tilde{x} = 2x - 1 \in \{-1, 1\}$, (14.36) and (14.37) can be combined together as

$$P(x_t = x) = P(\tilde{x}_t = \tilde{x})|_{\tilde{x}=2x-1} = \left[\frac{e^{-\lambda_{p,t}/2}}{1 + e^{-\lambda_{p,t}}}\right] e^{\tilde{x}\lambda_{p,t}/2}. \tag{14.38}$$

The factor in brackets does not depend upon the value of x_t and so for many circumstances can be regarded as a constant.

Log Posterior $\lambda_{s,t}^{(0)}$

For an AWGN channel with variance σ^2 and BPSK modulation as in (14.4), $\lambda_{s,t}^{(0)}$ can be computed as

$$\lambda_{s,t}^{(0)} = \log \frac{p(r_t^{(0)}|a_t^{(0)} = \sqrt{E_c})}{p(r_t^{(0)}|a_t^{(0)} = -\sqrt{E_c})} = \log \frac{\exp[-\frac{1}{2\sigma^2}(r_t^{(0)} - \sqrt{E_c})^2]}{\exp[-\frac{1}{2\sigma^2}(r_t^{(0)} + \sqrt{E_c})^2]} = \frac{2\sqrt{E_c}r_t^{(0)}}{\sigma^2} = L_c r_t^{(0)}, \tag{14.39}$$

where

$$L_c = \frac{2\sqrt{E_c}}{\sigma^2} = \frac{2\sqrt{RE_b}}{\sigma^2}$$

is the *channel reliability*, essentially just the signal to noise ratio. The quantity $\lambda_{s,t}^{(0)} = L_c r_t^{(0)}$ is often called the *soft channel input*. The posterior density can be expressed in terms of the channel reliability:

$$\begin{aligned} p(r_t^{(0)}|\tilde{x}_t = \tilde{x}) &= C_1 \exp\left[-\frac{1}{2\sigma^2}(r_t^{(0)} - \tilde{x}\sqrt{E_c})^2\right] \\ &= C_2 \exp\left[-\frac{1}{2\sigma^2}r_t^{(0)2}\right] \exp\left[-\frac{1}{2\sigma^2}E_c\right] \exp\left[\frac{L_c}{2}r_t^{(0)}\tilde{x}\right] \qquad (14.40) \\ &= C_3 \exp\left[\frac{L_c}{2}r_t^{(0)}\tilde{x}\right], \qquad (14.41) \end{aligned}$$

where the C_i are constants which do not depend on $\tilde{x}$.

14.3.13 Statement of the Turbo Decoding Algorithm

Let us combine the equations from the last section together and express iterations of the turbo algorithm. Let $\lambda^{[l,j]}(x_t|\mathbf{r})$ denote the log likelihood ratio computed at the lth iteration of the jth constituent encoder, $j \in \{1, 2\}$. Similarly, let $\lambda_{e,t}^{[l,j]}$ denote the extrinsic probability at the lth iteration for the jth decoder. The basic decomposition is described in (14.34). However, in light of the turbo principle, we replace the prior information $\lambda_{p,t}$ with the appropriate extrinsic information from the other decoder. Furthermore, we express the log posterior $\lambda_{s,t}^{(0)}$ in terms of the soft inputs $L_c r_t^{(0)}$.

Based on this notation, the turbo decoding algorithm (suppressing the interleavers) can be expressed as

$$\begin{aligned} \lambda^{[l,1]}(x_t|\mathbf{r}) &= \underbrace{L_c r_t^{(0)}}_{\text{channel input}} + \underbrace{\lambda_{e,t}^{[l-1,2]}}_{\substack{\text{extrinsic from other}\\ \text{decoder used as prior}}} + \underbrace{\lambda_{e,t}^{[l,1]}}_{\text{new extrinsic}} \\ \lambda^{[l,2]}(x_t|\mathbf{r}) &= \underbrace{L_c r_t^{(0)}}_{\text{channel input}} + \underbrace{\lambda_{e,t}^{[l,1]}}_{\substack{\text{extrinsic from other}\\ \text{decoder used as prior}}} + \underbrace{\lambda_{e,t}^{[l,2]}}_{\text{new extrinsic}} \end{aligned} \tag{14.42}$$

for $l = 1, 2, \ldots, M$, with $\lambda_{e,t}^{[0,2]} = 0$ to represent uniform prior probabilities.

14.3.14 Turbo Decoding Stopping Criteria

The turbo decoding algorithm is frequently run for a fixed number of iterations, which is determined by the worst case noise degradation. Most codewords, however, are not

corrupted by worst case noise and therefore need fewer iterations to converge. A *stopping criterion* is a way of determining if the decoding algorithm has converged so that iterations can be terminated. A properly designed stopping criterion reduces the average number of iterations, while maintaining the same probability of bit error performance.

We introduce here three different stopping criteria.

The Cross Entropy Stopping Criterion

From (14.42) it follows that

$$\lambda^{[l,2]}(x_t|\mathbf{r}) - \lambda^{[l,1]}(x_t|\mathbf{r}) = \lambda_{e,t}^{[l,2]} - \lambda_{e,t}^{[l,1]}.$$

We define $\Delta_{e,t}^{[l]} = \lambda_{e,t}^{[l,2]} - \lambda_{e,t}^{[l,1]}$. From the likelihood $\lambda^{[l,j]}(x_t|\mathbf{r})$ of (14.42), the probability of a bit output can be computed as

$$P^{[l,j]}(\tilde{x}_t = \tilde{x}|\mathbf{r}) = \frac{e^{\tilde{x}\lambda^{[l,j]}}}{1 + e^{\tilde{x}\lambda^{[l,j]}}}. \tag{14.43}$$

Let $P^{[l,j]}(\tilde{\mathbf{x}}|\mathbf{r})$ denote the probability of the entire sequence $\tilde{\mathbf{x}}$,

$$P^{[l,j]}(\tilde{\mathbf{x}}|\mathbf{r}) = P^{[l,j]}(\tilde{x}_0, \tilde{x}_1, \ldots, \tilde{x}_{N-1}|\mathbf{r}).$$

Under the assumption that the elements in $\tilde{\mathbf{x}}$ are statistically independent we have

$$P^{[l,j]}(\tilde{\mathbf{x}}|\mathbf{r}) = \prod_{k=0}^{N-1} P^{[l,j]}(\tilde{x}_k|\mathbf{r}).$$

We define the bit estimate at the lth iteration as $\hat{\tilde{x}}_t^{[l]} = \text{sign}(\lambda^{[l,2]}(x_t|\mathbf{r}))$.

Our first stopping criterion is based on the *cross entropy*, also known as the **relative entropy** or the **Kullback-Leibler distance** introduced in Section 1.12. The cross entropy between two probability distributions P and Q taking on values in some alphabet $\mathcal{A}$ is defined as

$$D(P||Q) = E_P\left[\log \frac{P}{Q}\right] = \sum_{x \in \mathcal{A}} P(x) \log \frac{P(x)}{Q(x)}. \tag{14.44}$$

The cross entropy is a measure of similarity between the two distributions P and Q. From Lemma 1.2 we have that $D(P||Q) = 0$ if and only if $P = Q$, that is, if the distributions are identical.

We use the cross entropy as a measure of similarity between the distributions $P^{[l,2]}$ and $P^{[l,1]}$. Since convergence implies a fixed point of the turbo decoding iterations, at convergence we should have $P^{[l,2]} = P^{[l,1]}$. In practice, we determine convergence has occurred when the cross entropy becomes sufficiently small.

We denote the cross entropy at the lth iteration by $T(l)$:

$$T(l) = D(P^{[l,2]}||P^{[l,1]}) = E_{P^{[l,2]}(\tilde{\mathbf{x}})}\left[\log \frac{P^{[l,2]}(\tilde{\mathbf{x}})}{P^{[l,1]}(\tilde{\mathbf{x}})}\right],$$

where the expectation is with respect to the probability $P^{[l,2]}(\tilde{\mathbf{x}})$. Under the independence assumption,

$$T(l) = \sum_{k=0}^{N-1} E_{P^{[l,2]}(\tilde{x}_k)}\left[\log \frac{P^{[l,2]}(\tilde{x}_k)}{P^{[l,1]}(\tilde{x}_k)}\right]. \tag{14.45}$$

From the definition of the expectation

$$E_{P^{[l,2]}(\tilde{x}_k)}\left[\log \frac{P^{[l,2]}(\tilde{x}_k)}{P^{[l,1]}(\tilde{x}_k)}\right] = P^{[l,2]}(\tilde{x}_k = 1) \log \frac{P^{[l,2]}(\tilde{x}_k = 1)}{P^{[l,1]}(\tilde{x}_k = 1)} + P^{[l,2]}(\tilde{x}_k = -1) \log \frac{P^{[l,2]}(\tilde{x}_k = -1)}{P^{[l,1]}(\tilde{x}_k = -1)}$$

and, using (14.43), it can be shown that

$$E_{P^{[l,2]}(\tilde{x}_k)}\left[\log \frac{P^{[l,2]}(\tilde{x}_k)}{P^{[l,1]}(\tilde{x}_k)}\right] = -\frac{\Delta_{e,k}^{[l]}}{1+\exp[\lambda^{[l,2]}(x_k|\mathbf{r})]} + \log \frac{1+\exp[-\lambda^{[l,1]}(x_k|\mathbf{r})]}{1+\exp[-\lambda^{[l,2]}(x_k|\mathbf{r})]}. \tag{14.46}$$

We now simplify this expectation using approximations which are accurate near convergence. We assume that the decisions do not change from among the different decoders, so

$$\operatorname{sign}(\lambda^{[l,1]}(x_k|\mathbf{r})) = \operatorname{sign}(\lambda^{[l,2]}(x_k|\mathbf{r})) = \hat{\tilde{x}}_k^{[l]}.$$

We invoke the approximation $\log(1+x) \approx x$, which is true when $|x| \ll 1$. Then it can be shown (see Exercise 9) that

$$E_{P^{[l,2]}(\tilde{x}_k)}\left[\log \frac{P^{[l,2]}(\tilde{x}_k)}{P^{[l,1]}(\tilde{x}_k)}\right] \approx \exp(-\lambda^{[l,1]}\hat{\tilde{x}}_k^{[l]})\left(1-\exp(-\hat{\tilde{x}}_k^{[l]}\Delta_{e,t}^{[l]})(1+\hat{\tilde{x}}_k^{[l]}\Delta_{e,t}^{[l]})\right). \tag{14.47}$$

Assuming further that $|\Delta_{e,t}^{[l]}| \ll 1$ we expand $\exp(-\hat{\tilde{x}}_k^{[l]}\Delta_{e,t}^{[l]})$ using the first two terms of its Taylor series expansion and write

$$E_{P^{[l,2]}(\tilde{x}_k)}\left[\log \frac{P^{[l,2]}(\tilde{x}_k)}{P^{[l,1]}(\tilde{x}_k)}\right] \approx \frac{(\hat{\tilde{x}}_k^{[l]}\Delta_{e,t}^{[l]})^2}{\exp(\lambda^{[l,1]}\hat{\tilde{x}}_k^{[l]})}.$$

Substituting this into (14.45) we obtain, using the fact that $(\hat{\tilde{x}}_k^{[l]})^2 = 1$,

$$T(l) \approx \sum_{k=0}^{N-1} \frac{(\Delta_{e,t}^{[l]})^2}{\exp(\lambda^{[l,1]}\hat{\tilde{x}}_k^{[l]})}.$$

Having found this approximate expression, we take as our stopping criterion: Stop if $T(l) <$ some threshold. Taking the threshold to be something in the range of $10^{-2}T(1)$ to $10^{-4}T(1)$ seems appropriate. For example, stopping when $T(l) < 10^{-3}T(1)$ is common.

It has been found experimentally that using this stopping criterion results in at most a few percent error degradation compared to a fixed number of iterations, with the amount of degradation being somewhat higher as the number of fixed iterations increases. At the same time, there is a significant decrease in the average number of iterations, with the amount of improvement being a function of the maximum number of iterations.

The Sign Change Ratio (SCR) Criterion

A stopping criterion which is simpler to compute than the cross entropy can be obtained as follows. Let $C(l)$ denote the number of changes of sign of $\lambda_{e,t}^{[l]}, t = 0, 1, \ldots, N-1$ compared with $\lambda_{e,t}^{[l-1]}$. Experimentally it has been found that if $C(l) < \epsilon N$, where ϵ is typically in the range of 0.005 to 0.03, then the stopping criterion performance is similar to that for the cross entropy criterion.

The Hard Decision Aided (HDA) Criterion

A third stopping criterion is obtained by comparing $\text{sign}(\lambda^{[l,2]}(x_t|\mathbf{r}))$ and $\text{sign}(\lambda^{[l-1,2]}(x_t|\mathbf{r}))$. When the signs are the same for all $t \in \{0, 1, \ldots, N-1\}$, then the decoding stops.

14.3.15 Modifications of the MAP Algorithm

The Max-Log-MAP Algorithm

The MAP algorithm is significantly more complex than the Viterbi algorithm, so that it is of interest to reduce the computational complexity of the MAP algorithm, if possible, even at the expense of some performance. In this section we introduce the max-log-MAP algorithm, which propagates approximations to *logarithms* of the α and β probabilities. This not only avoids some potential roundoff properties, but also has lower complexity than the MAP algorithm. Unfortunately, the algorithm is only approximate, so that some performance is lost. A further modification which recovers the lost performance with a slight increase in computational complexity is then discussed.

Define

$$A_t(p) = \ln(\alpha_t(p)) \qquad B_t(q) = \ln(\beta_t(q)) \qquad \text{and} \qquad \Gamma_t(p,q) = \ln(\gamma_t(p,q)).$$

Let us attempt to develop a recursion for computing $A_{t+1}(q)$. From (14.12) we have

$$\begin{aligned} A_{t+1}(q) = \ln(A_{t+1}(q)) &= \ln\left(\sum_{p=0}^{Q-1} \alpha_t(p)\gamma_t(p,q)\right) \\ &= \ln\left(\sum_{p=0}^{Q-1} \exp\left[A_t(p) + \Gamma_t(p,q)\right]\right). \end{aligned} \tag{14.48}$$

At this point, an approximation is made in the interest of developing a fast algorithm:

$$\ln\left(\sum_i e^{x_i}\right) \approx \max_i x_i. \tag{14.49}$$

Using this approximation in (14.48) we obtain

$$A_{t+1}(q) \approx \max_{p \in \{0,1,\ldots,Q-1\}} (A_t(p) + \Gamma_t(p,q)). \tag{14.50}$$

Thus to find $A_{t+1}(q)$, we add a branch cost $\Gamma_t(p,q)$ to $A_t(p)$, then compute the maximum value of the result over all paths leading to state q. The selected path to state q can then be thought of as the survivor path. The result is exactly the same operation as the Viterbi algorithm! The computational complexity is thus essentially the same as for the Viterbi algorithm: for each pair of merging paths, two additions and one comparison are required, except that the branch cost Γ is a posterior probability for the log-MAP algorithm and is a likelihood for the Viterbi algorithm. In the max-log-MAP algorithm $A_t(p)$ provides the (approximation of the logarithm of the) most probable path through the trellis to state p, rather than the probability of *any* path through the trellis to state p.

The recursion for $B_t(q)$ is obtained as follows:

$$B_t(p) = \ln(\beta_t(p)) = \ln\left(\sum_{q=0}^{Q-1} \gamma_t(p,q)\beta_{t+1}(q)\right)$$
$$= \ln\left(\sum_{q=0}^{Q-1} \exp\left[\Gamma_t(p,q) + B_{t+1}(q)\right]\right)$$
$$\approx \max_{q\in\{0,1,\ldots,Q-1\}} (\Gamma_t(p,q) + B_{t+1}(q)). \tag{14.51}$$

This amounts to a Viterbi algorithm working backward.

The branch metric $\Gamma_t(p,q)$ is computed using (14.27):

$$\Gamma_t(p,q) = \ln(P(x_t = x^{(p,q)})) + \ln(p(r_t^{(0)}|x_t)) + \ln(p(r_t^{(1)}|a^{(1)} = a^{(1,p,q)})).$$

Using (14.38) and (14.39) (and a similar expression for $p(r_t^{(1)}|x_t = 1)$) and throwing away unnecessary constants, we can write

$$\Gamma_t(p,q) = \underbrace{\tilde{x}^{(p,q)}\frac{\lambda_{p,t}}{2}}_{\text{prior}} + \underbrace{\frac{L_c}{2}\tilde{x}^{(p,q)}r_t^{(0)}}_{\text{systematic}} + \underbrace{\frac{L_c}{2}\tilde{v}^{(1,p,q)}r_t^{(1)}}_{\text{parity}}. \tag{14.52}$$

The log posterior $\lambda(x_t|\mathbf{r})$ is, using (14.33),

$$\lambda(x_t|\mathbf{r}) = \log\frac{\sum_{(p,q)\in\mathcal{S}_1}\alpha_t(p)\gamma_t(p,q)\beta_{t+1}(q)}{\sum_{(p,q)\in\mathcal{S}_0}\alpha_t(p)\gamma_t(p,q)\beta_{t+1}(q)}$$
$$= \log\frac{\sum_{(p,q)\in\mathcal{S}_1}\exp\left[A_t(p) + \Gamma_t(p,q) + B_{t+1}(q)\right]}{\sum_{(p,q)\in\mathcal{S}_0}\exp\left[A_t(p) + \Gamma_t(p,q) + B_{t+1}(q)\right]}$$
$$\approx \max_{(p,q)\in\mathcal{S}_1}(A_t(p) + \Gamma_t(p,q) + B_{t+1}(q)) - \max_{(p,q)\in\mathcal{S}_0}(A_t(p) + \Gamma_t(p,q) + B_{t+1}(q)). \tag{14.53}$$

This may be interpreted as follows [141, p. 132]: For each bit x_t, all the transitions from Ψ_t to Ψ_{t+1} are considered, grouped into those which might occur if $x_t = 1$ and those which might occur if $x_t = 0$. For each of these groups the transition which maximizes $A_t(p) + \Gamma_t(p,q) + B_{t+1}(q)$ is found, then the posterior log likelihood is computed based on these two optimal transitions.

If only the $A_t(p)$ values were needed, the max-log-MAP algorithm would have complexity essentially the same as the Viterbi algorithm; however, $B_t(p)$ must also be computed. It has been argued [355] that the complexity of the max-log-MAP algorithm is not more than three times that of the Viterbi algorithm. However, the storage requirements are higher, since the values of $A_t(p)$ and $B_t(p)$ must be stored. The storage can be reduced however, at the expense of an increase in computational complexity [355, 356].

14.3.16 Corrections to the Max-Log-MAP Algorithm

The approximation (14.49) has been shown [295] to result in approximately 0.35 dB of degradation compared to exact decoding. Another algorithm is obtained by using the "Jacobian logarithm":

$$\ln(e^{x_1} + e^{x_2}) = \max(x_1, x_2) + \ln(1 + e^{-|x_1 - x_2|}). \tag{14.54}$$

Let us write this as

$$\ln(e^{x_1} + e^{x_2}) = \max(x_1, x_2) + f_c(\delta) = g(x_1, x_2),$$

where $\delta = |x_1 - x_2|$ and where $f_c(\delta)$ is the "correction" term. Since the $f_c(\delta)$ function depends upon only a single variable, it is straightforward to precompute it and determine its values in use by table lookup. It has been found [295] that sufficient accuracy can be obtained when only eight values of $f_c(\delta)$ are stored for values of δ between 0 and 5.

In order to handle multiple term summations, the functions are composed as follows:

$$\ln\left(\sum_{i=1}^{I} e\right) = g(x_I, g(x_{I-1}, g(x_{I-2}, \ldots, g(x_3, g(x_2, x_1))\cdots))).$$

If a lookup table for f_c is used, the computational complexity is only sightly higher than the max-log-MAP algorithm, but the decoding is essentially *exact*.

14.3.17 The Soft Output Viterbi Algorithm

For turbo decoding, an alternative to MAP-type decoding algorithms (MAP, log-MAP, or max-log-MAP) is the soft output Viterbi algorithm (SOVA). This differs from the conventional Viterbi algorithm in two ways, which in combination make the algorithm suitable for use in decoding turbo codes. First, SOVA uses a modified path metric which takes account of prior probabilities of the input symbols. Second, SOVA produces a *soft* output indicating the reliability of the decision.

Before reading this section, the reader is advised to consult Appendix A, which introduces notation pertaining to arithmetic on log likelihood functions.

Recall that for the conventional Viterbi algorithm, the branch metric $\mu_t(\mathbf{r}_t, \mathbf{x}^{(p,q)}) = \log p(\mathbf{r}_t|\mathbf{x}^{(p,q)})$ was used (see (12.17)), where $p(\mathbf{r}_t|\mathbf{x}^{(p,q)})$ is the likelihood of the output $\mathbf{x}^{(p,q)}$ on the branch from state p at time t to state q at time $t+1$, based on the observation $\mathbf{r}_t$. We can incorporate prior information very simply: Use the logarithm of $\gamma_t(p,q)$ of (14.27) as the branch metric, which is the same as $\Gamma_t(p,q)$ (here assuming a single parity bit and systematic encoding):

$$\mu_t(\mathbf{r}_t, x^{(p,q)}) = \log(p(r_t^{(0)}|x_t)) + \log(p(r_t^{(1)}|a^{(1)} = a^{(1,p,q)})) + \log(P(x_t = x^{(p,q)})),$$

where the first two terms are equivalent to the log likelihood (for this $n = 2$ systematic code), and the third term represents the prior probabilities. As is evident from (14.52), this can be readily computed:

$$\mu_t(\mathbf{r}_t, x^{(p,q)}) = \tilde{x}^{(p,q)}\frac{\lambda_{p,t}}{2} + \frac{L_c}{2}\tilde{x}^{(p,q)}r_t^{(0)} + \frac{L_c}{2}\tilde{v}^{(1,p,q)}r_t^{(1)}.$$

This has an interesting interpretation. When the channel is good, so that L_c is large compared to the prior reliability $|\lambda_{p,t}|$, then the decoder relies more on the channel outputs $\mathbf{r}_t$. On the other hand, when the channel is poor, so that L_c is small compared to the reliability $|\lambda_{p,t}|$, then the decoder relies more on the reliability. Readers familiar with the Kalman filter will notice a similarity to the update of a Kalman filter: the Kalman filter relies more on observations when the observations are more reliable. (This attribute is common to all Bayesian methods.) Using the new branch metric, the path metric $M_t(p)$ is (essentially) equal to $\log p(\mathbf{x}_0^{t-1}|\mathbf{r}_0^{t-1})$, where $\mathbf{x}_0^{t-1}$ denotes the sequence of inputs from time 0 to time t, and $\mathbf{r}_0^{t-1}$ denotes the sequence of observations from time 0 to time $t-1$. We thus can compute the likelihood

$$p(\mathbf{x}_0^{t-1}|\mathbf{r}_0^{t-1}) = Ce^{M_t(p)}. \tag{14.55}$$

Let us consider now how to obtain soft outputs indicating the reliability of the decision for binary codes. Suppose that two paths merge at state q at time $t+1$, having path metrics $M_{t+1}^{(1)}(q)$ and $M_{t+1}^{(2)}(q)$, with $M_{t+1}^{(1)}(q) > M_{t+1}^{(2)}(q)$. The path with metric $M_{t+1}^{(1)}(q)$ is thus selected as the survivor path. Define the path metric difference as

$$\Delta_{t+1}^{q} = M_{t+1}^{(1)}(q) - M_{t+1}^{(2)}(q).$$

The probability of a correct decision is obtained by normalizing the likelihood of the choice by the likelihoods of all competing choices. Using (14.55), we obtain

$$P(\text{correct decision at } \Psi_{t+1} = q) = \frac{e^{M_{t+1}^{(1)}(q)}}{e^{M_{t+1}^{(1)}(q)} + e^{M_{t+1}^{(2)}(q)}} = \frac{e^{\Delta_{t+1}^{q}}}{1 + e^{\Delta_{t+1}^{q}}}.$$

The log likelihood ratio is

$$\ln \frac{P(\text{correct decision at } \Psi_{t+1} = q)}{1 - P(\text{correct decision at } \Psi_{t+1} = q)} = \Delta_{t+1}^{q}. \tag{14.56}$$

Thus the path metric difference where the paths merge in the Viterbi algorithm is equal to the log likelihood ratio of the probability that the decision is correct.

Application of this concept is somewhat complicated because of delayed decisions. Recall that the Viterbi algorithm typically makes a decision about a bit $\tilde{x}_t$ after some window of decoding delay, typically about five constraint lengths. Let us denote the decoding delay by δ. At time t, a decision is made about a bit $\tilde{x}_{t-\delta}$. Consider the window on the trellis in Figure 14.9, where $\delta = 6$. (For simplicity, not all paths to time t are shown.) The surviving path sequence selected at time t, denoted as $\boldsymbol{\Psi}_t$, is shown with a bold line; its metric is denoted as $M_t^{(1)}(p_t)$. The δ paths which were discarded by the Viterbi algorithm over this window are also shown. Let Δ_t^l denote the path metric difference between the metric along the surviving path $\boldsymbol{\Psi}_t$ and the paths discarded by the Viterbi algorithm at a lag of l steps back from t, $l = 0, 1, \ldots, \delta - 1$. We here refer to the path which was discarded at time $t - l$ as the lth path. Let $\tilde{x}_{t-\delta}$ denote the input bit along the selected path at time $t - \delta$, and let $\tilde{x}_{t-i}^l$ denote an input bit at time $t - i$ along the lth path.

If the bit $\tilde{x}_{t-\delta}^l$ on discarded path l is equal to $\tilde{x}_{t-\delta}$, then we would have made no bit error if we would have selected the discarded path. In this case, the reliability of this bit decision is ∞. If $\tilde{x}_{t-\delta}^l \neq \tilde{x}_{t-\delta}$, then there would be a bit error along the lth path, which we denote as

$$\tilde{e}_{t-\delta}^l = \tilde{x}_{t-\delta} \oplus \tilde{x}_{t-\delta}^l.$$

Here, $\oplus$ denotes addition in $GF(2)$, with identity 1:

$$1 \oplus 1 = 1 \qquad 1 \oplus -1 = -1 \qquad -1 \oplus 1 = -1 \qquad -1 \oplus -1 = 1. \tag{14.57}$$

The log likelihood value of the bit error is, by (14.56), equal to Δ_t^l. Combining these two cases we have

$$\lambda(\tilde{e}_{t-\delta}^l) = \log \frac{P(\tilde{e}_{t-\delta}^l = 1)}{P(\tilde{e}_{t-\delta}^l = -1)} = \begin{cases} \infty & \text{if } \tilde{x}_{t-\delta} = \tilde{x}_{t-\delta}^l \\ \Delta_t^l & \text{if } \tilde{x}_{t-\delta} \neq \tilde{x}_{t-\delta}^l. \end{cases} \tag{14.58}$$

Each path provides evidence about the likelihood that $\tilde{x}_{t-\delta}$ is correctly decoded. The total error resulting from the sum of all possible discarded paths for $\tilde{x}_{t-\delta}$ is

$$\tilde{e}_{t-\delta} = \sum_{l=0}^{\delta-1} \oplus \tilde{e}_{t-\delta}^l.$$

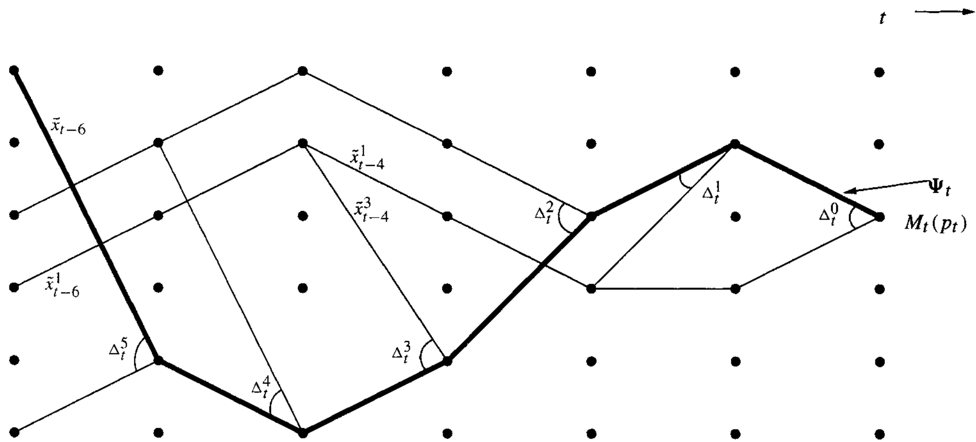


Figure 14.9: Trellis with metric differences and bits for SOVA.

We take the log likelihood ratio of $\tilde{x}_{t-\delta}$ to be

$$\lambda(\tilde{x}_{t-\delta}) = \tilde{x}_{t-\delta} \sum_{l=0}^{\delta-1} \boxplus \lambda(\tilde{e}^l_{t-\delta}), \tag{14.59}$$

where the factor $\tilde{x}_{t-\delta}$ sets the sign of the likelihood ratio and the sum of the errors represents the accumulation of evidence.

By (A.4) and (14.58), this sum is over only those indices l where $\tilde{x}_{t-\delta} \neq \tilde{x}^l_{t-\delta}$. Using (14.58) we can write (14.59) as

$$\lambda(\tilde{x}_{t-\delta}) = \tilde{x}_{t-\delta} \sum_{l=0}^{\delta-1} \boxplus \Delta^l_t.$$

Finally, using the approximation of (A.5), we can write

$$\lambda(\tilde{x}_{t-\delta}) \approx \tilde{x}_{t-\delta} \min_{l \in \{0,1,\ldots,\delta-1\}} \Delta^l_t. \tag{14.60}$$

The reliability of the bit decision for $\tilde{x}_{t-\delta}$ thus depends on the least reliable path decision which determines the path selection.

Implementation of the SOVA algorithm requires storing not only the path metric to each state, but also the metric difference Δ_t. The sequence $\tilde{e}^l_{t-i}$ is also updated for each decision. When a decision is made, the reliability of the decision is produced according to (14.60).

14.4 On the Error Floor and Weight Distributions

In this section we discuss briefly two questions relating to the performance of turbo codes. These questions are: Why is there an error floor? and What makes turbo codes so effective? Other discussions along these lines appear in [303].

14.4.1 The Error Floor

As observed from the plot in Figure 14.1, there is an error floor associated with turbo codes, so that for moderate to high SNRs, the probability of error performance fails to drop off as

rapidly as it does for low SNRs. This can be explained as follows. The probability of bit error for a turbo code can be approximated just as for convolutional codes. Thinking of the set of codes as block $(N/R, N)$ codes, there are 2^N codewords. Then the probability of bit error can be bounded as [303, p. 243]

$$P_b \leq \sum_{i=1}^{2^N} \frac{w_i}{N} Q\left(\sqrt{\frac{2d_i R E_b}{N_0}}\right),$$

where w_i is the weight of the message sequence of the ith message and d_i is the Hamming weight of the codeword. Grouping together codewords of the same Hamming weight, the bound on the probability of bit error can be written as

$$P_b \leq \sum_{d=d_{\text{free}}}^{N/R} \frac{W_d}{N} Q\left(\sqrt{\frac{2dRE_b}{N_0}}\right) = \sum_{d=d_{\text{free}}}^{N/R} \frac{N_d \tilde{w}_d}{N} Q\left(\sqrt{\frac{2dRE_b}{N_0}}\right), \tag{14.61}$$

where N_d is the multiplicity of codewords of weight d, and

$$\tilde{w}_d = \frac{W_d}{N_d},$$

where W_d is the total weight of all message sequences whose codewords have weight d. Thus $\tilde{w}_d$ is the average weight of the message sequences whose codewords have weight d. The quantity d_{free} is the free distance of the code, the minimum Hamming distance between codewords. The upper limit N/R of the summation comes from neglecting the length of the zero-forcing tail, if any.

As the SNR increases, the first term of the sum in (14.61) dominates. The asymptotic performance of the code is thus

$$P_b \approx \frac{N_{\text{free}} \tilde{w}_{\text{free}}}{N} Q\left(\sqrt{\frac{d_{\text{free}} 2RE_b}{N_0}}\right),$$

where N_{free} is the number of sequences at a distance d_{free} from each other and $\tilde{w}_{\text{free}}$ is the average weight of the message sequence causing the free-distance codewords. When plotted on a log-log scale (e.g., logarithmic probability with E_b/N_0 in dB), the slope of P_b is determined by d_{free}. If there is a small d_{free}, then the slope is small.

The error floor, which appears at higher SNRs, is thus ostensibly due to the presence of a small d_{free}, that is, due to low weight codewords.

Why should there be low weight codewords in a turbo code? We note first that the presence of a single 1 in the input sequence $\mathbf{x}$ can lead to many nonzero values in its parity sequence. If $x_t = 1$ for some t and $\mathbf{x}$ is zero at all other positions, then the encoder leaves the zero state when x_t arrives. Since the encoder is recursive, the remaining sequence of input zeros does not drive the encoder to the 0 state, so a sequence of 0s and 1s continues to be produced by the encoder. For example, for the encoder whose trellis is shown in Figure 14.4(b), a 1 followed by a string of 0s produces the parity sequence $\{1, 0, 1, 0, 1, 0, \ldots\}$. This alternating sequence output is typical of many recursive encoders. Having a high weight code sequence for a low weight input is one reason why recursive encoders are used in turbo encoders. If $x_t = 1$ happens to occur near the end of the input sequence, then the parity sequence $\mathbf{v}^{(1)}$ has low weight. But the interleaver may produce a sequence $\mathbf{x}'$ whose nonzero value occurs earlier, resulting in a parity sequence $\mathbf{v}^{(2)}$ of higher weight. From

one point of view, this is one of the sources of strength of turbo encoders: if one of the parity sequences has low weight, then there is a reasonable probability that the other parity sequence has higher weight. There are thus codewords with high weight.

Consider now a low-weight input sequence $\mathbf{x}$ that results in a low weight parity sequence in $\mathbf{v}^{(1)}$. For example, a sequence of all zeros, followed by a single 1, so that $x_{N-1} = 1$. This would result in a parity sequence with $\text{wt}(\mathbf{v}^{(1)}) = 1$. More generally, there might be a single 1 appearing somewhere near the end of $\mathbf{x}$. This would result in a low weight $\mathbf{v}^{(1)}$. When $\mathbf{x}$ is permuted, the resulting sequence may have the 1 appearing early in the sequence, causing the second encoder to leave the 0 state early on, after which, as mentioned, a parity sequence $\mathbf{v}^{(2)}$ of appreciable weight might be produced.

But circumstances may make it so that the second parity sequence also results in a low-weight codeword. In the first case, suppose that the parity sequence $\mathbf{v}^{(2)}$ is in fact a $\{1, 0\}$ alternating sequence, and that the parity sequences are now punctured with a puncturing phase that punctures all the 1s. All the weight from $\mathbf{v}^{(2)}$ is removed, so the weight of the codeword depends only upon the weight of $\mathbf{x}$ and $\mathbf{v}^{(1)}$, which may be very low.

A low weight codeword could also be obtained another way. If the interleaver is such that the single 1 appearing near the end of $\mathbf{x}$ also happens to appear near the end of $\mathbf{x}'$, then regardless of the puncturing, both $\mathbf{v}^{(1)}$ and $\mathbf{v}^{(2)}$ are low weight.

Thus it may occur that a sequence $\mathbf{x}$ which produces a low weight $\mathbf{v}^{(1)}$ can, after interleaving, produce a sequence $\mathbf{x}'$ which would also produce a low weight $\mathbf{v}^{(2)}$. At this point in the state of the art, methods of designing encoders and/or interleavers which completely avoid the low weight codeword problem are unknown.

14.4.2 Spectral Thinning and Random Interleavers

The difficulties of low-weight codewords notwithstanding, turbo codes are outstanding performers. This is because, while there are low weight codewords, there are not many of them! As mentioned, the interleaver helps ensure that if one parity sequence has low weight, the other has higher weight with high probability.

The *distance spectrum* of a code is a listing of the (N_d, W_d) information as a function of the codeword weight d, where N_d, again, is multiplicity of codewords at weight d, and W_d is the total weight of the message sequences producing codewords of weight d. Turbo codes are said to have a *sparse* distance spectrum if the multiplicities of the lower-weight codewords is relatively small. Since each term in the probability bound in (14.61) is scaled by the multiplicity N_d, higher multiplicities result in more contribution to the probability of error, so that a higher SNR must be achieved before the probability of error term becomes negligible.

For example, for the (37,21,65536) code, the distance spectrum computed using weight-2 message sequences, when the set of turbo codes is averaged over all possible interleavers, is [303]

d	N_d	W_d
6	4.5	9
8	11	22
10	20.5	41
12	75	150

(This data was found using the algorithm described in [306].) Note that N_d increases relatively slowly with d. Convolutional codes, on the other hand, frequently are *spectrally*

dense, meaning that N_d increases much more rapidly with d.

We now argue that the sparse distance spectrum is typical for long interleavers. The argument is based on enumerating aspects of the weight behavior of the codes, averaged over the set of all possible random interleavers. This argument is referred to as *random interleaving* [19, 303]. The sparse distance spectrum for turbo codes, compared to the distance spectrum of convolutional codes, is referred to as *spectral thinning*.

To characterize the weight spectrum, define the input redundancy weight enumerating function (IRWEF) $A(W, Z)$ [21]. The IRWEF $A(W, Z)$ is defined as

$$A(W, Z) = \sum_w \sum_z A_{w,z} W^w Z^z,$$

where $A_{w,z}$ is the number of codewords produced by message sequences $\mathbf{x}$ of weight w and parity sequences $\mathbf{v}^{(1)}$ and $\mathbf{v}^{(2)}$ of combined weight z. The quantities W and Z are formal variables used in the series expansion. Our interest here is not in the entire IRWEF $A(W, Z)$, but in the relationship between the low weight codewords of the turbo code and $A_{w,z}$. This requires enumerating possible state sequences in the constituent encoders.

In the first encoder, a message sequence $\mathbf{x}$ of weight w gives rise to a sequence of states $\boldsymbol{\Psi}^{(1)}$. We say that a *detour* occurs in the state sequence if a contiguous sequence of states deviates from the zero state then returns to the zero state. Let n_1 denote the number of detours in $\boldsymbol{\Psi}^{(1)}$ and let l_1 denote the total length of the detours. Let $d_1 = w + z_1$ denote the weight of the message and first parity word. Similarly, the permuted sequence $\mathbf{x}'$ gives rise to a state sequence $\boldsymbol{\Psi}^{(2)}$ in the second encoder; we denote the number of detours and the total length of the detours in $\boldsymbol{\Psi}^{(2)}$ by n_2 and l_2, respectively. Let $d_2 = w + z_2$ denote the weight of the message and second parity word.

Example 14.3 Let $\mathbf{x} = [0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1]$ and the interleaved bits $\mathbf{x}' = [1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0]$ be applied to the encoder of Example 14.1[2]. The parity sequences are

$$\mathbf{v}^{(1)} = [0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0] \qquad \mathbf{v}^{(2)} = [1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1].$$

Then $d_1 = 7$ and $d_2 = 12$. The combined weight of the parity bits is $z = 11$. The presence of this codeword contributes one codeword "count" to the coefficient $A_{4,12}$.

Figure 14.10 shows the state sequences for this codeword. The state sequence $\boldsymbol{\Psi}^{(1)}$ has $n_1 = 2$ detours whose total length is $l_1 = 8$; the state sequence $\boldsymbol{\Psi}^{(2)}$ has $n_2 = 2$ detours whose total length is $l_2 = 13$. □

Suppose the interleaver of length N is chosen at random, so there are $N!$ possible interleavers. If we assume that an interleaver is chosen according to a uniform distribution, then the probability of choosing any particular interleaver is $1/N!$. Suppose the message sequence $\mathbf{x}$ has weight w. Then the permuted sequence $\mathbf{x}'$ also has weight w. Since all 1 bits of $\mathbf{x}$ are indistinguishable from each other, and similarly all 0 bits, there are $w!(N-w)!$ interleavers out of the $N!$ that could all produce the same permuted sequence $\mathbf{x}'$. The probability that the mapping from $\mathbf{x}$ to $\mathbf{x}'$ occurs (where both have the same weight) is

$$\frac{w!(N-w)!}{N!} = \frac{1}{\binom{N}{w}}.$$

This is also the probability of occurrence of the codeword that results from the input sequences $\mathbf{x}$ and $\mathbf{x}'$.

[2]The interleaver is different from that example, since the length of the code is different.

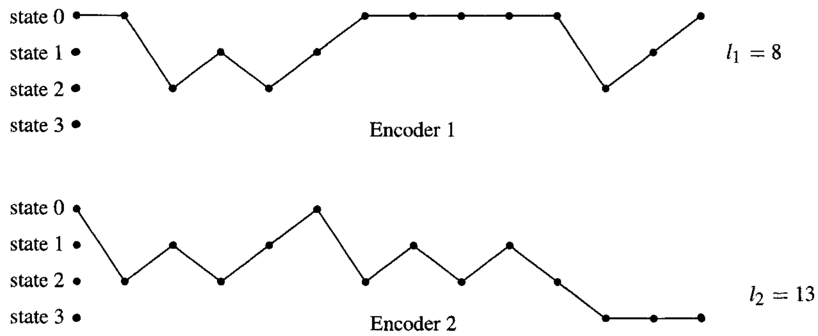


Figure 14.10: State sequences for an encoding example.

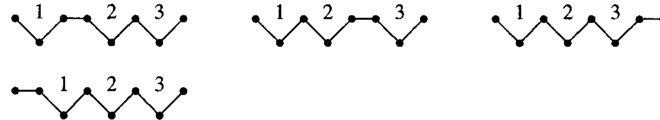


Figure 14.11: Arrangements of $n_1 = 3$ detours in a sequence of length $N = 7$.

Consider a sequence $\mathbf{x}$ of weight w, with corresponding encoder state sequences $\boldsymbol{\Psi}^{(1)}$ and $\boldsymbol{\Psi}^{(2)}$ and having parity weights z_1 and z_2. Since the particular codeword occurs with probability $1/\binom{N}{w}$, the contribution to A_{w,z_1+z_2}, averaged over all random interleavers, is

$$\frac{1}{\binom{N}{w}}.$$

The sequence of zeros connecting any two distinct state sequence detours has no effect on either the weight of the message sequence or its parity sequence. The detours can be moved around within the state sequence, without changing their order, without changing their contribution to A_{w,z_1+z_2}. Enumerating all the possible ways that the detours can be moved around, there are

$$\binom{N - l_1 + n_1}{n_1}$$

distinct ways that the n_1 detours can be arranged, without changing their order. Each of these therefore results in a contribution to A_{w,z_1+z_2}.

Example 14.4 Figure 14.11 shows the different ways that $n_1 = 3$ detours (each of length 2, so that $l_1 = 6$) can be arranged, in order, in a sequence of length $N = 7$. There are

$$\binom{7-6+3}{3} = 4$$

different arrangements. □

This applies to the first constituent encoder. For the second constituent encoder, the number of possible arrangements depends on whether the encoder ends in the 0 state. If the second encoder ends in the 0 state, then there are

$$\binom{N - l_2 + n_2}{n_2}$$

ways that the detours in $\Psi^{(2)}$ can be arranged in order, each of which contributes to the same A_{w,z_1+z_2}. If the second encoder does not return to the 0 state, then the last of the n_2 "detours" is not a true detour. There are thus

$$\binom{N - l_2 + (n_2 - 1)}{n_2 - 1}$$

ways the detours can be arranged, each of which contributes a codeword count to A_{w,z_1+z_2}.

The overall *average* contribution to A_{w,z_1+z_2} for a particular pattern of detours in $\Psi^{(1)}$ and $\Psi^{(2)}$ is

$$\frac{\binom{N-l_1+n_1}{n_1}\binom{N-l_2+n_2}{n_2}}{\binom{N}{w}} \tag{14.62}$$

if the last "detour" of the second encoder ends in the 0 state, or

$$\frac{\binom{N-l_1+n_1}{n_1}\binom{N-l_2+n_2-1}{n_2-1}}{\binom{N}{w}} \tag{14.63}$$

if the second encoder does not end in the 0 state.

Since our intent here is to explore codewords of low weight, we now assume that $n_1 \ll N$, $l_1 \ll N$, $n_2 \ll N$, $l_2 \ll N$, and $w \ll N$. (Because otherwise there would be either a large number of short detours or a few very long detours, either of which would be unlikely to result in codewords of low weight.) Under this assumption, the contribution to A_{w,z_1+z_2} of (14.62) can be approximated as

$$\frac{w!}{n_1!n_2!}N^{n_1+n_2-w} \tag{14.64}$$

and (14.63) can be approximated as

$$\frac{w!}{n_1!(n_2-1)!}N^{n_1+n_2-w-1}. \tag{14.65}$$

Each detour in the state sequence must be caused by a message sequence whose weight is at least 2 (i.e., one message bit to deviate from the 0 state, and one message bit to return back to the zero state), so $w \geq 2\max(n_1, n_2)$. In (14.64), as the block length (and interleaver length) approaches ∞, the exponent $N^{n_1+n_2-w} \to 0$ unless $w = n_1 + n_2$ and $n_1 = n_2$. In (14.65), $N^{n_1+n_2-w-1} \to 0$ as $N \to \infty$ for any values of n_1 and n_2. Thus the following conditions must be met by the codeword in order for the codeword to contribute to A_{w,z_1+z_2}:

1. The second encoder must terminate in the all zero state.
2. Both encoders must make the same number of detours.
3. Each state detour is caused by a message sequence of weight 2.

If these conditions are not all met, then asymptotically $A_{w,z}$ receives no contribution from the codeword.

The result of this is that, for large enough N, $A_{w,z}$ for low-weight codewords is rather small: the conditions simply are not met very often. Thus the distance spectrum for the code is "thinned."

One result of the thinned spectrum is that there are relatively few codewords of low weight, hence relatively few codewords near to other codewords. Thus codewords selected at random will, with high probability, be decoded correctly. However, when errors occur,

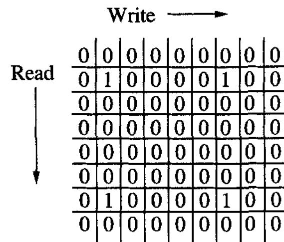


Figure 14.12: A 6 × 6 "square" sequence written into the 120 × 120 interleaver.

they tend to occur in clusters, since a decoding failure can cause several stages of the trellis to be corrupted in the BCJR algorithm. This observation is borne out in simulation: when the decoder is run for many iterations at low probability of error, most blocks are completely error free and errors, when they occur, tend to appear multiple times in the block.

14.4.3 On Interleavers

The interleaver is a key component of the turbo encoder, since it allows the extrinsic information passed into a decoder to be nearly independent of the observed data in the decoder. As we now argue, a rectangular interleaver, which is probably the easiest from an implementation point of view, leads to degraded coder performance compared to a (pseudo-) random interleaver, because it can lead to a large value of N_{free}. Thus it is important to use an interleaver which is closer to a true random interleaver.

We observe that the interleaved message sequence $\mathbf{x}' = \Pi\mathbf{x}$ has the same weight as the original message sequence $\mathbf{x}$. In the general case, the parity sequences $\mathbf{v}^{(1)}$ and $\mathbf{v}^{(2)}$ are different, however, because the inputs to the constituent convolutional encoders are different. Thus if $\mathbf{v}^{(1)}$ is a low-weight parity sequence, it may be hoped that $\mathbf{v}^{(2)}$ has higher weight. However, if the interleaved sequence $\mathbf{x}'$ not only has the same weight as $\mathbf{x}$, but is in fact equal to $\mathbf{x}$, and if $\mathbf{v}^{(1)}$ is low weight, then $\mathbf{v}^{(2)}$ has the same low weight, resulting in an overall low-weight codeword. Furthermore, as we show, a rectangular interleaver provides the possibility for many such low-weight codewords, resulting in a large N_{free}.

We give a specific example based on a code using the transfer function in (14.1) in a rate $R = 1/2$ code. Suppose that a rectangular (or square) interleaver is used in the turbo encoder, so that the message data $\mathbf{x}$ is written row by row, and read out column by column. Suppose, to be specific, that a 120×120 interleaver is used, resulting in a block code of length $N = 120^2 = 14400$ [303]. As will be shown, N_{free} =28900 for this code. When compared with the results for the $N = 65536$ code using a (pseudo-) random interleaver, the performance is about 2 dB worse at a probability of bit error of 10^{-5}. Some of the difference can be attributed to the shorter codeword length, but more significant is the fact that N_{free} is so large.

Consider a message sequence

$$\mathbf{x} = [\ldots, 1, 0, 0, 0, 0, 1, 0, 0, \ldots, 0, 1, 0, 0, 0, 0, 1, 0, \ldots],$$

where there are zeros such that the four ones form a 6×6 square in the interleaver, as shown in Figure 14.12. Thus the interleaved sequence $\mathbf{x}' = \Pi\mathbf{x}$ is equal to $\mathbf{x}$. The encoded parity sequence $\mathbf{v}^{(1)}$ has weight 4 after puncturing. Since $\mathbf{x}'$ is equal to $\mathbf{x}$, the parity sequence $\mathbf{v}^{(2)}$ also has weight 4. The entire codeword has weight $4 + 4 + 4 = 12$. There are

$(120 - 5) \times (120 - 5) = 13325$ different positions where the square pattern can be placed in the interleaver. There are also rectangular message patterns of size 11×6, 6×11, and 11×11 which also encode to sequences of weight 12. For the 6×11 and 11×6 patterns, the weight depends on the "phase" of the puncturing, depending on which parity sequence is punctured first, so only half of the positions result in the codeword of weight 12. There are thus

$$2 \times \frac{1}{2}(120 - 10) \times (120 - 5) = 12650$$

different input sequences producing a weight-12 codeword. For the 11×11 pattern, the weight of both sequences is affected by which is punctured first. As a result, only one-fourth of the possible input patterns result in a weight-12 codeword, so there are $\frac{1}{4}(120 - 10) \times (120 - 10) = 3025$ different codewords producing this pattern. Adding this up, we see that there are 28,900 weight-12 codewords.

It is clear that, for this example, increasing the size of the interleaver only results in more minimum codeword patterns, resulting in a larger N_{free}. In fact, N_{free} grows roughly linearly with N, so the effective multiplicity N_{free}/N does not change significantly for larger rectangular interleavers.

While this example was described for a particular code, the principles apply fairly generally. Attempts to design some kind of structured interleaver to reduce the implementation complexity frequently destroys the very randomness needed to obtain good performance at low SNRs.

14.5 EXIT Chart Analysis

In this section we introduce the extrinsic information transfer (EXIT) chart, a powerful method for analyzing iteratively decoded codes. While we present it here in the context of turbo codes, it can also be used for LDPC code analysis (see Section 15.9). The EXIT chart provides a means of characterizing a code which is both faster and more insightful than simulating the code. It reveals that there is a decoding threshold, an SNR below which correct decoding cannot be expected. EXIT analysis can also be used to search for good codes, or codes whose decoding converges quickly. It can also be used to approximate the probability of error in some regions of the curve.

The EXIT chart is expressed in terms of a likelihood ratio decoder. For our purposes, it will be convenient to use the labeling shown in Figure 14.13. The *a priori* information is labeled as A_i, $i = 1$ or 2, depending on which decoder is used. The extrinsic information is E_i, the decoder output information is D_i, and the soft input information is Z_i, with $D_i = Z_i + A_i + E_i$. We will also denote the transmitted information — the $\tilde{x}$ bits — as X. The key concept of the EXIT chart is measuring the amount of information that the prior A conveys about the transmitted data X and that the extrinsic information E conveys about the data information X. This information is measured using the *mutual information* in the form of $I(X; A_i)$ and $I(X; E_i)$. (Mutual information was introduced in Section 1.12.) To quantify these, it is necessary to model the distributions of the A_i and E_i data.

The soft output Z_i from the channel is obtained from the log likelihood ratio

$$Z_i = \log \frac{p(r_t|\tilde{x}_t = 1)}{p(r_t|\tilde{x}_t = -1)}$$

(where r_t generically represents either systematic or parity information). Following (14.39),

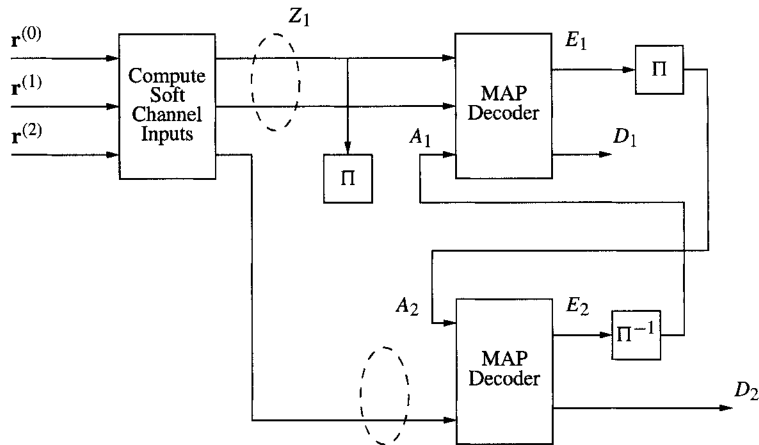


Figure 14.13: Variables used in iterative decoder for EXIT chart analysis.

we have

$$Z = L_c r_t = L_c(\sqrt{E_c}\tilde{x}_t + n),$$

where $n \sim \mathcal{N}(0, \sigma^2)$, with $\sigma^2 = N_0/2$ and $L_c = 2\sqrt{E_c}/\sigma^2$. We can write

$$Z = \mu_Z \tilde{x}_t + n_Z,$$

where $\mu_Z = 2E_c/\sigma^2$ and $n_Z \sim \mathcal{N}(0, \sigma_Z^2)$, where

$$\sigma_Z^2 = 4E_c/\sigma^2 = 2\mu_Z.$$

That is, we have $\sigma_Z^2 = 2\mu_Z$. A Gaussian distribution having the variance twice the mean is said to be *consistent*.

We make the following assumptions for the analysis:

1. For sufficiently large interleavers, the *a priori* values A_i are fairly uncorrelated from their respective channel observations Z_i over many iterations.
2. The probability density functions of the extrinsic information E_i — which are the prior inputs for the next decoder — approach Gaussian distributions with increasing iterations.

Mutual Information Between X and A Under these assumptions, we model the *a priori* probability input A_i to an encoder as

$$A_i = \mu_A \tilde{x}_t + n_A, \tag{14.66}$$

where n_A is an independent Gaussian random variable with variance σ_A^2 and zero mean. We assume that A_i is also consistent, so $\sigma_A^2 = 2\mu_A$. Then the conditional pdf of A can be written as

$$p_{A|X}(y|\tilde{x}) = \frac{1}{\sqrt{2\pi}\sigma_A} \exp\left[-\frac{1}{2\sigma_A^2}(y - (\sigma_A^2/2)\tilde{x})^2\right]. \tag{14.67}$$

Using the Kullback-Leibler distance introduced in (14.44), the mutual information $I(X;A)$ can be computed as (see Section 1.12)

$$I(X;A) = D(p_{XA}||p_X p_A) = \int p_{XA}(\tilde{x}, y) \log_2 \frac{p_{XA}(\tilde{x}, y)}{p_X(\tilde{x}) p_A(y)} d\tilde{x} dy,$$

where p_{XA} is the joint probability distribution of $\tilde{x}$ and A, and p_X and p_A are the marginal distributions. We assume that the $\tilde{x}$ occurs with equal probability for $\tilde{x} \in \{\pm 1\}$ and that, as mentioned, A is conditionally Gaussian. Then

$$\begin{aligned} I(X;A) &= \int p_{A|X}(y|\tilde{x}) p_X(\tilde{x}) \log_2 \frac{p_{A|X}(y|\tilde{x}) p_X(\tilde{x})}{p_X(\tilde{x}) p_A(y)} d\tilde{x} dy \\ &= \frac{1}{2} \int_{-\infty}^{\infty} \sum_{\tilde{x} \in \{\pm 1\}} p_{A|X}(y|\tilde{x}) \log_2 \frac{2 p_A(y|\tilde{x})}{p_A(y|1) + p_A(y|-1)} dy \qquad (14.68) \\ &= 1 - \int_{-\infty}^{\infty} \frac{e^{-\frac{1}{2\sigma_A^2}(y - \sigma_A^2/2)^2}}{\sqrt{2\pi}\sigma_A} \log_2(1 + e^{-y}) \, dy. \qquad (14.69) \end{aligned}$$

We will denote this as $I_A(\sigma_A) \triangleq I(X;A)$. Since this is information regarding a binary-valued random variable, we have the limits $0 \leq I(X;A) \leq 1$. We will furthermore define the function $J(\sigma) = I_A(\sigma_A)|_{\sigma_A = \sigma}$. Since this is mutual information, it can be shown that $J(\sigma)$ is a monotonic function of σ, so that there is an inverse:

$$\sigma_A = J^{-1}(I_A). \qquad (14.70)$$

Mutual Information Between X and E We can similarly write the mutual information between X and E. Following (14.68) we have

$$I_E = I(X;E) = \frac{1}{2} \int_{-\infty}^{\infty} \sum_{\tilde{x} \in \{\pm 1\}} p_{E|X}(y|\tilde{x}) \log_2 \frac{2 p_{E|X}(y|\tilde{x})}{p_{E|X}(y|1) + p_{E|X}(y|-1)} dy, \qquad 0 \leq I_E \leq 1. \qquad (14.71)$$

In this case, we do not consider E to be a Gaussian random variable. Instead, to compute I_E, a simulated channel is used to produce data which are passed through a stage of the decoder. Then, the extrinsic output of the decoder is used to estimate $p_E(y|\tilde{x})$ by creating a histogram of the extrinsic outputs. This estimated density is numerically integrated to produce I_E in (14.71). In this simulation, the parameter σ_A corresponding to some value of I_A via (14.70) is selected, and a Gaussian input vector A is generated according to (14.66), which is then passed through the BCJR algorithm at some SNR E_b/N_0. There is thus some functional relationship between I_A, E_b/N_0, and I_E, denoted abstractly as

$$I_E = T(I_A, E_b/N_0)$$

or, for a fixed E_b/N_0, simply as

$$I_E = T(I_A).$$

This function T denotes the "transfer" of information from the prior information A at the input of a decoder to the extrinsic information E at the output of the decoder, which, in the turbo decoding scheme, is then used as the prior input at the next decoder.

Figure 14.14 illustrates the qualitative shape of the function $T(I_A)$ for various values of SNR. As the SNR increases, the I_E available at the output of the decoder increases.

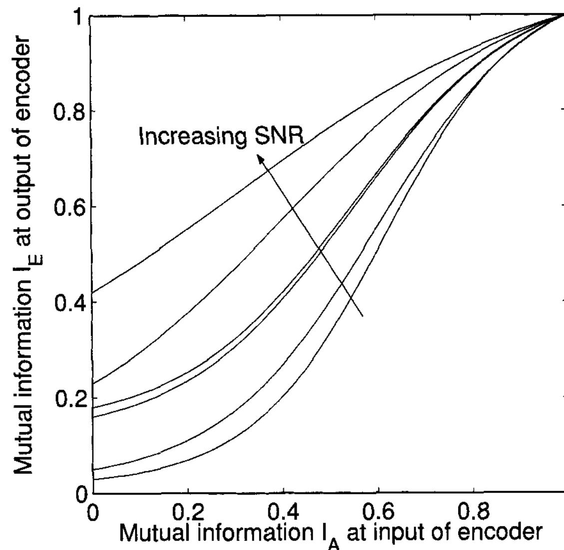


Figure 14.14: Qualitative form of the transfer characteristic $I_E = T(I_A)$.

14.5.1 The EXIT Chart

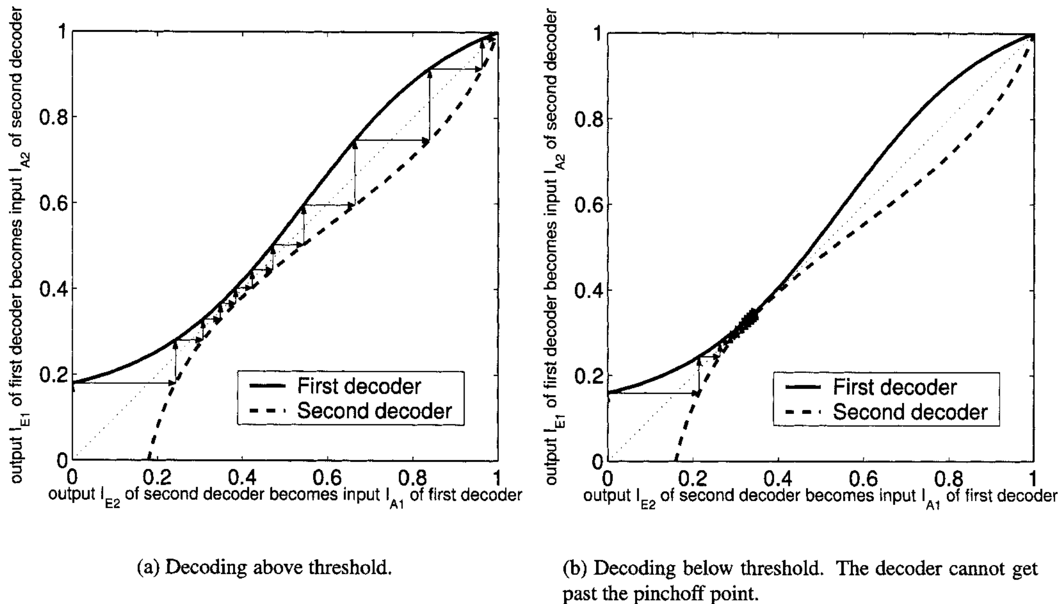
The plot in Figure 14.14 shows the mutual information I_E at the output of a single decoder as a function of the mutual information I_A at the input of the decoder. Let us now consider how this curve affects an iterative decoder. The extrinsic information E_1 at the output of the first decoder is permuted and used as the prior information A_2 at the next decoder. Let $I_{A_1}^{[n]}$ denote the mutual information $I(X; A_1)$ at the nth iteration of first the decoder, starting with zero *a priori* knowledge $I_{A_1}^{[0]} = 0$. Similarly let $I_{E_1}^{[n]} = I(X; E_1)$ denote the mutual information $I(X; E_1)$ at the output of the first decoder at the nth iteration, $I_{E_1}^{[n]} = T_1(I_{A_1}^{[n]})$. This is forwarded to the next decoder to become $I_{A_2}^{[n]} = I_{E_1}^{[n]}$. This passes through the second decoder to become $I_{E_2}^{[n]} = T_2(I_{A_2}^{[n]})$, which in turn is passed back to the first decoder as the prior, $I_{A_1}^{[n+1]} = I_{E_2}^{[n]}$. (Note that interleaving or de-interleaving does not change the mutual information.)

To portray this iteration graphically, the mutual information function $I_{E_1} = T(I_{A_1})$ is reflected across the line $y = x$ and plotted, so that the abscissa and ordinate of the plot are interchanged. Then starting at $I_{A_1}^{[0]} = 0$, each ordinate becomes an abscissa for the next iteration. Figure 14.15(a) shows the information decoding in a sequence of decoding steps, following the arrows. Ultimately (in this case), a point is reached where $I_A = 1$. If the prior information about a bit is sufficiently close to 1, then we conclude that the prior information is sufficient to accurately decode the bit. Thus, in this case, the decoder iterates until a correct decoding occurs. There is a "channel" or gap between the two curves in Figure 14.15(a). The decoding proceeds by walking through this channel.

Figure 14.15(b) shows how the iterative decoding process can break down. In this case $T(I_A)$ is plotted for a lower SNR, producing a function $T(I_A)$ which crosses the $y = x$ line. As a result, the iterations get stuck at the crossover point. The decoder is unable to "exit"

the channel in the EXIT chart.

Clearly, there is a threshold phenomenon taking place: for a sufficiently large SNR, after a sufficiently large number of iterations the decoder is able to decode correctly. For a sufficiently small SNR, the "channel" in the graph shuts down. The decoder never reaches the point that there is sufficient information about X in the extrinsic information to be able to correctly decode, no matter how many times the decoder iterates.



(a) Decoding above threshold.

(b) Decoding below threshold. The decoder cannot get past the pinchoff point.

Figure 14.15: Trajectories of mutual information in iterated decoding. The iterations follow the arrows.

Clearly, the farther the EXIT chart is from the line $y = x$, the faster the I_A will converge to 1. When the EXIT chart remains above the line $y = x$, convergence occurs. However, if it remains too near the $y = x$ line, then convergence is slowed. Such a line, having a derivative with value nearly equal to 1, is said to be "flat."

The EXIT chart reveals something fundamental about the iterative decoding process. As the SNR approaches the threshold, the number of decoding iterations must increase, because each step through the EXIT chart is smaller. This behavior occurs independent of the particular decoding algorithm used, or, as will be discussed in chapter 16, regardless of the fact that there are cycles in the associated factor graph. Decoding near capacity seems to be intrinsically difficult.

14.6 Block Turbo Coding

While the turbo code examples up to this point in the chapter have employed convolutional codes as their constituent codes, other block codes may also be used. As an example, Figure 14.16 illustrates a turbo coder built using parallel concatenated BCH codes in what is called Turbo BCH coding. One particular structure for the interleaver is suggested in Figure 14.17.

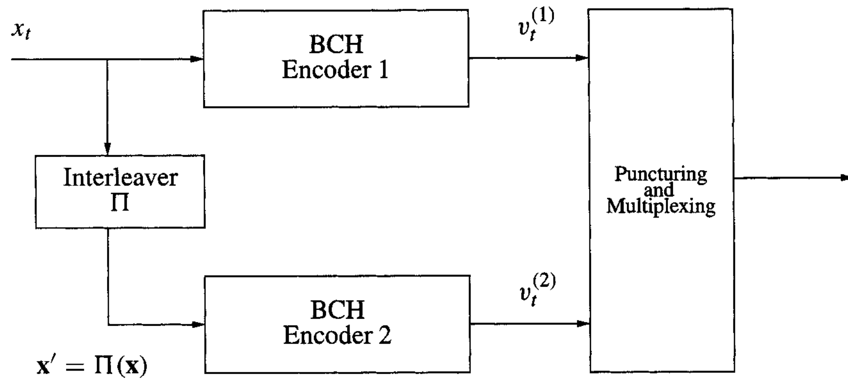


Figure 14.16: Turbo BCH encoding.

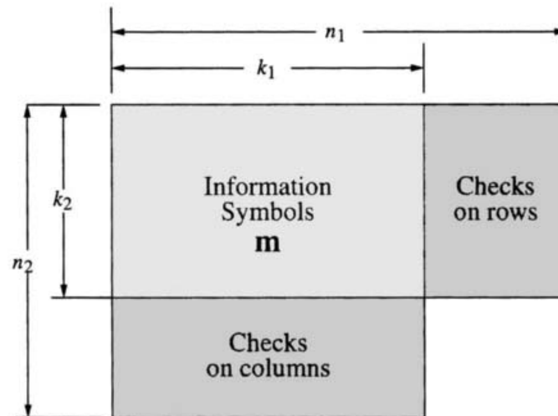


Figure 14.17: Structure of a particular implementation of a parallel concatenated code.

In this case, message data are written into a $k_2 \times k_1$ matrix. Data are read out in row order and passed to encoder 1, and data are read out in column order — constituting a permuted order — and passed through encoder 2. This framework is highly suggestive of the product codes introduced in Section 10.4, except that there is no portion of the codeword corresponding to the "parity on parity" that is present in a conventional product code. (Compare Figure 14.17 with Figure 10.5.)

Decoding of block turbo decoding proceeds as for the convolutional code: a decoder is used for each constituent code which produces a soft output in the form of an extrinsic probability, which is interleaved (or de-interleaved) and passed into the other decoder as a prior. The primary difficulty, then, is how to obtain soft output decoders for the codes.

Soft decoding of each code can be accomplished using the BCJR algorithm on the trellis representation for the code. To make the description explicit, we assume transmission over an AWGN. To make the description even more explicit, consider the trellis representation for the cyclically encoded (7,4,3) Hamming code with generator $g(x) = x^3 + x + 1$, shown in Figure 14.18 (and also originally in Figure 12.34). Unlike convolutional codes, where each branch of the trellis may convey several bits of systematic and/or parity information,

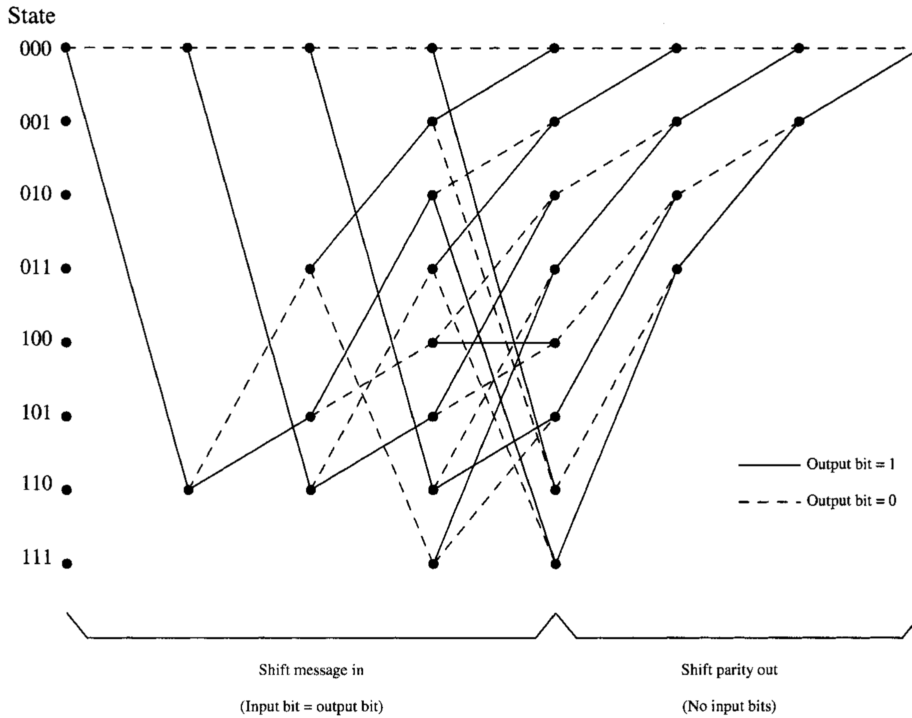


Figure 14.18: A trellis for a cyclically encoded (7,4,3) Hamming code.

each branch of the block code trellis carries only one bit of information, either message bits (for the first k stages) or parity bits (for the last $n-k$ stages). The transition probability $\gamma_t(p,q)$ is thus simplified compared to (14.27) and (14.52). We can write the transition probability as (neglecting uninformative factors)

$$\gamma_t(p,q) = \underbrace{\exp\left[\tilde{x}^{(p,q)}\frac{\lambda(\tilde{x}_t)}{2}\right]}_{\text{prior}}\underbrace{\exp\left[r_t\tilde{x}^{(p,q)}\frac{L_c}{2}\right]}_{\text{message or parity}}.$$

The likelihood ratio can then be computed as

$$\lambda(x_t|\mathbf{r}) = \log\frac{\sum_{(p,q)\in\mathcal{S}_1}\alpha_t(p)\gamma_t(p,q)\beta_{t+1}(q)}{\sum_{(p,q)\in\mathcal{S}_0}\alpha_t(p)\gamma_t(p,q)\beta_{t+1}(q)} \tag{14.72}$$

$$= \log\frac{\sum_{(p,q)\in\mathcal{S}_1}e^{\lambda(\tilde{x}_t)/2}e^{r_tL_c/2}\alpha_t(p)\beta_{t+1}(q)}{\sum_{(p,q)\in\mathcal{S}_0}e^{-\lambda(\tilde{x}_t)/2}e^{-r_tL_c/2}\alpha_t(p)\beta_{t+1}(q)}$$

$$= \lambda(\tilde{x}_t) + L_cr_t + \log\frac{\sum_{(p,q)\in\mathcal{S}_1}\alpha_t(p)\beta_{t+1}(q)}{\sum_{(p,q)\in\mathcal{S}_0}\alpha_t(p)\beta_{t+1}(q)} \tag{14.73}$$

$$= \lambda_{p,t} + \lambda_{s,t} + \lambda_{e,t}, \tag{14.74}$$

where $\lambda_{p,t} = \lambda(\tilde{x}_t)$ is the prior information, $\lambda_{s,t} = L_cr_t$ is the channel information, and

$$\lambda_{e,t} = \log\frac{\sum_{(p,q)\in\mathcal{S}_1}\alpha_t(p)\beta_{t+1}(q)}{\sum_{(p,q)\in\mathcal{S}_0}\alpha_t(p)\beta_{t+1}(q)}$$

is the extrinsic information. Note that, unlike the convolutional coded case, the extrinsic information depends only on α and β probabilities and not on any parity bits transmitted with the systematic bits along a branch. Given $\gamma_t(p, q)$, the computation of the α and β probabilities is identical to that for turbo codes based on convolutional codes.

14.7 Turbo Equalization

14.7.1 Introduction to Turbo Equalization

In this section we introduce a decoding technique applicable to a channel model which differs significantly from other channel models used throughout this book. Because of the close connection of turbo equalization with turbo decoding, it is deemed to be an appropriate topic to include here. In the rest of the book, the channel model has been a discrete memoryless model, specifically, an additive noise channel in which the received signal r_t is simply the transmitted signal s_t corrupted by additive noise (typically either Gaussian or Bernoulli): $r_t = s_t + n_t$. In this section, however, we consider the case that the channel has a response characterized by a transfer function $H(z) = h_0 + h_1 z^{-1} + \cdots + h_L z^{-L}$, so that the received signal is

$$r_t = \left(\sum_{i=0}^{L} s_{t-i} h_i \right) + n_t.$$

Such a channel could arise, for example, in a multipath environment or bandlimited channel.

The degradation of the received signal due to the channel can be severe, so that it is important to compensate in some way for the effect of the channel. Over the years, considerable work has been done on receivers for such channels. Various approaches include the following:

1. Linear equalization with a fixed filter $\tilde{H}(z)$, so that $r(z)\tilde{H}(z)$ "looks" a lot like the transmitted signal $s(z)$. The equalizer filter can be designed according to several criteria, such as zero forcing (cancel all interference, but neglect the influence of noise) or minimum mean-square error (minimize the average interference energy) [276].
2. Decision feedback equalization, a technique in which decisions on previous outputs are fed back to cancel their influence in the received signal [276].
3. Adaptive linear equalizers and decision feedback techniques, in which the receiver adaptively estimates the coefficients for the receiver (see, e.g., [276]).
4. Maximum likelihood sequence estimation (MLSE), in which the channel is regarded as having a state determined by the previous L bits, and a Viterbi decoder is used to decode [276].
5. Maximum a posteriori decoding, similar to MLSE, except that the MAP (or BCJR) algorithm is run. The latter two methods are arguably optimal, but run into computational difficulties because the number of states grows exponentially with the length L of the channel response. (Conventional turbo equalization also suffers from this problem.)
6. Suboptimal variations and interpolations of these ideas (such as [310]).

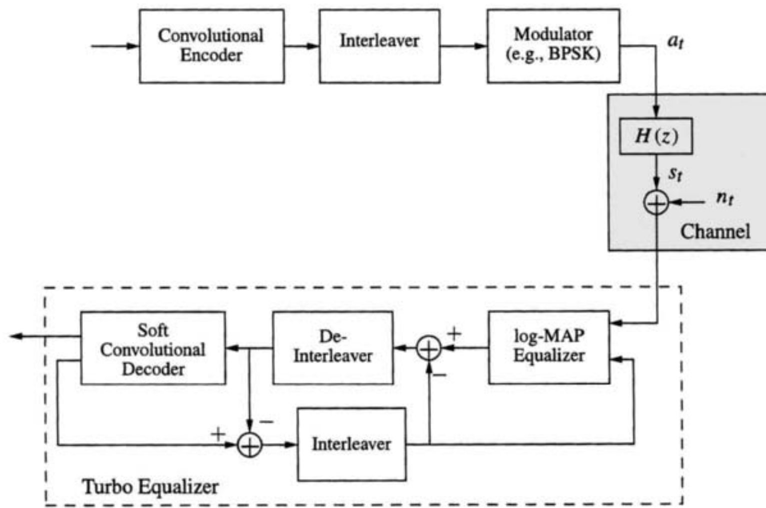


Figure 14.19: Framework for a turbo equalizer.

Until recently, most work in this area employed a nearly tacit separation principle: the equalization and detection was followed by the error correction decoding. However, the advent of turbo decoding algorithms has led to the development of *turbo equalization*, in which the channel impairments and the error correction are dealt with in an iterated structure.

14.7.2 The Framework for Turbo Equalization

A key observation is that if a convolutional encoder at the transmitter is followed by an interleaver, then the convolutive effects experienced as the signal traverses through the channel act like a second convolutional encoder, so the overall scheme acts like a serially concatenated code, with an interleaver between them. It is thus amenable to turbo decoding. Figure 14.19 shows the general framework for a system that can employ turbo decoding. The interleaver serves to decorrelate the values, so that extrinsic information computed in the decoder is nearly independent of the input values and can be used as a prior. In practice, the length of the interleaver is on the order of the length of the channel response.

In the turbo decoder, the channel response is first accounted for using a MAP equalizer, which produces information about the bits in the form of log likelihood ratios, $\lambda(x_t|\mathbf{r})$. The prior information is subtracted, leaving an extrinsic information that is passed through the interleaver and on to the convolutional decoder, where it is employed as a prior probability. The output of the decoder is again bit information in the form of log likelihood ratios $\lambda(x_t|\mathbf{r})$. The prior information associated with this is subtracted off, then the resulting extrinsic information is sent back to the equalizer for another iteration.

All elements of this equalizer should by now be familiar, with the possible exception of the MAP equalizer, which is described via an example.

Suppose ± 1-valued bits emerging from the convolutional encoder are denoted as $\tilde{v}_t$ (where, for the moment, we simply think of these as a string of bits, without regard to which are message bits and which are parity bits). The modulated bits are $a_t = \sqrt{E_c}\tilde{v}_t$. Suppose

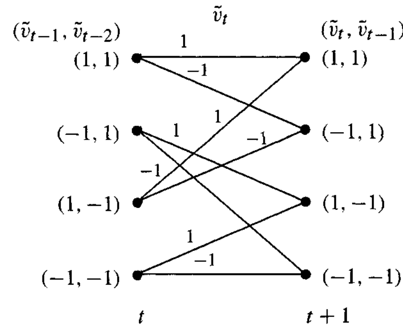


Figure 14.20: Trellis associated with a channel with $L = 2$.

that the channel has length $L = 2$, so that the received signal can be written as

$$r_t = h_0\sqrt{E_c}\tilde{v}_t + h_1\sqrt{E_c}\tilde{v}_{t-1} + h_2\sqrt{E_c}\tilde{v}_{t-2}.$$

We now form the trellis associated with this channel by defining the state at time t to be the L *previous* bits $(\tilde{v}_{t-1}, \tilde{v}_{t-2})$. Figure 14.20 shows the trellis associated with this channel. At a state $p = (\tilde{v}_{t-1}, \tilde{v}_{t-2})$ at time t, with the input $\tilde{v}_t = \tilde{v}^{(p,q)}$ leading to the state $q = (\tilde{v}_t, \tilde{v}_{t-1})$ at time $t+1$, let the channel output (excluding the noise) be denoted as

$$s_t(p, q) = h_0\sqrt{E_c}\tilde{v}_t + h_1\sqrt{E_c}\tilde{v}_{t-1} + h_2\sqrt{E_c}\tilde{v}_{t-2}.$$

Then for AWGN, the likelihood function is

$$p(r_t|s_t(p, q)) = C\exp\left[-\frac{1}{2\sigma^2}(r_t - s_t)^2\right].$$

Based on this, a transition probability suitable for use with the MAP algorithm is

$$\gamma_t(p, q) = p(r_t|s_t(p, q))P(\tilde{v}_t = \tilde{v}^{(p,q)}).$$

This can be subjected to the usual simplifications (e.g., expressed in log form, unimportant terms ignored, etc.).

Once the transition probability $\gamma_t(p, q)$ is established, the remainder of the MAP or log-MAP algorithm follows exactly as outlined in Section 14.3.12. The resulting decoder produces log likelihoods $\lambda(\tilde{v}_t|\mathbf{r})$, from which the extrinsic probabilities can be extracted for use in the convolutional decoder.

It is typical for turbo equalizers to have several dB of improvement compared to non-turbo equalizers. For a comparative study, we refer the reader to [141].

Of course, computing the transition probability, and hence the MAP decoding, requires knowledge of the channel coefficients $\{h_0, h_1, \ldots, h_L\}$. A variety of methods of estimating these are known. Many signals are prefaced with a training sequence which can be used to establish linear equations for computing the coefficients, using, for example, a minimum mean-squared error or least-squares criterion. Channel coefficients can also be estimated "blindly" by averaging out the unknown bits using an EM (expectation/maximization) type algorithm (see, e.g., [310]). Once transmission has begun, previous bits can be used to re-estimate the channel coefficients if the channel is time-varying.

Programming Laboratory 12: Turbo Code Decoding

Objective

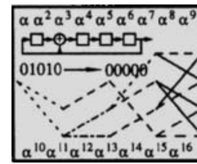
In this laboratory, you will finalize the decoding algorithms for the probabilistic form of the BCJR algorithm and use this to construct a turbo decoder.

Background

Reading: Sections 14.2, 14.3.

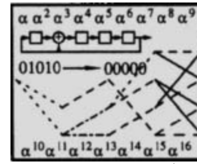
Programming Part

1) Using the class definitions and declarations shown in Algorithm 14.3, complete the function `alphabeta` to compute α and β using the forward and backward passes. Use normalized computations. Verify that α and β are computed correctly using the program `testbcjr`, comparing the results with Example 14.2.



Algorithm 14.3 BCJR Algorithm
File: `BCJR.h`
`BCJR.cc`
`testbcjr.cc`

2) Use your BCJR algorithm in conjunction with `testturbodec2` to reproduce (part of) the data shown in Figure 14.1 for a (37,21,65536) turbo code. *Note*: Do not take the SNR too large, or the computations will take excessively long.



Algorithm 14.4 Test the turbo decoder
File: `testturbodec2.cc`

As currently implemented, the interleaver is a very simple random interleaver.

14.8 Exercises

14.1 A convolutional encoder uses the parity generator $G(x) = \frac{1}{1+x+x^2}$. The length of the input sequence is 10. The interleaver is described by the sequence $\Pi = \{9, 4, 2, 7, 0, 6, 1, 8, 3, 5\}$.

(a) Draw the block diagram for the turbo encoder and the trellis for the convolutional code.

(b) The sequence $\mathbf{x} = [1, 1, 0, 0, 1, 0, 1, 0, 1, 1]$ is input to the turbo encoder. Determine the output sequences $\mathbf{v}^{(0)}$, $\mathbf{v}^{(1)}$, $\mathbf{v}^{(2)}$ and $\mathbf{v}$.

(c) The sequence is punctured to obtain a rate $R = 1/2$ code by taking the even bits of $\mathbf{v}^{(1)}$ and the odd bits of $\mathbf{v}^{(2)}$. Determine the output sequence $\mathbf{v}$ now.

14.2 [134] This exercise is meant to introduce concepts related to turbo decoding.

(a) Let r_t be the output of an AWGN channel. Suppose that BPSK modulation is employed. Let the transmitted signal s_t have energy E_b, where $s_t = \sqrt{E_b}\tilde{x}_t$ and $\tilde{x}_t \in \{\pm 1\}$. Let

$$\lambda(\tilde{x}_t|r_t) = \log \frac{P(\tilde{x}_t = 1|r_t)}{P(\tilde{x}_t = -1|r_t)}.$$

Show that

$$\lambda(\tilde{x}_t|r_t) = L_c r_t + \lambda(\tilde{x}_t), \tag{14.75}$$

where $L_c = 2\sqrt{E_b}/\sigma^2$ and $\lambda(\tilde{x}_t) = \log \frac{P(\tilde{x}=1)}{P(\tilde{x}=-1)}$.

(b) Suppose that the signal s_t is sent independently through two different channels, so the received values are

$$r_t^{(1)} = s_t + n_t^{(1)}$$
$$r_t^{(2)} = s_t + n_t^{(2)},$$

where $n_t^{(1)}$ and $n_t^{(2)}$ are independent. Show that

$$\lambda(\tilde{x}_t | r_t^{(1)}, r_t^{(2)}) = L_{c_1} r_t^{(1)} + L_{c_2} r_t^{(2)} + \lambda(\tilde{x}_t). \tag{14.76}$$

(c) Now consider the simple (3,2,2) parity check code shown in Figure 14.21(a). The parity check p_i^- (across the rows) is defined by

$$p_i^- = \tilde{x}_{i1} \oplus \tilde{x}_{i2}$$

and the parity check for the $p_i^|$ (down the columns) is defined by

$$p_i^| = \tilde{x}_{1i} \oplus \tilde{x}_{2i},$$

where $\oplus$ is the $GF(2)$ addition defined in (14.57). Explain why $\lambda(\tilde{x}_{12} \oplus p_1^- | r)$ is extrinsic information for the bit $\tilde{x}_{11}$. (Here the conditioning on r denotes conditioning based on the entire set of received codeword data.) Denote this extrinsic information as $\lambda_e(\tilde{x}_{11})^-$, that is, the likelihood ratio of the extrinsic information using the horizontal code parity checks.

Note that, using the $\boxplus$ operator defined in (A.3), we can write

$$\lambda_e(\tilde{x}_{11})^- = \lambda(\tilde{x}_{12} \oplus p_1^- | r) = \lambda(\tilde{x}_{12} | r) \boxplus \lambda(p_1^- | r). \tag{14.77}$$

(d) Suppose the channel input values $L_c r_t$ are as shown in Figure 14.21(b). Using (14.77) and the approximate formula (A.2), determine the extrinsic information for $\tilde{x}_{11}, \tilde{x}_{12}, \tilde{x}_{21}$ and $\tilde{x}_{22}$ using the horizontal parity bits p_i^-. That is, determine $\lambda_e(\tilde{x}_{11})^-$, $\lambda_e(\tilde{x}_{12})^-$, $\lambda_e(\tilde{x}_{21})^-$, and $\lambda_e(\tilde{x}_{22})^-$. Assume uniform priors. Computing this extrinsic information constitutes the horizontal stage of decoding.

For example, for $\tilde{x}_{11}$, we have

$$\begin{aligned}
\lambda_e(\tilde{x}_{11})^- &= \lambda(\tilde{x}_{12} \oplus p_1^- | r) = \lambda(\tilde{x}_{12} | r) \boxplus \lambda(p_1^- | r) \\
&= (L_c r_{12} + \lambda(\tilde{x}_{12})) \boxplus \lambda(p_1^- | r) \qquad \text{(using, e.g., (14.75))} \\
&= L_c r_{12} \boxplus \lambda(p_1^- | r) \qquad \text{(assuming uniform priors)} \\
&= 1.5 \boxplus 1.0 \approx 1.0
\end{aligned}$$

These likelihoods are available from $L_c r_t$ in Figure 14.21(b). Show that the extrinsic information $\lambda_e(\tilde{x}_{ij})^-$ is as shown here:

+1.0	+0.5
−1.0	−1.5

(e) For the second stage of decoding, determine the extrinsic information after the first vertical decoding using the information from the first (horizontal) extrinsic information as the prior. For example,

$$\begin{aligned}
\lambda_e(x_{11})^| &= \lambda(\tilde{x}_{21} \oplus p_1^| | r) = \lambda(\tilde{x}_{21} | r) \boxplus \lambda(p_1^| | r) \\
&= (L_c r_{21} + \lambda_e(\tilde{x}_{21})^-) \boxplus \lambda(p_1^| | r) \\
&= (4 + (-1)) \boxplus 2.0 \approx 2.0.
\end{aligned}$$

Show that $\lambda_e(\tilde{x}_{11})^|$, $\lambda_e(\tilde{x}_{12})^|$, $\lambda_e(\tilde{x}_{21})^|$, and $\lambda_e(\tilde{x}_{22})^|$ are as shown here:

+2.0	+0.5
+1.5	−2.0

This completes the vertical stage of decoding.

$\tilde{x}_{11}$	$\tilde{x}_{12}$	p_1^-
$\tilde{x}_{21}$	$\tilde{x}_{22}$	p_2^-
$p_1^{\vert}$	$p_2^{\vert}$	

(a) A simple (3, 2, 2) parity check code.

0.5	1.5	1.0
4.0	1.0	−1.5
2.0	−2.5	

(b) Received values $L_c r_t$.

Figure 14.21: Example for a (3, 2, 2) parity check code.

(f) The overall information after a horizontal and a vertical stage is

$$\lambda(\tilde{x}_{i,j}|r,-,|) = L_c r_{ij} + \lambda_e(\tilde{x}_{ij})^- + \lambda_e(\tilde{x}_{ij})^{|}.$$

The addition is justified by (14.76), since after this first complete round the three terms in the sum are independent. Show that this information is as shown here:

+3.5	+2.5
+4.5	−2.5

(g) Decoding can be accomplished by taking the sign of the total likelihoods. Determine the decoded values after this round of decoding.

(h) If iteration continues, the information $\lambda_e(\tilde{x}_{ij})^{|}$ is used as the prior for the next horizontal stage. Show that at the next stage, the extrinsic information is

+1.0	+1.0
−0.5	−1.5

(If iteration continues, then the independence assumption no longer holds, but is invoked anyway.)

14.3 Suppose that a $R = k/n$ convolutionally encoded sequence is passed through a BSC with channel crossover probability p_c. Determine the transition probability $\gamma_t(p, q)$ for this channel.

14.4 In Example 14.2:

(a) Compute α_1' and α_2' using the received data $\mathbf{r}$ in the example.

(b) Compute β_9' and β_8' using the received data $\mathbf{r}$ in the example.

(c) Using the data provided in Table 14.1, compute $P(x_0 = 0|\mathbf{r})$ and $P(x_1 = 0|\mathbf{r})$.

14.5 Show that the log likelihood of the sequence $\log p(\mathbf{r}) = \log p(\mathbf{r}_0^{N-1})$ can be written as

$$\log p(\mathbf{r}_0^{N-1}) = \log \sum_{\text{all valid} p} \alpha_N'(p) - \sum_{i=0}^{N} A_i.$$

14.6 Given the log probability ratio $\lambda = \log(P(X = 1)/P(X = 0))$, determine $P(X = 1)$ and $P(X = 0)$. Show that $P(X = x)$ can be written as $P(X = x) = [e^{-\lambda/2}/(1 + e^{-\lambda})]e^{\tilde{x}\lambda/2}$, where $\tilde{x} = 2x - 1$.

14.7 Show that (14.39) is correct.

14.8 Show that (14.46) is correct.

14.9 Show that (14.47) is correct. *Hint*: Consider separately the cases

$$\hat{\hat{x}}_k = 1,$$

$$\lambda^{[l,1]}(x_k|\mathbf{r}) \gg 0, \lambda^{[l,2]}(x_k|\mathbf{r}) \gg 0,$$

and

$$\hat{\hat{x}}_k = -1.$$

$$\lambda^{[l,1]}(x_k|\mathbf{r}) \ll 0, \lambda^{[l,2]}(x_k|\mathbf{r}) \ll 0.$$

14.10 (Examination of the approximation (14.49).) Let $x_1 = 1$ and make a plot of x_2 *vs.* $\log(e^{x_1} + e^{x_2}) - x_2$ for x_2 in the range $[1, 20]$. Comment on where the approximation becomes particularly accurate.

14.11 Show that (14.54) is correct.

14.12 Show how to obtain the approximation (14.64) from (14.62).

14.13 Show how (14.69) follows from (14.67) and (14.68).

14.14 The EXIT chart can be used to estimate the bit error rate after an arbitrary number of iterations. Using $D = Z + A + E$

(a) Argue that D is Gaussian distributed with variance σ_D^2 and mean $\sigma_D^2/2$, where

$$\sigma_D^2 = \sigma_Z^2 + \sigma_A^2 + \sigma_E^2.$$

(b) Show that the probability of bit error is therefore $P_b \approx Q(\mu_d/\sigma_D)$.

(c) Show that

$$P_b \approx Q\left(\frac{\sqrt{8RE_b/N_0 + J^{-1}(I_A)^2 + J^{-1}(I_E)^2}}{2}\right).$$

14.9 References

Turbo codes were originally described in [28, 27]; their decoding algorithm was somewhat different than that described here, since they used a Gaussian random variable to represent the extrinsic probability passed between the decoders. The original BCJR algorithm appears in [11]. The α and β probabilities are also fundamental in hidden Markov models; see, for example, [269, 280, 67]. The presentation here benefited from the discussion in [14], [303], and [141]. The latter reference provides an extensive comparison between the various decoding algorithms presented here as well as a wealth of information about turbo code performance and tradeoff studies. Several tutorial expositions are also available; see, for example [315]. A discussion of the weight distributions of turbo codewords appears in [21, 20]. Discussion on the structure of the codewords appears in [294], while some discussion of design issues appears in [20]. The paper [69] provides suggested tables of encoder polynomials and suggests that the feedback coefficients for the encoder should be a primitive polynomial.

The discussion on spectral thinning was drawn from [303], as was the discussion on interleaving. For more on distance spectrum of turbo codes, see [259]. A more extensive treatment of interleaving, discussing several different structured approaches to interleavers, appears in [146, Chapter 3] and [6, 7, 72, 328]. See also [12, 180, 329, 64] for discussions on interleaving.

The EXIT chart analysis is discussed in [336] and references therein.

Our discussion of the cross entropy stopping criterion is drawn from [134]. The other stopping criteria are described in [311, 312]; see also [380], [384], [1], and [313].

The max-log-MAP algorithm appeared first in [295]. The SOVA algorithm is widely attributed to [132]. It is also described in [131, 134]. The essential equivalence of the SOVA and max-log-MAP algorithm is discussed in [102].

Turbo block codes are discussed in [134]. Extensive simulation results of Turbo BCH codes appear in [141]. For an alternative viewpoint based upon the Chase algorithm, see [279] or [278].

On turbo equalization, see [343]. For an extensive, self-contained introduction to turbo equalization, see [190]. The book [141] provides detailed examples of turbo equalization. An example of turbo equalization using LDPC codes combined with blind estimation of the channel coefficients is in [127].

An excellent resource on material related to turbo codes, LDPC codes, and iterative decoding in general is the February 2001 issue of the *IEEE Transactions on Information Theory*, which contains many articles in addition to those articles cited here.

Chapter 15

Low-Density Parity-Check Codes

15.1 Introduction

Low-density parity-check (LDPC) codes were originally proposed in 1962 by Robert Gallager [112, 113]. LDPC codes (sometimes called Gallager codes [217]) have performance exceeding, in some cases, that of turbo codes, with iterative decoding algorithms which are easy to implement (with the per-iteration complexity much lower than the per-iteration complexity of turbo decoders), and are also parallelizable in hardware. There are other potential advantages to LDPC codes as well. In a very natural way, the decoder declares a decoding failure when it is unable to correctly decode, whereas turbo decoders must perform extra computations for a stopping criterion (and even then, the stopping criterion depends upon a threshold that must be established, and the stopping criterion does not establish that a codeword has been found). Also, LDPC codes of almost any rate and blocklength can be created simply by specifying the shape of the parity check matrix, while the rate of turbo codes is governed largely by a puncturing schedule, so flexibility in rate is obtained only through considerable design effort. Also, because the validity of a codeword is validated if its parity checks, even when errors do occur, they are almost always *detected* errors (especially for long codes). As an additional boon on the commercial side, LDPC codes are not patent protected. On the negative side, LDPC codes have a significantly higher *encode* complexity than turbo codes, being generically quadratic in the code dimension, although this can be reduced somewhat. Also, decoding may require many more iterations than turbo decoding, which has implications for latency.

It is a curious twist of history that LDPC codes, which are among the best codes in the world, should have been largely unnoticed for so long. Among the reasons that the codes might have been overlooked are that contemporary investigations in concatenated coding overshadowed LDPC codes, and that the hardware of the time could not support effective decoder implementations. As a result, LDPC codes remained largely unstudied for over thirty years, with only scattered references to them appearing in the literature, such as [330, 371, 370]. Recently, however, they have been strongly promoted, beginning with the work of MacKay [217, 66, 215, 216]. Both historically and recently, LDPC codes have been proved to be capable of closely approaching the channel capacity. In fact, the proof of the distance properties of LDPC codes demonstrates such strong performance for these codes that it has been termed a "semiconstructive proof of Shannon's noisy channel coding theorem" [217, p. 400]. In particular, using random coding arguments MacKay showed that LDPC code ensembles can approach the Shannon capacity limit exponentially fast in the length of the code. The powerful capabilities of LDPC codes have led to their recent inclusion in several standards, such as IEEE 802.16, IEEE 802.20, IEEE 802.3 and DBV-RS2.

15.2 LDPC Codes: Construction and Notation

We use N to denote the length of the code and K to denote its dimension and $M = N - K$.[1] Throughout this chapter, only binary LDPC codes are considered (although they can be constructed over other fields). Since the parity check matrices we consider are generally not in systemic form, we usually use the symbol A to represent parity check matrices, reserving the symbol H for parity check matrices in systematic form. Following the general convention in the literature for LDPC codes, we assume that vectors are *column* vectors. A message vector $\mathbf{m}$ is a $K \times 1$ vector; a codeword is a $N \times 1$ vector. The generator matrix G is $N \times K$ and the parity check matrix A is $(N - K) \times N$, such that $HG = \mathbf{0}$. We denote the rows of a parity check matrix as

$$A = \begin{bmatrix} \mathbf{a}_1^T \\ \mathbf{a}_2^T \\ \vdots \\ \mathbf{a}_M^T \end{bmatrix}.$$

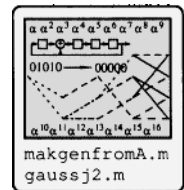
The equation $\mathbf{a}_i^T \mathbf{c} = 0$ is said to be a linear parity-check constraint on the codeword $\mathbf{c}$. We use the notation $z_m = \mathbf{a}_m^T \mathbf{c}$ and call z_m a parity check or, more simply, a check.

For a code specified by a parity check matrix A, it is expedient for encoding purposes to determine the corresponding generator matrix G. A systematic generator matrix may be found as follows. Using Gaussian elimination with column pivoting as necessary (with binary arithmetic) determine an $M \times M$ matrix A_p^{-1} so that

$$H = A_p^{-1} A = \begin{bmatrix} I & A_2 \end{bmatrix}.$$

(If such a matrix A_p does not exist, then A is rank deficient, $r = \text{rank}(A) < M$. In this case, form H by truncating the linearly dependent rows from $A_p^{-1} A$. The corresponding code has $R = K/N > (N - M)/N$, so it is a higher rate code than the dimensions of A would suggest.) Having found H, form

$$G = \begin{bmatrix} A_2 \\ I \end{bmatrix}.$$



Then $HG = \mathbf{0}$, so $A_p HG = AG = \mathbf{0}$, so G is a generator matrix for A. While A may be sparse (as discussed below), neither the systematic generator G nor H is necessarily sparse.

A matrix is said to be *sparse* if fewer than half of the elements are nonzero.

Definition 15.1 A low density parity check code is linear block code which has a very sparse parity check matrix.

For reasons to be made clear below, the parity check matrix should also be such that no two columns have more than one row in which elements in both columns are nonzero. (This corresponds to no cycles of length four in the Tanner graph.) □

The *weight* of a binary vector is the number of nonzero elements in it. The *column weight* of a column of a matrix is the weight of the column; similarly for *row weight*. An LDPC generator is *regular* if the column weights are all the same and the row weights are all the same. To generate a regular LDPC code, a column weight w_c is selected (typically a small integer such as $w_c = 3$) and values for N (the block length) and M (the redundancy)

[1] In other chapters, n, k, and m are used to describe the code. In this chapter we use n and m as indices, suggesting by them that they index length and redundancy components.

are selected. Then an $M \times N$ matrix A is generated which has weight w_c in each column and row weight w_r in each row. Attaining uniform row weight w_r requires that $w_c N = w_r M$. This structure says that every bit participates in w_c checks and each check involves w_r bits. Such a regular code is called a (w_c, w_r, N) code (or a (w_c, w_r) code, if a sequence of codes of increasing length N is considered). The *design rate* of a regular (w_c, w_r) code is $R = 1 - w_c/w_r$, provided that all the rows are linearly independent. (Because rows may be linearly dependent, the actual rate may be somewhat higher than the design rate.) Gallager showed that the minimum distance of a typical regular LDPC code increases linearly with N provided that $w_c \geq 3$. The parity check matrix need not be regular; codes having varying column weights are in general superior to regular codes. Irregular codes are introduced in Section 15.10.

The parity check matrix A may be generated at random with the appropriate column and row weights (although there are some restrictions on column overlap which should be met, as discussed in Section 15.11). We thus have, in constructive fulfillment of Shannon's original proof, a random code that is, as we shall see, easily decoded.

Example 15.1

$$A = \begin{bmatrix} \mathit{1} & \mathit{1} & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 \\ \mathit{1} & \mathit{1} & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 \end{bmatrix} \tag{15.1}$$

has column weight $w_c = 3$ and row weight $w_r = 6$. (Strictly speaking, A is not a sparse matrix, since more than half of its elements are nonzero. However, it was chosen as a small matrix to illustrate the concept of a random matrix with fixed-weight columns. Space considerations preclude explicit presentation of more realistic parity check matrices.)

For a codeword $\mathbf{c} = [c_1, c_2, \ldots, c_{10}]$, we must have

$$c_1 + c_2 + c_3 + c_6 + c_7 + c_{10} = 0,$$

$$c_1 + c_3 + c_5 + c_6 + c_8 + c_9 = 0,$$

etc. Thus, bits c_1, c_2, c_3, c_6, c_7 and c_{10} participate in check z_1. Also, from A it may be observed that bit c_1 is involved in checks z_1, z_2 and z_5. (The italicized elements of A are discussed below.) □

Example 15.2 One way to construct a (w_c, w_r) parity check matrix is as follows. Construct the matrix A_0,

$$A_0 = \begin{bmatrix} \underbrace{1\,1\,\cdots\,1}_{w_r} & & & \\ & \underbrace{1\,1\,\cdots\,1}_{w_r} & & \\ & & \ddots & \\ & & & \underbrace{1\,1\,\cdots\,1}_{w_r} \end{bmatrix}$$

with $N/w_r = M/w_c$ rows and N columns. This defines a $(1, w_r)$ regular parity check code, but one having minimum distance 2 (why?). Then we form A by stacking permutations of A_0,

$$A = \begin{bmatrix} \pi_1(A_0) \\ \pi_2(A_0) \\ \pi_3(A_0) \\ \vdots \\ \pi_{w_c}(A_0) \end{bmatrix},$$

where each $\pi_i(A_0)$ denotes a matrix obtained by permuting the columns of A_0. Obviously, the choice of the permutations determines the distance structure of the code. However, a random choice of permutation will, on average, produce a good code. (This is an instance of the *concentration principle*, which essentially states that a random code will, with high probability, have behavior like the average code.) Gallager showed that if each permutation is chosen at random out of the $N!$ possible permutations, then the average minimum distance increases linearly with N. Such codes are called good codes. This is thus a rather constructive way of achieving Shannon's original idea of random codes. While the code is increasing in size, the column and row weight are fixed, which means that the decoding complexity (per iteration) remains fixed. □

Agall.m
Agall.txt

Example 15.3 An example of an LDPC parity check matrix for a (3, 4)-regular LDPC code due to Gallager [113] is

$$A = \begin{bmatrix}
1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&1&1&1&1&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1\\
1&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&0&0&0&0\\
0&1&0&0&0&1&0&0&0&1&0&0&0&0&0&0&1&0&0&0\\
0&0&1&0&0&0&1&0&0&0&0&0&0&1&0&0&0&1&0&0\\
0&0&0&1&0&0&0&0&0&0&1&0&0&0&1&0&0&0&1&0\\
0&0&0&0&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&1\\
1&0&0&0&0&1&0&0&0&0&0&1&0&0&0&0&0&1&0&0\\
0&1&0&0&0&0&1&0&0&0&1&0&0&0&0&1&0&0&0&0\\
0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&0&1&0\\
0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&1&0&0&0\\
0&0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&1
\end{bmatrix}.$$

It can be shown that there are 13 linearly independent rows. The dimension of the code is thus $20 - 13 = 7$, so we have a (20,7) code with actual rate $R = 0.35$. □

writesparse.m

In many LDPC codes, N is taken to be quite large (such as $N > 10000$) while the column weight is held at around 3 or 4, so the density of 1s in the matrix is quite low. (A column weight of $w_c = 2$ has been found to be ineffective [217].)

Since the A matrix is sparse, it can be represented efficiently using lists of its nonzero locations. In this notation, bits are typically indexed by n or n' (e.g., $c_{n'}$) and the checks are typically indexed by m or m' (e.g., z_m). The set of bits that participate in check z_m (i.e., the nonzero elements on the mth row of A) is denoted

$$\mathcal{N}_m = \{n : A_{mn} = 1\}.$$

Thus we can write the mth check as

$$z_m = \sum_{n \in \mathcal{N}_m} c_n.$$

The set of bits that participate in check z_m except for bit n is denoted

$$\mathcal{N}_{m,n} = \mathcal{N}_m \backslash n.$$

The notation $|\mathcal{N}_m|$ indicates the number of elements in the set $\mathcal{N}_m$. These sets should be considered *ordered lists*, with the ith element of $\mathcal{N}_m$ being indicated by $\mathcal{N}_m(i)$.

The set of checks in which bit c_n participates (i.e., the nonzero elements of the nth column of A) is denoted

$$\mathcal{M}_n = \{m : A_{mn} = 1\}.$$

For a regular LDPC code, $|\mathcal{M}_n| = w_c$. Let

$$\mathcal{M}_{n,m} = \mathcal{M}_n \backslash m$$

be the set of checks in which bit c_n participates except for check m.

Example 15.4 For the parity check matrix of (15.1),

$$\mathcal{N}_1 = \{1, 2, 3, 6, 7, 10\}, \quad \mathcal{N}_2 = \{1, 3, 5, 6, 8, 9\}, \quad \text{etc.}$$

$$\mathcal{M}_1 = \{1, 2, 5\}, \quad \mathcal{M}_2 = \{1, 4, 5\}, \quad \text{etc.}$$
$$\mathcal{N}_{2,1} = \{3, 5, 6, 8, 9\}, \quad \mathcal{N}_{2,3} = \{1, 5, 6, 8, 9\}, \quad \text{etc.}$$
$$\mathcal{M}_{2,1} = \{4, 5\}, \quad \mathcal{M}_{2,4} = \{1, 5\}, \quad \text{etc.}$$

□

15.3 Tanner Graphs

Associated with a parity check matrix A is a graph called the *Tanner graph* containing two sets of nodes. The first set consists of N nodes which represent the N bits of a codeword; nodes in this set are called "bit" nodes. The second set consists of M nodes, called "check" nodes, representing the parity constraints. The graph has an edge between the nth bit node and the mth check node if and only if nth bit is involved in the mth check, that is, if $A_{mn} = 1$. Thus the Tanner graph is a graphical depiction of the parity check matrix. Figure 15.1 illustrates the graph for A from Example 15.1. A graph such as this, consisting of two distinct sets of nodes and having edges only between the nodes in different sets, is called a *bipartite* graph. The Tanner graph is used below to develop insight into the decoding algorithm. For the Tanner graph representation of the parity check matrix of a regular code, each bit node is adjacent to w_c check nodes and each check node is adjacent to w_r bit nodes.

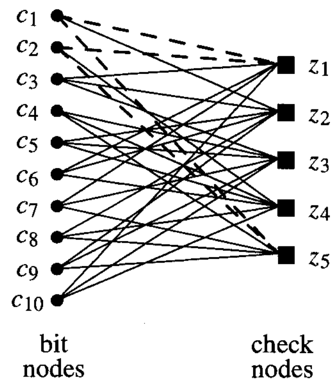


Figure 15.1: Bipartite graph associated with the parity check matrix A. (The dashed edges correspond to a cycle of length four, as discussed below.)

15.4 Transmission Through a Gaussian Channel

The decoding algorithm described below is a soft-decision decoder which makes use of channel information. We develop the decoder for codewords transmitted through an additive white Gaussian noise (AWGN) channel. When a codeword is transmitted through an AWGN channel the binary vector $\mathbf{c}$ is first mapped into a transmitted signal vector $\mathbf{t}$. For illustrative

purposes, a binary phase-shift keyed (BPSK) signal constellation is employed, so that the signal $a = \sqrt{E_c}$ represents the bit 1 and the signal $-a$ represents the bit 0. The energy per message bit E_b is related to the energy per transmitted coded bit E_c by $E_c = RE_b$, where $R = k/n$ is the rate of the code. The transmitted signal vector $\mathbf{t}$ has elements $t_n = (2c_n - 1)a$. This signal vector passes through a channel which adds a Gaussian noise vector $\boldsymbol{\nu}$, where each element of $\boldsymbol{\nu}$ is independent, identically distributed with zero mean and variance $\sigma^2 = N_0/2$. The received signal is

$$\mathbf{r} = \mathbf{t} + \boldsymbol{\nu}. \tag{15.2}$$

Given the received data, the posterior probability of detection can be computed as

$$\begin{aligned} P(c_n = 1|r_n) &= \frac{p(r_n|t_n = a)P(t_n = a)}{p(r_n)} \\ &= \frac{p(r_n|t_n = a)P(t_n = a)}{p(r_n|t_n = a)P(t_n = a) + p(r_n|t_n = -a)P(t_n = -a)} \\ &= \frac{p(r_n|t_n = a)}{p(r_n|t_n = a) + p(r_n|t_n = -a)}, \end{aligned} \tag{15.3}$$

where it assumed that $P(c_n = 1) = P(c_n = 0) = \frac{1}{2}$. In (15.3), the notation $P(\cdot)$ indicates a probability mass function and $p(\cdot)$ indicates a probability density function. For an AWGN channel,

$$p(r_n|t_n = a) = \left.\frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2\sigma^2}(r_n - t_n)^2}\right|_{t_n = a} = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2\sigma^2}(r_n - a)^2}. \tag{15.4}$$

Applying (15.4) in (15.3) we obtain

$$P(c_n = 1|r_n) = \frac{1}{1 + e^{-2ar_n/\sigma^2}}. \tag{15.5}$$

We shall refer to $P(c_n = x|r_n)$ as the *channel posterior* probability and denote it by $p_n(x)$.

Example 15.5 The message vector $\mathbf{m} = \begin{bmatrix}1 & 0 & 1 & 0 & 1\end{bmatrix}^T$ is encoded using a systematic generator G derived from (15.1) to obtain the code vector

$$\mathbf{c} = \begin{bmatrix}0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1\end{bmatrix}^T. \tag{15.6}$$

Then $\mathbf{c}$ is mapped to a signal constellation with amplitude $a = 2$ to obtain the vector

$$\mathbf{t} = \begin{bmatrix}-2 & -2 & -2 & 2 & -2 & 2 & -2 & 2 & -2 & 2\end{bmatrix}^T,$$

which is transmitted through an AWGN channel with $\sigma^2 = 2$. The vector

$$\mathbf{r} = \begin{bmatrix}-0.63 & -0.83 & -0.73 & -0.04 & 0.1 & 0.95 & -0.76 & 0.66 & -0.55 & 0.58\end{bmatrix}^T$$

is received. Using (15.5), it is found that the channel posterior probabilities are

$$P(\mathbf{c} = 1|\mathbf{r}) = \begin{bmatrix}0.22 & 0.16 & 0.19 & 0.48 & 0.55 & 0.87 & 0.18 & 0.79 & 0.25 & 0.76\end{bmatrix}^T. \tag{15.7}$$

If $\mathbf{r}$ were converted to a binary vector by thresholding the probabilities in (15.7) at 0.5, the estimated vector

$$\begin{bmatrix}0 & 0 & 0 & \underline{0} & \underline{1} & 1 & 0 & 1 & 0 & 1\end{bmatrix}$$

would be obtained, which differs from the original code vector at the two underlined locations. However, note that at the error locations, the channel posterior probability is only slightly different than the threshold 0.5, so that the bits only "weakly" decode to the values 0 and 1, respectively. Other bits more strongly decode to their true values. These weak and strong indications are exploited by a soft-decision decoder. □

15.5 Decoding LDPC Codes

Some insight can be gained into the soft, iterative decoding algorithm we will eventually develop by considering a preliminary, iterative, hard decision decoder:

Hard Decoder: For each bit c_n, compute the checks for those checks that are influenced by c_n. If the number of nonzero checks exceeds some threshold (say, the majority of the checks are nonzero), then the bit is determined to be incorrect. The erroneous bit is flipped and correction continues.

This simple scheme is capable of correcting more than one error, as we now explain. Suppose that c_n is in error and that other bits influencing its checks are also in error. Arrange the Tanner graph with c_n as a root (neglecting for now the possibility of cycles in the graph). In Figure 15.2, suppose the bits in the shaded boxes are in error. The bits that connect to the checks connected to the root node are said to be in tier 1. The bits that connect to the checks from the first tier are said to be in tier 2. Many such tiers could be established. Then, decode by proceeding from the "leaves" of the tree (the top of the figure). By the time decoding on c_n is reached, other erroneous bits may have been corrected. Thus bits and checks which are not directly connected to c_n can still influence c_n.

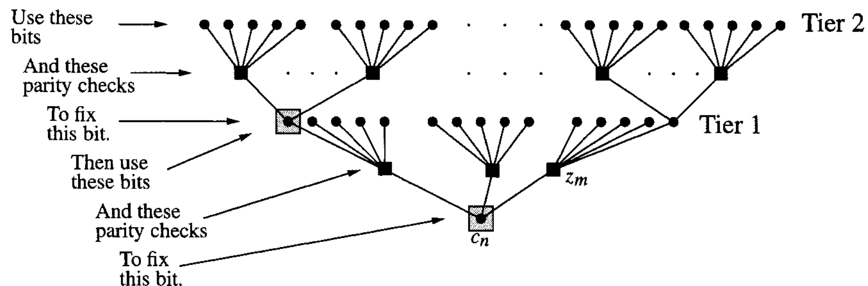


Figure 15.2: A parity check tree associated with the Tanner graph.

The soft decoder: In the soft decoder, rather than flipping bits (a hard operation), we propagate probabilities through the Tanner graph, thereby accumulating evidence that the checks provide about the bits. The optimal (minimum probability of decoding error) decoder seeks a codeword $\hat{\mathbf{c}}$ which maximizes $P(\mathbf{c}|\mathbf{r}, A\mathbf{c} = \mathbf{0})$, that is, the most probable vector which satisfies the parity checks, given set of received data $\mathbf{r} = [r_1, r_2, \ldots, r_N]$. However, the decoding complexity for the true optimum decoding of an unstructured (i.e., random) code is exponential in K, requiring an exhaustive search over all 2^K codewords. Instead, the decoder attempts to find a codeword having bits c_n which maximize

$$P(c_n|\mathbf{r}, \text{all checks involving bit } c_n \text{ are satisfied}),$$

that is, the posterior probability for a single bit given that only the checks on that bit are satisfied. As it turns out, even this easier, more computationally localized, task cannot be exactly accomplished due to approximations the practical algorithm must make. However, the decoding algorithm has excellent demonstrated performance and the complexity of the decoding is linear in the code length.

The decoding algorithm deals with two sets of probabilities. The first set of probabilities is related to the decoding criterion, $P(c_n|\mathbf{r}$, all checks involving bit c_n are satisfied). We denote this by

$$q_n(x) = P(c_n = x|\mathbf{r}, \text{all checks involving } c_n \text{ are satisfied}), \qquad x \in \{0, 1\},$$

or, using the notation defined in Section 15.2,

$$q_n(x) = P(c_n = x|\mathbf{r}, \{z_m = 0, m \in \mathcal{M}_n\}), \qquad x \in \{0, 1\}. \tag{15.8}$$

This probability is referred to as the *pseudoposterior* probability and is ultimately used to make the decisions about the decoded bits. A variant of this probability, called $q_{mn}(x)$, is also used, which is

$$q_{mn}(x) = P(c_n = x|\mathbf{r}, \text{all checks, except } z_m, \text{ involving } c_n \text{ are satisfied})$$

or, more briefly,

$$q_{mn}(x) = P(c_n = x|\mathbf{r}, \{z_{m'} = 0, m' \in \mathcal{M}_{n,m}\}).$$

The second set of probabilities has to do with the probability of checks given the bits. These indicate the probability that a check is satisfied, given the value of a single bit involved with that check and the observations associated with that check. This probability is denoted by $r_{mn}(x)$, with $r_{mn}(x) = P(z_m = 0|c_n = x, \mathbf{r})$.

The quantities $q_{mn}(x)$ and $r_{mn}(x)$ are computed only for those elements A_{mn} of A that are nonzero. The decoding algorithm incorporates information from the measured data to compute probabilities about the checks, as represented by $r_{mn}(x)$. The information about the checks is then used to find information about the bits, as represented by $q_{mn}(x)$. This, in turn, is used to update the probabilities about the checks, and so forth. This amounts to propagating through the "tree" derived from the Tanner graph. Iteration between bit and check probabilities (qs and rs) proceeds until all the parity checks are satisfied simultaneously, or until a specified number of iterations is exceeded.

15.5.1 The Vertical Step: Updating $q_{mn}(x)$

Consider Figure 15.3(a), which is obtained by selecting an arbitrary bit node c_n from the Tanner graph and using it as the root of a tree, with the subset of the Tanner graph connecting this bit to its checks and the other bits involved in these checks as nodes in the tree. The bits other than c_n which connect to the parity checks are referred to as bits in *tier 1* of the tree. We shall assume that the bits represented in the first tier of the tree are distinct, and hence *independent*.

In reality, the portion of the redrawn Tanner graph may not be a tree, since the bits on the first tier may not be distinct. For example, Figure 15.3(b) shows a portion of the actual Tanner graph from Figure 15.1 with the root c_1. In the figure, for example, bit c_2 is checked by both checks z_1 and z_5. There is thus a cycle of length four in the graph, indicated by the dashed lines. This cycle corresponds to the italicized elements of the matrix A in (15.1). Such a cycle means that the bits in the first tier are not independent (as ideally assumed). However, for a sufficiently large code, the probability of such cycles is small (at least for trees represented out to the first tier). We therefore assume a tree structure as portrayed in Figure 15.3(a), with its corresponding independence assumption.

Under the assumption of independence of bits in the first tier, the checks in the tree are statistically independent, given c_n. The decoding algorithm uses information that the checks provide about the bits, as indicated by the following.

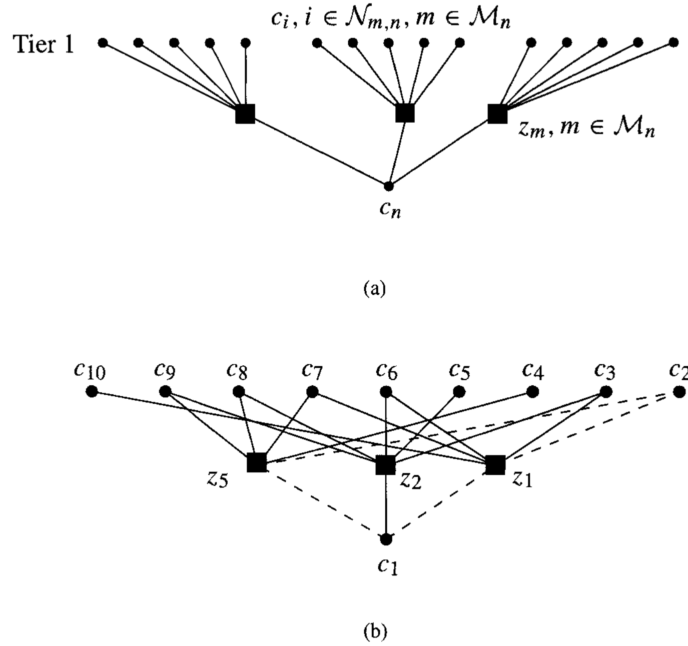


Figure 15.3: A subset of the Tanner graph. (a) Viewed as a tree, with node for c_n as the root. (b) An actual portion of the Tanner graph, with node for c_1 as root, showing a cycle.

Theorem 15.1 *For a bit c_n involved in parity checks $\{z_m, m \in \mathcal{M}_n\}$, if the checks are independent then*

$$q_n(x) = \alpha P(c_n = x|r_n) \prod_{m \in \mathcal{M}_n} P(z_m = 0|c_n = x, \mathbf{r}), \tag{15.9}$$

where α is a normalizing constant.

Proof

$$q_n(x) = P(c_n = 0|\mathbf{r}, \{z_m = 0, m \in \mathcal{M}_n\}) = \frac{P(c_n = 0, \{z_m = 0, m \in \mathcal{M}_n\}|\mathbf{r})}{P(\{z_m = 0, m \in \mathcal{M}_n\}|\mathbf{r})}$$
$$= \frac{1}{P(\{z_m = 0, m \in \mathcal{M}_n\}|\mathbf{r})} P(c_n = x|\mathbf{r}) P(\{z_m = 0, m \in \mathcal{M}_n\}|c_n = x, \mathbf{r}).$$

Due to independence of bits and noise, the conditional probability $P(c_n = x|\mathbf{r})$ can be written as $P(c_n = x|r_n)$. Under the assumption that the checks are independent, the joint probability on the checks can be factored, so that

$$q_n(x) = \frac{1}{P(\{z_m = 0\}, m \in \mathcal{M}_n\}|\mathbf{r})} P(c_n = x|r_n) \prod_{m \in \mathcal{M}_n} P(z_m = 0|c_n = x, \mathbf{r}).$$

The factor dividing this probability can be obtained by marginalizing:

$$q_n(x) = \frac{P(c_n = x|r_n)\prod_{m\in\mathcal{M}_n} P(z_m = 0|c_n = x, \mathbf{r})}{\sum_{x'} P(c_n = x'|r_n)\prod_{m\in\mathcal{M}_n} P(z_m = 0|c_n = x', \mathbf{r})};$$

that is, the factor is just a normalization constant, which we denote as α, which ensures that $q_n(x)$ is a probability mass function with respect to x. □

In (15.9), $q_n(x)$ has two probability factors. The factor $\prod_{m\in\mathcal{M}_n} P(z_m = 0|c_n = x, \mathbf{r})$ has been called the *extrinsic probability*. Like the extrinsic probability used in turbo code decoding, it expresses the amount of information there is about c_n based on the structure imposed by the code. The other factor in (15.9), $P(c_n|r_n)$, expresses how much information there is about c_n based on the measured channel output r_n corresponding to c_n; it has been called the intrinsic probability.

As before, let

$$r_{mn}(x) = P(z_m = 0|c_n = x, \mathbf{r}) \tag{15.10}$$

denote the probability that the mth check is satisfied, given bit c_n. We will derive in Section 15.5.2 an expression to compute this probability. Using (15.10) and Theorem 15.1 we can write

$$q_n(x) = \alpha P(c_n = x|\mathbf{r}) \prod_{m\in\mathcal{M}_n} r_{mn}(x). \tag{15.11}$$

Each bit in tier 1 of the tree has its own set of checks, each with their own corresponding checked bits. This leads to a situation as in Figure 15.4. To do decoding on the tree, we again invoke an independence assumption: the set of bits connected to a check subtree rooted at a bit in the first tier are independent. (As before, cycles in the Tanner graph violate this assumption.) The probability of a bit in the first tier of the tree is computed using bits from the second tier of the tree. Let n' be the index of a bit in the first tier connected to the check z_m. Let

$$q_{mn'}(x) = P(c_{n'} = x|\text{all checks involving } c_{n'}, \text{ except for check } z_m, \text{ are satisfied})$$

or, more briefly,

$$q_{mn'}(x) = P(c_{n'} = x|\{z_{m'} = 0, m' \in \mathcal{M}_{n',m}\}, \mathbf{r}).$$

Then, slightly modifying the results of Theorem 15.1,

$$q_{mn'}(x) = \alpha P(c_{n'} = x|r_{n'}) \prod_{m'\in\mathcal{M}_{n',m}} r_{m'n'}(x). \tag{15.12}$$

If there are w_c parity checks associated with each bit, then the computation (15.12) involves $w_c - 1$ checks. Using (15.12), the probabilities for bits in the first tier can be computed from the checks in the second tier, following which the probability at the root c_n can be computed using (15.11).

Since the product in (15.12) is computed down the columns of the A matrix (across the checks), updating $q_{mn}(x)$ is called the *vertical step* of the decoding algorithm. The process can be described in words as follows: For each nonzero position (m, n) of A, compute the product of $r_{m',n}(x)$ down the nth column of A, excluding the value at position (m, n), then multiply by the channel posterior probability. There are $w_c N$ values of q_{mn} to update, each requiring $O(w_c)$ operations, so this step has $O(N)$ complexity.

If the graph associated with the code were actually a tree with independence among the bits associated with the checks on each tier, this procedure could be recursively applied,

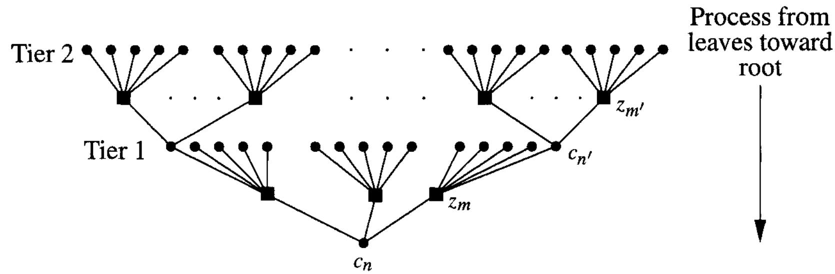


Figure 15.4: The two-tier tree.

starting at the leaf nodes of the tree — those not connected to further checks — and working toward the root. The probabilities at the leaf nodes could be computed using the channel posterior probabilities $p_n(x)$ defined in (15.5). Working from leaves toward the root, the probability $q_{mn'}(x)$ would be computed for each node $c_{n'}$ on the second-to-last tier, using the leaf nodes (the last tier). Then $q_{mn'}$ would be computed for each node on the third-to-last tier, using the probabilities obtained from the second-to-last tier, and so forth until the root node is reached.

However, it is time to face reality: The graph associated with the code is not actually a tree. The node c_n which we have called the root of the tree is not a distinguished root node, but is actually an arbitrary node. We deal with this reality by considering each node c_n in turn as if it were the "root" of a tree. For each c_n, we consider each parity check z_m, $m \in \mathcal{M}_n$ associated with it, and compute $q_{mn'}(x)$ as defined in (15.12), involving other $w_c - 1$ checks and the other bits of the first tier of the "tree" with that c_n as the root. The algorithm does not actually propagate information from leaf to root, but instead propagates information throughout the graph as if each node were the root. If there were no cycles in the tree, the algorithm would, in fact, result in an exact computation at each node of the tree. But as bits connect to checks to other bits through the iterations, there must eventually be some cycles in the graph. These violate the independence assumptions and lead to only approximate, but still very impressive, results.

15.5.2 Horizontal Step: Updating $r_{mn}(x)$

The probability $r_{mn}(x) = P(z_m = 0|c_n = x, \mathbf{r})$, depends on all of the bits $\{c_{n'}, n' \in \mathcal{N}_m\}$ that participate in z_m, so an expression involving all of the bits that influence check z_m is necessary. The desired probability can be computed by marginalizing,

$$P(z_m = 0|c_n = x, \mathbf{r}) = \sum_{\{x_{n'}, n' \in \mathcal{N}_{m,n}\}} P(z_m = 0, \{c_{n'} = x_{n'}, n' \in \mathcal{N}_{m,n}\}|c_n = x, \mathbf{r}), \tag{15.13}$$

where the sum is taken over all possible binary sequences $\{x_{n'}\}$ with $n' \in \mathcal{N}_{m,n}$. (Continuing Example 15.4, for $\mathcal{N}_{2,1} = \{3, 5, 6, 8, 9\}$, the variables in the sum $\{x_3, x_5, x_6, x_8, x_9\}$ take on all 2^5 possible binary values, from $(0, 0, 0, 0, 0)$ through $(1, 1, 1, 1, 1)$.) The joint

probability in (15.13) can be factored using conditioning as

$$r_{mn}(x) = \sum_{\{x_{n'}:n' \in \mathcal{N}_{m,n}\}} \big[P(z_m = 0|c_n = x, \{c_{n'} = x_{n'} : n' \in \mathcal{N}_{m,n}\}, \mathbf{r}) \\ \times P(\{c_{n'} = x_{n'} : n' \in \mathcal{N}_{m,n}\}|\mathbf{r})\big]. \tag{15.14}$$

Under the assumption that the bits $\{c_{n'}, n' \in \mathcal{N}_{m,n}\}$ are independent — which is true only if there are no cycles in the graph — (15.14) can be written as

$$r_{mn}(x) = \sum_{\{x_{n'}:n' \in \mathcal{N}_{m,n}\}} \big[P(z_m = 0|c_n = x, \{c_{n'} = x_{n'} : n' \in \mathcal{N}_{m,n}\}) \\ \times \prod_{l' \in \mathcal{N}_{m,n}} P(c_{l'} = x_{l'}|\mathbf{r})\big], \tag{15.15}$$

where the conditioning on $\mathbf{r}$ in the first probability has been dropped since the parity is independent of the observations, given the values of the bits. The conditional probability in the sum, $P(z_m = 0|c_n = x, \{c_{n'} = x_{n'} : n' \in \mathcal{N}_{m,n}\})$ is either 0 or 1, depending on whether the check condition for the bits in $\mathcal{N}_m$ is actually satisfied. Only those terms for which $\sum_{n' \in \mathcal{N}_m} x_{n'} = 0$ — or, in other words, for which $x_n = \sum_{n' \in \mathcal{N}_{m,n}} x_{n'}$ — contribute to the probability, so

$$r_{mn}(x) = \sum_{\{\{x_{n'}, n' \in \mathcal{N}_{m,n}\}: x = \sum_l x_l\}} \prod_{l \in \mathcal{N}_{m,n}} P(c_l = x_l|\mathbf{r}). \tag{15.16}$$

That is, $r_{mn}(x)$ is the total probability of sequences $\{x_{n'}, n' \in \mathcal{N}_{m,n}\}$ of length $|\mathcal{N}_{m,n}|$ whose sum is equal to x, where each element $x_{n'}$ in each sequence occurs with probability $P(c_{n'}|\mathbf{r})$. At first appearance, (15.16) seems like a complicated sum to compute. However, there are significant computational simplifications that can be made using a graph associated with this problem.

For a sequence of bit values $\{x_{n'}, n' \in \mathcal{N}_{m,n}\}$ involved in check m, let

$$\zeta(k) = \sum_{i=1}^{k} x_{\mathcal{N}_{m,n}(i)}$$

be the sum of the first k bits of $\{x_{n'}\}$. Then

$$\zeta(L) = \sum_{i=1}^{L} x_{\mathcal{N}_{m,n}(i)} = \sum_{l \in \mathcal{N}_{m,n}} x_l, \tag{15.17}$$

where $L = |\mathcal{N}_{m,n}|$. The sum in (15.17) appears in the index of the sum in (15.16).

The values of $\zeta(k)$ as a function of k may be represented using the trellis shown in Figure 15.5. There are two states in the trellis, 0 and 1, representing the possible values of $\zeta(k)$. Starting from the 0 state at $k = 0$, $\zeta(1)$ takes on two possible values, depending on the value of $x_{\mathcal{N}_{m,n}(1)}$. Then $\zeta(2)$ takes on two possible values, depending on the value of $\zeta(1)$ and $x_{\mathcal{N}_{m,n}(2)}$. The final state of the trellis represents $\zeta(L)$.

Let the values $x_{n'}$ occur with probability $P(x_{n'}) = P(c_{n'}|\mathbf{r})$. Using this graph, it may be observed that $r_{mn}(x)$ is the probability that $x = \zeta(L)$, where the probability is computed over *every possible path* through the trellis. The problem still looks quite complicated.

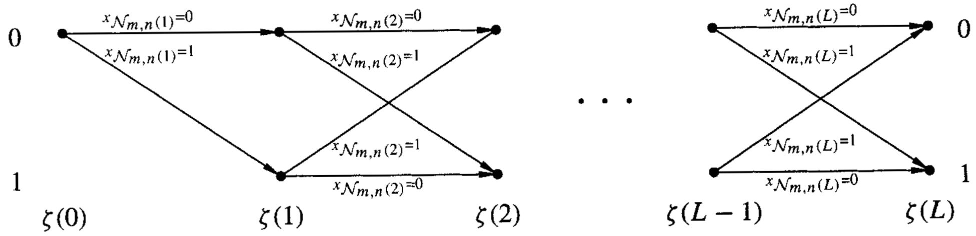


Figure 15.5: The trellis associated with finding $r_{mn}(x)$. The probability of all possible paths through the trellis is computed inductively.

However, due to the structure of the trellis, the probabilities can be recursively computed. Note that

$$P(\zeta(1)=x)=P(x_{\mathcal{N}_{m,n}(1)}=x)=P(c_{\mathcal{N}_{m,n}(1)}=x|\mathbf{r}).$$

Knowing $P(\zeta(1)=0)$ and $P(\zeta(1)=1)$, the event that $\zeta(2)=0$ can occur in two ways: either $\zeta(1)=0$ and $x_{\mathcal{N}_{m,n}(2)}=0$, or $\zeta(1)=1$ and $x_{\mathcal{N}_{m,n}(2)}=1$. Similarly, the event that $\zeta(2)=1$ can occur in two ways. Thus

$$\begin{aligned}P(\zeta(2)=0)&=P(\zeta(1)=0,x_{\mathcal{N}_{m,n}(2)}=0)+P(\zeta(1)=1,x_{\mathcal{N}_{m,n}}(2)=1)\\P(\zeta(2)=1)&=P(\zeta(1)=1,x_{\mathcal{N}_{m,n}(2)}=0)+P(\zeta(1)=0,x_{\mathcal{N}_{m,n}}(2)=1).\end{aligned}\tag{15.18}$$

By the (assumed) independence of the bits, the joint probabilities in (15.18) can be factored, so (15.18) can be written

$$\begin{aligned}P(\zeta(2)=0)&=P(\zeta(1)=0)P(x_{\mathcal{N}_{m,n}(2)}=0)+P(\zeta(1)=1)P(x_{\mathcal{N}_{m,n}}(2)=1)\\P(\zeta(2)=1)&=P(\zeta(1)=1)P(x_{\mathcal{N}_{m,n}(2)}=0)+P(\zeta(1)=0)P(x_{\mathcal{N}_{m,n}}(2)=1).\end{aligned}\tag{15.19}$$

Let $w_k(x)$ be the probability

$$w_k(x)=P(\zeta(k)=x).$$

Then (15.19) can be written

$$\begin{aligned}w_2(0)&=w_1(0)P(x_{\mathcal{N}_{m,n}(2)}=0)+w_1(1)P(x_{\mathcal{N}_{m,n}(2)}=1)\\w_2(1)&=w_1(1)P(x_{\mathcal{N}_{m,n}(2)}=0)+w_1(0)P(x_{\mathcal{N}_{m,n}(2)}=1),\end{aligned}$$

and, in general (under assumptions of independence)

$$\begin{aligned}w_k(0)&=w_{k-1}(0)P(x_{\mathcal{N}_{m,n}(k)}=0)+w_{k-1}(1)P(x_{\mathcal{N}_{m,n}(k)}=1)\\w_k(1)&=w_{k-1}(1)P(x_{\mathcal{N}_{m,n}(k)}=0)+w_{k-1}(0)P(x_{\mathcal{N}_{m,n}(k)}=1).\end{aligned}\tag{15.20}$$

The recursion is initialized with $w_0(0)=1$, $w_0(1)=0$. (It may be noted that the recursion (15.20) is an instance of the BCJR algorithm. In this case, the w probabilities are directly analogous to the forward probabilities of the BCJR algorithm, usually denoted by α.)

By the recursive computation (15.20), the probabilities of all possible paths through the trellis are computed. By the definition of $w_k(x)$, $P(\zeta(L)=x)=w_L(x)$ and by (15.16),

$$r_{mn}(0)=w_L(0)\qquad r_{mn}(1)=w_L(1).\tag{15.21}$$

Now consider computing $P(z_m=0|c_n=x,\mathbf{r})$ for a check node z_m in the first tier of a multi-tier tree, such as that portrayed in Figure 15.4. The bit probabilities $P(x_n')=$

$P(c_{n'}|r_{n'})$ necessary for the recursion (15.20) are obtained from the bit nodes in tier 1 of the tree, which depend also upon check nodes in tier 2, excluding check z_m. That is, we use

$$P(c_{n'} = x|\mathbf{r}) = q_{mn'}(x)$$

as the bit probabilities. Then the recursive update rule is written

$$\begin{aligned} w_k(0) &= w_{k-1}(0)q_{m,\mathcal{N}_{m,n}(k)}(0) + w_{k-1}(1)q_{m,\mathcal{N}_{m,n}(k)}(1) \\ w_k(1) &= w_{k-1}(1)q_{m,\mathcal{N}_{m,n}(k)}(0) + w_{k-1}(0)q_{m,\mathcal{N}_{m,n}(k)}(1). \end{aligned} \quad (15.22)$$

This computation is used for each iteration of the decoding algorithm.

The probability update algorithm (15.22) can be extended to codes with larger than binary alphabets by creating a trellis with more states. The LDPC decoding algorithm is thus applicable to linear codes over arbitrary fields. For binary codes, however, (15.22) can be re-expressed in an especially elegant way in terms of differences of probabilities. Let $\delta q_{ml} = q_{ml}(0) - q_{ml}(1)$ and $\delta r_{ml} = r_{ml}(0) - r_{ml}(1)$. From (15.22),

$$w_k(0) - w_k(1) = (w_{k-1}(0) - w_{k-1}(1))(q_{m\mathcal{N}_{m,n}(k)}(0) - q_{m\mathcal{N}_{m,n}(k)}(1)).$$

Inductively,

$$w_k(0) - w_k(1) = \prod_{i=1}^{k}(q_{m\mathcal{N}_{m,n}(i)}(0) - q_{m\mathcal{N}_{m,n}(i)}(1)).$$

Using $\delta r_{mn} = r_{mn}(0) - r_{mn}(1)$, we have

$$\delta r_{mn} = \prod_{l' \in \mathcal{N}_{m,n}} \delta q_{m,l'}. \quad (15.23)$$

In words what this update says is: For each nonzero element (m, n) of A, compute the product of the $\delta q_{mn'}$ *across* the mth row, except for the value at column n. This step is therefore called the horizontal step. The entire update has complexity $O(N)$.

Having found the δr_{mn} and using the fact that $r_{mn}(0) + r_{mn}(1) = 1$, the probabilities can be computed as

$$r_{mn}(0) = (1 + \delta r_{mn})/2 \qquad r_{mn}(1) = (1 - \delta r_{mn})/2. \quad (15.24)$$

15.5.3 Terminating and Initializing the Decoding Algorithm

As before, let $q_n(x) = P(c_n = x|\{z_m : m \in \mathcal{M}_n\})$. This can be computed as

$$q_n(x) = \alpha_n p_n(x) \prod_{m \in \mathcal{M}_n} r_{mn}(x),$$

where α_n is chosen so that $q_n(0) + q_n(1) = 1$. These pseudoposterior probabilities are used to make decisions on x: if $q_n(1) > 0.5$ a decision is made to set $\hat{c}_n = 1$.

Since the decoding criterion computes $P(c_n = x|\mathbf{r}$, checks involving $c_n)$, with each bit probability computed separately, it is possible that the set of bit decisions obtained by the decoding algorithm do not initially *simultaneously* satisfy all of the checks. This observation can be used to formulate a stopping criterion. If $A\hat{\mathbf{c}} = 0$, that is, all checks are simultaneously satisfied, then decoding is finished. Otherwise, the algorithm repeats from the horizontal step.

It may happen that $A\hat{\mathbf{c}} = 0$ is not satisfied after the specified maximum number of iterations. In this case, a *decoding failure* is declared; this is indicative of an error event

which exceeds the ability of the code to correct within that number of iterations. Knowledge of a decoding failure is important, but not available with many codes, including turbo codes. In some systems, a decoding failure may invoke a retransmission of the faulty codeword.

The iterative decoding algorithm is initialized by setting $q_{mn}(x) = p_n(x)$. That is, the probability conditional on the checks $q_{mn}(x)$ is set to the channel posterior probability, which is what would be used if the Tanner graph were actually a tree.

15.5.4 Summary of the Algorithm

Algorithm 1 is a concise statement of the decoding algorithm; the details are discussed in the sections below. (This particular formulation of the decoding algorithm is due to [217], while concepts from the description are from [113].)

Algorithm 15.1 Iterative Decoding Algorithm for Binary LDPC Codes

---

Input: A, the channel posterior probabilities $p_n(x) = P(c_n = x|r_n)$, and the maximum # of iterations, L.
Initialization: Set $q_{mn}(x) = p_n(x)$ for all (m, n) with $A(m, n) = 1$.
Horizontal step: For each (m, n) with $A(m, n) = 1$:
Compute $\delta q_{ml} = q_{ml}(0) - q_{ml}(1)$
Compute

$$\delta r_{mn} = \prod_{\{n' \in \mathcal{N}_{m,n}\}} \delta q_{mn'} \tag{15.25}$$

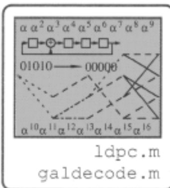
Compute $r_{mn}(1) = (1 - \delta r_{mn})/2$ and $r_{mn}(0) = (1 + \delta r_{mn})/2$.
Vertical Step: For each (m, n) with $A(m, n) = 1$:
Compute

$$q_{mn}(0) = \alpha_{mn} p_n(0) \prod_{\{m' \in \mathcal{M}_{n,m}\}} r_{m'n}(0) \quad \text{and} \quad q_{mn}(1) = \alpha_{mn} p_n(1) \prod_{\{m' \in \mathcal{M}_{n,m}\}} r_{m'n}(1) \tag{15.26}$$

where α_{mn} is chosen so $q_{mn}(0) + q_{mn}(1) = 1$.
Also compute the "pseudoposterior" probabilities

$$q_n(0) = \alpha_n p_n(0) \prod_{\{m' \in \mathcal{M}_n\}} r_{m'n}(0) \quad \text{and} \quad q_n(1) = \alpha_n p_n(1) \prod_{\{m' \in \mathcal{M}_n\}} r_{m'n}(1)$$

where α_n is chosen so that $q_n(0) + q_n(1) = 1$.
Make a tentative decision: Set $\hat{c}_n = 1$ if $q_n(1) > 0.5$, else set $\hat{c}_n = 0$.
If $A\hat{\mathbf{c}} = 0$, then **Stop**. Otherwise, if #iterations $< L$, loop to **Horizontal Step**
Otherwise, declare a decoding failure and **Stop**.



This algorithm (or its log likelihood equivalent) is sometimes referred to as the *sum-product* decoding algorithm.

Example 15.6 For the parity check matrix (15.1) of Example 15.1 and the received probability vector of Example 15.5, the decoding proceeds as follows:
Initialization: Set $q_{mn}(x) = p_n(x)$ from (15.7)

$$q_{mn}(1)$$

$$\begin{bmatrix} 0.22 & 0.16 & 0.19 & & & 0.87 & 0.18 & & & 0.76 \\ 0.22 & & 0.19 & & 0.55 & 0.87 & & 0.79 & 0.25 & \\ & & 0.19 & 0.48 & 0.55 & & 0.18 & & 0.25 & 0.76 \\ & 0.16 & & 0.48 & 0.55 & 0.87 & & 0.79 & & 0.76 \\ 0.22 & 0.16 & & 0.48 & & & 0.18 & 0.79 & 0.25 & \end{bmatrix}$$

Iteration 1: Horizontal Step:

$$\delta r_{mn}$$

$$\begin{bmatrix} 0.1 & 0.086 & 0.094 & & & -0.079 & 0.091 & & & -0.11 \\ -0.013 & & -0.012 & & 0.075 & 0.01 & & 0.013 & -0.015 & \\ & & 0.00067 & 0.01 & -0.0041 & & 0.00064 & & 0.00083 & -0.00079 \\ & 0.00089 & & 0.015 & -0.0061 & -0.00082 & & -0.001 & & -0.0012 \\ -0.005 & -0.0042 & & -0.071 & & & -0.0044 & 0.0049 & -0.0057 & \end{bmatrix}$$

$$r_{mn}(1)$$

$$\begin{bmatrix} 0.45 & 0.46 & 0.45 & & & 0.54 & 0.45 & & & 0.56 \\ 0.51 & & 0.51 & & 0.46 & 0.49 & & 0.49 & 0.51 & \\ & & 0.5 & 0.49 & 0.5 & & 0.5 & & 0.5 & 0.5 \\ & 0.5 & & 0.49 & 0.5 & 0.5 & & 0.5 & & 0.5 \\ 0.5 & 0.5 & & 0.54 & & & 0.5 & 0.5 & 0.5 & \end{bmatrix}$$

Iteration 1: Vertical Step:

$$q_{mn}(1)$$

$$\begin{bmatrix} 0.23 & 0.16 & 0.19 & & & 0.87 & 0.18 & & & 0.76 \\ 0.19 & & 0.16 & & 0.56 & 0.89 & & 0.79 & 0.25 & \\ & & 0.17 & 0.51 & 0.52 & & 0.16 & & 0.26 & 0.8 \\ & 0.14 & & 0.51 & 0.51 & 0.88 & & 0.78 & & 0.8 \\ 0.19 & 0.14 & & 0.47 & & & 0.15 & 0.79 & 0.26 & \end{bmatrix}$$

$$q_n(1)$$

$$\begin{bmatrix} 0.19 & 0.14 & 0.17 & 0.5 & 0.52 & 0.88 & 0.16 & 0.78 & 0.26 & 0.8 \end{bmatrix} \tag{15.27}$$

$$\hat{\mathbf{c}} = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 \end{bmatrix} \quad \mathbf{z} = \begin{bmatrix} 0 & 1 & 1 & 1 & 0 \end{bmatrix}$$

At the end of the first iteration, the parity check condition is not satisfied. The algorithm runs through two more iterations (not shown here). At the end, the decoded value

$$\hat{\mathbf{c}} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}$$

is obtained, which exactly matches the transmitted code vector $\mathbf{c}$ of (15.6).

Even though the minimum distance of the code is 4, the code was able to decode beyond the minimum distance (in this case) and correct two errors. □

15.5.5 Message Passing Viewpoint

The decoding algorithm can be viewed as an instance of a message passing algorithm. Messages are passed among the nodes in the Tanner graph. In the horizontal step, "messages" in the form of probability vectors $q_{mn}(x)$ are passed to the check nodes, where the messages are combined using (15.23). In the vertical step, "messages" in the form of probability vectors $r_{mn}(x)$ are passed to the bit nodes, where the messages are combined using (15.12). The iteration of the algorithm may be viewed as message passing through the graph obtained by concatenating several copies of the Tanner graph, as shown in Figure 15.6.

In the absence of cycles, such message passing algorithms compute exact probabilities [258]. However, the presence of cycles in the graph means that the decoding algorithm computes only approximate solutions. Careful analysis of graphs with cycles, however, [367] has shown theoretically that the approximate algorithm still provides effective decoding capability; this conclusion is borne out by repeated simulation studies.

15.5.6 Likelihood Ratio Decoder Formulation

In this section we re-derive the decoding algorithm, this time in terms of log likelihood ratios. This derivation serves to emphasize some of the likelihood arithmetic presented in Appendix A and reinforce the concept of extrinsic probability that arose in the context of turbo decoding. Computationally, it avoids having to compute normalizations. Furthermore,

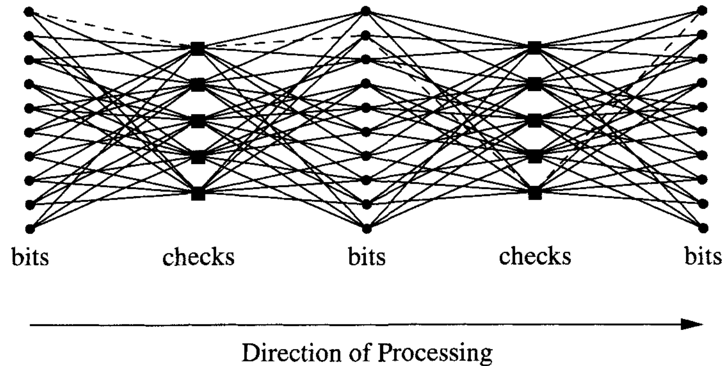


Figure 15.6: Processing information through the graph determined by A. The dashed line illustrates a cycle of length 4.

the likelihood ratio algorithm is used in the density evolution analysis presented in Section 15.8. However, it is only applicable to binary codes, so the general formulation above has value for nonbinary codes. (The reader is advised to review the log likelihood rule in Appendix A before reading this section.)

Let

$$\lambda(c_n|\mathbf{r}) = \log \frac{P(c_n = 1|\mathbf{r})}{P(c_n = 0|\mathbf{r})} = \log \frac{P(c_n = 1|r_n, \{r_i, i \neq n\})}{P(c_n = 0|r_n, \{r_i, i \neq n\})}. \tag{15.28}$$

By application of Bayes' rule, the numerator can be expressed as

$$\begin{aligned}
P(c_n = 1|r_n, \{r_i, i \neq n\}) &= \frac{p(r_n, c_n = 1, \{r_i, i \neq n\})}{p(r_n, \{r_i, i \neq n\})} \\
&= \frac{p(r_n|c_n = 1, \{r_i, i \neq n\})p(c_n = 1, \{r_i, i \neq n\})}{p(r_n|\{r_i, i \neq n\})p(\{r_i, i \neq n\})} \\
&= \frac{p(r_n|c_n = 1)p(c_n = 1, \{r_i, i \neq n\})}{p(r_n|\{r_i, i \neq n\})p(\{r_i, i \neq n\})} \\
&= \frac{p(r_n|c_n = 1)P(c_n = 1|\{r_i, i \neq n\})}{p(r_n|\{r_i, i \neq n\})},
\end{aligned}$$

where we have used the fact that, given c_n, r_n is independent of $\{r_i, i \neq n\}$. A similar expression follows for the denominator. The likelihood ratio (15.28) can thus be written

$$\lambda(c_n|\mathbf{r}) = \log \frac{p(r_n|c_n = 1)}{p(r_n|c_n = 0)} + \log \frac{P(c_n = 1|\{r_i, i \neq n\})}{P(c_n = 0|\{r_i, i \neq n\})}.$$

For a Gaussian channel, we have seen (see, e.g., (14.39)) that

$$\log \frac{p(r_n|c_n = 1)}{p(r_n|c_n = 0)} = L_c r_n,$$

where $L_c = 2\sqrt{E_c}/\sigma^2$ is the channel reliability. We observe that the terms in the sum can be identified as

$$\lambda(c_n|\mathbf{r}) = \underbrace{L_c r_n}_{\text{intrinsic}} + \underbrace{\log \frac{P(c_n = 1|\{r_i, i \neq n\})}{P(c_n = 0|\{r_i, i \neq n\})}}_{\text{extrinsic}}, \tag{15.29}$$

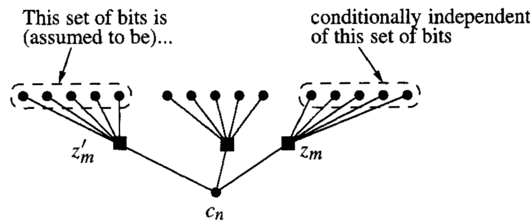


Figure 15.7: Conditional independence among the sets of bits.

where the intrinsic term is determined by the explicit measurement r_n affecting the bit c_n and the extrinsic term is determined by the information provided by all the *other* observations and the code structure.

Let us now express the probabilities in the extrinsic term in terms of the parity checks. Let $z_{m,n}$ denote the parity check computed using the mth check associated with c_n, except for c_n. That is,

$$z_{m,n} = \sum_{i \in \mathcal{N}_{m,n}} c_i.$$

If $c_n = 1$, then $z_{m,n} + c_n = 0$; that is, $z_{m,n} = 1$ for all the checks $m \in \mathcal{M}_n$ in which c_n participates. Similarly, if $c_n = 0$, then $z_{m,n} = 0$ for all $m \in \mathcal{M}_n$. We can write (15.29) as

$$\lambda(c_n|\mathbf{r}) = L_c r_n + \log \frac{P(z_{m,n} = 1 \text{ for all } m \in \mathcal{M}_n | \{r_i, i \neq n\})}{P(z_{m,n} = 0 \text{ for all } m \in \mathcal{M}_n | \{r_i, i \neq n\})}.$$

We now invoke the assumption that the graph associated with the code is cycle-free. Then the set of bits associated with $z_{m,n}$ are independent of the bits associated with $z_{m',n}$, for $m' \neq m$. (See Figure 15.7.) We thus have

$$\begin{aligned} \lambda(c_n|\mathbf{r}) &= L_c r_n + \log \frac{\prod_{m \in \mathcal{M}_n} P(z_{m,n} = 1 | \{r_i, i \neq n\})}{\prod_{m \in \mathcal{M}_n} P(z_{m,n} = 0 | \{r_i, i \neq n\})} \\ &= L_c r_n + \sum_{m \in \mathcal{M}_n} \log \frac{P(z_{m,n} = 1 | \{r_i, i \neq n\})}{P(z_{m,n} = 0 | \{r_i, i \neq n\})}. \end{aligned}$$

Let us define the log likelihood ratio

$$\lambda(z_{m,n} | \{r_i, i \neq n\}) = \frac{P(z_{m,n} = 1 | \{r_i, i \neq n\})}{P(z_{m,n} = 0 | \{r_i, i \neq n\})}.$$

Then

$$\lambda(c_n|\mathbf{r}) = L_c r_n + \sum_{m \in \mathcal{M}_n} \lambda(z_{m,n} | \{r_i, i \neq n\}) = L_c r_n + \sum_{m \in \mathcal{M}_n} \lambda\left(\sum_{j \in \mathcal{N}_{m,n}} c_j | \{r_i, i \neq n\} \right).$$

Under the assumption that the checks in $z_{m,n}$ are conditionally independent (if there are no cycles in the graph), we invoke the tanh rule of (A.1) to write

$$\lambda(c_n|\mathbf{r}) = L_c r_n - 2 \sum_{m \in \mathcal{M}_n} \tanh^{-1}\left(\prod_{j \in \mathcal{N}_{m,n}} \tanh\left(-\frac{\lambda(c_j | \{r_i, i \neq n\})}{2} \right) \right). \tag{15.30}$$

Computation now requires knowing the $\lambda(c_j|\{r_i, i \neq n\}$, the conditional likelihoods of the bits which connect to the checks of c_n. How are these obtained? They are obtained the same way as $\lambda(c_n)$: that is, we remove from c_n its distinguished role, and treat all bit nodes alike. However, we must be careful to deal only with the extrinsic information.

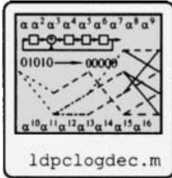
Let

$$\eta_{m,n} = -2\tanh^{-1}\left(\prod_{j\in\mathcal{N}_{m,n}} \tanh\left(-\frac{\lambda(c_j|\{r_i, i\neq n\})}{2}\right)\right). \tag{15.31}$$

This can be thought of as the "message" which is passed from the check node m to the bit node n. Then (15.30) can be written

$$\lambda(c_n|\mathbf{r}) = L_c r_n + \sum_{m\in\mathcal{M}_n} \eta_{m,n}. \tag{15.32}$$

This can be thought of as a message that the bit node c_n sends to its check nodes.



If we were to employ an iterative decoder alternating between (15.31) and (15.32), a problem would develop. The likelihoods $\lambda(c_n|\mathbf{r})$ each contain the prior information $L_c r_n$, which would lead to a bias in $\eta_{m,n}$. What we need to do is remove from the "message" that bit node n sends to check node m the message that it has already received from that check node. This represents the extrinsic information passed by the decoder.

The log likelihood decoder can now be described.

Algorithm 15.2 Iterative Log Likelihood Decoding Algorithm for Binary LDPC Codes

Input: A, the received vector $\mathbf{r}$, the maximum # of iterations L, and the channel reliability L_c.
Initialization: Set $\eta_{m,n}^{[0]} = 0$ for all (m,n) with $A(m,n) = 1$.
Set $\lambda_n^{[0]} = L_c r_n$
Set the loop counter $l = 1$.
Check node update: For each (m,n) with $A(m,n) = 1$: Compute

$$\eta_{m,n}^{[l]} = -2\tanh^{-1}\left(\prod_{j\in\mathcal{N}_{m,n}} \tanh\left(-\frac{\lambda_j^{[l-1]} - \eta_{m,j}^{[l-1]}}{2}\right)\right) \tag{15.33}$$

Bit node update: For $n = 1, 2, \ldots, N$: Compute

$$\lambda_n^{[l]} = L_c r_n + \sum_{m\in\mathcal{M}_n} \eta_{m,n}^{[l]} \tag{15.34}$$

Make a tentative decision: Set $\hat{c}_n = 1$ if $\lambda_n[l] > 0$, else set $\hat{c}_n = 0$.
If $A\hat{\mathbf{c}} = 0$, then **Stop**. Otherwise, if #iterations $< L$, loop to **Check node update**
Otherwise, declare a decoding failure and **Stop**.

Example 15.7 For the code of Example 15.1 and the received vector of Example 15.5, we obtain the following. Initially,

$$\lambda^{[0]} = \begin{bmatrix} -1.3 & -1.7 & -1.5 & -0.08 & 0.2 & 1.9 & -1.5 & 1.3 & -1.1 & 1.2 \end{bmatrix}.$$

In the first iteration,

$$\eta^{[1]} = \begin{bmatrix} -0.21 & -0.17 & -0.19 & & & 0.16 & -0.18 & & & 0.22 \\ 0.027 & & 0.024 & & -0.15 & -0.02 & & -0.026 & 0.03 & \\ & & -0.0013 & -0.021 & 0.0083 & & -0.0013 & & -0.0017 & 0.0016 \\ & -0.0018 & & -0.03 & 0.012 & 0.0016 & & 0.0021 & & 0.0023 \\ 0.01 & 0.0083 & & 0.14 & & & 0.0088 & -0.0097 & 0.011 & \end{bmatrix}$$

and

$$\lambda^{[1]} = \begin{bmatrix} -1.4 & -1.8 & -1.6 & 0.011 & 0.072 & 2 & -1.7 & 1.3 & -1.1 & 1.4 \end{bmatrix}$$

which corresponds to the probability vector

$$p = \begin{bmatrix} 0.19 & 0.14 & 0.17 & 0.5 & 0.52 & 0.88 & 0.16 & 0.78 & 0.26 & 0.8 \end{bmatrix}.$$

This matches exactly the probability vector found by the probability decoder. Other iterations proceed similarly, yielding identical probability values and decoded values. □

The check node update (15.33) can be also simplified using the min-sum approximation (see Appendix A), at a cost of about 0.5 dB in performance.

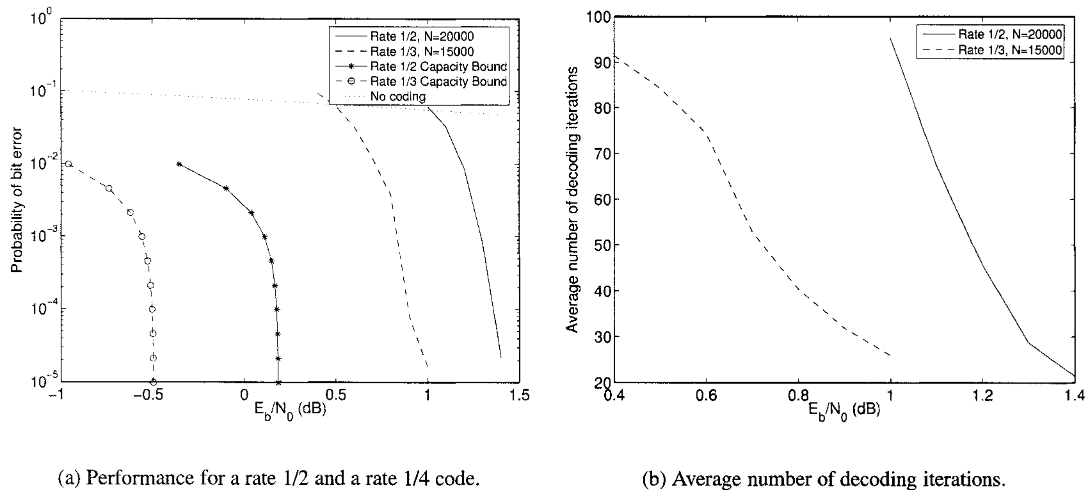
15.6 Why Low-Density Parity-Check Codes?

LDPC codes have excellent distance properties. Gallager showed that for random LDPC codes, the minimum distance $d_{\min}$ between codewords increases with N when column and row weights are held fixed [112, p. 5], that is, as they become increasingly sparse. Sequences of LDPC codes as $N \to \infty$ have been proved to reach channel capacity [217]. LDPC codes thus essentially act like the random codes used in Shannon's original proof of the channel coding theorem. Note, however, that for the nonrandom constructive techniques summarized in Section 15.11, there may be an error floor (see [291]).

The decoding algorithm is tractable. As observed, the decoding algorithm has complexity linearly proportional to the length of the code. Thus we get the benefit of a random code, but without the exponential decoding complexity usually associated with random codes. These codes thus fly in the face of the now outdated conventional coding wisdom, that there are "few known constructive codes that are good, fewer still that are practical, and none at all that are both practical and very good." [217, p. 399]. It is the extreme sparseness of the parity check matrix for LDPC codes that makes the decoding particularly attractive. The low-density nature of the parity check matrix thus, fortuitously, contributes both to good distance properties and the relatively low complexity of the decoding algorithm.

For finite length (but still long) codes, excellent coding gains are achievable as we briefly illustrate. Figure 15.8(a) shows the BPSK probability of error performance for two LDPC codes, a rate 1/2 code with $(N, K) = (20000, 10000)$ and a rate 1/4 code with $(N, K) = (13298, 3296)$ from [215], compared with uncoded BPSK. These plots were made by adding simulated Gaussian noise to a codeword then iterating the algorithm up to 1000 times. As many as 100000 blocks of bits were simulated to get the performance points at the higher SNRs. In all cases, the errors counted in the probability of error are *detected* errors; in no case did the decoder declare a successful decoding that was erroneous! (This is not always the case. We have found that for very short toy codes, the decoder may terminate with the condition $A\hat{\mathbf{c}} = \mathbf{0}$, but $\hat{\mathbf{c}}$ is erroneous. However, for long codes, decoding success meant correct decoding.)

Figure 15.8(b) shows the *average* number of iterations to complete the decoding. (The peak number of iterations, not shown, was in many instances much higher.) The high number of iterations suggests a rather high potential decoding complexity, even though



(a) Performance for a rate 1/2 and a rate 1/4 code.

(b) Average number of decoding iterations.

Figure 15.8: Illustration of the decoding performance of LPDC codes and the number of iterations to achieve decoding.

each iteration is readily computed. As suggested by EXIT chart analysis, as the decoding threshold is approached, the number of iterations must increase.

There are, of course, some potential disadvantages to LDPC codes. First, the best code performance is obtained for very long codes (as predicted by the channel coding theorem). This long block length, combined with the need for iterative decoding, introduces latency which is unacceptable in many applications. Second, since the G matrix is not necessarily sparse, the encoding operation may have complexity $O(N^2)$. Some progress in reducing complexity is discussed in Section 15.12.

LDPC codes have an error floor, just as turbo codes do. Some efforts to lower the error floor have been made by increasing the girth of the Tanner graph, but this has met with only limited success. A tradeoff between decoding thresholds (from the density evolution analysis) and the error floor has been observed. Codes having very low error floors tend to perform around half a dB worse in terms of their decoding thresholds. However, this has not yet led to any design methodologies to reduce error floor.

15.7 The Iterative Decoder on General Block Codes

There initially seems to be nothing impeding the use of the sum-product decoder for a general linear block code: it simply relies on the parity check matrix. This would mean that there is a straightforward iterative soft decoder for every linear block code. In fact, Figure 15.9 shows the use of the soft decoder on a (7, 4) Hamming code. The soft decoding works better than conventional hard decoding by about 1.5 dB.

However, for larger codes a serious problem arises. Given a generator matrix G, the corresponding H matrix that might be found for it is not likely to be very sparse, so the

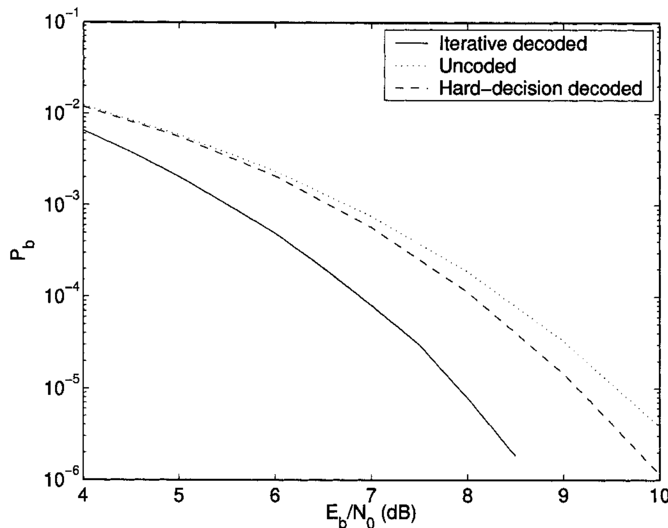


Figure 15.9: Comparison of hard-decision Hamming decoding and sum-product (iterative) decoding.

resulting Tanner graph has many cycles in it. In fact, a threshold is reached at some density of the parity check matrix at which the decoder seems to break down completely. Also, the problem of finding the sparsest representation of a matrix is as computationally difficult (NP-complete) as performing a ML decoding.

15.8 Density Evolution

Having described LDPC codes and the decoding, we now turn attention to some analytical techniques associated with the codes. Density evolution is an analytical technique which has been used to understand limits of performance of LDPC decoders. It also provides a tool which can be used in the design of families of LDPC codes, since their performance can be predicted using density evolution much more rapidly than the performance can be simulated. Density evolution introduces the idea of a channel threshold, above which the code performs well and below which the probability of error is non-negligible. This provides a single parameter characterizing code performance which may be used to gauge the performance compared to the ultimate limit of the channel capacity.

In density evolution, we make a key assumption that the block length $N \to \infty$, under which it may be assumed there are no cycles in the Tanner graph. Since the code is linear, and we have assumed a symmetric channel, it suffices for this analysis to assume that the all-zero codeword $\mathbf{c} = \mathbf{0}$ is sent. We also assume that the LDPC code is regular, with $|\mathcal{M}_n| = w_c$ and $|\mathcal{N}_m| = w_r$ for each n and m.

Local Convention: We assume furthermore that a bit of zero is mapped to a signal amplitude of $+\sqrt{E_c}$ (i.e., $0 \to 1$ and $1 \to -1$; note that this mapping is consistent with the assumption made in Appendix A, but different from the convention used throughout most of the book).

Based on this convention, the received signal is $r_t = \sqrt{E_c} + n_t$, where $n_t \sim \mathcal{N}(0, \sigma^2)$.

Hence in the likelihood decoding algorithm, the initial likelihood ratio is

$$\lambda_t^{[0]} = \frac{2\sqrt{E_c}}{\sigma^2}(\sqrt{E_c} + n_t),$$

which is Gaussian. The mean and variance of $\lambda_t^{[0]}$ are

$$m^{[l]} = E[\lambda_t^{[0]}] = \frac{2E_c}{\sigma^2} \qquad \mathrm{var}(\lambda_t^{[0]}) = \frac{4E_c}{\sigma^2} = 2m^{[l]}.$$

That is, the variance is equal to twice the mean. Thus

$$\lambda_t^{[0]} \sim \mathcal{N}(m^{[l]}, 2m^{[l]}).$$

A Gaussian random variable having the property that its variance is equal to twice its mean is said to be **consistent**. Consistent random variables are convenient because they are described by a single parameter, the mean.

Clearly, the initial $\lambda_t^{[0]}$ are Gaussian, but at other iterations they are not Gaussian. However, as $\lambda_t^{[l]}$ is computed as the sum of other random variables, a central limit argument can be made that it should tend toward Gaussian. Furthermore, numerical experiments confirm that they are fairly Gaussian. The messages $\eta_{m,n}^{[l]}$ sent by check nodes are nongaussian, but again numerical experiments confirm that they can be approximately represented by Gaussians. In the interest of analytical tractability, *we assume that all messages are not only Gaussian, but consistent.* The density evolution analysis tracks the parameters of these Gaussian random variables through the decoding process.

Let $\mu^{[l]} = E[\eta_{m,n}^{[l]}]$ denote the mean of a randomly chosen $\eta_{m,n}^{[l]}$. (Under the assumption that the nodes are randomly chosen and that the code is regular, we also assume that the mean does not depend on m or n.) Let $m^{[l]}$ denote the mean of $\lambda_n^{[l]}$. We assume that both the $\eta_{m,n}^{[l]}$ and $\lambda_n^{[l]}$ are consistent random variables, so

$$\eta_{m,n}^{[l]} \sim \mathcal{N}(\mu^{[l]}, 2\mu^{[l]}) \qquad \lambda_n^{[l]} \sim \mathcal{N}(m^{[l]}, 2m^{[l]}).$$

Under the local convention (which changes the signs in the tanh rule), by the tanh rule (15.31),

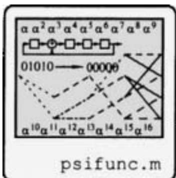
$$\tanh\left(\frac{\eta_{m,n}^{[l]}}{2}\right) = \prod_{j \in \mathcal{N}_{m,n}} \tanh\left(\frac{\lambda_j^{[l]}}{2}\right).$$

Taking the expectation of both sides we have

$$E\left[\tanh\left(\frac{\eta_{m,n}^{[l]}}{2}\right)\right] = E\left[\prod_{j \in \mathcal{N}_{m,n}} \tanh\left(\frac{\lambda_j^{[l]}}{2}\right)\right]. \tag{15.35}$$

Now define the function

$$\begin{aligned}\Psi(x) &= E[\tanh(y/2)] \text{ where } y \sim \mathcal{N}(x, 2x) \\ &= \frac{1}{\sqrt{4\pi x}} \int_{-\infty}^{\infty} \tanh(y/2) e^{-(y-x)^2/(4x)}\, dy,\end{aligned}$$



which is plotted in Figure 15.10 compared with the function $\tanh(x/2)$. The $\Psi(x)$ function is monotonic and looks roughly like $\tanh(x/2)$ stretched out somewhat.[2]

[2] It has been found [51] that $\Psi(x)$ can be closely approximated by $\Psi(x) \approx 1 - e^{-.4527x^{0.86}+0.0218}$.

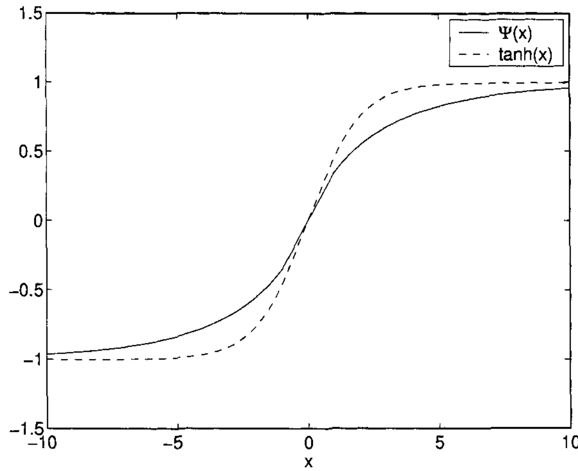


Figure 15.10: The function $\Psi(x)$ compared with $\tanh(x/2)$.

Using the Ψ function we can write (15.35) as

$$\Psi(\mu^{[l]}) = \left(\Psi(m^{[l]})\right)^{w_r-1}. \tag{15.36}$$

The bit update equation (15.32) can be expressed (with some shuffling of the order of computation) as

$$\lambda_n^{[l]} = \lambda_n^{[0]} + \sum_{m \in \mathcal{M}_n} \eta_{m,n}^{[l-1]}.$$

Taking expectations of both sides we obtain

$$m^{[l]} = \frac{2E_c}{\sigma^2} + (w_c - 1)\mu^{[l-1]}.$$

Substituting into (15.36), we obtain

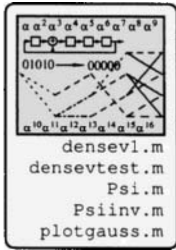
$$\Psi(\mu^{[l]}) = \left(\Psi\left(\frac{2E_c}{\sigma^2} + (w_c - 1)\mu^{[l-1]}\right)\right)^{w_r-1},$$

or

$$\mu^{[l]} = \Psi^{-1}\left(\left(\Psi\left(\frac{2E_c}{\sigma^2} + (w_c - 1)\mu^{[l-1]}\right)\right)^{w_r-1}\right). \tag{15.37}$$

The recursion is initialized with $\mu^{[0]} = 0$. The dynamics are completely determined by the row weight w_r, the column weight w_c, and the SNR E_c/σ^2.

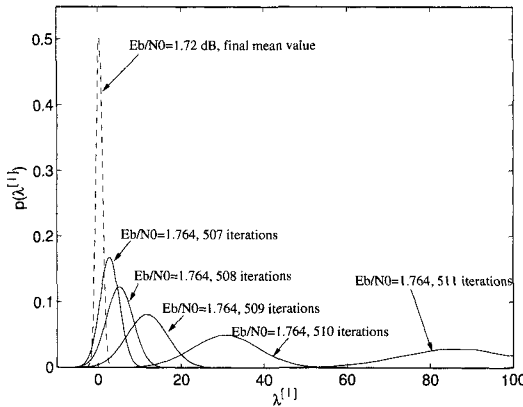
For some values of SNR, the mean $\mu^{[l]}$ converges to a small fixed point. The Gaussian pdf it represents would thus have both positive and negative outcomes, meaning that the λ represented by this distribution could have negative values, or that (recalling that the all-zero codeword is assumed) there is a nonnegligible probability that there are decoding errors. On the other hand, for some values of SNR, the mean $\mu^{[l]}$ tends to infinity. The pdf has all of its probability on positive values. Thus, for a sufficiently large number of iterations the decoder would decoder correctly.



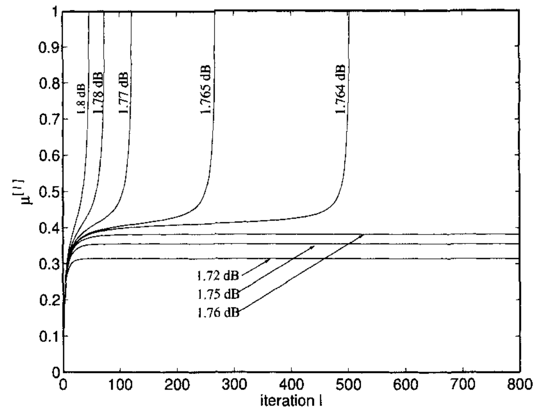
Example 15.8 Let $w_c = 4$ and $w_r = 6$, resulting in a $R = 1 - 4/6 = 1/3$ code. Recall that $E_c = RE_b$ and $\sigma^2 = N_0/2$, so that $E_c/\sigma^2 = 4RE_b/N_0$. Let $E_b/N_0 = 1.72$ dB. Then the iteration (15.37) achieves a fixed point at $\mu^* = \lim_{l\to\infty} \mu^{[l]} = 0.3155$. The corresponding density $\mathcal{N}(0.3155, 0.631)$ is shown using dashed lines in Figure 15.11(a). The mean is small enough that there is a high probability that $\lambda^{[l]} < 0$ for any iteration; hence, decoding errors are probable.

When $E_b/N_0 = 1.764$ dB, the mean (15.37) tends to ∞: $\mu^{[l]} \to \infty$ as $l \to \infty$. Figure 15.11(a) shows the distributions for iterations 507 through 511. Clearly, for high iteration numbers, the decoder is almost certain to decode correctly.

Figure 15.11(b) shows the mean values $\mu^{[l]}$ as a function of the iteration number l for various SNRs. For sufficiently small SNR, the mean converges to a finite limit, implying a nonzero probability of error. As the SNR increases, the mean "breaks away" to infinity after some number of iterations, where the number of iterations required decreases with increasing SNR.



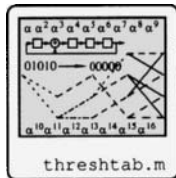
(a) The pdf of $\lambda^{[l]}$ for $E_b/N_0 = 1.72$ (final), and $E_b/N_0 = 1.764$ (various iterations).



(b) The mean of the pdf of $\lambda^{[l]}$ as a function of iteration l for different values of E_b/N_0.

Figure 15.11: Behavior of density evolution for a $R = 1/3$ code.

□



As this example shows, there is a value of E_b/N_0 above which reliable decoding can be expected ($\mu^{[l]} \to \infty$) and below which it cannot. This is called the *threshold* of the decoder. Table 15.1 [51] shows a table of thresholds for regular LDPC codes of various rates, as well as channel capacity at that rate. (*Note*: The recursion (15.37) is sensitive to numerical variation.) The thresholds are shown both in terms of E_b/N_0 and in terms of a channel standard deviation σ_τ, where

$$\frac{2RE_b}{N_0} = \frac{1}{\sigma_\tau^2}.$$

As the table shows, there is a tendency toward decrease (improvement) in the E_b/N_0 threshold as the rate of the code decreases. However, even within a given rate, there is variation depending on the values of w_c and w_r. It appears that values of $w_c > 3$ generally raise the threshold. Note that, since the analysis does not take cycles in the graph into account, this has nothing to do with problems in the decoding algorithm associated with cycles; it is an intrinsic part of the structure of the code.

Table 15.1: Threshold Values for Various LDPC Codes for the Binary AWGN Channel

w_c	w_r	Rate	Threshold σ_τ	Threshold E_b/N_0 (dB)	Capacity (dB)	Gap (dB)
3	12	0.75	0.6297	2.2564	1.6264	0.63
3	9	2/3	0.7051	1.7856	1.0595	0.7261
4	10	0.6	0.7440	1.7767	0.6787	1.098
3	6	0.5	0.8747	1.1628	0.1871	0.9757
4	8	0.5	0.8323	1.5944	0.1871	1.4073
5	10	0.5	0.7910	2.0365	0.1871	1.8494
3	5	0.4	1.0003	0.9665	-0.2383	1.2048
4	6	1/3	1.0035	1.7306	-0.4954	2.226
3	4	0.25	1.2517	1.0603	-0.7941	1.8544

15.9 EXIT Charts for LDPC Codes

Recall from Section 14.5 that an EXIT chart is a method for representing how the mutual information between the decoder output and the transmitted bits changes over turbo decoding iterations. EXIT charts can also be established for LDPC codes, as we now describe.

Consider the fragments of a Tanner graph in Figure 15.12. In these fragments, there are bit-to-check messages and check-to-bit messages, denoted as $\mu_{B\to C}$ and $\mu_{C\to B}$, respectively, where the messages are the log likelihood ratios. Let $r_{mn}^{[i]}(x)$ and $q_{mn}^{[i]}(x)$ denote the probabilities computed in the horizontal and vertical steps of Algorithm 15.1, respectively, at the ith iteration of the algorithm. Using the original probability-based decoding algorithm of Algorithm 15.1, the messages from bit nodes (n) to check nodes (m) or back are

$$C \to B : \mu_{C\to B} = \log\frac{r_{mn}(1)}{r_{mn}(0)}$$

$$B \to C : \mu_{B\to C} = \log\frac{q_{mn}(1)}{q_{mn}(0)}.$$

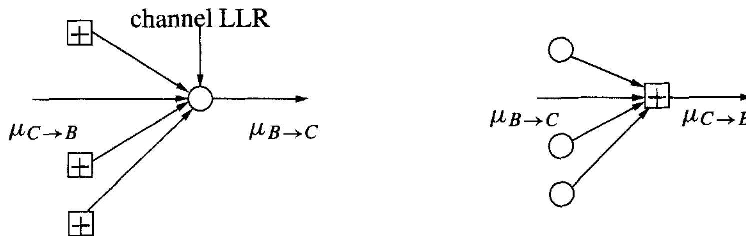


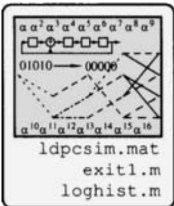
Figure 15.12: A portion of a Tanner graph, showing messages from bits to checks and from checks to bits.

Let X denote the original transmitted bits. The mutual information (see Section 1.12) between a check-to-bit message $\mu_{C\to B}$ and the transmitted data symbol for that bit X is denoted as $I(X, \mu_{C\to B}) = I_{C\to B}$. The iteration number i may also be indicated, as in $I_{C\to B}^{[i]}$. The actual mutual information is computed experimentally as follows. Histograms of the message data $\mu_{C\to B}^{[i]}$ are used to estimate the probability distribution. These histograms are

obtained from the outputs $\log r_{mn}(1)/r_{mn}(0)$ of *all* the check nodes in the Tanner graph. (Alternatively a single node could be used, by sending the codeword multiple times through the channel with independent noise.) These histograms are normalized to form estimated probability density functions, here denoted $\hat{p}(\mu)$, of the random variable $\mu_{C\to B}$. Then these estimated density functions are used in the mutual information integral (1.40) wherever $p(y|-a)$ appears. Because of symmetry, the likelihood $p(y|a)$ is computed using $\hat{p}(-\mu)$. The numerical evaluation of the integral then gives the desired mutual information.

In a similar way, the mutual information between a bit-to-check message $\mu_{B\to C}$ and the transmitted data symbol for that bit X, $I(X, \mu_{B\to C}) = I_{B\to C}$, is computed from the histograms of the outputs $\log q_{mn}(1)/q_{mn}(0)$ to estimate the densities in (1.40).

The first trace of the EXIT chart is now formed by plotting $I_{C\to B}^{[i]}, I_{B\to C}^{[i+1]}$ for values of i as the decoding algorithm proceeds. The horizontal axis is thus the check-to-bit axis. The second trace of the EXIT chart uses $I_{B\to C}^{[i+1]}$ as the independent variable, but plotted on the vertical axis, with $I_{C\to B}^{[i+1]}$ — that is, the mutual information at the *next* iteration — on the horizontal axis. The "transfer" of information in the EXIT chart results because the check-to-bit message at the output of the $i+1$th stage becomes the check-to-bit message at the input of the next stage.



Example 15.9 Figure 15.13 shows the density estimated from the histogram of the log likelihood ratios $L = \log r_{mn}(1)/r_{mn}(0)$ for a (15000, 10000) LDPC code at an SNR of 1.6 dB for various iterations of the algorithm. At iteration 0, the log likelihoods from the received signal data are plotted. At the other iterations, the log likelihoods of the bit-to-check information are plotted. Observe that the histogram has a rather Gaussian appearance (justifying the density evolution analysis of Section 15.8) and that as the iterations proceed the mean becomes increasingly negative. The decoder thus becomes increasingly certain that the transmitted bits are 0.

Figure 15.14 shows the mutual information as a function of decoder iteration number for bit-to-check and check-to-bit information for various SNRs. Perhaps the most interesting is for an SNR of 0.4 dB: after an initial increase in information, the decoder stalls and no additional increases occur. The EXIT chart is essentially obtained by eliminating the iteration number parameter from these two plots and plotting them against each other.

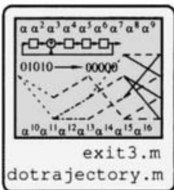
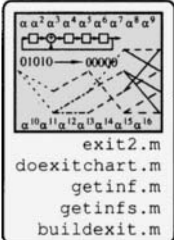


Figure 15.15 shows the EXIT chart for this code at various SNRs. The solid bold line plots the points $(I_{C\to B}^{[i]}, I_{B\to C}^{[i+1]})$ and the dashed bold line plots the points $(I_{B\to C}^{[i+1]}, I_{C\to B}^{[i+1]})$, with the horizontal axis representing $I_{C\to B}$ and the vertical axis representing $I_{B\to C}$. The narrow solid line plots the progress of the decoding algorithm: the decoder essentially follows the stair-step pattern between the two traces of the plot. At an SNR of 0.8 dB, the code is fairly close to the decoding threshold, so it takes many iterations for the decoder to pass through the channel. At an SNR of 0.4 dB, the decoder is below the decoding threshold: the information is not able to make it through the channel. At an SNR of 1.2 dB, the channel is open somewhat wider, so fewer decoding iterations are required, and at 1.8 dB the channel is open wider still.



□

15.10 Irregular LDPC Codes

An irregular (or nonuniform) LDPC code has a very sparse parity check matrix in which the column weight (resp. row weight) may vary from column to column (resp. row to row). Considering that the results in Table 15.1 suggest that for the same rate, different column/row weights perform differently, the ability to allocate weights flexibly provides potentially useful design capability. In fact, the best known LDPC codes are irregular; gains of up to 0.5 dB compared to regular codes are attainable [212]. In this section, we

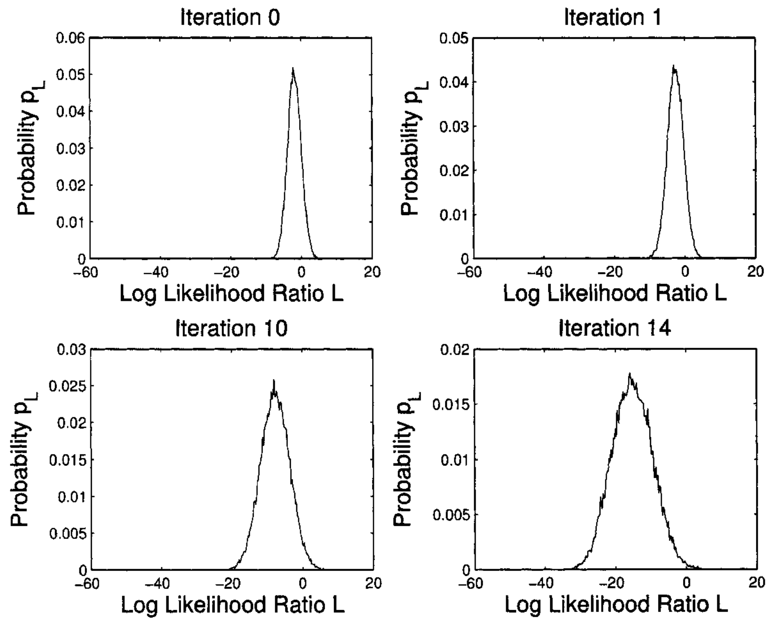


Figure 15.13: Histograms of the bit-to-check information for various decoder iterations at 1.6 dB.

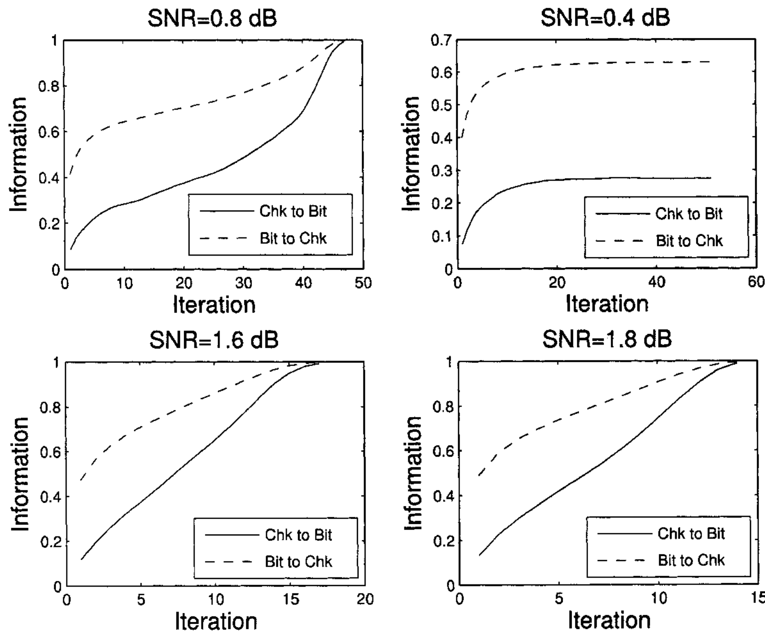


Figure 15.14: Decoder information at various signal-to-noise ratios.

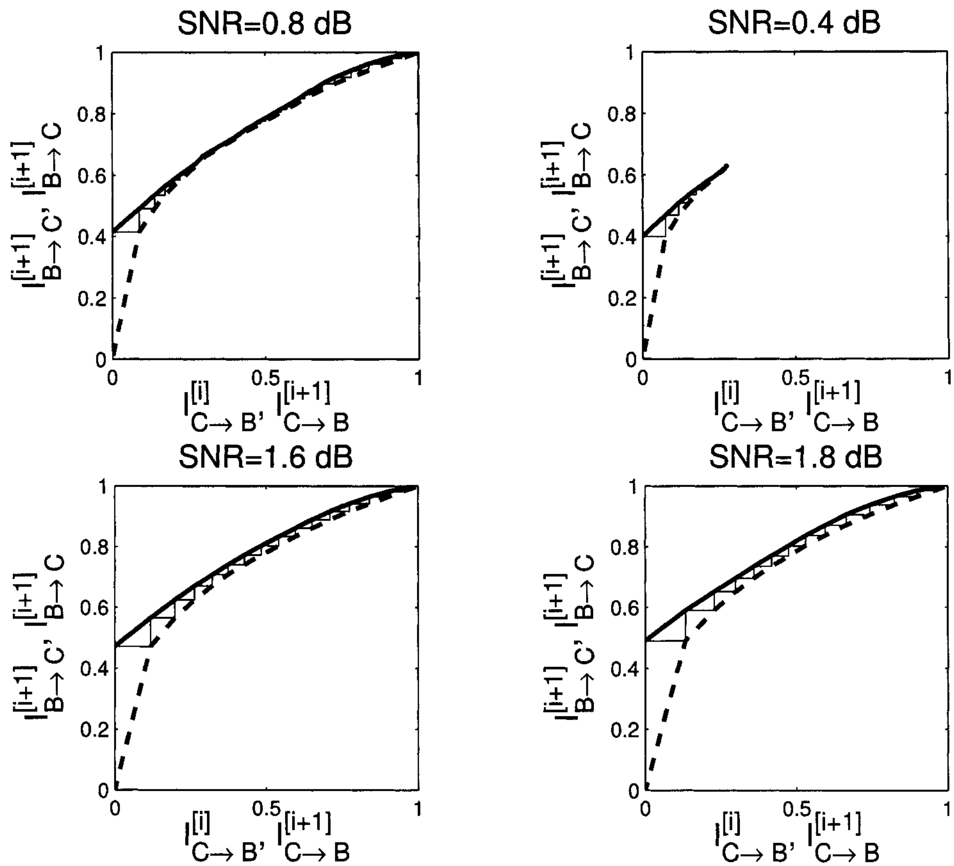


Figure 15.15: EXIT charts at various signal-to-noise ratios.

present some results of the design of irregular codes. This is followed by a sketch of the density evolution analysis which leads to these results.

15.10.1 Degree Distribution Pairs

The distribution of the weights of the columns and rows of the parity check matrix is described as follows. We let ν_i represent the fraction of *edges* emanating from a bit (variable) node in the Tanner graph for the code and let χ_i represent the fraction of *edges* emanating from a check node. Let d_v denote the maximum number of edges connected to a variable node and let d_c be the maximum number of edges connected to a check node. The polynomial

$$\nu(x) = \sum_{i=2}^{d_v} \nu_i x^{i-1}$$

represents the distribution of variable node weights and

$$\chi(x) = \sum_{i=2}^{d_c} \chi_i x^{i-1}$$

represents the distribution of check node weights. These are called the variable node and check node distributions, respectively. The degree distributions satisfy $\nu(1) = 1$ and $\chi(1) = 1$. The pair $(\nu(x), \chi(x))$ is called a *degree distribution pair*. For example, for the (3, 6) regular code, $\nu(x) = x^2$ and $\chi(x) = x^5$. The number of variable nodes is N and the number of check nodes is M.

The number of variable nodes of degree i is (see Exercise 15.12)

$$N\frac{\nu_i}{\sum_{j\geq 2}\nu_j/j} = N\frac{\nu_i}{\int_0^1 \nu(x)\,dx}. \tag{15.38}$$

The total number of edges emanating from all nodes is

$$E = N\sum_{i\geq 2} i\frac{\nu_i/i}{\int_0^1 \nu(x)\,dx} = N\frac{1}{\int_0^1 \nu(x)\,dx}. \tag{15.39}$$

Similarly, the number of check nodes of degree i is

$$N\frac{\chi_i/i}{\sum_{j\geq 2}\chi(j)/j} = N\frac{\chi_i/i}{\int_0^1 \chi(x)\,dx}$$

and the total number of edges is

$$E = M\frac{1}{\int_0^1 \chi(x)\,dx}. \tag{15.40}$$

Equating (15.39) and (15.40) we find

$$\frac{M}{N} = \frac{\int_0^1 \chi(x)\,dx}{\int_0^1 \nu(x)\,dx}.$$

Under the assumption that the corresponding check equations are all linearly independent, the rate of the code is

$$R(\nu, \chi) = \frac{N-M}{N} = 1 - \frac{\int_0^1 \chi(x)\,dx}{\int_0^1 \nu(x)\,dx}.$$

Example 15.10 Suppose $\eta_3 = 0.5$ and $\eta_4 = 0.5$ and $N = 1000$. Then

$$\eta(x) = 0.5x^2 + 0.5x^3$$

and there are

$$N\frac{0.5/3}{0.5/3 + 0.5/4} = 571$$

(rounding) variable nodes of degree 3 and

$$N\frac{0.5/4}{0.5/3 + 0.5/4} = 429$$

variable nodes of degree 4, for a total of

$$E = 571 \cdot 3 + 429 \cdot 4 = 3426$$

edges in the graph. □

15.10.2 Some Good Codes

Assuming that the message passing decoder can be analyzed using a density evolution similar to that of the regular code, a threshold σ_τ can be established such that the mean message $\mu^{[l]} \to \infty$ if the channel deviation $\sigma > \sigma_\tau$. This leads to a design optimization problem: choose variable and check node distributions $\nu(x)$ and $\chi(x)$ in such a way as to *maximize* the corresponding σ_τ (i.e., minimize the SNR at which the code correctly decodes). Some results of such an optimization for some values of d_v are shown in Table 15.2, along with the corresponding threshold σ_τ, the corresponding E_b/N_0, and the gap to channel capacity. Note that in some cases the codes are within significantly less then half a dB away from capacity (up to the idealizations of the analysis: arbitrarily long block lengths, and cycles in the graph do not affect the decoding).

Table 15.2: Example Degree Distribution Pairs for $R = 1/2$ codes for Binary Transmission over an AWGN [292, p. 623]

d_v:	4	5	6	7	8	9	10	11	12
η_2	0.38354	0.32660	0.33241	0.31570	0.30013	0.27684	0.25105	0.23882	0.24426
η_3	0.04327	0.11960	0.24632	0.41672	0.28395	0.28342	0.30938	0.29515	0.25907
η_4	0.57409	0.18393	0.11014				0.00104	0.03261	0.01054
η_5		0.36988							0.05510
η_6			0.3112						
η_7				0.43810					
η_8					0.41592				0.01455
η_9						0.43974			
η_{10}							0.43853		0.01275
η_{11}								0.43342	
η_{12}									0.40373
χ_5	0.24123								
χ_6	0.75877	0.78555	0.76611	0.43810	0.22919	0.01568			
χ_7		0.21445	0.23389	0.56190	0.77081	0.85244	0.63676	0.43011	0.25475
χ_8						0.13188	0.36324	0.56989	0.73438
χ_9									0.01087
σ_τ	0.9114	0.9194	0.9304	0.9424	0.9497	0.9540	0.9558	0.9572	0.9580
E_b/N_0 (dB)	0.8058	0.7299	0.6266	0.5153	0.4483	0.4090	0.3927	0.3799	0.3727
gap(dB)	0.6187	0.5428	0.4395	0.3282	0.2612	0.2219	0.2056	0.1928	0.1856

15.10.3 Density Evolution for Irregular Codes

We now summarize how the density evolution is described for these irregular codes. We present the highlights of the technique, leaving aside some technicalities which are covered in [292]. In the current analysis, a more explicit representation of the distribution of the messages is needed than the consistent Gaussian approximation made in Section 15.8.

The decoder algorithms can be summarized as

$$\eta_{m,n}^{[l]} = -2\tanh^{-1}\left(\prod_{j \in \mathcal{N}_{m,n}} \tanh\left(-\frac{\lambda_{j,m}^{[l-1]}}{2}\right)\right), \tag{15.41}$$

$$\lambda_{n,m}^{[l]} = \lambda_{n,m}^{[0]} + \sum_{j \in \mathcal{M}_{n,m}} \eta_{j,n}^{[l]}. \tag{15.42}$$

Write (15.41) as

$$\tanh \frac{-\eta_{m,n}^{[l]}}{2} = \left(\prod_{j \in \mathcal{N}_{m,n}} \tanh(-\frac{\lambda_{j,m}^{[l-1]}}{2}) \right)$$

and take the log of both sides. In doing this, we have to be careful about the signs. Therefore we will separate out the sign,

$$(\operatorname{sgn}(\eta_{m,n}^{[l]}), -\log \left| \tanh \frac{\eta_{m,n}^{[l]}}{2} \right|) = \sum_{j \in \mathcal{N}_{m,n}} (-\operatorname{sgn}(\lambda_{j,m}^{[l-1]}), \log \left| \tanh \frac{\lambda_{j,m}^{[l-1]}}{2} \right|). \tag{15.43}$$

We employ here a somewhat different definition of the sgn function:

$$\operatorname{sgn}(x) = \begin{cases} 0 & x > 0 \\ 0 & \text{with probability } \frac{1}{2} \text{ if } x = 0 \\ 1 & \text{with probability } \frac{1}{2} \text{ if } x = 0 \\ 1 & x < 0 \end{cases}$$

so that $\operatorname{sgn}(x) = 1$ means that $x < 0$. Then the sum for the signs is performed in $\mathbb{Z}_2$ and the sum for the magnitude is the ordinary sum in $\mathbb{R}$.

Now let γ be the function

$$\gamma(x) : [-\infty, +\infty] \to \{0, 1\} \times [0, \infty]$$

defined by

$$\gamma(x) = (\gamma_1(x), \gamma_2(x)) = (\operatorname{sgn}(x), -\log \tanh(|x/2|)). \tag{15.44}$$

Then (15.43) is

$$\gamma(\eta_{m,n}^{[l]}) = \sum_{j \in \mathcal{N}_{m,n}} \gamma(\lambda_{j,m}^{[l-1]})$$

so we can express (15.41) as

$$\eta_{m,n}^{[l]} = \gamma^{-1} \left(\sum_{j \in \mathcal{N}_{m,n}} \gamma(\lambda_{j,m}^{[l-1]}) \right). \tag{15.45}$$

The equation (15.45) has the feature that the product is converted to a sum; in the analysis below this is useful because sums of independent random variables have convolved distributions.

We describe the evolution in terms of distribution functions: Let $\mathcal{F}$ denote the space of right-continuous, nondecreasing functions F_z defined on $\mathbb{R}$, such that for $F_z \in \mathcal{F}$, $\lim_{x \to -\infty} F_z(x) = 0$ and $\lim_{x \to \infty} F_z(x) \le 1$, allowing for the possibility of a point probability mass at ∞:

$$P(z = \infty) = 1 - \lim_{x \to \infty} F_z(x).$$

A function $F_z \in \mathcal{F}$ represents the usual cumulative distribution function of the random variable z:

$$F_z(x) = P(z \le x).$$

We define the left limit of F_z as

$$F_z^{-} x = \lim_{y \uparrow x} F_z(y).$$

so that $1 - F_z^-(x) = P(z \geq x)$. Derivatives (more precisely, Radon-Nikodyn derivatives [30]) of the distribution functions are probability densities.

Suppose we have a random variable z with distribution F_z. We wish to describe the distribution of the random variable $\gamma(z) = (\gamma_1(z), \gamma_2(z))$, with γ defined in (15.44). Note that any function $G(s, x)$ defined over $\{0, 1\} \times [0, \infty)$ can be written as

$$G(s,x) = I_{s=0}G^0(x) + I_{s=1}G^1(x),$$

where $I_{s=a}$ is the indicator (or characteristic) function:

$$I_{s=a} = \begin{cases} 1 & \text{if } s = a \\ 0 & \text{if } s \neq a. \end{cases}$$

Using this notation, we define the distribution of $\gamma(z)$ as

$$\Gamma(F_z)(s,x) = I_{s=0}\Gamma_0(F_z)(x) + I_{s=1}, \Gamma_1(F_z)(x) \tag{15.46}$$

where

$$\Gamma_0(F_z)(x) = 1 - F_z^-(-\log\tanh(x/2)) = P(z \geq -\log\tanh(x/2))$$

and

$$\Gamma_1(F_z)(x) = F_z(\log\tanh(x/2)) = P(z \leq \log\tanh(x/2)).$$

It can be shown that

$$\lim_{x\to\infty} \Gamma_0(F_z)(x) - \lim_{x\to\infty} \Gamma_1(F_z)(x) = P(z = 0).$$

The function Γ has an inverse: for a function

$$G(s,x) = I_{s=0}G^0(x) + I_{s=1}G^1(x),$$

define Γ^{-1} by

$$\Gamma^{-1}(G)(x) = I_{x>0}G^0(-\log\tanh(x/2)) + I_{x<0}G^1(-\log\tanh(-x/2)) \tag{15.47}$$

and

$$\Gamma^{-1}(G)(0) = \lim_{x\to\infty} G^0(x).$$

It can be verified that $\Gamma^{-1}(\Gamma(F)) = F$ for all $F \in \mathcal{F}$.

For notational convenience, Γ and Γ^{-1} are also applied to densities, where it is to be understood that the notation is a representation of the operation applied to the associated distributions.

Let G and H be two distributions,

$$G = I_{s=0}G^0 + I_{s=1}G^1 \qquad H = I_{s=0}H^0 + I_{s=1}H^1.$$

Let $\otimes$ denote the operation of convolution on distribution functions. Then we define the convolution $\otimes$ on G and H by

$$G \otimes H = I_{s=0}\left((G^0 \otimes H^0) + (G^1 \otimes H^1)\right) + I_{s=1}\left((G^0 \otimes H^1) + (G^1 \otimes H^0)\right).$$

Again for notational convenience we allow the convolution operator to act on densities, where it is to be understood that it applies to the associated distributions. We denote repeated convolution as $^{\otimes}$:

$$\underbrace{G \otimes G \otimes \cdots \otimes G}_{p \text{ factors}} = G^{\otimes p}.$$

Let $P^{[l]}$ and $Q^{[l]}$ be the densities of the random variables $\lambda_{n,m}^{[l]}$ and $\eta_{m,n}^{[l]}$, respectively. The corresponding distribution functions are denoted $\int P^{[l]}$ and $\int Q^{[l]}$.

Let the graph associated with the code have the distribution pair (ν, χ),

$$\nu(x) = \sum_{i \geq 2} \nu_i x^{i-1} \qquad \chi(x) = \sum_{i \geq 2} \chi_i x^{i-1}.$$

Recall that the fraction of edges connected to a variable node of degree i is ν_i and the fraction of edges connected to a check node is χ_i. Thus a randomly chosen edge in the graph is connected to a check node of degree i with probability χ_i. Therefore, with probability χ_i, the sum in (15.45) has $(i-1)$ terms, corresponding to the edges connecting check m with its neighbors except bit n. We now invoke the independence assumption, that these neighboring nodes are independent. Combining (15.45) with the definition of the Γ function, we obtain

$$Q^{[l]} = \Gamma^{-1}\left(\sum_{i \geq 2} \chi_i \left[\Gamma(P^{[l-1]})\right]^{\otimes(i-1)}\right).$$

We use the shorthand notation for this

$$Q^{[l]} = \Gamma^{-1}\chi(\Gamma(P^{[l-1]})). \tag{15.48}$$

(This explains the unusual definition $\sum_{i \geq 2} \chi_i x^{i-1}$, with the exponent $i-1$.) The recursion for $P^{[l]}$ is straightforward, since only sums are involved:

$$P^{[l]} = P_0 \otimes \sum_{i \geq 2} \nu_i \left[Q^{[l-1]}\right]^{\otimes(i-1)}.$$

Again we use the shorthand notation

$$P^{[l]} = P_0 \otimes \nu(Q^{[l]}). \tag{15.49}$$

Combining (15.48) and (15.49) we obtain the overall recursion,

$$P^{[l]} = P_0 \otimes \nu(\Gamma^{-1}(\nu(\Gamma(P^{[-1]})))). \tag{15.50}$$

The original density P_0 is Gaussian, just as it was in Section 15.8.

15.10.4 Computation and Optimization of Density Evolution

It can be shown that the recursion (15.50) always converges to some fixed distribution, although it may be the distribution with its probability mass at ∞. It can further be shown that the probability of error is a nonincreasing function of the iteration number l.

The convolutions implied in (15.50) can be efficiently computed by quantizing the distributions and employing an FFT for fast convolution. This corresponds to a quantized message passing algorithm, which is suboptimal compared to exact message passing. Any decoding threshold σ_τ thus obtained is therefore a lower bound on the actual threshold.

The basic problem is to choose coefficients $\{\eta_i\}$ and $\{\chi_i\}$ so that the decoding threshold σ_τ is as large as possible. The basic outline for the computation is as follows. Starting with a given degree distribution pair $(\eta(x), \chi(x))$, an error probability ϵ and a maximum number of iterations L is selected. From this, a maximum admissible channel parameter σ is selected, which is the largest channel parameter such that the error probability after L iterations is is less than ϵ. Then a *hill climbing* approach is used. A small change to the

degree distribution pair is introduced. If the change leads to a target error probability after L iterations, or if the maximum admissible channel parameter is larger, then the new degree distribution pair is accepted, otherwise the old degree distribution pair is retained. The hill climbing process repeats until some termination criterion is satisfied.

Clearly, there is a very large search space. Some acceleration of the search process can be obtained by limiting the scope of the search. It has been found that very good degree distribution pairs exist with only a few nonzero terms. In particular, it suffices to allow only two or three nonzero check node degrees (which may be chosen consecutively) and to limit the nonzero variable node degrees to 2, 3, or d_v.

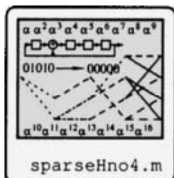
A variation on this density evolution concept has been used to design rate $R = 1/2$ codes which (theoretically) perform to within 0.0045 dB of the capacity limit [50]. Simulations of actual codes with block lengths $N = 10^7$ indicate that the actual performance is within 0.04 dB of capacity. So, while the analysis and design are somewhat idealized, the theory matches the practice rather well.

15.10.5 Using Irregular Codes

The procedure outlined above determines a degree distribution pair $(\eta(x), \chi(x))$. This can be used to construct an actual code as follows. Choose a code length N (usually quite large). Determine the number of variable nodes N_i having i edges and the number of check nodes M_i having i edges. Randomly generate a matrix A having the given column and row weights. Some iteration of this is probably necessary to avoid cycles of length 4 in the graph (and possibly other short cycles). Then the decoding algorithms described above apply to this parity check matrix without any change.

15.11 More on LDPC Code Construction

It is straightforward to generate random LDPC codes: simply generate columns of A at random having the appropriate weight. However, there are some practicalities to be dealt with. First, if the columns of A are not linearly independent, some columns can be eliminated, which serves to *increase* the rate of the code by decreasing N. Second, it is important to reduce the number of cycles in the graph associated with the code. Therefore, eliminating or regenerating columns which would contribute to short cycles is advised. It can be seen that when two columns of A have an overlap of more than 1 bit (as in the italicized elements of (15.1)) there is a cycle of length 4 in the iterated graph. For large N, this is a very easy condition to check for and eliminate in the random generation process. With somewhat more work, longer cycles can be detected and removed.


sparseHno4.m

Besides such random constructions, there have been more recent constructions which attempt to introduce additional structure into the parity check matrix and/or the generator matrix. While space limitations preclude more than a mention of these results, it is important to be aware that such construction techniques exist. The interested reader is encouraged to check the references cited below.

15.11.1 A Construction Based on Finite Geometries

A finite geometry is a collection of "points," which are m-tuples, $\mathbf{a} \in GF(q)^m$, and "lines," upon which sets of points lie. For constructing LDPC codes, the finite geometry $\mathbf{G}$ is employed, where $\mathbf{G}$ is a finite geometry with N points and M lines with the following properties: (1) Every line consists of w_r points; (2) any two points are connected by exactly

one line; (3) every point lies on w_c lines; (4) either two lines are parallel (having no point in common), or they intersect at exactly one point.

For such a geometry $\mathbf{G}$, form a binary incidence matrix $H_{\mathbf{G}}$ whose rows and columns correspond to the lines and points of $\mathbf{G}$, respectively, with $h_{i,j} = 1$ if and only if the ith line of $\mathbf{G}$ contains the jth point of $\mathbf{G}$. Then each row of $H_{\mathbf{G}}$ portrays the points of $\mathbf{G}$ (and has weight w_r) and each column portrays the lines of $\mathbf{G}$ (and has weight w_c).

Based on this idea, parity check matrices for codes are constructed [194] that offer the following potential advantages:

1. Several different decoding algorithms exist, from one step majority logic decoders with low complexity, through the usual sum-product algorithm with higher complexity.
2. The code can be extended by splitting each column of H. If done properly, performance within a few dB of capacity can be achieved.
3. Codes derived from finite geometries may be cyclic or nearly cyclic, structure that enables them to be efficiently *encoded.*

15.11.2 Constructions Based on Other Combinatoric Objects

Several LDPC constructions have been reported based on combinatoric objects.

Constructions based on Kirkman triple systems are reported in [177, 176], which produce $(3, k)$-regular LDPC codes whose Tanner graph is free of 4-cycles for any k.

Constructions based on Latin rectangles are reported in [352], with reportedly low encode and decode complexity.

Designs based on Steiner 2-designs is reported in [351] and [179] and [300]. See also [353].

High rate LDPC codes based on constructions from unital designs are reported in [178].

Constructions based on disjoint difference sets permutation matrices for use in conjunction with the magnetic recording channel are reported in [319].

15.12 Encoding LDPC Codes

While LDPC codes have an efficient decoding algorithm, with complexity linear in the code length, the *encoding* efficiency is quadratic in the block length, since it requires multiplication by the generator matrix which is not sparse. This complexity is in contrast to the turbo code case, which has linear encode complexity. However, as we present here [289], it is possible to encode with a reasonable complexity, provided that some preprocessing is performed prior to encoding.

Before encoding, we perform the following preprocessing steps. By row and column permutations, we bring H into the form indicated in Figure 15.16, where the upper right corner can be identified as a lower triangular matrix. Because it is obtained only by permutations, the H matrix is still sparse. We denote the permutation/decomposition as

$$H = \begin{bmatrix} A & B & T \\ C & D & E \end{bmatrix}$$

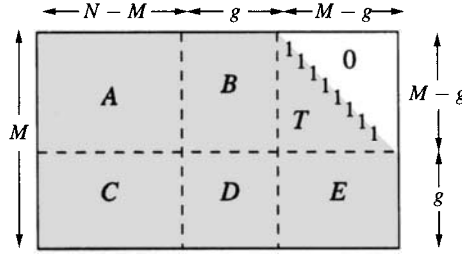


Figure 15.16: Result of permutation of rows and columns.

and say that H is in *approximate lower triangular* form. We say that g is the *gap* of this representation. T is a $(M-g) \times (M-g)$ lower triangular matrix with ones along the diagonal and hence is invertible. Now multiply H on the left by the matrix

$$\begin{bmatrix} I & \mathbf{0} \\ -ET^{-1} & I \end{bmatrix}.$$

This amounts to doing Gaussian elimination to clear the matrix E, which produces the form

$$\tilde{H} = \begin{bmatrix} I & \mathbf{0} \\ -ET^{-1} & I \end{bmatrix} H = \begin{bmatrix} A & B & T \\ -ET^{-1}A + C & -ET^{-1}B + D & \mathbf{0} \end{bmatrix}.$$

Note that $\tilde{H}$ is the parity check matrix for an equivalent code.

For a message vector $\mathbf{m}$ of length K, we write the codeword as

$$\mathbf{c} = \begin{bmatrix} \mathbf{m} \\ \mathbf{p}_1 \\ \mathbf{p}_2 \end{bmatrix},$$

where $\mathbf{p}_1$ and $\mathbf{p}_2$ represent parity information. The parity check equation $\tilde{H}\mathbf{c} = \mathbf{0}$ gives rise to two equations,

$$A\mathbf{m} + B\mathbf{p}_1 + T\mathbf{p}_2 = \mathbf{0} \tag{15.51}$$

$$(-ET^{-1}A + C)\mathbf{m} + (-ET^{-1}B + D)\mathbf{p}_1 = \mathbf{0}. \tag{15.52}$$

Letting $X = (-ET^{-1}B + D)$ and assuming for the moment that X is nonsingular, we have from (15.52)

$$\mathbf{p}_1 = -X^{-1}(-ET^{-1}A + C)\mathbf{m}.$$

The $g \times (N - M)$ matrix $-X^{-1}(-ET^{-1}A + C)$ can be precomputed and saved, so that $\mathbf{p}_1$ can be computed with a complexity of $O(g(N - M))$. The complexity can be further reduced, as is outlined below.

Once $\mathbf{p}_1$ is known, then $\mathbf{p}_2$ can be obtained from (15.51) by

$$\mathbf{p}_2 = -T^{-1}(A\mathbf{m} + B\mathbf{p}).$$

Note that since T^{-1} is lower triangular, $\mathbf{p}_2$ can be found by backsubstitution.

If it turns out that X is singular, then columns of $\tilde{H}$ can be permuted to obtain a nonsingular X.

The process of computing $\mathbf{p}_1$ and $\mathbf{p}_2$ constitutes the encoding process. The steps for the computation as well as their computational complexity are outlined here (assuming that the preprocessing steps have already been accomplished). For the sake of clarity, intermediate variables are used to show the steps which may not be necessary in a final implementation.

Steps to compute $\mathbf{p}_1 = -X^{-1}(-ET^{-1}A + C)\mathbf{m}$:

Operation	Comment	Complexity
$\mathbf{x}_1 = A\mathbf{m}$	Multiplication by a sparse matrix	$O(N)$
$\mathbf{x}_2 = T^{-1}\mathbf{x}_1$	Solve $T\mathbf{x}_2 = \mathbf{x}_1$ by backsubstitution (T is sparse)	$O(N)$
$\mathbf{x}_3 = -E\mathbf{x}_2$	Multiplication by a sparse matrix	$O(N)$
$\mathbf{x}_4 = C\mathbf{m}$	Multiplication by a sparse matrix	$O(N)$
$\mathbf{x}_5 = \mathbf{x}_3 + \mathbf{x}_4$	Addition	$O(N)$
$\mathbf{p}_1 = -X^{-1}\mathbf{x}_5$	Multiplication by dense $g \times g$ matrix	$O(g^2)$

Steps to compute $\mathbf{p}_2 = -T^{-1}(ET^{-1}A + C)\mathbf{m}$.

Operation	Comment	Complexity
$\mathbf{x}_1 = A\mathbf{m}$	Multiplication by sparse matrix (already done)	0
$\mathbf{x}_6 = P\mathbf{p}_1$	Multiplication by sparse matrix	$O(N)$
$\mathbf{x}_7 = \mathbf{x}_1 + \mathbf{x}_7$	Addition	$O(N)$
$\mathbf{p}_2 = T^{-1}(\mathbf{x}_7)$	Solve $T\mathbf{p}_2 = \mathbf{x}_7$ by backsubstitution (T is sparse)	$O(N)$

The overall algorithm is $O(N + g^2)$. Clearly, the smaller g (the "gap") can be made, the lower the complexity of the algorithm. A heuristic greedy search method for performing the initial permutations is described in [289].

15.13 A Variation: Low-Density Generator Matrix Codes

As a variation on the LDPC theme, it is interesting to consider low density generator matrix (LDGM) codes. These are codes in which the generator matrix G is very sparse. Let $G = \begin{bmatrix} I \\ P \end{bmatrix}$ be a very sparse generator in systematic form. Then the corresponding parity check matrix $H = \begin{bmatrix} -P & I \end{bmatrix}$ is also very sparse, so the code is amenable to decoding using the sum-product algorithm. The LDGM code is thus straightforward to encode and decode.

However, it is clear that since G is very sparse the code has low-weight codewords, which results in a significant error floor. For this reason, LGDM codes have not been of as much interest. It has been shown, however, that a straightforward concatenation of two LGDM codes (which is still easy to encode) has good performance when used with an iterative decoder between the concatenated stages [122].

15.14 Serial Concatenated Codes; Repeat-Accumulate Codes

The parallel concatenated codes presented in chapter 14 are not the only types of concatenated codes that can take advantage of iterative decoding algorithms. The (serially) concatenated codes introduced in chapter 10 can also be iteratively decoded. Figure 15.17 shows the encoder and decoder block diagram for an iteratively decoded serial concatenated coding scheme. The MAP decoder block operates essentially identical to that for the block turbo code of Section 14.6. Note that the outer MAP decoder does not have the received data as an input (in contrast to parallel concatenated decoders, for which both decoders use received data as an input).

This discussion about concatenated codes might seem more germane to the chapter on turbo codes. But we now present an example of a serially concatenated code whose decoder is more in line with the flavor of LDPC decoders. This is the set of codes known as repeat-

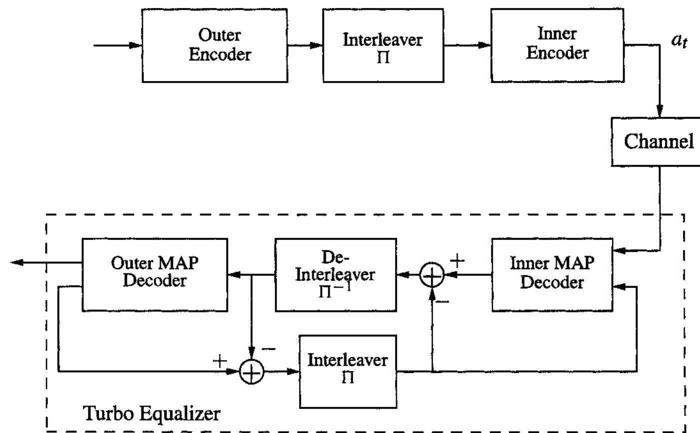


Figure 15.17: Serially concatenated codes and their iterative decoding.

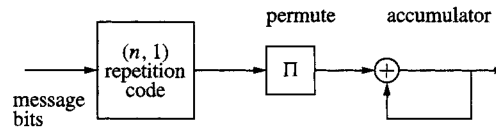


Figure 15.18: A repeat-accumulate encoder.

accumulate (RA) codes. The RA code encoder consists of three trivial encoding blocks, as shown in Figure 15.18.

1. The outer code is an $(n, 1)$ repetition code. These are the simplest known error correction codes, having good distance properties but very low rate.
2. A pseudorandom interleaver Π.
3. The inner code is a rate-1 recursive convolutional code with generator $G(x) = \frac{1}{1+x}$. This acts as an accumulator, because the output is the mod-2 sum (accumulation) of the inputs.

While this code could be decoded much like turbo codes using iterated BCJR algorithms, we present an alternative point of view here using Tanner graphs.

Suppose that a *systematic* RA code is employed, in which the original message bits are transmitted along with the accumulator output. This linear code would have a parity check matrix, which, in turn, would have a Tanner graph representation.

Example 15.11 [15, p. 623] Consider a systematic RA code on $K = 2$ message bits with a $(3, 1)$ repetition code. The interleaver is $\Pi = (1, 4, 6, 2, 3, 5)$. The operation of the code is as follows. Two message bits, m_1 and m_2 arrive at the encoder and are replicated three times:

$$m_1, m_1, m_1, m_2, m_2, m_2.$$

These bits pass through the interleaver, which produces the output sequence

$$m_1, m_2, m_2, m_1, m_1, m_2.$$

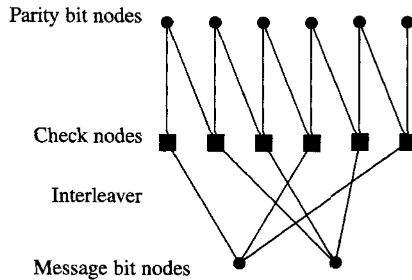


Figure 15.19: The Tanner graph for a (3, 1) RA code with two input bits.

Then the accumulator produces the running sum output:

$$p_1 = m_1, \quad p_2 = m_2 + p_1, \quad p_3 = m_2 + p_2, \quad p_4 = m_1 + p_3, \quad \cdots$$

□

The Tanner graph for such a code is shown in Figure 15.19. The variable nodes of the graph have been split into the (systematic) message bits and the parity bits to help distinguish the structure of the graph. The Tanner graph can be interpreted as follows. Reading left-to-right, the first check node constrains the parity bit to be equal to the first bit, m_1. Each succeeding check node constrains the parity bit to be the sum of the previous parity bit and the next input, where the input sequence is determined by the interleaver. This does not define a regular code, since each message bit is connected to n check nodes and the parity bits are connected to one or two check nodes.

Once we observe the structure of the RA code on the Tanner graph, we may observe that, regardless of the length N of the code, the Tanner graph retains its sparseness. Each parity bit node is connected to no more than two check nodes and each message bit node is connected to n check nodes.

To obtain the original, nonsystematic RA code, the systematic RA code can be punctured. The Tanner graph and decoding does not change, except that the channel observations $L_c r_t$ for the corresponding punctured bits would be zero.

15.14.1 Irregular RA Codes

The RA encoder structure presented above can be generalized to an *irregular* repeat-accumulate structure [167]. In this code there are K input bits. Instead of repeating all K bits an equal number of times, we choose fractions $f_2, f_3, \ldots, f_q$ such that

$$\sum_{i=2}^{q} f_i = 1.$$

Then the first block of $f_1 K$ bits is repeated two times, the next block of $f_2 K$ bits is repeated three times, and so forth. Furthermore, the parity check nodes are generalized to connect to $a + 1$ nodes, of which a are message bit nodes. This structure is illustrated in Figure 15.20. From this general structure, assignment of the code parameters can be optimized using density evolution, so that the decoding SNR threshold can be minimized.

While an irregular RA code is simply a special case of an irregular LDPC code, it has an important advantage: it can be very efficiently encoded. The RA codes thus provide an

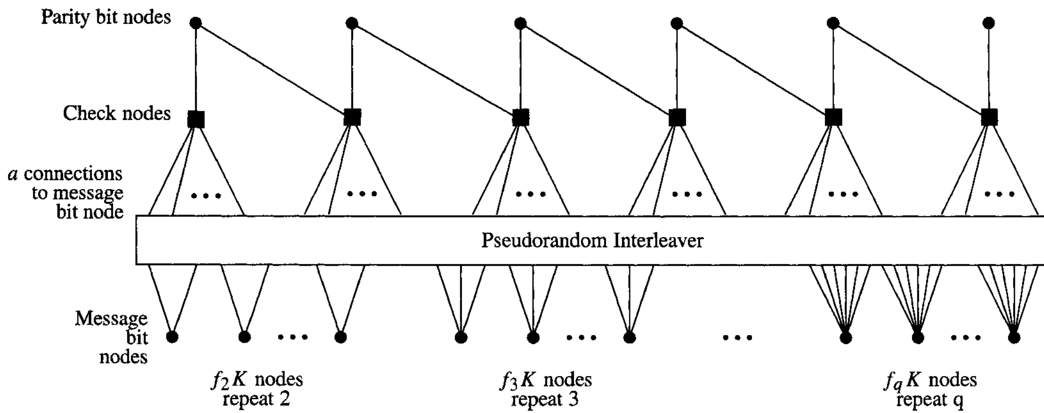


Figure 15.20: Tanner graph for an irregular repeat-accumulate code.

instance of a code which has linear encode complexity and linear decode complexity, while still achieving excellent decoding performance.

Programming Laboratory 13: Programming an LDPC Decoder

Objective

You are to implement the decoding algorithm for the low-density parity-check code, as described in Algorithm 15.1, and test it using (1) a small code, to verify that your algorithm is working correctly and (2) a couple of long codes.

Background

Reading: Sections 15.2, 15.5.

Because the parity check matrix for a long code would be huge if explicitly represented, it is important to represent only the nonzero elements of the sparse matrix. To store a sparse matrix in a file, the following format is used.

```
N M
maxcolweight maxrowweight
colwt colwt colwt ... colwt
rowwt rowwt rowwt ... rowwt
M1(1) M1(2) M1(3) ...
M2(1) M2(2) M2(3) ...
...
N1(1) N1(2) N1(3) N1(4) N1(5) N1(6) ...
N2(1) N2(2) N2(3) N2(4) N2(5) N2(6) ...
...
```

In this file representation, `N` and `M` are the N and M parameters for the code, where $M = N - K$, `maxcolweight` and `maxrowweight` represent the maximum weight of the columns (typically 3) and `maxrowweight` represents the maximum weight of the rows. The list `colwt colwt colwt ... colwt` is the list the column weights of each of the N columns of A. The list `rowwt rowwt ... rowwt` is the list of the row weights of each of the M rows of A. Then the list `M1(1) M1(2) M1(3) ...` is the list of the data $\mathcal{M}_1$, that is, the checks that bit 1 participates in (representing the first column of A). The other `M` data describe the other columns of A. Then `N1(1) N1(2) ...` describe $\mathcal{N}_1$, the set of bits that participate in check 1, and so forth. As an example, the description of the A matrix of (15.1) provided in the file `Asmall.txt`.

Since the matrices for real codes are very large, it important to use a sparse representation in the internal computer representation as well. That is, rather than allocate space for a $M \times N$ matrix to represent A, you only need to allocate space for a $w_c \times N$ matrix or a $M \times w_r$ matrix (depending on how you do your internal representation).

There are a variety of ways in which you can represent the sparse data. It takes some work, however, to represent the data in such a way that you can access data in both row-oriented and a column-oriented ways, since both directions are used in the vertical and horizontal steps. We describe here one method to sparsely represent the data.

Think of the sparse elements in the A matrix "floating" to the top of the matrix. With this representation, it is easy to access down the column to do a vertical step. Here is the computation of the pseudoposteriors:

```
// Vertical step:
// Work across the columns
```

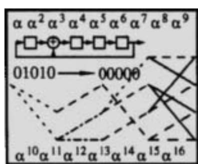
```
for(n = 0; n < N; n++) {
   prod0 = 1-pn[n];  prod1 = pn[n];
   // pn represents the channel posterior
   // compute the pseudoposteriors
   // Now work down each column
   for(l = 0; l < Mnlen[n]; l++) {
      prod0 *= r0[l][n];    prod1 *= r1[l][n];
   }
   alpha = 1/(prod0 + prod1);
   q0p[n] = alpha*prod0;    q1p[n] = alpha*prod1;
}
```

However, when doing the horizontal step, it is necessary to keep track of which row the compressed data comes from. This is done by counting the number of nonzero elements above an element, in an array called `na` ("number above"). The `na` array is set to zero initially (for every iteration). As the elements in a column are used, the row indexing into the "compressed" matrix skips down to row `na[column]`. The following code shows the horizontal step implemented this way:

```
// Make sure na[] is set to zero before this step
// Horizontal step:
for(row = 0; row < M; row++) {
   // Copy the data on this row into a
   // temporary array of deltaq values
   for(l = 0; l < Nmlen[row]; l++) {
      // for each nonzero value on this row
      idx = Nm[row][l];
      deltaq[l] = 1-2*q1[na[idx]][idx];
      // compute delta q
   }
   // Work over nonzero elements of
   // this row, taking products
   for(l = 0; l < Nmlen[row]; l++) {
      prod = 1;
      for(k = 0; k < Nmlen[row]; k++) {
         if(k==l) continue;  // skip when k==l
         prod *= deltaq[k];
      }
      // assign the product back into
      // sparse structure
      idx = Nm[row][l];
      r1[na[idx]][idx] = (1-prod)/2;   // r1 value
      r0[na[idx]++][idx] = (1+prod)/2; // r0 value
   }
}
```

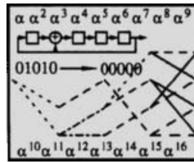
Assignment

1) Complete the class `galdec` by finishing the details on the `decode` member function. Test the decode function using the `galtest` program, which uses the 5×10 parity check matrix represented in `Asmall.txt`. You should obtain numerical results similar to those in Example 15.6.



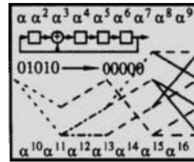
Algorithm 15.3 LDPC class declaration and definition
File: `galdec.h`
`galdec.cc`
`galtest.cc`

For debugging purposes, it may be helpful to compare with a Matlab version of the decoder. Note, however, that this implementation does *not* treat the sparse matrices efficiently, and so will have trouble scaling to larger codes.



Algorithm 15.4 Matlab code to test LDPC decoding
File: `ldpc.m`
`galdecode.m`

2) Using the program `galtest2`, produce the probability of error plot and average number of decoding iterations plot as in the chapter for a rate 1/4 and a rate 1/2 code, defined in `A1-2.txt` and `A1-4.txt`.



Algorithm 15.5 Make performance plots for LDPC codes
File: `galtest2.cc`
`A1-2.txt`
`A1-4.txt`

Numerical Considerations

Because the probabilities eventually tend toward either 0 or 1, some of the computations can be somewhat sensitive. Suppose that the numbers p_0' and p_1' are to be normalized to form probabilities according to

$$p_0 = \frac{p_0'}{p_0' + p_1'} \qquad p_1 = \frac{p_1'}{p_0' + p_1'}.$$

Suppose also that $p_0' > p_1'$. Then the probabilities can be computed as

$$p_0 = \frac{1}{1 + \frac{p_1'}{p_0'}} \qquad p_1 = \frac{\frac{p_1'}{p_0'}}{1 + \frac{p_1'}{p_0'}}.$$

This is a stable way of numerically computing the result. If $p_1' > p_0'$, then the result can similarly be written in terms of the ratio p_0'/p_1'.

15.15 Exercises

15.1 Determine a low-density parity check matrix for the $(n, 1)$ repetition code. Show that there are no cycles of girth 4 in the Tanner graph.

15.2 Let H be a binary matrix whose columns are formed from the $\binom{m}{2}$ m-tuples of weight 2. Determine the minimum nonzero weight of a code that has H as its parity check matrix. Show that there are no cycles of girth 4 in the Tanner graph.

15.3 Let $h(x) = 1 + x + x^3 + x^7$. Form a 15×15 parity check matrix by the cyclical shifts of the coefficients of $h(x)$. Show that there are no cycles of length 4 in the Tanner graph. What is the dimension of the code represented by this matrix?

15.4 For the parity check matrix

$$A = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix}$$

(a) Construct the Tanner graph for the code.
(b) Determine the girth of the minimum-girth cycle.
(c) Determine the number of cycles of length 6.
(d) Determine a generator matrix for this code.
(e) Express the $\mathcal{N}$ and $\mathcal{M}$ lists describing this parity check matrix.

15.5 Let $\{c_1, \ldots, c_n\}$ be independent bits, $c_i \in \{0, 1\}$ and let $\lambda_i(c) = \log \frac{P(c_i=1)}{P(c_i=0)}$. Let $z = \sum_{i=1}^n c_i$ be the parity check of the c_i. Let

$$\lambda(z) = \log \frac{P(z = 1)}{P(z = 0)}$$

be the likelihood ratio of the parity check.

(a) Show that

$$\tanh\left(-\frac{\lambda(z)}{2}\right) = \prod_{i=1}^{n} \tanh\left(\frac{-\lambda_i(c)}{2}\right).$$

This is the tanh rule. Thus

$$\lambda(z) = -2\tanh^{-1}\left(\prod_{i=1}^{n} \tanh\left(\frac{-\lambda_i(c)}{2}\right)\right). \tag{15.53}$$

(b) Let

$$f(x) = \log \frac{e^x + 1}{e^x - 1}, \qquad x > 0.$$

Show that $f(f(x)) = x$ for $x > 0$.

(c) Plot $f(x)$.

(d) Let $\sigma_z = -\prod_{i=1}^n \operatorname{sign}(-\lambda_i(c))$ be the product of the signs of the bit likelihoods. Show that (15.53) can be written as

$$\operatorname{sign}(-\lambda(z)) = \prod_{i=1}^{n} \operatorname{sign}(-\lambda_i(c))$$

$$\tanh(|\lambda(z)/2|) = \prod_{i=1}^{n} \tanh(|\lambda_i(c)|/2). \tag{15.54}$$

(e) Show that (15.54) can be written as

$$\lambda(z) = \sigma_z f\left(\sum_{i=1}^{n} f(|\lambda_i(c)|)\right).$$

Hint: $\tanh(z/2) = \frac{e^x-1}{e^x+1}$.

(f) Show that if c_k is equally likely to be 0 or 1, then $\lambda(z) = 0$. Explain why this is reasonable.

(g) Explain why $\lambda(z) \approx -(-1)^{\hat{z}}|\lambda_{\min}(c)|$, where $|\lambda_{\min}(c)| = \min_i |\lambda_i(c)|$.

15.6 Let $\mathbf{a}_m$ be the mth row of a parity check matrix A and let z_m be the corresponding parity check. That is, for some received vector $\mathbf{e}$, $z_m = \mathbf{e}\mathbf{a}_m^T$. Let $\mathbf{a}_m$ be nonzero in positions $i_1, i_2, \ldots, i_p$, so that

$$z_m = e_{i_1} + e_{i_2} + \cdots + e_{i_p}.$$

Assume that each e_i is 1 with probability p_c. Let $z_m(w)$ be the sum of the first w terms in z_m and let $p_z(w)$ be the probability that $z_m(w) = 1$.

(a) Show that $p_z(w+1) = p_z(w)(1-p_c)+(1-p_z(w))p_c$, with initial condition $p_z(0) = 0$.

(b) Show that $p_z(w) = \frac{1}{2} - \frac{1}{2}(1-2p_c)^w$.

15.7 [217] Hard decision decoding on the BSC. Let A be the $m \times n$ parity-check matrix for a code and let $\mathbf{r}$ be a binary-valued received vector. A simple decoder can be implemented as follows:

Set $\hat{\mathbf{c}} = \mathbf{r}$ (initial codeword guess)
[*] Let $\mathbf{z} = \mathbf{c}A^T \pmod 2$ (compute checks)
If $\mathbf{z} = \mathbf{0}$, end. (everything checks — done)
Evaluate the vote vector $\mathbf{v} = \mathbf{z}A$ (not modulo 2), which counts for each bit the number of unsatisfied checks to which it belongs. The bits of $\hat{\mathbf{c}}$ that get the most votes are viewed as the most likely candidates for being wrong. So flip all bits $\hat{\mathbf{c}}$ that have the largest vote.
Go to [*]

(a) Let $\mathbf{r} = [1, 0, 1, 0, 1, 0, 0, 1, 0, 1]$. Compute $\mathbf{s}$, $\mathbf{v}$, and the updated $\hat{\mathbf{c}}$ using the parity check matrix in (15.1). Is the decoding complete at this point?

(b) Repeat for $\mathbf{r} = [1, 0, 1, 0, 1, 1, 0, 1, 0, 1]$. Continue operation until correct decoding occurs.

(c) Show that $\mathbf{v} = \hat{\mathbf{s}}A$ counts the number of unsatisfied checks.

(d) Some analysis of the algorithm. We will develop an expression for the average number of bits changed in an iteration. Let w_r be the row weight of A (assumed fixed) and w_c be the column weight (assumed fixed). Determine the largest possible number of votes w a check can be involved in.

(e) Let $\mathbf{e}$ be the (binary) error vector. Let bit l of $\mathbf{e}$ participate in check z_m (that is, $A_{ml} = 1$). Show that when $e_l = 0$, the probability that bit l receives a vote of w is $a = P(\text{vote}_l = w|e_l = 0) = [p_z(w_r - 1)]^w$, where $p_z(w)$ is the function defined in Exercise 15.6.

(f) Show that when $e_l = 1$, the probability that this bit receives the largest possible number of votes w is the probability $b = P(\text{vote}_l = w|e_l = 1) = [1 - p_z(w_r - 1)]^w$.

(g) Show that $P(e_l = 0|\text{vote}_l = w) \propto a(1 - p_c)$ and $P(e_l = 1|\text{vote}_l = w) \propto bp_c$, where p_c is the crossover probability for the channel.

(h) Hence show that the expected change in the weight of the error vector when a bit is changed after one round of decoding is $(a(1 - p_c) - bp_c)/(a(1 - p_c) + bp_c)$.

15.8 [113] Consider a sequence of m independent bits in which the jth bit is 1 with probability p_j. Show that the probability that an even number of the bits in the sequence are 1 is $(1+\prod_{j=1}^m(1-2p_j))/2$. (For an expression when all the probabilities are the same, see Exercise 3.3.)

15.9 The original Gallager decoder: Let P_{il} be the probability that in the ith parity check the lth bit of the check is equal to 1, $i = 1, 2, \ldots, m$, $l = 1, 2, \ldots, w_r$. Show that

$$\frac{P(c_n = 0|\mathbf{r}, \{z_m = 0, m \in \mathcal{M}_n\})}{P(c_n = 1|\mathbf{r}, \{z_m = 0, m \in \mathcal{M}_n\})} = \frac{1 - P(c_n = 1|\mathbf{r})}{P(c_n = 1|\mathbf{r})} \prod_{i=1}^{w_c} \frac{1 + \prod_{l=1}^{w_r-1}(1 - 2P_{il})}{1 - \prod_{l=1}^{w_r-1}(1 - 2P_{il})}$$

Use Exercise 15.8.

15.10 Suppose that each bit is checked by $w_c = 3$ checks, and that w_r bits are involved in each check. Let $p_0 = p_c$ be the probability that a bit is received in error.

(a) Suppose that r_n is received incorrectly (which occurs with probability p_0). Show that a parity check on this bit is unsatisfied with probability $(1 + (1 - 2p_0)^{w_r-1})/2$.

(b) Show that the probability that a bit in the first tier is received in error and then corrected is $p_0((1 + (1 - 2p_0)^{w_r-1})/2)^2$.

(c) Show that the probability that a bit in the first tier is received correctly and then changed because of unsatisfied parity checks is $(1 - p_0)((1 - (1 - 2p_0)^{w_r-1})/2)^2$.

(d) Show that the probability of error p_1 of a digit in the first tier after applying the decoding process is

$$p_1 = p_0 - p_0\left[\frac{1 + (1 - 2p_0)^{w_r-1}}{2}\right]^2 + (1 - p_0)\left[\frac{1 - (1 - 2p_0)^{w_r-1}}{2}\right]^2$$

and that after i steps the probability of error of processing a digit in the ith tier is

$$p_{i+1} = p_0 - p_0\left[\frac{1 + (1 - 2p_i)^{w_r-1}}{2}\right]^2 + (1 - p_0)\left[\frac{1 - (1 - 2p_i)^{w_r-1}}{2}\right]^2.$$

15.11 A bound on the girth of a graph. The girth of a graph is the length of the smallest cycle. In this exercise, you will develop a bound on the girth. Suppose a regular LDPC code of length n has m parity checks, with column weight w_c and row weight w_r. Let $2l$ be the girth of the associated Tanner graph.

(a) Argue that for any node, the neighborhood of edges on the graph of depth $l - 1$ forms a tree (i.e., the set of all edges up to $l - 1$ edges away), with nodes of odd depth having "out-degree" w_r and nodes of even depth having "out-degree" w_c.

(b) Argue that the number of nodes at even depths of the tree should be at most equal to n, and that the number of nodes at odd depths is equal to m.

(c) Determine the number of nodes at depth 0 (the root), at depth 2, at depth 4, at depth 6, and so forth. Conclude that the total number of nodes at even depth is

$$1 + w_c(w_r - 1)\frac{[(w_c - 1)(w_r - 1)]^{\lfloor l/2 \rfloor} - 1}{(w_c - 1)(w_r - 1) - 1}.$$

This number must be less than or equal to n. This yields an upper bound on l.

15.12 (Irregular codes) Show that (15.38) is true.

15.13 Show that $\sum_{j\geq 2} \nu_j/j = \int_0^1 \nu(x)\,dx$.

15.14 Show that the decoder algorithm of Algorithm 15.2 can be written as in (15.42) and (15.41).

15.15 Draw the Tanner graph for a (4, 1) Repeat Accumulate code with three input bits with an interleaver $\Pi = (6, 12, 8, 2, 3, 4, 10, 1, 9, 5, 11, 7)$.

15.16 References

Many references appear throughout this chapter; see especially Section 15.1. Tanner graphs and codes built using graphs are discussed in [330]. On the analysis of LDPC codes, the paper [290] is highly recommended, which uses density evolution. Density evolution is also discussed in [51] (our discussion comes from there) and from [15]. See also [74]. The EXIT chart for turbo codes was presented in [334, 335].

The discussion of irregular LDPC codes comes from [292]. See also [212].

One of the potential drawbacks to LDPC codes is the large number of decoding iterations that may be required, resulting in increased latency and decoding hardware complexity. An approach to reduce the number of iterations is presented in [47, 62], in which the messages around a cycle in the Tanner graph establish an eigenvalue problem or a least-squared problem.

Our discussion of encoding comes from [289]. Another class of approaches is based on iterative use of the *decoding* algorithm in the encoding process. This could allow for the same hardware to be used for both encoding and decoding. These approaches are described in [135]. Using the sum-product decoder on general matrices is explored in [245]. An excellent resource on material related to turbo codes, LDPC codes, and iterative decoding in general is the February 2001 issue of the *IEEE Transactions on Information Theory*, which contains many articles in addition to those articles cited here. Interesting results on the geometry of the iterative decoding process are in [156].

Repeat accumulate codes are presented in [167]. Our presentation has benefited from the discussion in [15].

Chapter 16

Decoding Algorithms on Graphs

16.1 Introduction

In this chapter, the seemingly distinct algorithms applied to turbo codes and to low-density parity-check codes are shown to be instances of a more general algorithm for message passing on graphs. In fact, this algorithm also circumscribes the Viterbi algorithm and the fast Hadamard transform and the fast Fourier transform, which have been employed in this book, as well as many other useful algorithms such as the Kalman filter and state space models, fast matrix multiplication, directed acyclic graphs, and hidden Markov modeling, which are beyond the purview of these pages.

Somewhat amazingly, the computational efficiency ascribed to all of these algorithms can be attributed to the following observation about the distributive law $ab+ac = a(b+c)$: The first computation requires two multiplications and one addition, while the second requires only one addition and one multiplication. Application of this distributive property arises in many contexts, where we want to "marginalize" out some variables.

Example 16.1 [2] Suppose that $f(x, y, w)$ and $g(x, z)$ are real-valued functions, where x, y, z and w are variables taking values in a finite set A with q elements. Suppose we are to compute

$$\alpha(x, w) = \sum_{y,z \in A} f(x, y, w)g(x, z) \quad \text{and} \quad \beta(y) = \sum_{x,z,w \in A} f(x, y, w)g(x, z).$$

That is, $\alpha(x, w)$ is obtained by marginalizing out the variables y and z, while $\beta(y)$ is obtained by marginalizing out the variables x, z and w. The marginalization for $\alpha(x, w)$ requires summation over q^2 different values for each of the q^2 values of (x, w), for a total complexity of $2q^4$ (one addition and one multiplication for each). The function $\beta(y)$ is obtained by marginalizing over the variables x, w, and z, at a complexity of $2q^3$ per each of the q values of y, for a complexity of $2q^4$. The overall complexity to compute both the α and β marginalizations is $4q^4$.

Contrastingly, by means of the distributive law we can write

$$\alpha(x, w) = \left(\sum_{y \in A} f(x, y, w)\right)\left(\sum_{z \in A} g(x, z)\right).$$

Now define the functions $\alpha_1(x, w)$ and $\alpha_2(x)$ by

$$\alpha_1(x, w) = \sum_{y \in A} f(x, y, w) \qquad \alpha_2(x) = \sum_{z \in A} g(x, z).$$

All the values of $\alpha_1(x, w)$ can be computed in q^3 additions, and all the values of $\alpha_2(x)$ can be computed in q^2 additions. Then the values $\alpha(x, w) = \alpha_1(x, w)\alpha_2(x)$ can be computed in q^2 multiplications, resulting in a total complexity of $q^3 + 2q^2$. Employing the distributive law again, we obtain

$$\beta(y) = \sum_{x,w \in A} f(x, y, w)\alpha_2(x).$$

By reusing $\alpha_2(x)$, only another $2q^3$ operations are necessary to compute $\beta(y)$. The total complexity to compute both $\alpha(x, w)$ and $\beta(y)$ is $3q^3 + 2q^2$ operations, compared to the $4q^4$ operations for the direct method. □

16.2 Operations in Semirings

While marginalization seems like a rather specialized operation, the development below demonstrates that it arises in a variety of settings. In order for the algorithm to have broad applicability, we express it in the framework of a commutative semiring.

Definition 16.1 A **commutative semiring** $\langle K, +, \cdot \rangle$ is a set K together with two binary operations $+$ and $\cdot$ which satisfy the following three axioms:

SR1 The operation $+$ is associative and commutative and there is an additive identity called "0" such that $k + 0 = k$ for all $k \in K$. (No additive inverse is necessary, so this does not form a group; this algebraic structure is called a **commutative monoid**.)

SR2 The operation "$\cdot$" is associative and commutative. There is a multiplicative identity called "1" such that $1 \cdot k = k$ for all $k \in K$.

SR3 The distributive law holds: $(a \cdot b) + (a \cdot c) = a \cdot (b + c)$.

Often the semiring $\langle K, +, \cdot \rangle$ is denoted simply by K. □

There are a variety of sets/operations which form interesting and useful semirings. Some of these are summarized in Table 16.1.

Table 16.1: Some Commutative Semirings [2]

	Set K	"$(+, 0)$"	"$(\cdot, 1)$"	Name
1	A	$(+, 0)$	$(\cdot, 1)$	(conventional $+$ and $\cdot$ operations)
2	$A[x]$	$(+, 0)$	$(\cdot, 1)$	(conventional $+$ and $\cdot$ operations)
3	$A[x, y, \ldots]$	$(+, 0)$	$(\cdot, 1)$	(conventional $+$ and $\cdot$ operations)
4	$[0, \infty)$	$(+, 0)$	$(\cdot, 1)$	sum-product
5	$(0, \infty]$	$(\min, \infty)$	$(\cdot, 1)$	min-product
6	$[0, \infty)$	$(\max, 0)$	$(\cdot, 1)$	max-product
7	$(-\infty, \infty]$	$(\min, \infty)$	$(+, 0)$	min-sum
8	$[-\infty, \infty)$	$(\max, -\infty)$	$(+, 0)$	max-sum
9	$\{0, 1\}$	(or, 0)	(and, 1)	Boolean
10	2^S	$(\cup, \emptyset)$	$(\cap, S)$	

Whatever semiring we are working in, we generically employ the $+$ or the Σ (summation) operator to indicate the "$+$" operation and juxtaposition, $\cdot$, or Π to indicate the "$\cdot$" operator.

16.3 Functions on Local Domains

Let $x_1, x_2, \ldots, x_n$ be a set of variables such that x_i takes on values in a set A_i, and let $|A_i| = q_i$. For $S = \{i_1, i_2, \ldots i_r\}$ a subset of $\{1, 2, \ldots, n\}$, we denote the Cartesian product $A_{i_1} \times A_{i_2} \times \cdots \times A_{i_r}$ by A_S and denote the variable list $\{x_{i_1}, x_{i_2}, \ldots, x_{i_r}\}$ by x_S. The set

$A_1 \times A_2 \times \cdots A_n$ is denoted simply by $\mathbf{A}$. The entire set of variables $\{x_1, x_2, \ldots, x_n\}$ is denoted $\mathbf{x}$.

Let $\mathcal{S} = \{S_1, S_2, \ldots, S_M\}$ be M subsets of $\{1, \ldots, n\}$. For each $i = 1, 2, \ldots, M$, suppose there is a function α_i: $A_{S_i} \to K$, where K is a commutative semiring. The function α_i is called a *local kernel* function. When $S_i = \{i_1, i_2, \ldots i_r\}$, then α_i is a function of the variables $x_{i_1}, x_{i_2}, \ldots, x_{i_r}$. The set

$$A_{S_i} = A_{i_1} \times A_{i_2} \times \cdots \times A_{i_r}$$

is called the *configuration space* of α_i; each element of A_{S_i} is a particular configuration of the variables, assigning a value to each variable x_i from the set A_i. The set $L_i = \{x_{i_1}, x_{i_2}, \ldots, x_{i_r}\} = x_{S_i}$ is called the *local domain* of the local kernel function. The *global kernel function* $\beta : \mathbf{A} \to K$ is defined by

$$\beta(x_1, x_2, \ldots, x_n) = \prod_{i=1}^{M} \alpha_i(x_{S_i}) \tag{16.1}$$

(where the Π symbol denotes the "$\cdot$" operation in the semiring K).

The algorithm to be developed computes *marginal functions* of global kernel functions, which we define as follows. Let $S \subset \{1, 2, \ldots, n\}$ be a set of variable indices and let S^c denote the complement of the set S relative to the universal set $\{1, 2, \ldots, n\}$. Then the S-marginalization is the function $\beta_S : A_S \to K$ defined by

$$\beta_S(x_S) = \sum_{x_{S^c} \in A_{S^c}} \beta(\mathbf{x}) \tag{16.2}$$

(where Σ denotes "+" in the semiring K). In other words, all the variables not in the set x_S are "summed out." We also sometimes use the notation $\beta_L(x_L)$, where L is the local domain corresponding to the set S. In addition to the set complement notation $x_{S^c} \in A_{S^c}$, we also use the "summary" notation $\sim \{x_S\}$ to indicate "every variable not in x_S," where the set A_S is implicit:

$$\sum_{\sim\{x_S\}} \text{ means the same as } \sum_{x_{S^c} \in A_{S^c}} .$$

Note: In many instances a normalization of the marginal functions is computed. In this case, (16.2) would be more properly written as

$$\beta_S(x_S) \propto \sum_{x_{S^c} \in A_{S^c}} \beta(\mathbf{x}).$$

The algorithm to be developed is sometimes called the *marginalize a product of functions* (MPF) algorithm. It is also called (because semiring 4 in Table 16.1 is a "generic" ring) the *sum-product algorithm*. Other equivalent or nearly equivalent algorithms are called "message-passing" or "belief propagation" [258]. We now provide examples showing how this framework can be applied to a variety of problems.

Example 16.2 We express the problem of Example 16.1 using these concepts. Let $L_1 = \{x_1, x_2, x_4\}$ be a local domain and $L_2 = \{x_1, x_3\}$ be another local domain, where each x_i takes value in a set A having q elements. Let

$$\alpha_1(x_1, x_2, x_4) = f(x_1, x_2, x_4) \qquad \alpha_2(x_1, x_3) = g(x_1, x_3)$$

be local kernel functions. Then the global kernel function is

$$\beta(x_1, x_2, x_3, x_4) = \alpha_1(x_1, x_2, x_4)\alpha_2(x_1, x_3).$$

Let $S_3 = \{1, 4\}$. The S_3-marginalization of β is

$$\beta_{S_3}(x_1, x_4) = \sum_{x_2 \in A, x_3 \in A} \beta(x_1, x_2, x_3, x_4) = \sum_{x_2 \in A, x_3 \in A} \alpha_1(x_1, x_2, x_4)\alpha_2(x_1, x_3)$$
$$= \sum_{\sim\{x_1, x_4\}} \alpha_1(x_1, x_2, x_4)\alpha_2(x_1, x_3).$$

Let $S_4 = \{x_2\}$. The S_4-marginalization of the β is

$$\beta_{S_4}(x_2) = \sum_{x_1, x_3, x_4 \in A} \alpha_1(x_1, x_2, x_4)\alpha_2(x_1, x_3) = \sum_{\sim\{x_2\}} \alpha_1(x_1, x_2, x_4)\alpha_2(x_1, x_3).$$

□

Example 16.3 [2] This example demonstrates how marginalization can be used to express a useful problem, that of computing the Hadamard transform. (The efficient algorithm to be developed gives rise to the fast Hadamard transform.) Let $x_1, x_2, x_3, y_1, y_2, y_3$ be six variables each taking values in the set $A = \{0, 1\}$. Let $f(y_1, y_2, y_3)$ be a real-valued function of its arguments. Define the following sets, domains, and kernels:

i	Local Domain L_i	Local Kernel α_i
1	$\{y_1, y_2, y_3\}$	$f(y_1, y_2, y_3)$
2	$\{x_1, y_1\}$	$(-1)^{x_1 y_1}$
3	$\{x_2, y_2\}$	$(-1)^{x_2 y_2}$
4	$\{x_3, y_3\}$	$(-1)^{x_3 y_3}$
5	$\{x_1, x_2, x_3\}$	1

We observe that the local kernel function associated with L_5 is the trivial, identity, kernel function. Introduction of trivial local kernels is often a useful trick to mapping problems into the "marginalize a product of functions" framework.

The global kernel function is

$$\beta(x_1, x_2, x_3, y_1, y_2, y_3) = f(y_1, y_2, y_3)(-1)^{x_1 y_1 + x_2 y_2 + x_3 y_3}.$$

Now let $L = \{x_1, x_2, x_3\}$. The marginalization of β with respect to this local domain is

$$\beta_L(x_1, x_2, x_3) = \sum_{y_1, y_2, y_3 \in \{0,1\}} f(y_1, y_2, y_3)(-1)^{x_1 y_1 + x_2 y_2 + x_3 y_3}$$
$$= \sum_{\sim\{x_1, x_2, x_3\}} f(y_1, y_2, y_3)(-1)^{x_1 y_1 + x_2 y_2 + x_3 y_3}.$$

This is a Hadamard transform of the function $f(y_1, y_2, y_3)$. □

Example 16.4 [195] In this example, we show how the power-of-two discrete Fourier transform (DFT) can be expressed in this formalism. Let $(w_0, w_1, \ldots, w_{N-1})$ be a complex-valued N-tuple, where N is a power of 2. The DFT is

$$W_k = \sum_{n=0}^{N-1} w_n e^{-j2\pi nk/N}, \qquad k = 0, 1, \ldots, N-1.$$

Let us represent n and k as a binary representation using binary-valued variables x_i to represent n and binary-valued variables y_i to represent k. For the sake of a specific example, suppose that $N = 8$. Then we write

$$n = 4x_2 + 2x_1 + x_0 \qquad k = 4y_2 + 2y_1 + y_0, \qquad x_i, y_i \in \{0, 1\}.$$

Write the global kernel function as

$$\beta(x_0, x_1, x_2, y_0, y_1, y_2) = w_n e^{-j2\pi nk/N} = w_{4x_2+2x_1+x_0} e^{-j2\pi(4x_2+2x_1+x_0)(4y_2+2y_1+y_0)/N}. \tag{16.3}$$

It can be shown that (see Exercise 16.6)

$$\beta(x_0, x_1, x_2, y_0, y_1, y_2) = w_{4x_2+2x_1+x_0}(-1)^{x_2 y_0}(-1)^{x_1 y_1}(-1)^{x_0 y_2}(j)^{-x_0 y_1} j^{-x_1 y_0}\Omega^{-x_0 y_0}, \tag{16.4}$$

where $\Omega = e^{j2\pi/N}$. We thus identify the following local domain and local kernel functions:

i	Local Domain L_i	Local Kernel α_i
1	$\{x_0, x_1, x_2\}$	$w_{4x_2+2x_1+x_0}$
2	$\{x_2, y_0\}$	$(-1)^{x_2 y_0}$
3	$\{x_1, y_1\}$	$(-1)^{x_1 y_1}$
4	$\{x_0, y_2\}$	$(-1)^{x_0 y_2}$
5	$\{x_0, y_1\}$	$(j)^{-x_0 y_1}$
5	$\{x_1, y_0\}$	$(j)^{-x_1 y_0}$
6	$\{x_0, y_0\}$	$\Omega^{-x_0 y_0}$

Then for $L = \{y_1, y_2, y_3\}$, the marginalization is

$$\beta_L(y_1, y_2, y_3) = \sum_{x_0, x_1, x_2 \in \{0,1\}} w_{4x_2+2x_1+x_0} e^{-j2\pi(4x_2+2x_1+x_0)(4y_2+2y_1+y_0)} = W_{4y_2+2y_1+y_0}.$$

□

Example 16.5 [2] **Maximum likelihood decoding.** We now consider application of these principles to error correction, demonstrating the idea with the (7,4,3) Hamming code having parity check matrix

$$H = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix}. \tag{16.5}$$

The codewords are transmitted through a memoryless channel, so that for the received vector $(y_1, y_2, \ldots, y_7)$ the likelihood of a particular codeword $(x_1, x_2, \ldots, x_7)$ is

$$p(y_1, y_2, \ldots, y_7 | x_1, x_2, \ldots, x_7) = \prod_{i=1}^{7} p(y_i | x_i). \tag{16.6}$$

The maximum likelihood decoder seeks the codeword which maximizes (16.6). Codewords $\mathbf{x}$ must satisfy the parity check equation $H\mathbf{x} = \mathbf{0}$. To describe the parity check conditions, we use the "Iverson convention" [126, p. 14]: If P is a Boolean proposition, then we take $[P]$ to be the $\{0, 1\}$-valued function indicating the truth of P:

$$[P] = \begin{cases} 1 & \text{if } P \text{ is true} \\ 0 & \text{if } P \text{ is false.} \end{cases}$$

We also have

$$-\log[P] = \begin{cases} 0 & \text{if } P \text{ is true} \\ \infty & \text{if } P \text{ is false.} \end{cases}$$

Then the parity check equations give rise to the functions

$$\chi_1(x_1, x_4, x_5, x_7) = -\log[x_1 + x_4 + x_5 + x_7 = 0]$$
$$\chi_2(x_2, x_4, x_6, x_7) = -\log[x_2 + x_4 + x_6 + x_7 = 0]$$
$$\chi_3(x_3, x_5, x_6, x_7) = -\log[x_3 + x_5 + x_6 + x_7 = 0].$$

The domains and kernels for this problem are as follows:

i	Local Domain L_i	Local Kernel α_i
1	$\{x_1\}$	$-\log p(y_1 \mid x_1)$
2	$\{x_2\}$	$-\log p(y_2 \mid x_2)$
$\vdots$	$\vdots$	
7	$\{x_7\}$	$-\log p(y_7 \mid x_7)$
8	$\{x_1, x_4, x_5, x_7\}$	$\chi_1(x_1, x_4, x_5, x_7)$
9	$\{x_2, x_4, x_6, x_7\}$	$\chi_2(x_2, x_4, x_6, x_7)$
10	$\{x_3, x_5, x_6, x_7\}$	$\chi_3(x_3, x_5, x_6, x_7)$

Using the min-sum semiring, the global kernel function is

$$\beta(x_1, x_2, \ldots, x_7) = \begin{cases} -\sum_{i=1}^{7} \log p(y_i \mid x_i) & \text{if } (x_1, \ldots, x_7) \text{ is a codeword} \\ \infty & \text{if } (x_1, \ldots, x_7) \text{ is not a codeword} \end{cases}$$

Let $S_i = \{i\}$ and $L_i = \{x_i\}$. The S_i-marginalization is

$$\beta_i(x_i) = \min_{\substack{\sim \{x_i\}, \text{ where} \\ (x_1, \ldots, x_7) \\ \text{is a codeword}}} \left(-\sum_{i=1}^{7} \log p(y_i \mid x_i) \right).$$

The value of x_i for which $\beta_i(x_i)$ is the smallest is the value of the ith component of a maximum-likelihood codeword, that is, a codeword for which $\log p(y_1, \ldots, y_7 \mid x_1, \ldots, x_7)$ is largest.

As we will see, cycles in the factor graph preclude exact calculation of this marginalization. Otherwise, this would generalize to be a computationally efficient means of finding the maximum likelihood codeword for an arbitrary linear block code. □

Example 16.6 [195] **MAP decoding.** We consider again a coding problem. Let $(x_1, \ldots, x_n)$ be selected according to a uniform probability from a code $\mathcal{C}$ of length n and transmitted over a memoryless channel, whose output is the vector $\mathbf{y} = (y_1, y_2, \ldots, y_n)$. The posterior distribution $p(\mathbf{x}|\mathbf{y})$ is proportional to the function $p(\mathbf{y}|\mathbf{x})p(\mathbf{x})$. We write

$$\beta(\mathbf{x}) \propto p(\mathbf{y}|\mathbf{x})p(\mathbf{x}),$$

representing a global kernel function.

Since the prior distribution of the transmitted codeword is assumed to be uniform over the code, we have

$$p(\mathbf{x}) = \chi_{\mathcal{C}}(\mathbf{x})/|\mathcal{C}|,$$

where $\chi_{\mathcal{C}}(\mathbf{x})$ is 1 if $\mathbf{x} \in \mathcal{C}$, and 0 otherwise. Then we have

$$\beta(\mathbf{x}) \propto \frac{1}{|\mathcal{C}|} \chi_{\mathcal{C}}(x_1, \ldots, x_n) \prod_{i=1}^{n} p(y_i \mid x_i).$$

For the parity check matrix of (16.5), we have (redefining the χ functions)

$$\beta(x_1,\ldots,x_7) \propto$$

$$\frac{1}{|C|}[x_1+x_4+x_5+x_7=0][x_2+x_4+x_6+x_7=0][x_3+x_5+x_6+x_7=0]\prod_{i=1}^{7} p(y_i|x_i)$$

$$=\frac{1}{|C|}\chi_1(x_1+x_4+x_5+x_7)\chi_2(x_2+x_4+x_6+x_7)\chi_3(x_3+x_5+x_6+x_7)\prod_{i=1}^{7} p(y_i|x_i).$$

The domains and kernels for this are very similar to those for the maximum likelihood case, except that we use semiring of conventional addition and multiplication.

i	Local Domain L_i	Local Kernel α_i
1	$\{x_1\}$	$p(y_1|x_1)$
2	$\{x_2\}$	$p(y_2|x_2)$
$\vdots$	$\vdots$	
7	$\{x_7\}$	$p(y_7|x_7)$
8	$\{x_1, x_4, x_5, x_7\}$	$\chi_1(x_1, x_4, x_5, x_7)$
9	$\{x_2, x_4, x_6, x_7\}$	$\chi_2(x_2, x_4, x_6, x_7)$
10	$\{x_3, x_5, x_6, x_7\}$	$\chi_3(x_3, x_5, x_6, x_7)$

Now the marginalization is computed by

$$\beta_i(x_i) = \sum_{\sim\{x_i\}} \beta(x_1,\ldots,x_7).$$

This represents a marginalization of the posterior density, which computes a quantity proportional to $p(x_i|y_1,\ldots y_7)$. □

16.4 Factor Graphs and Marginalization

Factor graphs are a way of representing the computations in the marginalize a product of functions operation which reveal how to take advantage of the distributive law to reduce the computational complexity.

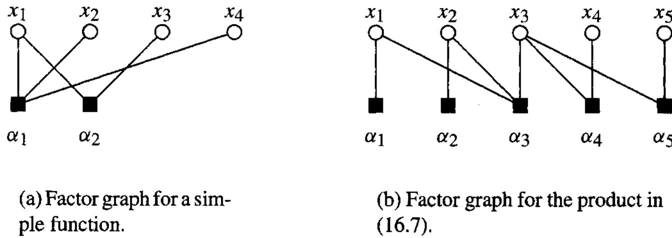
Definition 16.2 Factor graphs are bipartite graphs that represent the factorization of the global kernel function (16.1), having a variable node (or vertex) for each (single) variable x_i and a factor node for each local kernel function α_j, and an edge connecting a variable node x_i to a factor node α_j if and only if x_i is an argument to the function α_j. □

Example 16.7 Figure 16.1(a) shows the factor graph for the functions of Example 16.2. The function nodes are indicated with filled blocks and the variable nodes are circles. □

Example 16.8 [195] Let $\alpha_1(x_1)$, $\alpha_2(x_2)$, $\alpha_3(x_1, x_2, x_3)$, $\alpha_4(x_3, x_4)$, and $\alpha_5(x_3, x_5)$ be local kernel functions. Let

$$\beta(x_1, x_2, x_3, x_4, x_5) = \alpha_1(x_1)\alpha_2(x_2)\alpha_3(x_1, x_2, x_3)\alpha_4(x_3, x_4)\alpha_5(x_3, x_5) \tag{16.7}$$

be the global kernel function. The factor graph corresponding to this factorization is shown in Figure 16.1(b). □



(a) Factor graph for a simple function.

(b) Factor graph for the product in (16.7).

Figure 16.1: Factor graph for some examples.

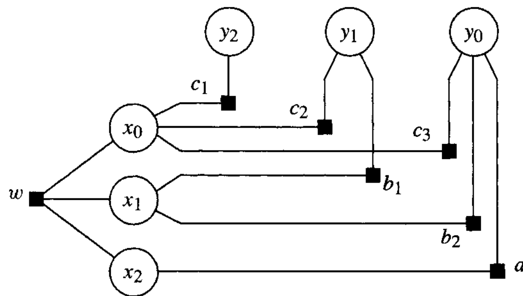


Figure 16.2: Factor graph for a DFT.

Example 16.9 The factor graph for the DFT factorization (16.4) is shown in Figure 16.2. The functions are

$$c_1(x_0, y_2) = (-1)^{x_0 y_2} \qquad c_2(x_0, y_1) = j^{-x_0 y_1} \qquad c_3 = \Omega^{-x_0 y_0}$$

$$b_1(x_1, y_1) = (-1)^{x_1 y_1} \qquad b_2(x_1, y_0) = j^{-x_1 y_0} \qquad a(x_2, y_0) = (-1)^{x_2 y_0}.$$

□

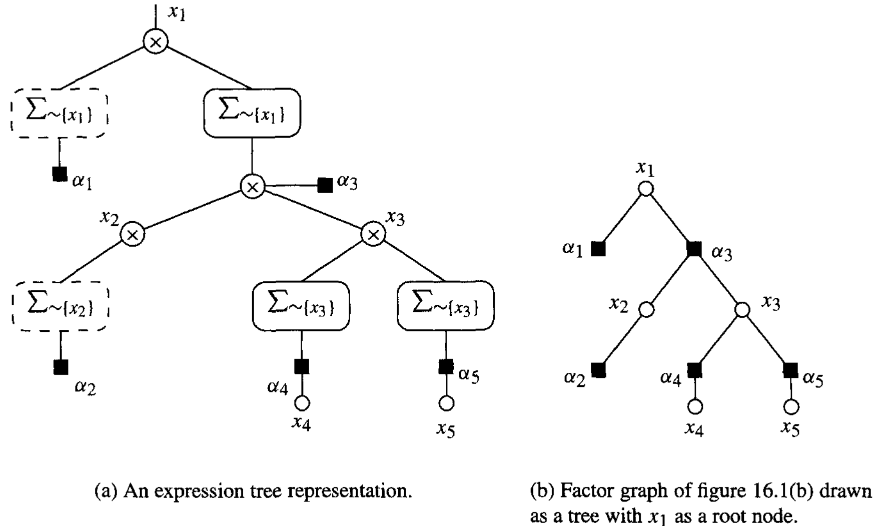
16.4.1 Marginalizing on a Single Variable

Let us now consider the marginalization of the global kernel function of (16.7) to produce a function of a single variable. Letting $S = \{1\}$ we have

$$\begin{aligned}
\beta_S(x_1) &= \sum_{x_2,x_3,x_4,x_5} \beta(x_1, x_2, x_3, x_4, x_5) \\
&= \sum_{x_2,x_3,x_4,x_5} \alpha_1(x_1)\alpha_2(x_2)\alpha_3(x_1, x_2, x_3)\alpha_4(x_3, x_4)\alpha_5(x_3, x_5) \\
&= \sum_{\sim\{x_1\}} \alpha_1(x_1)\alpha_2(x_2)\alpha_3(x_1, x_2, x_3)\alpha_4(x_3, x_4)\alpha_5(x_3, x_5).
\end{aligned}$$

Applying the distributive law, we can write this as

$$\beta_S(x_1) = \alpha_1(x_1)\Big(\sum_{x_2} \alpha_2(x_2)\Big(\sum_{x_3} \alpha_3(x_1, x_2, x_3)\Big(\sum_{x_4} \alpha_4(x_3, x_4)\Big)\Big(\sum_{x_5} \alpha_5(x_3, x_5)\Big)\Big)\Big). \tag{16.8}$$



(a) An expression tree representation.

(b) Factor graph of figure 16.1(b) drawn as a tree with x_1 as a root node.

Figure 16.3: Graphical representations of marginalization.

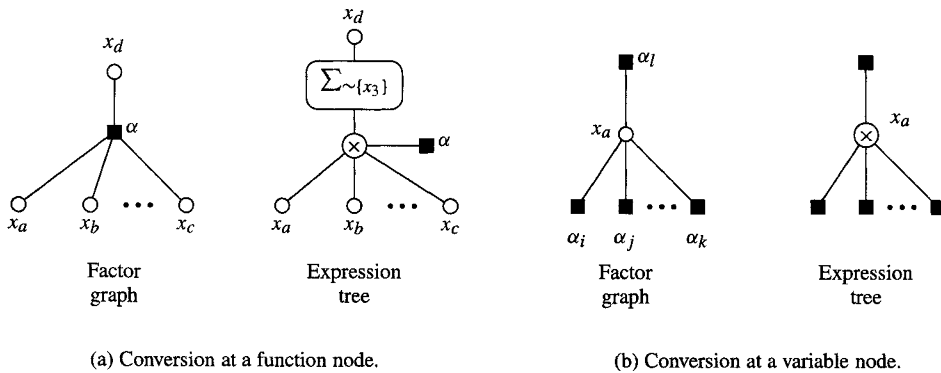
This requires fewer computations than the direct method, but raises the question of how this distributive structure can be discovered in a general problem. Using the summary notation, we can rewrite (16.8) as

$$\beta_S(x_1) = \alpha_1(x_1) \sum_{\sim\{x_1\}} \Big(\alpha_2(x_2)\alpha_3(x_1, x_2, x_3)\Big(\sum_{\sim\{x_3\}} \alpha_4(x_3, x_4)\Big)\Big(\sum_{\sim\{x_3\}} \alpha_5(x_3, x_5)\Big)\Big). \tag{16.9}$$

Our next step is to portray the computations in (16.9) as an expression tree, which is a graphical representation of how the computations are organized. Figure 16.3(a) shows an expression tree representing the computations in (16.9). In the expression tree, leaf nodes are either variables or functions. As in the factor graph, edges still indicate functional dependency. However, the summaries in (16.9) are also portrayed. (The dashed box around a summary notation indicates that there are no other variables to sum out). Now consider the factor graph of Figure 16.1(b) redrawn with the x_1 node as the root node, as shown in Figure 16.3(b). The structural similarity of the graph is evident. The transformations to go between the expression tree representation and the factor graph representation are summarized in Figure 16.4.

Observe that in going from the α_4 function node to the x_3 variable node, all variables except x_3 in the α_4 branch of the tree are summed out. Similarly, in going from the α_5 node to the x_3 node of the tree, all variables below the x_3 node in the α_5 branch of the tree are summed out. At the x_3 branch, the product of the information coming from the branches below is passed to the next higher level of the tree. In going from the α_2 branch to the x_2 branch, all other variables except x_2 in the branch are summed out (there are none). At the α_3 node, the information from the branches below is multiplied together. In going from the α_3 node to the x_1 node, all variables except x_1 are summed out.

We can think of the passing of information among the levels of parenthesization in the



(a) Conversion at a function node.

(b) Conversion at a variable node.

Figure 16.4: Conversion from factor graph to expression tree.

equation (16.9) (and the levels of the tree) as *message-passing*. A *message* is simply a representation of a function or a variable. For functions over discrete variables, the message is simply a list of the function values evaluated at all the outcomes. For example, suppose that x_1 takes the values $0, 1, \ldots, q-1$. Then the "message" associated with the function $\alpha_1(x_1)$ is the vector (or list) of function evaluations $(\alpha_1(0), \alpha_1(1), \ldots, \alpha_1(q-1))$.

We denote the message from a function node α to a variable node x as $\mu_{\alpha\to x}$. The message from a variable node x to a function node α is denoted as $\mu_{x\to\alpha}$. Consider again the summation in (16.9). In light of the expression tree/factor graph representation, we can designate the various parts of the summation as follows:

$$\beta_S(x_1) = \underbrace{\underbrace{\alpha_1(x_1)}_{\mu_{\alpha_1\to x_1}(x_1)} \underbrace{\sum_{\sim\{x_1\}}\Big(\alpha_3(x_1,x_2,x_3)\underbrace{\underbrace{\alpha_2(x_2)}_{\mu_{\alpha_2\to x_2}(x_2)}}_{\mu_{x_2\to\alpha_3}(x_2)} \underbrace{\underbrace{\Big(\sum_{\sim\{x_3\}}\alpha_4(x_3,x_4)\Big)}_{\mu_{\alpha_4\to x_3}(x_3)}\underbrace{\Big(\sum_{\sim\{x_3\}}\alpha_5(x_3,x_5)\Big)}_{\mu_{\alpha_5\to x_3}(x_3)}}_{\mu_{x_3\to\alpha_3}(x_3)}\Big)}_{\mu_{\alpha_3\to x_1}(x_1)}}_{\mu_{x_1\to\beta_S}(x_1)}.$$

For example,

$$\mu_{\alpha_5\to x_3}(x_3) = \sum_{\sim\{x_3\}} \alpha_5(x_3, x_5)$$

is the message from function node α_5 to variable node x_3, as a function of x_3. For finite domains this can be represented as a vector. For example, if x_3 takes on the values $0, 1, \ldots, q-1$, then the message $\mu_{\alpha_5\to x_3}$ is the vector

$$\mu_{\alpha_5\to x_3} = \begin{bmatrix} \sum_{\sim\{x_3\}}\alpha_5(0, x_5) \\ \sum_{\sim\{x_3\}}\alpha_5(1, x_5) \\ \vdots \\ \sum_{\sim\{x_3\}}\alpha_5(q-1, x_5) \end{bmatrix}.$$

The point of this example is this:

> The computational organization which puts the distributive law into effect when marginalizing to compute $\beta_i(x_i)$ for a single variable node x_i is found by drawing the factor graph as a tree with x_i as the root node, then starting at the leaf nodes of the tree, pass "messages" toward the root node.

In the procedure described below, it is also helpful to think of the marginalized function $\beta_{\{i\}}(x_i)$ as an implicit node in the factor graph.

Let us now state explicitly how the messages are formed. We first consider a message from a variable node to an adjacent function node, $\mu_{x\to\alpha}(x)$. Let e denote the edge connecting the x node and the α node. The message is formed by computing the product of all messages on edges incident to x *except* the edge e. Let $\text{Nb}(x)$ denote the set of nodes which are neighbors (adjacent) to x. Note that $\alpha \in \text{Nb}(x)$. The set $\text{Nb}(x) \setminus \alpha$ consists of all nodes that are adjacent to x except α. Then the message is computed by:

Message from variable node to function node:

$$\boxed{\mu_{x\to\alpha}(x) = \prod_{y\in\text{Nb}(x)\setminus\alpha} \mu_{y\to x}(x).} \tag{16.10}$$

The message from a function node to a variable node $\mu_{\alpha\to x}(x)$ is the product of the local kernel function α with all messages received at α from nodes other than the x node, summarized (marginalized) for the variable x. As before, let $\text{Nb}(\alpha)$ be the set of neighbors of the α node (i.e., nodes adjacent to it) and let $\text{Nb}(\alpha) \setminus x$ be this set excluding the x node. Then the message is computed by:

Message from a function node to a variable node:

$$\boxed{\mu_{\alpha\to x}(x) = \sum_{\sim\{x\}} \left(\alpha(x, \text{Nb}(\alpha)) \prod_{y\in\text{Nb}(\alpha)\setminus x} \mu_{y\to\alpha}(x) \right).} \tag{16.11}$$

The notation $\alpha(x, \text{Nb}(\alpha))$ indicates that α is a function of the variable x and the other variable nodes adjacent to α.

The algorithm based on these two steps is referred to as the *sum-product algorithm* (bearing in mind that both "sum" and "product" depend upon the semiring invoked for the problem) or the *message passing* algorithm. Following the usual convention, empty products are equal to 1. Summaries $\sum_{\sim\{x_i\}}$ over variables which are not in the summand simply return the summand.

Both the message from a variable node to a function node $\mu_{x\to\alpha}(x)$ and the message from a function node to a variable node $\mu_{\alpha\to x}(x)$ are functions of the variable x. That is, messages in either direction along the edge $\{x, \alpha\}$ are functions of x. This happens because at the function node the variables not associated with the edge $\{x, \alpha\}$ are marginalized out, while at the variable node all messages are functions of that variable. The two message rules (16.10) and (16.11) can be summarized as follows [195]:

> **The Message Passing Rule:** The message from a node v on an edge e is the product of the local kernel function at v (or the unit function if v is a variable node) with all messages received at v on edges other than e, summarized for the variable associated with e.

To restate: when a node v has messages from all of its neighbors *except one*, w, then v computes its message and sends it to w, using (16.10) if v is a variable node or (16.11) if v is a function node.

We may observe that the message passing rules have the important special cases shown in Table 16.2.

Table 16.2: Some Special Cases of Message Passing

Rule 1	By (16.10), the message sent from a variable node with only one neighbor is equal to 1 (because there is an empty product). This is called the *trivial message*.
Rule 2	By (16.10), a variable node with exactly two neighbors simply passes the message from one of its neighbors on to its other neighbor.
Rule 3	By (16.11), the message sent from a function node with only one neighbor is the function's value itself.

It is helpful to think of the factor graph as having an additional set of nodes — implicit nodes — representing the actual marginal functions $\beta_i(x_i)$. These nodes don't send messages, but they receive the final message sent from a variable node x_i.

16.4.2 Marginalizing on All Individual Variables

In some problems, it is desired to compute all the marginal functions $\beta_{\{i\}}(x_i)$ at all the variable nodes. One way to accomplish this, of course, is to consider each variable node in turn as the root node of the factor graph, and to pass messages from the leaf nodes in the tree to the respective root nodes. However, this may unnecessarily duplicate many of the computations. Instead, it is more efficient take no particular vertex as the root. Each vertex v in turn regards a neighbor vertex w as a parent vertex in the tree, processing and passing messages along from all its other vertices as they are received, which are regarded as children in the tree. Once a vertex v has received messages from all of its adjacent nodes *except one*, it sends its message to that node. The idea is summarized in Figure 16.5. The message passing is not deadlocked (unable to get started), since leaf nodes only have one neighbor, so they are immediately able to pass their messages to their neighbors. If the graph is actually a tree, as are all the examples we have seen so far, then processing moves from the leaf nodes to the root. In general, the actual sequence of messages passed among the nodes may vary. It may be helpful to think of each node as an independent processor, passing messages to a neighbor once all the messages from other neighbors have been received. In fact, this viewpoint implemented in hardware can give rise to efficient parallel architectures.

Example 16.10 The apparent inscrutability of messages sloshing around a factor graph can be alleviated somewhat by considering a sequence of messages for a particular example. We return to the factor graph of Figure 16.1(b). Marginalizing on all the variables can be accomplished with six steps of message passing. At each step, each node that has messages from all its neighbors except one passes its message to that neighbor. Figure 16.6, which shows the factor graph of Figure 16.1(b) redrawn, summarizes the steps of the algorithm and includes the "implicit" $\beta_{\{i\}}$ nodes for the marginalized functions. The numbers in the circles indicate the step in which the message is passed along the edge.

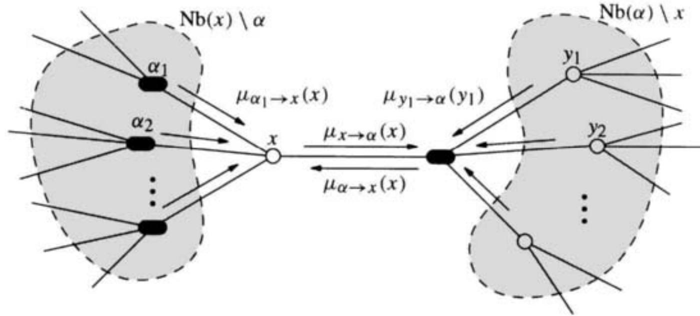


Figure 16.5: Message passing in the sum-product algorithm.

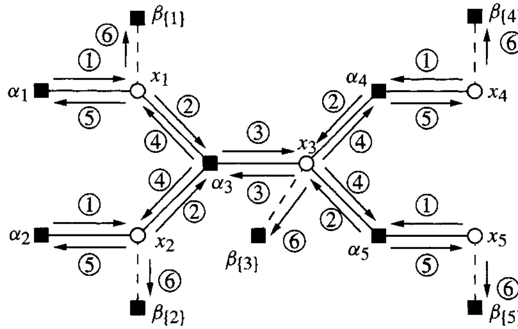


Figure 16.6: Steps of processing in the sum-product algorithm.

Step 1

$$\mu_{\alpha_1\to x_1}(x_1) = \sum_{\sim\{x_1\}} \alpha_1(x_1) \underbrace{\prod_{y\in \mathrm{Nb}(\alpha_1\setminus\{x_1\})} \mu_{y\to\alpha_1}(y)}_{\text{empty product}} = \sum_{\sim\{x_1\}} \alpha_1(x_1) = \alpha_1(x_1)$$

(Cf. rule 3)

$$\mu_{\alpha_2\to x_2}(x_2) = \sum_{\sim\{x_2\}} \alpha_2(x_2) = \alpha_2(x_2)$$

$$\mu_{x_4\to\alpha_4}(x_4) = \underbrace{\prod_{y\in \mathrm{Nb}(x_4)\setminus\{x_4\}} \mu_{y\to x_4}(x_4)}_{\text{empty product}} = 1 \qquad \text{(Cf. rule 1)}$$

$$\mu_{x_5\to\alpha_5}(x_5) = 1$$

Step 2

$$\mu_{x_1\to\alpha_3}(x_1) = \prod_{\gamma\in \mathrm{Nb}(x_1)\backslash\alpha_3} \mu_{\gamma\to x_1}(x_1) = \mu_{\alpha_3\to x_1}(x_1) \quad \text{(Cf. rule 2)}$$

$$\mu_{x_2\to\alpha_3}(x_2) = \prod_{\gamma\in \mathrm{Nb}(x_2)\backslash\alpha_3} \mu_{\gamma\to x_2}(x_2) = \mu_{\alpha_3\to x_2}(x_2)$$

$$\mu_{\alpha_4\to x_3}(x_3) = \sum_{\sim\{x_3\}} \alpha_4(x_3,x_4) \prod_{y\in \mathrm{Nb}(\alpha_4)\backslash\{x_3\}} \mu_{y\to\alpha_3}(y) = \sum_{\sim\{x_3\}} \alpha_4(x_3,x_4)\mu_{x_4\to\alpha_3}(x_4)$$

$$\mu_{\alpha_5\to x_3}(x_3) = \sum_{\sim\{x_3\}} \alpha_5(x_3,x_5) \prod_{y\in \mathrm{Nb}(\alpha_5)\backslash\{x_3\}} \mu_{y\to\alpha_3}(y) = \sum_{\sim\{x_3\}} \alpha_5(x_3,x_5)\mu_{x_5\to\alpha_3}(x_5)$$

Step 3

$$\begin{aligned}\mu_{\alpha_3\to x_3}(x_3) &= \sum_{\sim\{x_3\}} \alpha_3(x_1,x_3) \prod_{y\in \mathrm{Nb}(\alpha_3)\backslash\{x\}} \mu_{y\to\alpha_3}(y)\\ &= \sum_{\sim\{x_3\}} \alpha_3(x_1,x_2,x_3)\mu_{x_2\to\alpha_3}(x_2)\mu_{x_1\to\alpha_3}(x_1)\end{aligned}$$

$$\mu_{x_3\to\alpha_3}(x_3) = \prod_{\gamma\in \mathrm{Nb}(x_3)\backslash\{\alpha_3\}} \mu_{\gamma\to x_3}(x_3) = \mu_{\alpha_4\to x_3}(x_3)\mu_{\alpha_5\to x_3}(x_3)$$

Step 4

$$\begin{aligned}\mu_{\alpha_3\to x_1}(x_1) &= \sum_{\sim\{x_1\}} \alpha_3(x_1,x_2,x_3) \prod_{\gamma\in \mathrm{Nb}(\alpha_3)\backslash\{x_1\}} \mu_{y\to\alpha_3}(y)\\ &= \sum_{\sim\{x_1\}} \alpha_3(x_1,x_2,x_3)\mu_{x_2\to\alpha_3}(x_2)\mu_{x_3\to\alpha_3}(x_3)\end{aligned}$$

$$\begin{aligned}\mu_{\alpha_3\to x_2}(x_2) &= \sum_{\sim\{x_2\}} \alpha_3(x_1,x_2,x_3) \prod_{\gamma\in \mathrm{Nb}(\alpha_3)\backslash\{x_2\}} \mu_{y\to\alpha_3}(y)\\ &= \sum_{\sim\{x_2\}} \alpha_3(x_1,x_2,x_3)\mu_{x_1\to\alpha_3}(x_1)\mu_{x_3\to\alpha_3}(x_3)\end{aligned}$$

$$\mu_{x_3\to\alpha_4}(x_3) = \prod_{\gamma\in \mathrm{Nb}(x_3)\backslash\{\alpha_4\}} \mu_{\gamma\to x_3}(x_3) = \mu_{\alpha_3\to x_3}(x_3)\mu_{\alpha_5\to x_3}(x_3)$$

$$\mu_{x_3\to\alpha_5}(x_3) = \prod_{\gamma\in \mathrm{Nb}(x_3)\backslash\{\alpha_5\}} \mu_{\gamma\to x_3}(x_3) = \mu_{\alpha_3\to x_3}(x_3)\mu_{\alpha_4\to x_3}(x_3)$$

Step 5

$$\begin{aligned}\mu_{x_1\to\alpha_1}(x_1) &= \mu_{\alpha_3\to x_1}(x_1)\\ \mu_{x_2\to\alpha_2}(x_2) &= \mu_{\alpha_3\to x_2}(x_2)\\ \mu_{\alpha_4\to x_4}(x_4) &= \sum_{\sim\{x_4\}} \alpha_4(x_3,x_4)\mu_{x_3\to\alpha_4}(x_3)\\ \mu_{\alpha_5\to x_5}(x_5) &= \sum_{\sim\{x_5\}} \alpha_5(x_3,x_5)\mu_{x_3\to\alpha_5}(x_3)\end{aligned}$$

Step 6 The actual marginal values are computed by thinking of the functions $\beta_{\{i\}}(x_i)$ as nodes in the graph, adjacent only to the nodes for the x_i. Then the marginal functions are computed as the

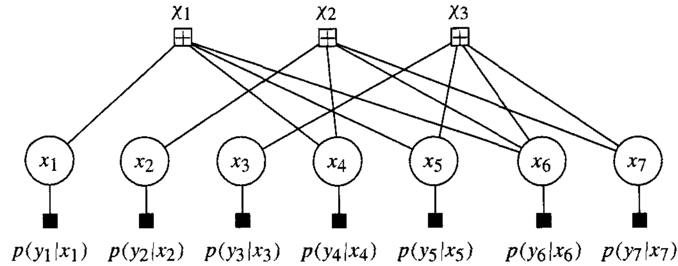


Figure 16.7: The factor (Tanner) graph for a Hamming code.

messages from the variable nodes to these implicitly defined nodes

$$\beta_{\{1\}}(x_1) = \mu_{x_1 \to \beta_{\{1\}}}(x_1) = \prod_{\gamma \in \text{Nb}(x_1)\backslash\{\beta_{\{1\}}\}} \mu_{\gamma \to x_1}(x_1) = \mu_{\alpha_1 \to x_1}(x_1)\mu_{\alpha_3 \to x_1}(x_1)$$

$$\beta_{\{2\}}(x_2) = \mu_{x_2 \to \beta_{\{2\}}}(x_2) = \prod_{\gamma \in \text{Nb}(x_2)\backslash\{\beta_{\{2\}}\}} \mu_{\gamma \to x_2}(x_2) = \mu_{\alpha_2 \to x_2}(x_2)\mu_{\alpha_3 \to x_2}(x_2)$$

$$\beta_{\{3\}}(x_3) = \mu_{x_3 \to \beta_{\{3\}}}(x_3) = \prod_{\gamma \in \text{Nb}(x_3)\backslash\{\beta_{\{3\}}\}} \mu_{\gamma \to x_3}(x_3)$$

$$= \mu_{\alpha_3 \to x_3}(x_3)\mu_{\alpha_4 \to x_3}(x_3)\mu_{\alpha_5 \to x_3}(x_3)$$

$$\beta_{\{4\}}(x_4) = \mu_{x_4 \to \beta_{\{4\}}}(x_4) = \prod_{\gamma \in \text{Nb}(x_4)\backslash\{\beta_{\{4\}}\}} \mu_{\gamma \to x_4}(x_4) = \mu_{\alpha_4 \to x_4}(x_4)$$

$$\beta_{\{5\}}(x_5) = \mu_{x_5 \to \beta_{\{5\}}}(x_5) = \prod_{\gamma \in \text{Nb}(x_5)\backslash\{\beta_{\{5\}}\}} \mu_{\gamma \to x_5}(x_5) = \mu_{\alpha_5 \to x_5}(x_5).$$

□

16.5 Applications to Coding

Let us now consider application of message passing to some coding problems.

16.5.1 Block Codes

Figure 16.7 shows the factor graph for the Hamming code of Example 16.6. This is essentially the Tanner graph, with the local functions representing parity checks indicated by ⊞ and the local functions representing the channel inputs indicated by ■, associated with the functions $p(y_i|x_i)$. For brevity, we refer to these local functions here as the y_i functions. Let us consider a few of the explicit messages passed on this graph and make connections to the LDPC decoder. The messages from the y_i nodes to the x_i nodes are, by rule 1, the likelihoods,

$$\mu_{y_i \to x_i}(x_i) = p(y_i|x_i) \qquad i = 1, 2, \ldots, 7.$$

By rule 2, the messages $\mu_{x_i \to \chi_i}(x_i)$, $i = 1, 2, 3$ simply pass these likelihoods along. More interesting are the messages passed from node x_7, which is involved in multiple parity checks. For example,

$$\mu_{x_7 \to \chi_1}(x_7) = \mu_{y_7 \to x_7}\mu_{\chi_2 \to x_7}\mu_{\chi_3 \to x_7} = p(y_7|x_7)\mu_{\chi_2 \to x_7}\mu_{\chi_3 \to x_7}.$$

This should be compared with (15.12). Structurally they are the same: the message from variable node n to check node m corresponds exactly to the $q_{mn}(x_n)$ of the LDPC decoder.

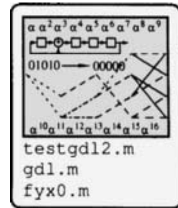
Now consider messages from check nodes to variable nodes. For example,

$$\mu_{\chi_1 \to x_1}(x_1) = \sum_{x_4, x_5, x_6, x_7} \chi_1(x_1, x_4, x_5, x_7) \mu_{x_4 \to \chi_1} \mu_{x_5 \to \chi_1} \mu_{x_5 \to \chi_1}.$$

This should be compared with (15.15). The messages from check node m to variable node n correspond to $r_{mn}(x_n)$ of the LDPC decoder.

Finally, consider the marginal functions obtained from the message from the node x_i to $\beta_i(x_i)$. For example,

$$\mu_{x_7 \to \beta_7}(x_7) = p(y_7|x_7) \mu_{\chi_1 \to x_7} \mu_{\chi_2 \to x_7} \mu_{\chi_3 \to x_7}.$$



This should be compared with (15.9). The pseudoposterior functions $q_n(x_n)$ are the marginal functions computed by the message passing algorithm.

The files indicated provide a Matlab implementation of a fairly general message passing algorithm. It can be verified that the results of these computations are the same as in Example 15.6.

In the general case, the factor graph (i.e., the Tanner graph) has cycles in it. While the message passing paradigm is exact for factor graphs without cycles, the presence of cycles in the graph leads to a bias, which means that the results computed are not exact.

16.5.2 Modifications to Message Passing for Binary Variables

For binary codes, the message passing rules (16.10) and (16.11) can frequently be simplified. In this case, all the variables are Bernoulli variables, so that the messages represent probabilities such as $P(x_i = 0)$ and $P(x_i = 1)$. Furthermore, the functions we consider are only check functions.

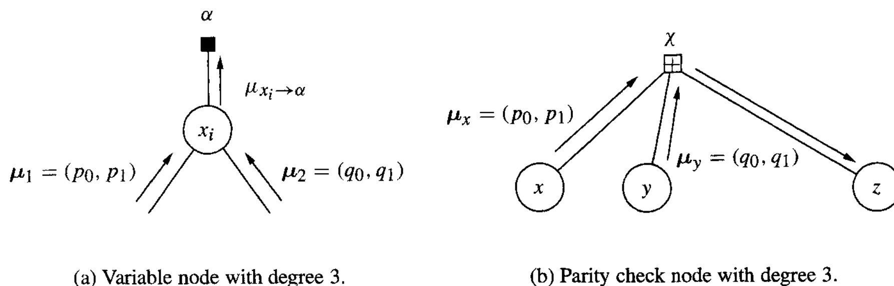
Consider the portion of the graph shown in Figure 16.8(a), with a variable node of degree 3, with incoming message vectors $\boldsymbol{\mu}_1 = (p_0, p_1)$ and $\boldsymbol{\mu}_2 = (q_0, q_1)$. According to (16.10), the *normalized* message vector $\boldsymbol{\mu}_{x_i \to \alpha}$, which we denote here as $\text{var}(p_0, p_1, q_0, q_1)$ (that is, the message from a variable node), is

$$\boldsymbol{\mu}_{x_i \to \alpha} = \text{var}(p_0, p_1, q_0, q_1) = (\mu_{x_i \to \alpha}(0), \mu_{x_i \to \alpha}(1)) = \left(\frac{p_0 q_0}{p_0 q_0 + p_1 q_1}, \frac{p_1 q_1}{p_0 q_0 + p_1 q_1} \right). \tag{16.12}$$

Now consider the parity check node with function $\chi(x, y, z) = [x + y + z = 0]$, where the message from x is represented by the probability vector $\boldsymbol{\mu}_x = (p_0, p_1)$ and the message from y is represented by the probability vector $\boldsymbol{\mu}_y = (q_0, q_1)$. The normalized message vector from χ to z, which we denote as $\text{chk}(p_0, p_1, q_0, q_1)$ is

$$\boldsymbol{\mu}_{\chi \to z} = \text{chk}(p_0, p_1, q_0, q_1) = (\mu_{\chi \to z}(0), \mu_{\chi \to z}(1)) = (p_0 q_0 + p_1 q_1, p_0 q_1 + p_1 q_0). \tag{16.13}$$

Since the messages are probabilities, they can be more efficiently parameterized by a single value, rather than the vector (p_0, p_1). There are three different parameterizations which are convenient to use. We describe them and characterize the messages (16.12) and (16.13) in these parameterizations [195].



(a) Variable node with degree 3.

(b) Parity check node with degree 3.

Figure 16.8: Graph portions to illustrate simplifications.

Likelihood ratio Let $\lambda(p_0, p_1) = p_1/p_0$.

$$\text{var}(\lambda_1, \lambda_2) = \mu_{x_i \to \alpha} = \lambda_1 \lambda_2 \qquad \text{chk}(\lambda_1, \lambda_2) = \mu_{\chi \to z} = \frac{\lambda_1 + \lambda_2}{1 + \lambda_1 \lambda_2}. \tag{16.14}$$

Log likelihood ratio $\Lambda(p_0, p_1) = \log(p_1/p_0)$.

$$\begin{aligned} \text{var}(\Lambda_1, \Lambda_2) &= \mu_{x_i \to \alpha} = \Lambda_1 + \Lambda_2 \\ \text{chk}(\Lambda_1, \Lambda_2) &= \mu_{\chi \to z} = \log\cosh\left(\frac{\Lambda_1 - \Lambda_2}{2}\right) - \log\cosh\left(\frac{\Lambda_1 + \Lambda_2}{2}\right) \\ &= -2\tanh^{-1}\left(\tanh(\Lambda_1/2)\tanh(\Lambda_2/2)\right). \end{aligned} \tag{16.15}$$

This corresponds to the tanh rule.

Likelihood difference $\delta(p_0, p_1) = p_0 - p_1$

$$\text{var}(\delta_1, \delta_2) = \mu_{x_i \to \alpha} = \frac{\delta_1 + \delta_2}{1 + \delta_1 \delta_2} \qquad \text{chk}(\delta_1, \delta_2) = \mu_{\chi \to z} = \delta_1 \delta_2. \tag{16.16}$$

In the log likelihood ratio case, there is a convenient approximation. Since for $|x| \gg 1$,

$$\log(\cosh(x)) \approx |x| - \log(2)$$

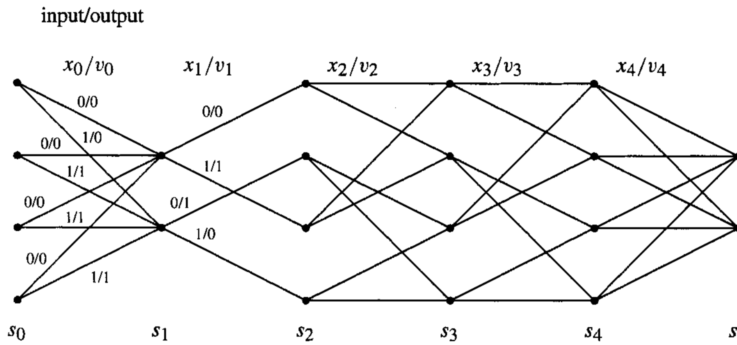
the formula (16.15) can be written

$$\text{chk}(\Lambda_1, \Lambda_2) \approx |(\Lambda_1 - \Lambda_2)/2| - |(\Lambda_1 + \Lambda_2)/2| = -\operatorname{sign}(\Lambda_1)\operatorname{sign}(\Lambda_2)\min(|\Lambda_1|, |\Lambda_2|).$$

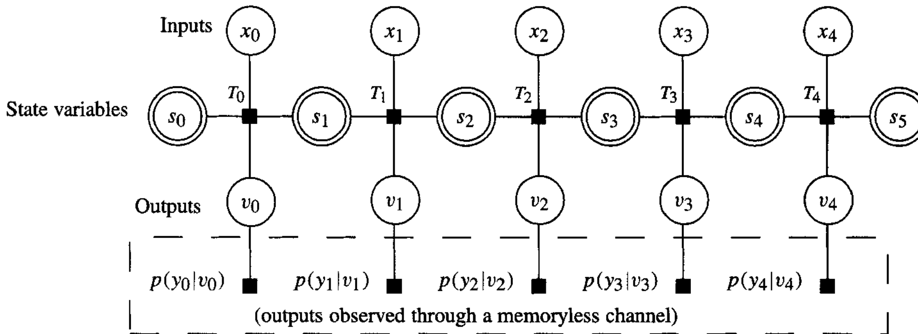
This corresponds to doing operations in the min-sum semiring.

16.5.3 Trellis Processing and the Forward/Backward Algorithm

As we have seen many times, it is frequently convenient to use a trellis description of a code. We describe here how such trellises can be given a factor graph description. Figure 16.9(a) shows a (time-varying, for interest's sake) trellis, where the state at time t is denoted by s_t and the next state s_{t+1} is determined by s_t and an input x_t. The transition from s_t to s_{t+1} produces the output v_t. The states are typically considered hidden. A factor graph corresponding to this trellis is shown in Figure 16.9(b). The state variables, being not



(a) Trellis.



(b) Corresponding factor graph.

Figure 16.9: A trellis and a factor graph representation of it.

directly observable, are indicated with double circles. If the outputs v_i are observed through a memoryless channel with outputs y_i, then an additional set of function nodes is included in the factor graph, as shown.

The local functions $T_i(s_i, x_i, v_i, s_{i+1})$ describe the state transitions. Associated with each function $T_i(x_i, x_i, v_i, s_{i+1})$ is a set T_i describing the behavior, which consists of *allowed* tuples of the form (s_i, x_i, v_i, s_{i+1}). For example, the table here shows T_0 and T_1 for the trellis of Figure 16.9(a), where the states at each time instant are numbered from top to bottom.

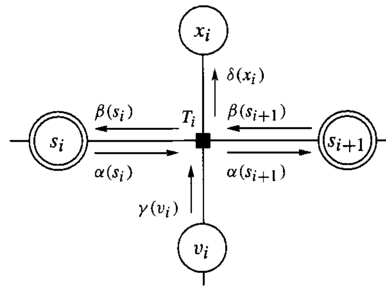


Figure 16.10: Message designation for forward/backward algorithm.

T_0 (s_0, x_0, v_0, s_1)	T_1 (s_1, x_1, v_1, s_2)
(0,0,0,0)	(0,0,0,0)
(0,1,0,1)	(0,1,1,2)
(1,0,0,0)	(1,0,1,1)
(1,1,1,1)	(1,1,0,3)
(2,0,0,0)	
(2,1,1,1)	
(3,0,0,0)	
(3,1,1,1)	

Then the local kernel function is

$$T_i(s_i, x_i, v_i, s_{i+1}) = [(s_i, x_i, v_i, s_{i+1}) \in T_i].$$

For comparison with the forward/backward (BCJR) algorithm, we use the following designation for messages, as shown in Figure 16.10.

Message	Designation
$\mu_{T_i \to s_{i+1}}(s_{i+1})$	$\alpha(s_{i+1})$
$\mu_{v_i \to T_i}(v_i)$	$\gamma(v_i)$
$\mu_{s_{i+1} \to T_i}(s_{i+1})$	$\beta(s_{i+1})$
$\mu_{T_i \to x_i}(x_i)$	$\delta(x_i)$

Let us now consider the message passing algorithm on the factor graph starting at the left end. To begin with, the s_0 and x_0 nodes send the trivial message 1. The v_0 node passes along the message $p(y_0|v_0)$. At this point, the function node T_0 has messages from all but one of its edges and can send out a message.

Then by the message passing rules,

$$\alpha(s_{i+1}) = \sum_{\sim\{s_{i+1}\}} T_i(s_i, x_i, v_i, s_{i+1})\alpha(s_i)\gamma(v_i). \tag{16.17}$$

The sum receives contributions from those edges $e = (s_i, x_i, v_i, s_{i+1})$ such that $T_i(e) = 1$. For each such edge e, we let $\alpha(e) = \alpha(s_i)$ (the initial state of the edge) and $\gamma(e) = \gamma(v_i)$. Let $E_i(s)$ denote the set of edges incident on a state s in the ith section of the trellis. Then the sum in (16.17) can be written as

$$\alpha(s_{i+1}) = \sum_{e \in E_i(s_{i+1})} \alpha(e)\gamma(e).$$

This is equivalent to (14.12) in the BCJR algorithm.

Suppose the trellis proceeds to a state s_N. Starting at the right and working backward, the message passing rules indicate

$$\beta(s_i) = \sum_{\sim\{s_i\}} T_i(s_i, x_i, v_i, s_{i+1})\beta(s_{i+1})\gamma(v_i).$$

This can be written in the form

$$\beta(s_i) = \sum_{e \in E_i(s_i)} \beta(e)\gamma(e),$$

which is equivalent to (14.14) in the BCJR algorithm. Having computed the α and β at a node T_i, we can compute the message

$$\delta(x_i) = \sum_{\sim\{x_i\}} T_i(s_i, x_i, v_i, s_{i+1})\alpha(s_i)\beta(s_{i+1})\gamma(x_i),$$

which is equivalent to (14.9).

Using the message passing rules, it is straightforward to derive posterior probabilities for other variables as well, such as the posterior probability for a state s_i or an output v_i.

If we were to employ the min-sum ring, instead of the conventional addition and multiplication, the algorithm obtained would be the Viterbi algorithm.

16.5.4 Turbo Codes

Figure 16.11 shows the factor graph corresponding to a particular turbo code; see Figure 14.2 for the encoding framework. The form of the graph makes the decoding algorithm clear: the forward/backward message passing algorithm is used on the first encoder. The resulting messages are interleaved and passed in to the other decoder, where the forward/backward message passing is again performed. The graph typically has loops in it, through the interleaver, so the decoding algorithm is not exact.

16.6 Summary of Decoding Algorithms on Graphs

A general, unstructured, parity check matrix describes a Tanner graph, which implies a message-passing decoder. However, unless the check matrix is very sparse there are cycles on the graph which bias the results. Hence, this decoder is appropriate for LDPC codes.

The parity check matrix corresponding to convolutional codes has a Toeplitz structure. With the introduction of *state* variables, the factor graph is again amenable to message passing decoding. This decoder is appropriate for turbo codes.

However, there are still many codes that do not fall into either of these categories. Some important work in this direction, however, appears in [236]. The process of encoding $c(z) = m(z)g(z)$ is represented by a filterbank using the Cook-Toom fast convolution algorithm, which, by some reorganization, is represented as a critically sampled filter bank. Working backward from the filter bank, a parity-checking graph is obtained, which is decomposed into stages suitable for iterative decoding. Iterative message passing decoding is feasible, resulting in a soft-input/soft-output Reed-Solomon decoder.

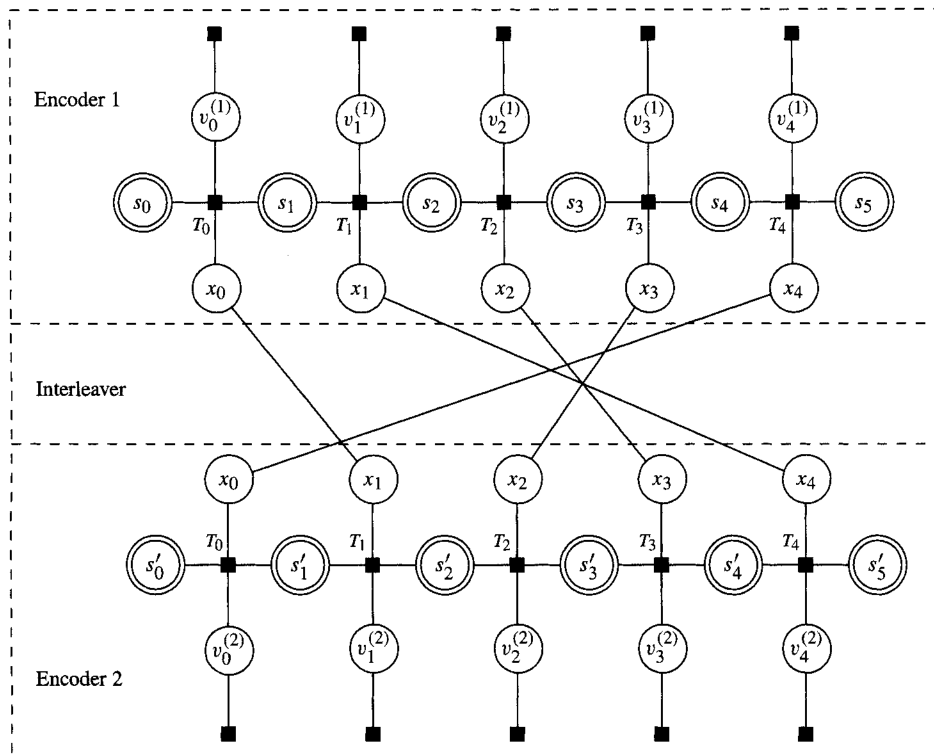


Figure 16.11: The factor graph for a turbo code.

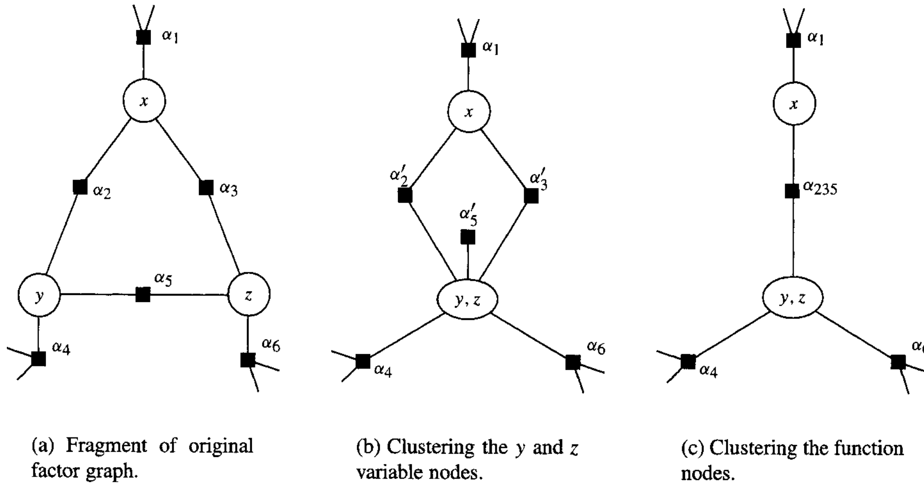
16.7 Transformations of Factor Graphs

The factor graphs presented so far are adequate for decoding purposes. However, there are modifications that can be made to factor graphs that can be used to extend their applicability, for example, by eliminating cycles in the graph or dealing with nodes representing multiple variables. These give the factor graph technique the ability to represent algorithms such as the fast Hadamard transform or the DFT. Two transformations in particular are introduced, namely, clustering and stretching. These transformations are presented in [195], which this discussion closely follows.

16.7.1 Clustering

Clustering is the combining together of two or more nodes into a single node. Either variable nodes or function nodes may be clustered together. To cluster the nodes v and w, delete v and w and any incident edges from the factor graph, introduce a new node representing the clustered pair (v, w), and connect this new node to all the nodes that were neighbors of v or w in the original graph.

Variable nodes. If v and w are **variable** nodes with domains A_v and A_w, respectively, the new variable node (v, w) has domain $A_v \times A_w$, so the size of the message is $|A_v|\,|A_w|$. This multiplication of the domain sizes can result in a significant increase in computational



(a) Fragment of original factor graph.

(b) Clustering the y and z variable nodes.

(c) Clustering the function nodes.

Figure 16.12: Demonstration of clustering transformations [195].

complexity. Any function f in the original factor graph that was a function of v or w is replaced with a function f' that has (v, w) as an argument.

Example 16.11 [195] Figure 16.12(a) shows a fragment of a factor graph with a cycle. The result of clustering the y and z variable nodes is shown in Figure 16.12(b). Here, for example, the relabeled function $\alpha_2'(x, y, z)$ is defined as $\alpha_2'(x, y, z) = \alpha_2(x, y)$. □

Function nodes. When v and w are local kernel function nodes, the pair (v, w) indicates the *product* of the functions, whose domain is the union of the domains of the original functions. The global kernel function for this clustered graph is identical to the global kernel function of the original graph.

Example 16.12 Figure 16.12(c) shows the clustering of three function nodes, resulting in a function node

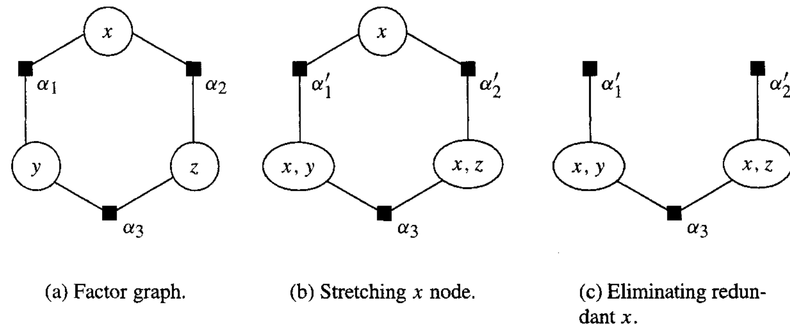
$$\alpha_{235}(x, y, z) = \alpha_2(x, y)\alpha_3(x, z)\alpha_5(y, z).$$

□

As these examples show, clustering variables can be used to eliminate cycles in a graph. If the sum-product algorithm is used on the graph in Figure 16.12(c), then the algorithm would be exact, while if used on the graph of Figure 16.12(a) it would not be exact due to the cycles. However, the computational complexity in the exact case is higher, because there is a larger domain on the merged variable nodes.

16.7.2 Stretching Variable Nodes

The influence of a variable node on the functions adjacent to it can be represented by "stretching" that variable node to the other variable nodes incident to those functions. Let x be a variable node of a graph and let $\mathrm{Nb}_2(x)$ be the set of nodes which can be reached from



(a) Factor graph.

(b) Stretching x node.

(c) Eliminating redundant x.

Figure 16.13: Stretching transformations.

x by a path of length two. Then $\mathrm{Nb}_2(x)$ is the set of variable nodes that are also arguments to the functions that x is. Stretching is accomplished by replacing each node $y \in \mathrm{Nb}_2(x)$ with the node (x, y). Functions incident on these modified nodes are simply modified to reflect the change in arguments.

Once a variable x has been stretched to all variable nodes in $\mathrm{Nb}(x)$, its influence on its adjacent functions is represented by the modified nodes. Thus, the original x node is redundant and can, if desired, be removed from the graph. This is another way of eliminating cycles from factor graphs.

Example 16.13 Figure 16.13(a) shows a segment of a factor graph. In Figure 16.13(b), the x variable node has been stretched into the y and z variable nodes. In Figure 16.13(c), the variable x is redundant and is eliminated, breaking the cycle. □

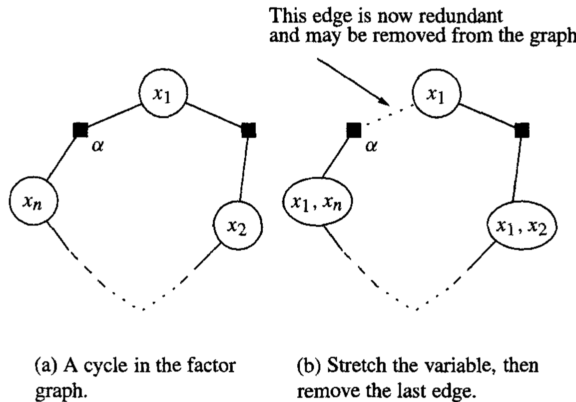
Stretching can be carried out further than to $\mathrm{Nb}_2(x)$. If B is the set of nodes to which x has been stretched, then x can be further stretched to any node in $\mathrm{Nb}_2(B)$. Thus, the set of edges to which x is stretched forms a connected subgraph of the factor graph. In the message-passing algorithm, since x appears in all the variable nodes of the factor graph, it is *not* summarized out. This provides a modeling ability essentially equivalent to that of the junction graphs presented in [2].

When multiple variables are stretched and a variable already exists in the node into which it is stretched, it is only necessary to retain one instance of the variable.

As seen in the last example, stretching can be used as a means of eliminating cycles. If a variable node x_1 is involved in a cycle, then it is first stretched to all the variables in the cycle. Let (α, x_1) denote the last edge in the cycle and let $((x_1, x_n), \alpha)$ be the penultimate edge in the cycle after x_1 is stretched. See Figure 16.14. Since α obtains the variable x_1 from the node (x_1, x_n), the edge (α, x_1) is redundant and may now be eliminated.

16.7.3 Exact Computation of Graphs with Cycles

By the methods outlined above, cycles in graphs can be eliminated and exact computation can result. A more-or-less constructive way to eliminate cycles is to identify spanning trees in the graph, then use the spanning tree to direct how the variables should be stretched. A *spanning tree* T for a connected graph G is a connected *cycle-free* subgraph of G having the same vertex set as G. Algorithms for finding a spanning tree of a graph are well known



(a) A cycle in the factor graph.

(b) Stretch the variable, then remove the last edge.

Figure 16.14: Eliminating an edge in a cycle.

(see, e.g., [305]). The spanning tree is not generally unique. To systematically eliminate cycles in a graph G, identify a spanning tree T, noting the edges in the graph G which are *not* in the spanning tree T. For each such removed edge, stretch the incident variable around a cycle in the graph, so that the removed edge is the last edge in the cycle. Then the edge may be removed as described above.

Example 16.14 To clarify this technique, consider the problem of computing the DFT introduced in Example 16.4, whose factor graph is shown in Figure 16.2. This graph has cycles, so to obtain exact computation a spanning tree must be identified. A particular spanning tree is shown in Figure 16.15(a), where the dashed lines indicate the edges *not* included in the spanning tree. Three edges were removed, so we must consider three cycles.

First select the edge (w, x_0) which was not used in the spanning tree. Stretch the variable x_0, which is incident to that edge, around the cycle

$$x_0 \to c_2 \to y_1 \to b_1 \to x_1 \to b_2 \to y_0 \to a \to x_2 \to w \to x_0,$$

as shown in Figure 16.15(b). Then the edge (w, x_0) is redundant and may be removed.

Now select the edge $(w, (x_0, x_1))$, which was not used in the spanning tree. Stretch the *original* (single) variable node x_1 around the cycle

$$(x_0, x_1) \to b_2 \to (x_0, y_0) \to a \to (x_0, x_2) \to w \to (x_0, x_1),$$

as shown in Figure 16.15(c). The edge $(w, (x_0, x_1))$ may be removed.

Now select the edge $(c_3, (x_0, x_1, y_0))$. Stretch the original variable y_0 around the cycle

$$(x_0, x_1, y_0) \to b_2 \to (x_0, x_1) \to b_1 \to (x_0, y_1) \to c_2 \to x_0 \to c_3 \to (x_0, x_1, y_0),$$

as shown in Figure 16.15(d).

Then a judicious clustering of functions is selected: all of the b_i and all of the c_i functions are clustered into two function nodes, as shown in Figure 16.15(e). The functions are

$$b = b_1(x_1, y_1)b_2(x_1, y_0) = (-1)^{x_1 y_1} j^{-x_1 y_0} = b(x_1, y_0, y_1)$$

$$c = c_1(x_0, y_2)c_2(x_0, y_1)c_3(x_0, y_0) = (-1)^{x_0 y_2} j^{-x_0 y_1} \Omega^{-x_0 y_0} = c(x_0, y_0, y_1, y_2).$$

Finally, the factor graph is extracted; the terminal node is to be a function of (y_0, y_1, y_2), so a special node is created for them, as shown in Figure 16.15(f).

The messages in the transform are passed from left to right.

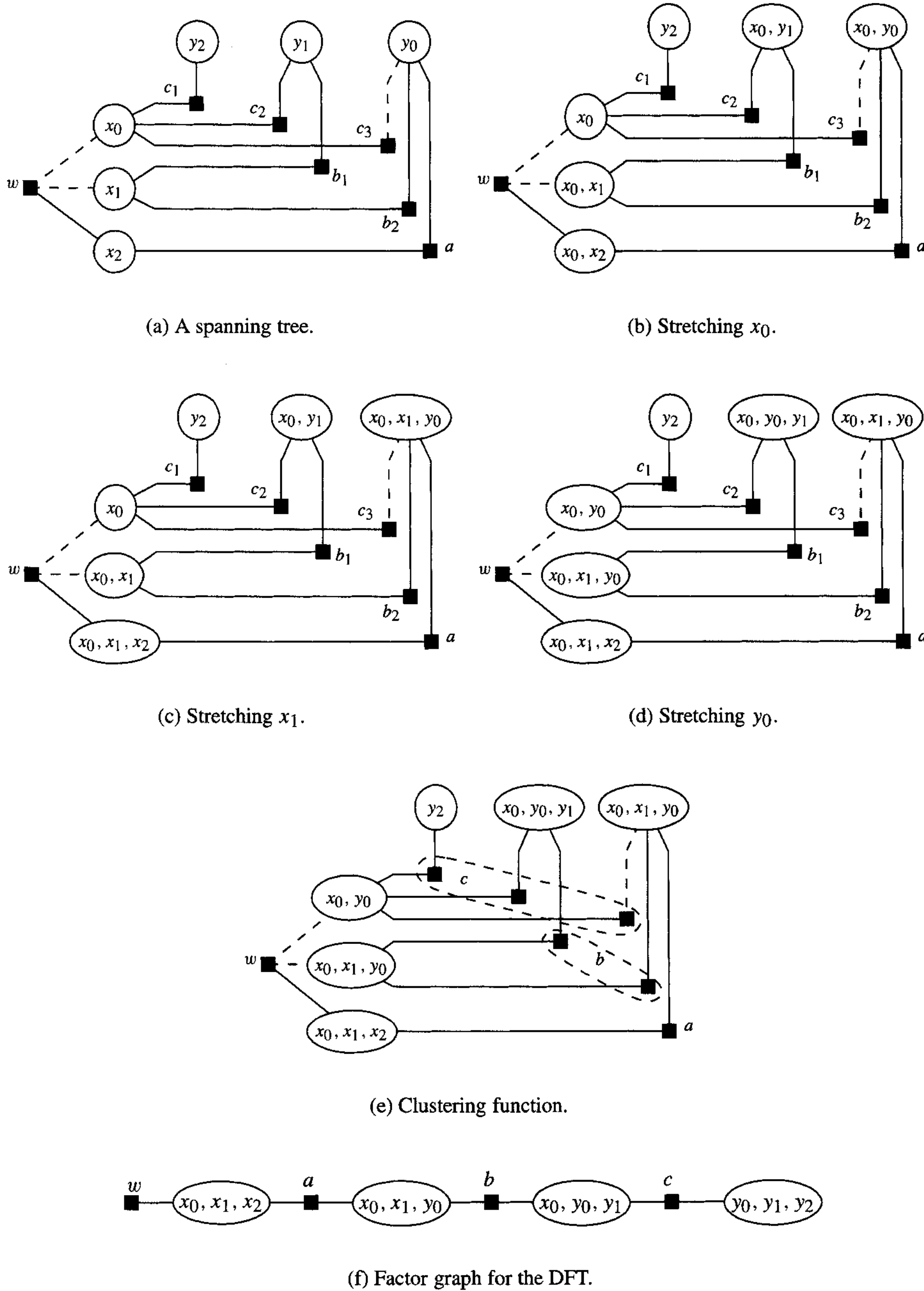
$$\mu_{w\to(x_0,x_1,x_2)} = w_{4x_2+2x_1+x_0}$$
$$\mu_{(x_0,x_1,x_2)\to a} = w_{4x_2+2x_1+x_0}$$
$$\mu_{a\to(x_0,x_1,y_0)} = \sum_{\sim(x_0,x_1,y_0)} a(x_2, y_0) w_{4x_2+2x_1+x_0} = \sum_{x_2} (-1)^{x_2 y_0} w_{4x_2+2x_1+x_0}$$

(The sum is over x_2, because that is the variable in $\sim(x_0, x_1, y_0)$ which appears in the nodes to the left.)

$$\mu_{(x_0,x_1,y_0)\to b} = \mu_{a\to(x_0,x_1,y_0)}$$
$$\mu_{b\to(x_0,y_0,y_1)} = \sum_{\sim(x_0,y_0,y_1)} b(x_1, y_0, y_1)\mu_{(x_0,x_1,y_0)\to b}$$
$$= \sum_{x_1}\sum_{x_2} (-1)^{x_2 y_0}(-1)^{x_1 y_1} j^{-x_1 y_0} w_{4x_2+2x_1+x_0}$$
$$\mu_{(x_0,y_0,y_1)\to c} = \mu_{b\to(x_0,y_0,y_1)}$$
$$\mu_{(c\to(y_0,y_1,y_2)} = \sum_{x_0}\sum_{x_1}\sum_{x_2} (-1)^{x_2 y_0}(-1)^{x_1 y_1} j^{-x_1 y_0}(-1)^{x_0 y_2} j^{-x_0 y_1}\Omega^{-x_0 y_0} w_{4x_2+2x_1+x_0}$$

Since there are generally different spanning trees, different algorithms can be organized. In the DFT case, these might correspond, for example, to decimation-in-time, decimation-in-frequency, or others.

□



(a) A spanning tree.

(b) Stretching x_0.

(c) Stretching x_1.

(d) Stretching y_0.

(e) Clustering function.

(f) Factor graph for the DFT.

Figure 16.15: Transformations on the DFT factor graph.

16.8 Exercises

16.1 Semirings: Show that $\min(a+b, a+c) = a + \min(b, c)$.

16.2 Show that rings 5—8 of Table 16.1 are all isomorphic by identifying the isomorphisms among them.

16.3 The basic sum-product algorithm: Let $f(X, Y)$ have the following table:

X	Y	$f(X,Y)$
0	0	1
0	1	0
1	0	0
1	1	1

Let $p_x = P(X=1)$ and $p_y = P(Y=1)$.

(a) Determine $p_f = P(f(X, Y) = 1)$ in terms of p_x and p_y

(b) Let $l_x = \log \frac{P(X=1)}{P(X=0)}$ and $l_y = \log \frac{P(Y=1)}{P(Y=0)}$. Determine $l_f = \log \frac{P(f(X,Y)=1)}{P(f(X,Y)=0)}$.

16.4 Consider the following computation problem:

$$\beta(x_1, x_2, \ldots, x_9) = \alpha_1(x_1, x_3)\alpha_2(x_1, x_4, x_5)\alpha_3(x_1, x_6, x_7)\alpha_4(x_4, x_8, x_9)$$

and

$$\beta_1(x_1) = \sum_{\sim x_1} \beta(x_1, x_2, \ldots, x_9).$$

Assume that each x_i comes from an alphabet $\mathcal{A}$ with q elements in it.

(a) Determine the number of computations required to compute $\beta_1(x_1)$ if the distributive law is not used.

(b) Draw a tree representation for the computation of $\beta_1(x_1)$.

(c) Determine the messages on each edge of the tree to compute $\beta_1(x_1)$. What is the computational complexity in this case?

(d) Suppose that in addition to $\beta_1(x_1)$, it is desired to compute

$$\beta_7(x_7) = \sum_{\sim x_7} \beta(x_1, x_2, \ldots, x_9).$$

Determine the messages necessary for this and the additional computational complexity for this computation.

16.5 When normalized computations are employed, we have the marginal functions

$$\beta_{\{i\}}(x_i) = \mathsf{N} \prod_{\gamma \in \text{Nb}(x_i)\setminus\{\beta_{\{i\}}\}} \mu_{\gamma \to x_i}(x_i) \propto \prod_{\gamma \in \text{Nb}(x_i)\setminus\{\beta_{\{i\}}\}} \mu_{\gamma \to x_i}(x_i),$$

that is, the product of all messages directed toward x_i. Here N is a normalizing operator. Show that the marginal function can be computed as a function proportional to the product of any two messages that were passed in opposite directions over any single edge incident on x_i. As a specific example of this, the following are equivalent ways of computing $\beta_{\{3\}}(x_3)$ for the factor graph of Figure 16.1(b):

$$\begin{aligned}\beta_{\{3\}}(x_3) &= \mu_{\alpha_3 \to x_3}(x_3)\mu_{x_3 \to \alpha_3}(x_3) = \mu_{\alpha_4 \to x_3}(x_3)\mu_{x_3 \to \alpha_4}(x_3) \\ &= \mu_{\alpha_5 \to x_3}(x_3)\mu_{x_3 \to \alpha_5}(x_3).\end{aligned}$$

16.6 Show that (16.3) can be written as (16.4).

16.7 Probabilistic functions. Let $P(x_1, x_2, x_3, x_4)$ be a joint probability mass function (pmf).

(a) Show the factor graph representing $P(x_1, x_2, x_3, x_4)$.

(b) The pmf can always be factored as

$$P(x_1, x_2, x_3, x_4) = P(x_1)P(x_2|x_1)P(x_3|x_1, x_2)P(x_4|x_1, x_2, x_3).$$

Show the factor graph for this factorization.

(c) If x_1, x_2, x_3, x_4 form a Markov chain, this factorization can be simplified to

$$P(x_1, x_2, x_3, x_4) = P(x_1)P(x_2|x_1)P(x_3|x_2)P(x_4|x_3).$$

Show the factor graph for this factorization.

(d) Now suppose that the x_i are not directly observable, but are observed through a memoryless channel via $p(y_i|x_i)$. Then

$$P(x_1, \ldots, x_4, y_1, \ldots, y_4) = \prod_{i=1}^{4} P(x_i|x_{i-1})p(y_i|x_i).$$

Show the factor graph corresponding to this factorization.

16.8 For the Tanner graph in Figure 16.7, let $\mu_{\chi_1 \to x_5}(x_5)$ be the message from parity check node χ_1 to bit node x_5 and let $\mu_{x_4 \to \chi_1}(x_4)$ be the message from bit node x_4 to parity check χ_1.

(a) Write the message passing updates for $\mu_{\chi_1 \to x_5}(x_5)$ and $\mu_{x_4 \to \chi_1}$.

(b) Write these messages as vectors, such as $\mu_{\chi_1 \to x_5} = \begin{bmatrix} \mu_{\chi_1 \to x_5}(0) \\ \mu_{\chi_1 \to x_5}(1) \end{bmatrix}$. Show that the message passing algorithm can be expressed as a linear relationship

$$\mu_{\chi_1 \to x_5} = L_1 \mu_{x_4 \to \chi_1}.$$

Determine the matrix L_1.

(c) Similarly determine the matrix L_2 in the message passing rule

$$\mu_{x_5 \to \chi_3} = L_2 \mu_{\chi_1 \to x_5}.$$

(d) Determine a matrix L such that

$$\mu_{x_5 \to \chi_3} = L \mu_{x_4 \to \chi_1}.$$

16.9 [2] Let M_i be a $(q_i - 1) \times q_i$ matrix, for $i = 1, 2, \ldots, n$. Let the elements of the matrix M_i be denoted as $M_i[x_{i-1}, x_i]$. We can express the computation of the product $M = M_1 M_2 \cdots M_n$ as a factor graph. When $n = 2$, $M[x_0, x_2] = \sum_{x_1} M_1[x_0, x_1]M_2[x_1, x_2]$.

(a) Show that when $M = M_1 M_2 \cdots M_n$,

$$M[x_0, x_n] = \sum_{x_1}\sum_{x_2} \cdots \sum_{x_{n-1}} M_1[x_0, x_1] \cdots M_n[x_{n-1}, x_n]$$

(b) Determine $n + 1$ local domains and local kernels to describe this problem. *Hint*: At the local domain $n + 1$, the local domain is $\{x_0, x_n\}$.

16.10 Derive the rules for binary message passing for both the var and chk rules in (16.12), (16.13), (16.14), (16.15), and (16.16). *Hint*: For the log likelihood formulation, $\tanh(x/2) = \frac{e^x - 1}{e^x + 1}$ and $\log \frac{1-x}{1+x} = -2 \tanh^{-1} x$.

16.11 Draw the factor graph corresponding to the Hadamard transform of Example 16.3. By clustering, eliminate loops in the graph. Write down the message passing equations for this graph and verify that it computes the Hadamard transform.

16.9 References

Our description of the sum-product algorithm closely follows [195], while the description of semirings comes from [2]. A closer look at the latter, providing constructive ways of identifying simplifying rules, is in [257]. The graphical model idea is well explored in the February 2001 issue of *IEEE Transactions on Information Theory*. The connection between turbo decoding and this family of algorithms seems to have been observed first in [232]. There are actually several different kinds of graphical models employed — including junction trees, Bayesian networks, Markov random fields — which are all more or less isomorphic [195] and to which the algorithms described here can be applied. A generalization of these graphical models is in [235]. Monographs treating graphical models are [374] and [107]. Factor graphs are also treated in the tutorial [209], where the *normal factor graph* or *Forney factor graph* (FFG) is introduced [95]. With the material of this chapter as background, the FFG is a straightforward and useful extension which provides for block-diagram-like system modeling.

For iteratively decoded codes, the sum-product algorithm is inexact due to cycles. Some results on graphs with cycles are examined in [233, 381, 367, 108, 368]. A means of approximately treating the cycles and accelerating the convergence of LDPC decoding is treated in [47].

Part V

Space-Time Coding

Chapter 17

Fading Channels and Space-Time Codes

17.1 Introduction

For most of this book the codes have been designed for transmission through either an AWGN channel or a BSC. One exception is the convolutive channel, for which turbo equalization was introduced in Section 14.7. In this chapter, we introduce a coding technique appropriate for Rayleigh flat fading channels. Fading is a multiplicative change in the amplitude of the received signal. As will be shown in Section 17.2, fading is mitigated by means of *diversity*, that is, multiple, independent transmissions of the signal. Space-time coding provides a way of achieving diversity for multiple transmit antenna systems with low-complexity detection algorithms.

A very important "meta-lesson" from this chapter is that the coding employed in communicating over a channel should match the particular requirements of the channel. In this case, space-time codes are a response to the question: Since diversity is important to communicating over a fading channel, how can coding be used to obtain diversity for portable receivers?

A discussion of the fading channel and its statistical model are presented in Section 17.2. In section 17.3, the importance of diversity in combating fading is presented. Section 17.4 presents space-time codes, which are able to provide diversity with only a single receive antenna and moderate decode complexity. Trellis codes used as space-time codes are presented in Section 17.5.

17.2 Fading Channels

In the channels most frequently used in this book, communication has been from the transmitter directly to the receiver, with only additive noise and attenuation distorting the received signal. In many communication problems, however, the transmitted signal may be subject to multiple reflections. Furthermore, the reflectors and transmitter may be moving with respect to each other, as suggested by Figure 17.1. For example, in an urban environment, a signal transmitted from a cell phone base station may reflect off of buildings, cars, or even trees, so that the signal received at a cell phone may consist only of the superposition of reflected signals. In fact, such impediments are very typical of most wireless channels. The received signal may thus be represented in the (complex baseband) form

$$\begin{aligned} r(t) &= \sum_n \alpha_n(t) e^{-j2\pi f_c \tau_n(t)} s(t - \tau_n(t)) + n(t) \\ &= \sum_n \alpha_n(t) e^{-j\theta_n(t)} s(t - \tau_n(t)), + n(t) \end{aligned} \tag{17.1}$$

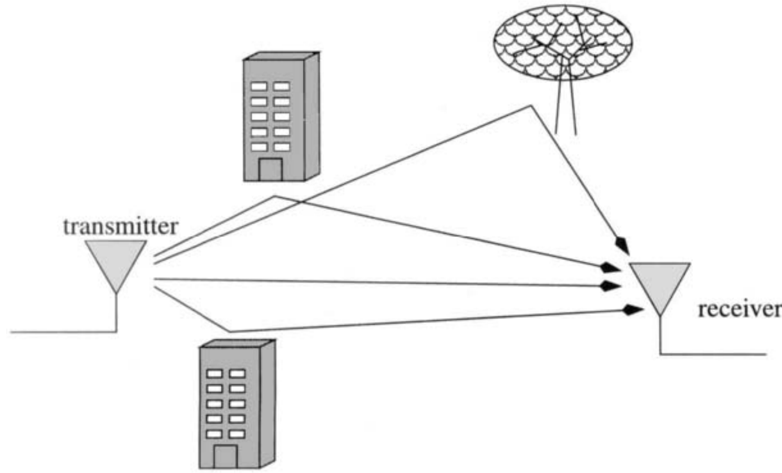


Figure 17.1: Multiple reflections from transmitter to receiver.

where $s(t)$ is the transmitted signal, $\alpha_n(t)$ is the attenuation on the nth path (which may be time-varying), $\tau_n(t)$ is the delay on the nth path, and $e^{-j2\pi f_c \tau_n(t)}$ represents the phase change in the carrier with frequency f_c due to the delay, with phase $\theta_n(t) = 2\pi\tau_n(t) f_c$. $\alpha_n(t)$, $\theta_n(t)$, and $\tau_n(t)$ can be considered as random processes. The noise $n(t)$ is a complex, stationary, zero-mean Gaussian random process with independent real and imaginary parts and $E[n(t)n^*(s)] = N_0\delta(t-s)$. In the limit, if the number of reflectors can be regarded as existing over a continuum (e.g., for signals reflecting off the ionosphere), the received signal can be modeled as

$$r(t) = \int \alpha_s(t) e^{-j\theta_s(t)} s(t - \tau_s(t))\, ds + n(t).$$

Frequently, the delays are similar enough relative to the symbol period that for all practical purposes the delayed signals $s(t-\tau_n(t))$ are the same, so $s_n(t-\tau_n(t)) = s(t-\tau(t))$ for all n. However, even in this case the changes in phase due to delay can be significant. Since f_c is usually rather large (in the megahertz or gigahertz range), small changes in delay can result in large changes in phase.

Example 17.1 A signal transmitted at $f_c = 900$ Mhz is reflected from two surfaces in such a way that at some particular instant of time, one signal to receiver travels 0.16 m farther than the other signal. The time difference is therefore

$$\tau = \frac{0.16}{c} = 5.33 \times 10^{-10} \text{ s}$$

and the phase difference is

$$\theta = 2\pi\tau f_c = 3.0159 \text{ radians} = 172.8^\circ.$$

The change in phase in the carrier results in a factor of $e^{j2\pi 3.0159} \approx -1$ between the two signals — the two received signals will almost cancel out! □

Fading in this case is thus due primarily to changes in the phase $\theta_n(t)$. The randomly varying phase $\theta_n(t)$ associated with the factor $\alpha_n e^{-j\theta_n}$ results in signals that at times add

constructively and at times add destructively. When the signals add destructively, then *fading* occurs.

If all of the delays are approximately equal, say, to a delay τ, then

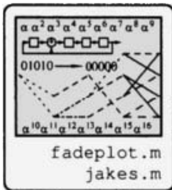
$$r(t) = \sum_n \alpha_n(t)e^{-j\theta_n(t)}s(t-\tau) + n(t) = g(t)s(t-\tau) + n(t), \tag{17.2}$$

where

$$g(t) = \sum_n \alpha_n(t)e^{-j\theta_n(t)} \triangleq g_I(t) + jg_Q(t) \triangleq \alpha(t)e^{-j\phi(t)}$$

is a time-varying complex amplitude factor. The channel transfer function is then $T(t, f) = \alpha(t)e^{-\phi(t)}$, with magnitude response $|T(t, f)| = |g(t)| = \alpha(t)$. Since all frequency components are subjected to the same gain $\alpha(t)$, the channel is said to induce **flat fading**. The channel model for which the space-time codes of this chapter are applicable is flat fading.

The effect of fading on the received signal can be severe. For example, suppose that 90% of the time the signals add constructively, resulting in an SNR at the receiver so that the probability of error is essentially 0, but that 10% of the time the channel introduces a deep fade, so that the probability of error is essentially 0.5. The probability of error averaged over time is then 0.05, much too high for most practical purposes, even though the receiver works perfectly most of the time!



As we shall see, the way to combat fading is through *diversity*, sending multiple copies of the signal in the anticipation that not all of the signals will fade simultaneously. Consider, for example, the plot in Figure 17.2, which shows a simulation of $|g(t)|$ (in dB) for a particular channel for two different realizations of the channel. From the plot, it is clear that both of the signals are not necessarily highly attenuated at the same time. If these represented two different paths from transmitter to receiver, there is hope that at least one of the paths would present a reliable channel. Looked at from another point of view, if at one instant of time one channel is bad, at another instant of time, that channel might be good. These observations lead to various forms of diversity.

In *time diversity*, the transmitter sends the same signal at different times, with sufficient delay between symbols that the transmissions experience independent fading. Time diversity may be accomplished using error control coding in conjunction with interleaving.

A second means of diversity is *frequency diversity*, in which the signal is transmitted using carriers sufficiently separated that the channel over which the signals travel experience independent fading. This can be accomplished using spread spectrum techniques or multiple carriers.

A third means of diversity is *spatial diversity*, in which the signal is transmitted from or received by multiple antennas, whose spatial separation is such that the paths from transmitter antennas to receiver antennas experience independent fading. Spatial diversity has the advantage of good throughput (not requiring multiple transmission for time diversity) and good bandwidth (not requiring broad bandwidth for frequency diversity), at the expense of some additional hardware. Space-time codes are essentially a means of achieving spatial diversity.

17.2.1 Rayleigh Fading

Since the amplitude factor $g(t)$ is the summed effect of many reflectors, it may be regarded (by the central limit theorem) as a complex Gaussian random variable. That is, $g(t) = g_I(t) + jg_Q(t)$ has $g_I(t)$ and $g_Q(t)$ as independent, identically distributed random variables.

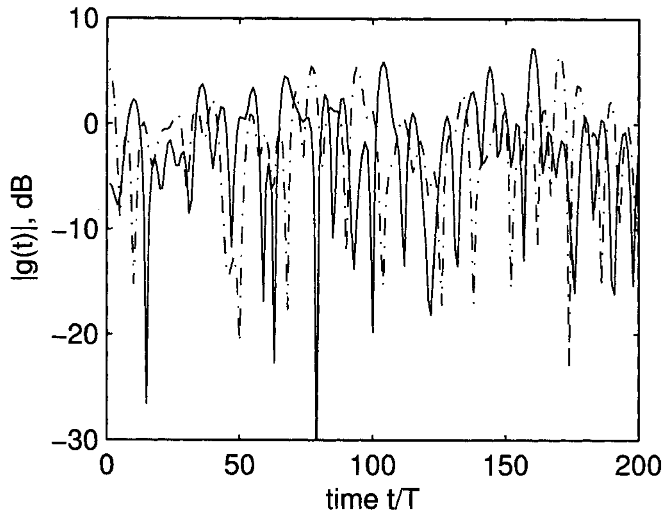


Figure 17.2: Simulation of a fading channel.

If there is no strong direct path signal from transmitter to receiver, then these random variables are modeled as zero-mean random variables, so $g_I(t) \sim \mathcal{N}(0, \sigma_f^2)$ and $g_Q(t) \sim \mathcal{N}(0, \sigma_f^2)$, where σ_f^2 is the fading variance. It can be shown that the magnitude $\alpha = |g(t)|$ is *Rayleigh distributed* (see Exercise 2), so

$$f_\alpha(\alpha) = \frac{\alpha}{\sigma_f^2} e^{-\alpha^2/2\sigma_f^2} \qquad \alpha \geq 0. \tag{17.3}$$

A flat fading channel with magnitude distributed as (17.3) is said to be a **Rayleigh fading channel.**

In the channel model (17.2), it is frequently assumed that τ is known (or can be estimated), so that it can be removed from consideration. On this basis, we write (17.2) as

$$r(t) = g(t)s(t) + n(t).$$

Let this $r(t)$ represent a BPSK-modulated signal transmitted with energy per bit E_b. Suppose furthermore that $g(t) = \alpha(t)e^{-j\phi(t)}$ is such that the magnitude $\alpha(t)$ is essentially constant over at least a few symbols, and that random phase $\phi(t)$ varies slowly enough that it can be estimated with negligible error. This is the *quasistatic* model. Then conventional BPSK detection can be used. This results in a probability of error (see (1.23)) for a particular value of α as

$$P_2(\alpha) = P(\text{bit error}|\alpha) = Q\left(\sqrt{\frac{2\alpha^2 E_b}{N_0}}\right). \tag{17.4}$$

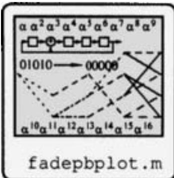
The probability of bit error is then obtained by averaging out the dependence on α:

$$P_2 = \int_0^\infty P_2(\alpha) f_\alpha(\alpha)\, d\alpha.$$

Substituting (17.3) and (17.4) into this integral and integrating by parts (twice) yields

$$P_2 = \frac{1}{2}\left(1 - \sqrt{\frac{\overline{\gamma}_b}{1+\overline{\gamma}_b}}\right), \tag{17.5}$$

where



$$\overline{\gamma}_b = \frac{E_b}{N_0}E[\alpha^2].$$

In Figure 17.3, the line corresponding to $m = 1$ illustrates the performance of the narrowband fading channel, plotted as a function of $\overline{\gamma}_b$ (in dB). Clearly, there is significant degradation compared to BPSK over a channel impaired only by AWGN.

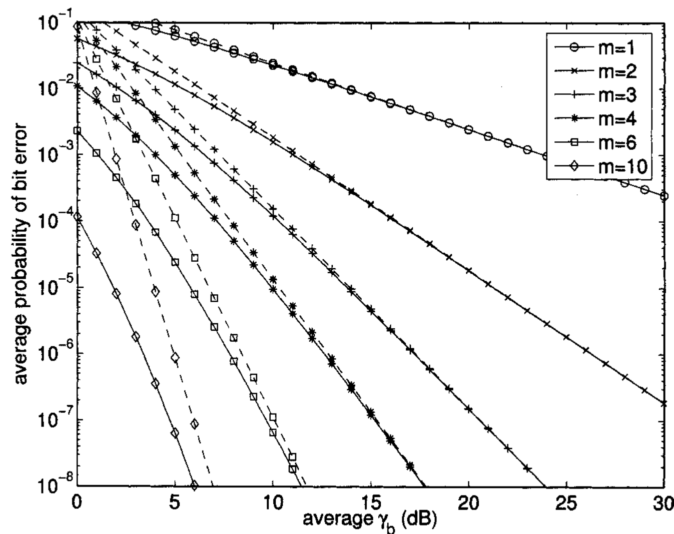


Figure 17.3: Diversity performance of quasi-static, flat-fading channel with BPSK modulation. Solid is exact; dashed is approximation.

17.3 Diversity Transmission and Reception: The MIMO Channel

To provide some background for diversity receivers, let us now consider the somewhat more general problem of transmission through a general linear multiple-input/multiple-output (MIMO) channel. For insight, we first present the continuous-time channel model, then a discrete-time equivalent appropriate for detection (e.g., after filtering and sampling).

Suppose that the signal

$$s_i(t) = \sum_{k=-\infty}^{\infty} a_{ik}\varphi(t - kT)$$

is transmitted from the ith antenna in a system of n antennas, where a_{ik} is the (complex) signal amplitude drawn from a signal constellation for the kth symbol period, and $\varphi(t)$ is the transmitted pulse shape, normalized to unit energy. (See Figure 17.4.) This signal passes

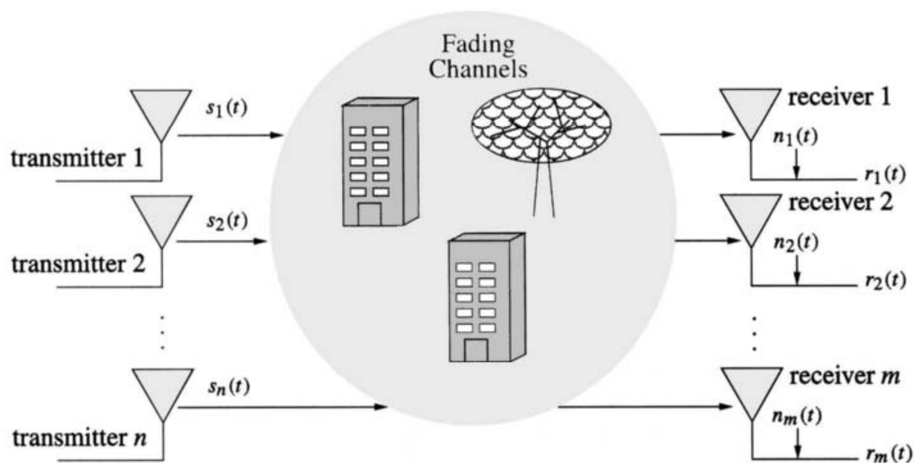


Figure 17.4: Multiple transmit and receive antennas across a fading channel.

through a channel with impulse response $\tilde{h}_{ji}(t)$ and is received by the jth receiver antenna out of m antennas, producing

$$r_j(t) = s_i(t) * \tilde{h}_{ji}(t) + n_j(t) = \sum_{k=-\infty}^{\infty} a_{ik}[\varphi(t - kT) * \tilde{h}_{ji}(t)] + n_j(t).$$

Each $\tilde{h}_{ji}(t)$ may be the response due to scattering and multipath reflections, just as for a single fading channel. The total signal received at the jth receiver due to all transmitted signals is

$$r_j(t) = \sum_{k=-\infty}^{\infty} \sum_{i=1}^{n} a_{ik}[\varphi(t - kT) * \tilde{h}_{ji}(t)] + n_j(t)$$

(assuming that the noise $n_j(t)$ is acquired *at* the receiver, not through the separate channels). Stacking up the vectors as

$$\mathbf{a}(k) = \begin{bmatrix} a_{1,k} \\ a_{2,k} \\ \vdots \\ a_{n,k} \end{bmatrix} \qquad \mathbf{r}(t) = \begin{bmatrix} r_1(t) \\ r_2(t) \\ \vdots \\ r_m(t) \end{bmatrix} \qquad \mathbf{n}(t) = \begin{bmatrix} n_1(t) \\ n_2(t) \\ \vdots \\ n_m(t) \end{bmatrix},$$

we can write

$$\mathbf{r}(t) = \sum_{k=-\infty}^{\infty} \mathbf{H}(t - kT)\mathbf{a}_k + \mathbf{n}(t), \tag{17.6}$$

where $\mathbf{H}(t)$ is the $m \times n$ matrix of impulse responses with

$$h_{ji}(t) = \varphi(t) * \tilde{h}_{ji}(t) = \int_{-\infty}^{\infty} \varphi(\tau)\tilde{h}_{ji}(t - \tau)\, d\tau.$$

If the vector noise process $\mathbf{n}(t)$ is white and Gaussian, with independent components, then the log likelihood function can be maximized by finding the sequence of vector signals

$\mathbf{a} \in \mathcal{S}^n$ minimizing

$$J = \int_{-\infty}^{\infty} \|\mathbf{r}(t) - \sum_{k=-\infty}^{\infty} \mathbf{H}(t-kT)\mathbf{a}_k\|^2\, dt$$
$$= \int_{-\infty}^{\infty} \|\mathbf{r}(t)\|^2 - 2\sum_{k=-\infty}^{\infty} \mathrm{Re}\left[\mathbf{a}_k^* \int_{-\infty}^{\infty} \mathbf{H}^H(t-kT)\mathbf{r}(t)\, dt\right]$$
$$+ \int_{-\infty}^{\infty} \|\sum_k \mathbf{H}(t-kT)\mathbf{a}_k\|^2\, dt.$$

where $\|\mathbf{x}\|^2 = \mathbf{x}^H\mathbf{x}$ and where ^{H} denotes the transpose-conjugate and * denotes complex conjugation. In this general case, the minimization can be accomplished by a maximum-likelihood vector sequence estimator, that is, a Viterbi algorithm. Let us denote

$$\mathbf{r}_k = \int_{-\infty}^{\infty} \mathbf{H}^H(t-kT)\mathbf{r}(t)\, dt \tag{17.7}$$

as the outputs of a *matrix matched filter*, matched to the transmitted signal and channel. Substituting (17.6) into (17.7) we can write

$$\mathbf{r}_k = \sum_{l=-\infty}^{\infty} \mathbf{S}_{k-l}\mathbf{a}_l + \mathbf{n}_k,$$

where

$$\mathbf{S}_k = \int_{-\infty}^{\infty} \mathbf{H}^H(t-kT)\mathbf{H}(t)\, dt$$

and

$$\mathbf{n}_k = \int_{-\infty}^{\infty} \mathbf{H}^H(t-kT)\mathbf{n}(t)\, dt.$$

17.3.1 The Narrowband MIMO Channel

The formulation of the previous section is rather more general than is necessary for our future development. Consider now the narrowband MIMO channel, in which the frequency response is essentially constant for the signals that are transmitted over the channel. In this case, the transmitted waveform $\varphi(t)$ (transmitted by all antennas) is received as

$$\mathbf{H}(t) = \varphi(t)\mathbf{H}.$$

The matched filter $\mathbf{H}^H(-t)$ can be decomposed into multiplication by $\mathbf{H}^H$ followed by conventional matched filtering with $\varphi(-t)$.

Note that the narrowband case can occur in the case of a flat fading channel, where the channel coefficients h_{ji} are randomly time-varying, due, for example, to multiple interfering signals obtained by scattering.

As a specific and pertinent example, suppose that $s(t) = \sum_k a_k\varphi(t-kT)$ is transmitted (i.e., there is only a single transmit antenna) over a channel, and two receive antennas are used, with

$$r_1(t) = h_1 s(t) + n_1(t) \qquad r_2(t) = h_2 s(t) + n_2(t),$$

where h_1 and h_2 are complex and constant over at least one symbol interval. Thus, $\mathbf{H} = \begin{bmatrix} h_1 \\ h_2 \end{bmatrix}$. The receiver first computes

$$\begin{aligned} \mathbf{H}^H \mathbf{r}(t) &= h_1^* h_1 s(t) + h_2^* h_2 s(t) + (h_1^* n_1(t) + n_2^* n_2(t)) \\ &= (|h_1|^2 + |h_2|^2) s(t) + (h_1^* n_1(t) + n_2^* n_2(t)), \end{aligned}$$

then passes this signal through the matched filter $\varphi(-t)$, as shown in Figure 17.5, to produce the signal

$$r_k = (|h_1|^2 + |h_2|^2) a_k + n_k,$$

where n_k is a complex Gaussian random variable with $E[n_k] = 0$ and $E[n_k n_k^*] = (|h_1|^2 + |h_2|^2) N_0$. Thus the maximum likelihood receiver employs the decision rule

$$\hat{a}_k = \arg\min_a |r_k - (|h_1|^2 + |h_2|^2) a|^2.$$

This detector is called a *maximal ratio combiner*, since it can be shown that it maximizes the signal to noise ratio.

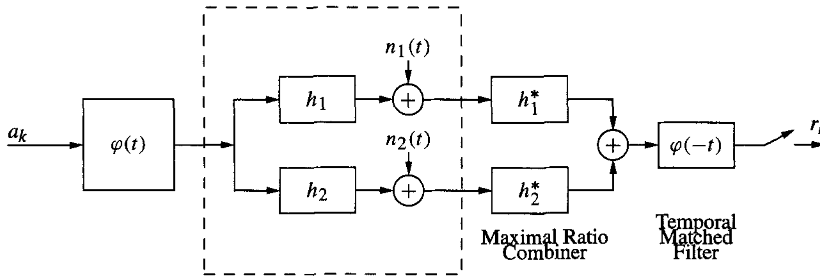


Figure 17.5: Two receive antennas and a maximal ratio combiner receiver.

17.3.2 Diversity Performance with Maximal-Ratio Combining

Let us now consider the performance of the single-transmitter, m-receiver system using maximal ratio combining. Suppose that the signal $a \in \mathcal{S}$ is sent. The matched filter output is

$$r_k = \|\mathbf{h}\|^2 a_k + n_k,$$

where $n_k = \mathbf{h}^H \mathbf{n}$ is a complex Gaussian random variable with $E[|n_k|^2] = N_0 \|\mathbf{h}\|^2 = N_0 \sum_{j=1}^m |h_j|^2$. Let us define an effective signal to noise ratio as

$$\gamma_{\text{eff}} = \|\mathbf{h}\|^2 \frac{E_b}{N_0} = \sum_{j=1}^m |h_j|^2 \frac{E_b}{N_0} \triangleq \sum_{j=1}^m \gamma_j,$$

where $\gamma_j = |h_j|^2 E_b / N_0$ is the effective SNR for the jth channel. We assume a calibration so that the average SNR is $E[\gamma_j] = E_b/N_0$ for $j = 1, 2, \ldots, m$.

Transmitting from a single antenna then recombining the multiple received signals through a maximal ratio combiner results in a single-input, single-output channel. For

BPSK transmission (assuming that the channel varies sufficiently slowly that $\mathbf{h}$ can be adequately estimated) the probability of error as a function of the effective signal to noise ratio is

$$P_2(\gamma_{\text{eff}}) = Q(\sqrt{2\gamma_{\text{eff}}}).$$

As for the case of a single fading channel, the overall probability of error is obtained by averaging over channel coefficients. Assume, as for the single fading case, that each coefficient h_i is a complex Gaussian random variable, so γ_i is a χ^2 distribution with 2 degrees of freedom. If each h_i varies independently (which can be assumed if the receive antennas are at least a half wavelength apart) then γ_{eff} is a χ^2 distribution with $2m$ degrees of freedom. The pdf of such a distribution can be shown to be (see, e.g., [255])

$$f_{\gamma_{\text{eff}}}(t) = \frac{1}{(m-1)!(E_b/N_0)^m} t^{m-1} e^{-tN_0/E_b} \qquad t \geq 0.$$

The overall probability of error is

$$P_2 = \int_0^\infty f_{\gamma_{\text{eff}}}(t) P_2(t)\, dt.$$

The result of this integral (integrating by parts twice) is

$$P_2 = p^m \sum_{k=0}^{m-1} \binom{m-1+k}{k} (1-p)^k,$$

where

$$p = \frac{1}{2}\left(1 - \sqrt{\frac{\gamma_{\text{eff}}}{1+\gamma_{\text{eff}}}}\right)$$

is the probability of error we found in (17.5) for single-channel diversity. Figure 17.3 shows the result for various values of m. It is clear that diversity provides significant performance improvement.

At high SNR, the quantity p can be approximated by

$$p \approx \frac{1}{4\gamma_{\text{eff}}}.$$

For $m > 1$, the probability of error can be approximated by observing that $(1-p)^k \approx 1$, so

$$P_2 \approx p^m \sum_{k=0}^{m-1} \binom{m-1+k}{k} = p^m \binom{2m-1}{k} \triangleq K_m p^m. \tag{17.8}$$

Using the $m = 1$ approximation, we find

$$P_2 \approx \left(\frac{1}{4\gamma_{\text{eff}}}\right)^m \binom{2m-1}{m}. \tag{17.9}$$

While the probability of error in an AWGN channel decreases *exponentially* with the signal to noise ratio, in a fading channel the probability of error only decreases *reciprocally* with the signal to noise ratio, with an exponent equal to the diversity m. We say that this scheme has diversity order m.

17.4 Space-Time Block Codes

We have seen that performance in a fading channel can be improved by receiver diversity. However, in many systems the receiver is required to be small and portable — such as a cell phone or personal digital assistant — and it may not be practical to deploy multiple receive antennas. The transmitter at a base station, however, may easily accommodate multiple antennas. It is of interest, therefore, to develop means of diversity which employ multiple transmit antennas instead of multiple receive antennas. This is what space-time codes provide.

17.4.1 The Alamouti Code

To introduce space-time codes, we present the Alamouti code, an early space-time code and still one of the most commonly used. Consider the transmit diversity scheme of Figure

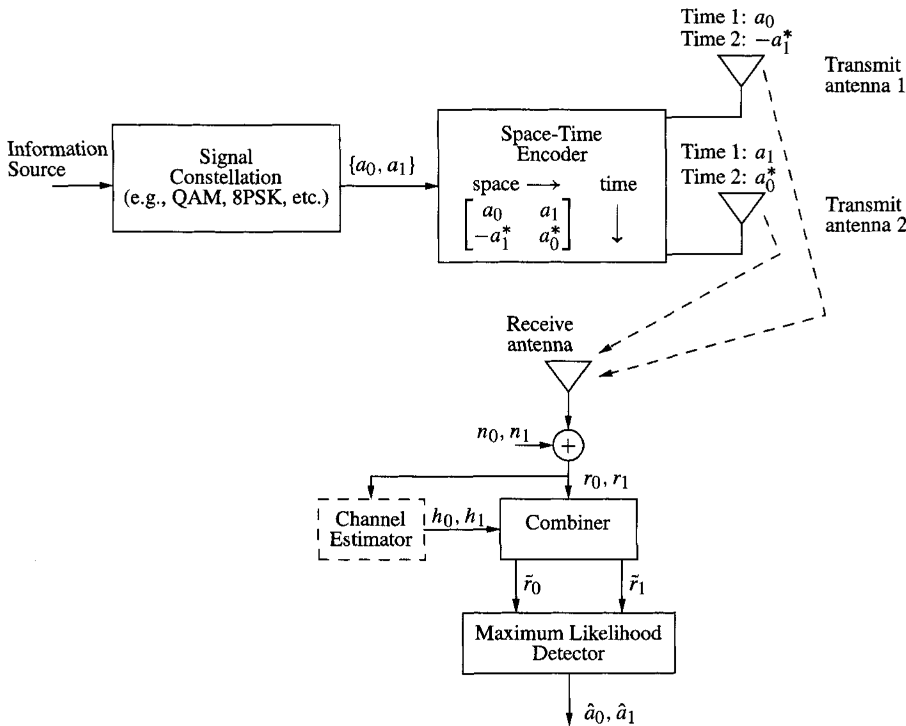


Figure 17.6: A two-transmit antenna diversity scheme: the Alamouti code.

17.6. Each antenna sends *sequences* of data from a signal constellation $\mathcal{S}$. In the Alamouti code, a frame of data lasts for two symbol periods. In the first symbol time, antenna 1 sends the symbol $a_0 \in \mathcal{S}$ while antenna 2 sends the symbol $a_1 \in \mathcal{S}$. In the second symbol time, antenna 1 sends the symbol $-a_1^*$ while antenna 2 sends the symbol a_0^*. It is assumed that the fading introduced by the channel varies sufficiently slowly that it is *constant* over two symbol times (e.g., the quasistatic assumption applies for the entire duration of the codeword). The channel from antenna 1 to the receiver is modeled as $h_0 = \alpha_0 e^{-j\phi_0}$ and the channel from antenna 2 to the receiver is modeled as $h_1 = \alpha_1 e^{-j\phi_1}$. The received signal

for the first signal (i.e., the sampled matched filter output) is

$$r_0 = h_0 a_0 + h_1 a_1 + n_0 \tag{17.10}$$

and for the second signal is

$$r_1 = -h_0 a_1^* + h_1 a_0^* + n_1. \tag{17.11}$$

The receiver now employs a *combining scheme*, computing

$$\begin{aligned} \tilde{r}_0 &= h_0^* r_0 + h_1 r_1^* \\ \tilde{r}_1 &= h_1^* r_0 - h_0 r_1^*. \end{aligned} \tag{17.12}$$

Substituting (17.10) and (17.11) into (17.12), we have

$$\begin{aligned} \tilde{r}_0 &= (|h_0|^2 + |h_1|^2) a_0 + h_0^* n_0 + h_1 n_1^* \\ \tilde{r}_1 &= (|h_0|^2 + |h_1|^2) a_1 - h_0^* n_1 + h_1^* n_0. \end{aligned}$$

The key observation is that $\tilde{r}_0$ *depends only on* a_0, so that detection can take place with respect to this single quantity. Similarly, $\tilde{r}_1$ *depends only on* a_1, again implying a single detection problem.

The receiver now employs the maximum likelihood decision rule on each signal separately

$$\begin{aligned} \hat{a}_0 &= \arg\min_{a \in \mathcal{S}} |\tilde{r}_0 - (|h_0|^2 + |h_1|^2) a|^2 \\ \hat{a}_1 &= \arg\min_{a \in \mathcal{S}} |\tilde{r}_1 - (|h_0|^2 + |h_1|^2) a|^2. \end{aligned}$$

Overall the scheme is capable of sending two symbols over two symbol periods, so this represents a rate 1 code. However, it also provides a diversity of 2. Assuming that the total transmitter power with two antennas in this coded scheme is equal to the total transmitted power of a conventional receiver diversity method, the transmit power must be split into two for each antenna. This power split results in a 3 dB performance reduction compared to $m = 2$ using two receive antennas, but otherwise equivalent performance.

Suppose that, unlike this Alamouti scheme, the combining scheme produced values which are a mixture of the transmitted signals. For example, suppose

$$\begin{aligned} \tilde{r}_0 &= a a_0 + b a_1 + \tilde{n}_0 \\ \tilde{r}_1 &= c a_0 + d a_1 + \tilde{n}_1 \end{aligned}$$

for some coefficients a, b, c, d. We could write this as

$$\tilde{\mathbf{r}} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \end{bmatrix} + \begin{bmatrix} \tilde{n}_0 \\ \tilde{n}_1 \end{bmatrix} \triangleq A \begin{bmatrix} a_0 \\ a_1 \end{bmatrix} + \begin{bmatrix} \tilde{n}_0 \\ \tilde{n}_1 \end{bmatrix}.$$

Then the maximum likelihood decision rule must maximize *jointly*:

$$[\hat{a}_0, \hat{a}_1] = \arg\min_{\mathbf{a} \in \mathcal{S}^2} \|\tilde{\mathbf{r}} - A\mathbf{a}\|^2.$$

The search is over the vector of length 2, so that if the constellation has M points in it, then the search complexity is $O(M^2)$. In the case of m-fold diversity the complexity rises as $O(M^m)$. This increase of complexity is avoided in the Alamouti code case because of the orthogonality of the encoding matrix.

It is important for computational simplicity that a symbol appears in only one combined received waveform. The remainder of the development of space-time block codes in this

chapter is restricted to considerations of how to achieve this kind of separation, using orthogonal designs.

The Alamouti code has been adopted by the IEEE 802.11a and IEEE 802.16a wireless standards.

17.4.2 A More General Formulation

Let us now establish a more general framework for space-time codes, in which the Alamouti scheme is a special case. In the interest of generality, we allow for both transmit and receive diversity, with n transmit antennas and m receive antennas. The code frames exist for l symbol periods. (Thus, for the Alamouti scheme, $n = 2$, $m = 1$, and $l = 2$.) At time t, the symbols $c_i(t), i = 1, 2, \ldots, n$ are transmitted simultaneously from the n transmit antennas. The signal $r_j(t)$ at antenna j is

$$r_j(t) = \sum_{i=1}^{n} h_{j,i} c_i(t) + n_j(t), \qquad j = 1, 2, \ldots, m,$$

where the $c_i(t)$ is the coded symbol transmitted by antenna i at time t. The codeword for this frame is thus the sequence

$$\mathbf{c} = c_1(1), c_2(1), \ldots, c_n(1), c_1(2), c_2(2), \ldots, c_n(2), \ldots, c_1(l), c_2(l), \ldots, c_n(l) \quad (17.13)$$

of length nl.

17.4.3 Performance Calculation

Before considering how to design the codewords, let us first establish the diversity order for the coding scheme. Consider the probability that a maximum-likelihood decoder decides in favor of a signal

$$\mathbf{e} = e_1(1), e_2(1), \ldots, e_n(1), e_1(2), e_2(2), \ldots, e_n(2), \ldots, e_1(l), e_2(l), \ldots, e_n(l)$$

over the signal $\mathbf{c}$ of (17.13) which was transmitted, for a given channel state. We denote this probability as

$$P(\mathbf{c} \rightarrow \mathbf{e} | h_{j,i}, j = 1, 2, \ldots, m, i = 1, 2, \ldots, n).$$

Over the AWGN channel, this probability can be bounded (using a bound on the Q function) by

$$P(\mathbf{c} \rightarrow \mathbf{e} | h_{j,i}, j = 1, 2, \ldots, m, i = 1, 2, \ldots, n) \leq \exp(-d^2(\mathbf{c}, \mathbf{e}) E_s / 4N_0),$$

where

$$d^2(\mathbf{c}, \mathbf{e}) = \sum_{j=1}^{m} \sum_{t=1}^{l} \left| \sum_{i=1}^{n} h_{j,i}(c_i(t) - e_i(t)) \right|^2.$$

Let $\mathbf{h}_j^T = [h_{j,1}, h_{j,2}, \ldots, h_{j,n}]^T$ and $\mathbf{c}_t^T = [c_1(t), c_2(t), \ldots, c_n(t)]^T$ and similarly $\mathbf{e}_t$. Then

$$d^2(\mathbf{c}, \mathbf{e}) = \sum_{j=1}^{m} \mathbf{h}_j^T \left[\sum_{t=1}^{l} (\mathbf{c}_t - \mathbf{e}_t)(\mathbf{c}_t - \mathbf{e}_t)^H \right] \mathbf{h}_j^*.$$

Let

$$A(\mathbf{c}, \mathbf{e}) = \sum_{t=1}^{l} (\mathbf{c}_t - \mathbf{e}_t)(\mathbf{c}_t - \mathbf{e}_t)^H.$$

Then

$$d^2(\mathbf{c}, \mathbf{e}) = \sum_{j=1}^{m} \mathbf{h}_j^T A(\mathbf{c}, \mathbf{e}) \mathbf{h}_j^*,$$

so that

$$P(\mathbf{c} \to \mathbf{e} | h_{j,i}, j = 1, 2, \ldots, m, i = 1, 2, \ldots, n) \leq \prod_{j=1}^{m} \exp(-\mathbf{h}_j^T A(\mathbf{c}, \mathbf{e}) \mathbf{h}_j^* E_s / 4N_0).$$

The matrix $A(\mathbf{c}, \mathbf{e})$ can be written as

$$A(\mathbf{c}, \mathbf{e}) = B(\mathbf{c}, \mathbf{e}) B^H(\mathbf{c}, \mathbf{e}),$$

where

$$B(\mathbf{c}, \mathbf{e}) = \begin{bmatrix} e_1(1) - c_1(1) & e_1(2) - c_1(2) & \cdots & e_1(l) - c_1(l) \\ e_2(1) - c_2(1) & e_2(2) - c_2(2) & \cdots & e_2(l) - c_2(l) \\ \vdots & & & \\ e_n(1) - c_n(1) & e_n(2) - c_n(2) & \cdots & e_n(l) - c_n(l) \end{bmatrix}. \tag{17.14}$$

In other words, $A(\mathbf{c}, \mathbf{e})$ has $B(\mathbf{c}, \mathbf{e})$ as a square root. It is known (see [153]) that any matrix having a square root is nonnegative definite; among other implications of this, all of its eigenvalues are nonnegative.

The symmetric matrix $A(\mathbf{c}, \mathbf{e})$ can be written as (see, e.g., [246])

$$V A(\mathbf{c}, \mathbf{e}) V^H = D,$$

where V is a unitary matrix formed from the eigenvectors of $A(\mathbf{c}, \mathbf{e})$ and D is diagonal with real nonnegative diagonal elements λ_i. Let $\boldsymbol{\beta}_j = V \mathbf{h}_j^*$. Then

$$\begin{aligned} P(\mathbf{c} \to \mathbf{e} | h_{j,i}, j = 1, 2, \ldots, m, i = 1, 2, \ldots, n) &\leq \prod_{j=1}^{m} \exp\big(-\boldsymbol{\beta}_j^H D \boldsymbol{\beta}_j E_s / 4N_0\big) \\ &= \prod_{j=1}^{m} \exp\Big(-\sum_{i=1}^{n} \lambda_i |\beta_{i,j}|^2 4 E_s / 4N_0\Big). \end{aligned}$$

Assuming the elements of $\mathbf{h}_i$ are zero mean Gaussian and normalized to have variance 0.5 per dimension, then the $|\beta_{i,j}|$ are Rayleigh distributed with density

$$p(|\beta_{i,j}|) = 2|\beta_{i,j}| \exp(-|\beta_{i,j}|^2).$$

The average performance is obtained by integrating

$$P(\mathbf{c} \to \mathbf{e}) \leq \int \prod_{j=1}^{m} \exp(-\sum_{i=1}^{n} \lambda_i |\beta_{i,j}|^2 E_s / 4N_0) \prod_{i,j} 2|\beta_{i,j}| \exp(-|\beta_{i,j}|^2) d|\beta_{1,1}| \cdots d|\beta_{n,m}|.$$

After some effort, this can be shown to be

$$P(\mathbf{c} \to \mathbf{e}) \leq \left(\frac{1}{\prod_{i=1}^{n} (1 + \lambda_i E_s / 4N_0)} \right)^m. \tag{17.15}$$

Let r be the rank of $A(\mathbf{c}, \mathbf{e})$, so there are $n - r$ eigenvalues of $A(\mathbf{c}, \mathbf{e})$ equal to 0. Then (17.15) can be further approximated as

$$P(\mathbf{c} \rightarrow \mathbf{e}) \leq \left(\prod_{i=1}^{r} \lambda_i\right)^{-m} (E_s/4N_0)^{-rm}.$$

Comparing with (17.9), we see that the probability of error decreases reciprocally with the signal-to-noise ratio to the rmth power. We have thus proved the following theorem:

Theorem 17.1 *The order of the diversity for this coding is rm.*

From this theorem we obtain the **rank criterion**: To obtain the maximum diversity mn, the matrix $B(\mathbf{c}, \mathbf{e})$ must of full rank for any pair of codewords $\mathbf{c}$ and $\mathbf{e}$.

The factor $\left(\prod_{i=1}^{r} \lambda_i\right)^{-m}$ in (17.15) is interpreted as the coding advantage. In combination with the diversity from the other factor, we obtain the following two design criteria for Rayleigh space-time codes [332].

- In order to achieve maximum diversity, $B(\mathbf{c}, \mathbf{e})$ of (17.14) must be full rank for any pair of codewords $\mathbf{c}$ and $\mathbf{e}$. The smallest r over any pair of codewords leads to a diversity of rm.
- The coding benefit is maximized by maximizing the sum of the determinants of all $r \times r$ principle cofactors of $A(\mathbf{c}, \mathbf{e}) = B(\mathbf{c}, \mathbf{e})B(\mathbf{c}, \mathbf{e})^H$, since this sum is equal to the product of the determinants of the cofactors.

Real Orthogonal Designs

Let us now turn attention to the problem of designing the transmitted codewords. Recall that to minimize decoder complexity, it is desirable to be able to decompose the decision problem so that optimal decisions can be made on the basis of a single symbol at a time, as was possible for the Alamouti code. This can be achieved using *orthogonal designs*. For the moment, we consider only real orthogonal designs which are associated with real signal constellations.

Definition 17.1 A **real orthogonal design** of size n is an $n \times n$ matrix $\mathcal{O} = \mathcal{O}(x_1, x_2, \ldots, x_n)$ with elements drawn from $\pm x_1, \pm x_2, \ldots, \pm x_n$ such that

$$\mathcal{O}^T \mathcal{O} = I \sum_{i=1}^{n} x_i^2 \triangleq IK.$$

That is, $\mathcal{O}$ is proportional to an orthogonal matrix. □

By means of column permutations and sign changes, it is possible to arrange $\mathcal{O}$ so that the first row has all positive signs. Examples of orthogonal designs are

$$\mathcal{O}_2 = \begin{bmatrix} x_1 & x_2 \\ -x_2 & x_1 \end{bmatrix} \qquad \mathcal{O}_4 = \begin{bmatrix} x_1 & x_2 & x_3 & x_4 \\ -x_2 & x_1 & -x_4 & x_3 \\ -x_3 & x_4 & x_1 & -x_2 \\ -x_4 & -x_3 & x_2 & x_1 \end{bmatrix}$$

$$\mathcal{O}_8 = \begin{bmatrix} x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 & x_8 \\ -x_2 & x_1 & x_4 & -x_3 & x_6 & -x_5 & -x_8 & x_7 \\ -x_3 & -x_4 & x_1 & x_2 & x_7 & x_8 & -x_5 & -x_6 \\ -x_4 & x_3 & -x_2 & x_1 & x_8 & -x_7 & x_6 & -x_5 \\ -x_5 & -x_6 & -x_7 & -x_8 & x_1 & x_2 & x_3 & x_4 \\ -x_6 & x_5 & -x_8 & x_7 & -x_2 & x_1 & -x_4 & x_3 \\ -x_7 & x_8 & x_5 & -x_6 & -x_3 & x_4 & x_1 & -x_2 \\ -x_8 & -x_7 & x_6 & x_5 & -x_4 & -x_3 & x_2 & x_1 \end{bmatrix}.$$

In fact, it is known that these are the *only* orthogonal designs [117]. Each row of an orthogonal design $\mathcal{O}$ is a permutation with sign changes of the $\{x_1, x_2, \ldots, x_n\}$. Let us denote the (i, j)th element of $\mathcal{O}$ as $o_{ij} = x_{\epsilon_i(j)}\delta_i(j)$, where $\epsilon_i(j)$ is a permutation function for the ith row and $\delta_i(j)$ is the sign of the (i, j) entry. Observe that permutations in the orthogonal matrices above are symmetric so that $\epsilon_i(j) = \epsilon_i^{-1}(j)$. By the orthogonality of $\mathcal{O}$,

$$[\mathcal{O}^T\mathcal{O}]_{i,k} = \sum_{l=1}^{n} o_{li}o_{lk} = \begin{cases} \sum_{i=1}^{n} x_i^2 & i = k \\ 0 & \text{otherwise} \end{cases} \triangleq K\delta_{i-k}$$

or

$$\sum_{l=1}^{n} x_{\epsilon_l(i)}x_{\epsilon_l(k)}\delta_l(i)\delta_l(j) = K\delta_{i-k}. \tag{17.16}$$

Encoding and Decoding Based on Orthogonal Designs

At the encoder, the symbols $a_1, a_2, \ldots, a_n$ are selected from the (for the moment real) signal constellation $\mathcal{S}$ and used to fill out the $n \times n$ orthogonal design $\mathcal{O}(a_1, a_2, \ldots, a_n)$. At time slot $t = 1, 2, \ldots, n$, the elements of the tth row of the orthogonal matrix are simultaneously transmitted using n transmit antennas. The frame length of the code is $l = n$. At time t, the jth antenna receives

$$\begin{aligned} r_j(t) &= \sum_{i=1}^{n} h_{j,i}c_i(t) + n_j(t) \\ &= \sum_{i=1}^{n} h_{j,i}\delta_t(i)a_{\epsilon_t(i)} + n_j(t). \end{aligned}$$

Now let $l = \epsilon_t(i)$, or $i = \epsilon_t^{-1}(l)$. The received signal can be written as

$$r_j(t) = \sum_{l=1}^{n} h_{j,\epsilon_t^{-1}(l)}\delta_t(\epsilon_t^{-1}(l))a_l + n_j(t).$$

Let $\mathbf{h}_{j,t}^T = [h_{j,\epsilon_t^{-1}(1)}\delta_t(\epsilon_t^{-1}(1)), \ldots, h_{j,\epsilon_t^{-1}(n)}\delta_t(\epsilon_t^{-1}(n))]$ and $\mathbf{a}^T = [a_1, \ldots, a_n]$. Then

$$r_j(t) = \mathbf{h}_{j,t}^T\mathbf{a} + n_j(t).$$

Stacking the received signals in time, the jth receive antenna receives

$$\mathbf{r}_j = \begin{bmatrix} r_j(1) \\ \vdots \\ r_j(n) \end{bmatrix} = \begin{bmatrix} \mathbf{h}_{j,1}^T \\ \vdots \\ \mathbf{h}_{j,n}^T \end{bmatrix} \mathbf{a} + \begin{bmatrix} n_j(1) \\ \vdots \\ n_j(n) \end{bmatrix} \triangleq H_{j,\text{eff}}\mathbf{a} + \mathbf{n}_j. \tag{17.17}$$

The maximum likelihood receiver computes

$$J(\mathbf{a}) = \sum_{j=1}^{m} \|\mathbf{r}_j - H_{j,\text{eff}}\mathbf{a}\|^2$$

over all vectors $\mathbf{a}$ and selects the codeword with the minimizing value. However, rather than having to search jointly over all $|S|^n$ vectors $\mathbf{a}$, each component can be selected independently, as we now show. We can write

$$J(\mathbf{a}) = \sum_{j=1}^{m} \|\mathbf{r}_j\|^2 - 2\,\text{Re}\left[\mathbf{a}^H \sum_{j=1}^{m} H_{j,\text{eff}}^H \mathbf{r}\right] + \mathbf{a}^H \left(\sum_{j=1}^{m} H_{j,\text{eff}}^H H_{j,\text{eff}}\right)\mathbf{a}.$$

Using (17.16), we see that each $H_{j,\text{eff}}$ is an orthogonal matrix $\mathcal{O}(h_{j,1}, h_{j,2}, \ldots, h_{j,n})$, so that

$$\sum_{j=1}^{m} H_{r,\text{eff}}^H H_{r,\text{eff}} \triangleq I \sum_{j=1}^{m} K_j \triangleq IK$$

for some scalar K. Let

$$\mathbf{v} = \sum_{j=1}^{m} H_{j,\text{eff}}^H \mathbf{r}.$$

Then minimizing $J(\mathbf{a})$ is equivalent to minimizing

$$J'(\mathbf{a}) = -2\,\text{Re}[\mathbf{a}^H \mathbf{v}] + \|\mathbf{a}\|^2 K.$$

Now let $S_i = -2\,\text{Re}\, a_i^* v_i + K|a_i|^2$. Minimizing $J'(\mathbf{a})$ is equivalent to minimizing $\sum_{i=1}^n S_i$, which amounts to minimizing each S_i separately. Since each S_i depends only on a_i, we have

$$\hat{a}_i = \arg\min_a K|a|^2 - 2\,\text{Re}\, a^* v_i.$$

Thus using orthogonal designs allows for using only scalar detectors instead of vector detectors.

Let us now examine the diversity order of these space-time block codes.

Theorem 17.2 *[331] The diversity order for coding with real orthogonal designs is nm.*

Proof The matrix $B(\mathbf{c}, \mathbf{e})$ of Theorem 17.1 is formed by

$$B(\mathbf{c}, \mathbf{e}) = \mathcal{O}(\mathbf{c}) - \mathcal{O}(\mathbf{e}) = \mathcal{O}(\mathbf{c} - \mathbf{e}).$$

Since $\det(\mathcal{O}^T(\mathbf{c} - \mathbf{e})\mathcal{O}(\mathbf{c} - \mathbf{e})) = \left[\sum_i |c_i - e_i|^2\right]^n$ is not equal to 0 for any $\mathbf{c}$ and $\mathbf{e} \neq \mathbf{c}$, $\mathcal{O}(\mathbf{c} - \mathbf{e})$ must be full rank. By the rank criterion, then, the maximum diversity of nm obtains. □

This encoding and decoding scheme provides a way of sending n message symbols over n symbol times, for a rate $R = 1$ coding scheme. However, as mentioned above, it applies only to $n = 2, 4$, or 8.

Generalized Real Orthogonal Designs

In the interest of obtaining more possible designs, we now turn to a generalized real orthogonal design $\mathcal{G}$.

Definition 17.2 [331] A **generalized orthogonal design** of size n is a $p \times n$ matrix $\mathcal{G}$ with entries $0, \pm x_1, \pm x_2, \ldots, \pm x_k$ such that $\mathcal{G}^T \mathcal{G} = KI$, where K is some constant. The **rate** of $\mathcal{G}$ is $R = k/p$. □

Example 17.2 The following are examples of generalized orthogonal designs.

$$\mathcal{G}_3 = \begin{bmatrix} x_1 & x_2 & x_3 \\ -x_2 & x_1 & -x_4 \\ -x_3 & x_4 & x_1 \\ -x_4 & -x_3 & x_2 \end{bmatrix} \qquad \mathcal{G}_5 = \begin{bmatrix} x_1 & x_2 & x_3 & x_4 & x_5 \\ -x_2 & x_1 & x_4 & -x_3 & x_6 \\ -x_3 & -x_4 & x_1 & x_2 & x_7 \\ -x_4 & x_3 & -x_2 & x_1 & x_8 \\ -x_5 & -x_6 & -x_7 & -x_8 & x_1 \\ -x_6 & x_5 & -x_8 & x_7 & -x_2 \\ -x_7 & x_8 & x_5 & -x_6 & -x_3 \\ -x_8 & -x_7 & x_6 & x_5 & -x_4 \end{bmatrix}$$

$$\mathcal{G}_6 = \begin{bmatrix} x_1 & x_2 & x_3 & x_4 & x_5 & x_6 \\ -x_2 & x_1 & x_4 & -x_3 & x_6 & -x_5 \\ -x_3 & -x_4 & x_1 & x_2 & x_7 & x_8 \\ -x_4 & x_3 & -x_2 & x_1 & x_8 & -x_7 \\ -x_5 & -x_6 & -x_7 & -x_8 & x_1 & x_2 \\ -x_6 & x_5 & -x_8 & x_7 & -x_2 & x_1 \\ -x_7 & x_8 & x_5 & -x_6 & -x_3 & x_4 \\ -x_8 & -x_7 & x_6 & x_5 & -x_4 & -x_3 \end{bmatrix}$$

$$\mathcal{G}_7 = \begin{bmatrix} x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 \\ -x_2 & x_1 & x_4 & -x_3 & x_6 & -x_5 & -x_8 \\ -x_3 & -x_4 & x_1 & x_2 & x_7 & x_8 & -x_5 \\ -x_4 & x_3 & -x_2 & x_1 & x_8 & -x_7 & x_6 \\ -x_5 & -x_6 & -x_7 & -x_8 & x_1 & x_2 & x_3 \\ -x_6 & x_5 & -x_8 & x_7 & -x_2 & x_1 & -x_4 \\ -x_7 & x_8 & x_5 & -x_6 & -x_3 & x_4 & x_1 \\ -x_8 & -x_7 & x_6 & x_5 & -x_4 & -x_3 & x_2 \end{bmatrix}.$$

□

The encoding for a generalized orthogonal design is as follows. A set of k symbols $a_1, a_2, \ldots, a_k$ arrives at the encoder. The encoder builds the design matrix $\mathcal{G}$ by setting $x_i = a_i$. Then for $t = 1, 2, \ldots, p$, the n antennas simultaneously transmit the n symbols from the tth row of $\mathcal{G}$. In p symbol times, then, k message symbols are sent, resulting in a rate $R = k/p$ code.

In the interest of maximizing rate, minimizing encoder and decoder latency, and minimizing complexity, it is of interest to determine designs having the smallest p possible. A design having rate at least R with the smallest possible value of p is said to be delay-optimal. (The designs of Example 17.2 yield delay-optimal, rate 1 codes.)

A technique for constructing generalized orthogonal designs is provided in [331].

17.4.4 Complex Orthogonal Designs

We now extend the concepts of real orthogonal designs to complex orthogonal designs.

Definition 17.3 A **complex orthogonal design** of size n is a matrix $\mathcal{U}$ formed from the elements $\pm x_1, \pm x_2, \ldots, \pm x_n$, their conjugates $\pm x_1^*, \pm x_2^*, \ldots, \pm x_n^*$, or multiples of these by $j = \sqrt{-1}$, such that $\mathcal{U}^H\mathcal{U} = (|x_1|^2 + \cdots + |x_n|^2)I$. That is, $\mathcal{U}$ is proportional to a unitary matrix. □

Without loss of generality, the first row can be formed of the elements $x_1, x_2, \ldots, x_n$.

The same method of encoding is used for complex orthogonal designs as for real orthogonal designs: n antennas send the rows for each of n time intervals.

Example 17.3 The matrix

$$\mathcal{U}_2 = \overset{\text{time}}{\downarrow} \quad \overset{\text{space} \longrightarrow}{\begin{bmatrix} x_1 & x_2 \\ -x_2^* & x_1^* \end{bmatrix}}$$

is a complex orthogonal design. As suggested by the arrows, if the columns are distributed in space (across two antennas) and the rows are distributed in time (over different symbol times) this can be used as a space-time code. In fact, this is the Alamouti code. □

Unfortunately, as discussed in Exercise 6, complex orthogonal designs of size n exist only for $n = 2$ or $n = 4$. The Alamouti code is thus, in some sense, almost unique.

We turn therefore to generalized complex orthogonal designs.

Definition 17.4 [331] A **generalized complex orthogonal design** is a $p \times n$ matrix $\mathcal{U}$ whose entries are formed from $0, \pm x_1, \pm x_1^*, \ldots, \pm x_k, \pm x_k^*$ or their product with $j = \sqrt{-1}$ such that $\mathcal{U}^H\mathcal{U} = (|x_1|^2 + \cdots |x_k|^2)I$. □

A generalized orthogonal design can be used to create a rate $R = k/p$ space-time code using n antennas, just as for orthogonal designs above.

Example 17.4 The following are examples of generalized complex orthogonal designs.

$$\mathcal{U}_3 = \begin{bmatrix} x_1 & x_2 & x_3 \\ -x_2 & x_1 & -x_4 \\ -x_3 & x_4 & x_1 \\ -x_4 & -x_3 & x_2 \\ x_1^* & x_2^* & x_3^* \\ -x_2^* & x_1^* & -x_4^* \\ -x_3^* & x_4^* & x_1^* \\ -x_4^* & -x_3^* & x_2^* \end{bmatrix} \qquad \mathcal{U}_4 = \begin{bmatrix} x_1 & x_2 & x_3 & x_4 \\ -x_2 & x_1 & -x_4 & x_3 \\ -x_3 & x_4 & x_1 & -x_2 \\ -x_4 & -x_3 & x_2 & x_1 \\ x_1^* & x_2^* & x_3^* & x_4^* \\ -x_2^* & x_1^* & -x_4^* & x_3^* \\ -x_3^* & x_4^* & x_1^* & -x_2^* \\ -x_4^* & -x_3^* & x_2^* & x_1^* \end{bmatrix}.$$

These provide $R = 1/2$ coding using 3 and 4 antennas, respectively.

A rate $R = 3/4$ code using an orthogonal design is provided by [338] is

$$\mathcal{U}_4 = \begin{bmatrix} x_1 & -x_2^* & -x_3^* & 0 \\ x_2 & x_1^* & 0 & x_3^* \\ x_3 & 0 & x_1^* & -x_2^* \\ 0 & -x_3 & x_2 & x_1 \end{bmatrix}.$$

□

Higher rate codes are also known using *linear processing orthogonal designs*, which form matrices not by individual elements, but by linear combinations of elements. These produce what are known as *generalized complex linear processing orthogonal designs*.

Example 17.5 Examples of linear processing designs producing $R = 3/4$ codes are

$$\mathcal{U} = \begin{bmatrix} x_1 & x_2 & x_3/\sqrt{2} \\ -x_2^* & x_1^* & x_3/\sqrt{2} \\ x_3^*/\sqrt{2} & x_3^*/\sqrt{2} & (-x_1 - x_1^* + x_2 - x_2^*)/2 \\ x_3^*/\sqrt{2} & -x_3^*/\sqrt{2} & (x_2 + x_2^* + x_1 - x_1^*)/2 \end{bmatrix}$$

$$\mathcal{U} = \begin{bmatrix} x_1 & x_2 & x_3/\sqrt{2} & x_3/\sqrt{2} \\ -x_2^* & x_1^* & x_3/\sqrt{2} & -x_3/\sqrt{2} \\ x_3^*/\sqrt{2} & x_3^*/\sqrt{2} & (-x_1 - x_1^* + x_2 - x_2^*)/2 & (-x_2 - x_2^* + x_1 - x_1^*)/2 \\ x_3^*/\sqrt{2} & -x_3^*/\sqrt{2} & (x_2 + x_2^* + x_1 - x_1^*)/2 & -(x_1 + x_1^* + x_2 - x_2^*)/2 \end{bmatrix}$$

for $n = 3$ and $n = 4$ antennas, respectively. □

Future Work

Several specific examples of designs leading to low-complexity decoders have been presented in the examples above. However, additional work in high-rate code designs is still a topic of ongoing research.

17.5 Space-Time Trellis Codes

Trellis codes can also be used to provide diversity. We present here several examples from the literature.

Example 17.6 Consider the coding scheme presented in Figure 17.7. This is an example of *delay diversity*, which is a hybrid of spatial diversity and time diversity. The signal transmitted from two antennas at time t consists of the current symbol a_t and the previous symbol a_{t-1}. The received signal is $r_t = h_0 a_t + h_1 a_{t-1} + n_t$. If the channel parameters $\{h_0, h_1\}$ is known, then an equalizer can be used to detect the transmitted sequence.

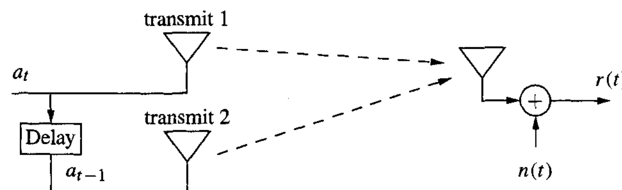


Figure 17.7: A delay diversity scheme.

Because the encoder has a memory element in it, this can also be regarded as a simple trellis code, so that decoding can be accomplished using the Viterbi algorithm. If this encoder is used in conjunction with an 8-PSK signal constellation then there are 8 states. The trellis and the output sequence are shown in Figure 17.8. The outputs listed beside the states are the pair (a_{k-1}, a_k) for each input. (A similar four-state trellis is obtained for a 4-PSK using delay diversity.)

For a sequence of symbols $a_1, a_2, \ldots, a_n$, a delay diversity coder may also be written as a space-time *block code*, with codeword

$$A(a_1, a_2, \ldots, a_n) = \begin{bmatrix} a_1 & a_2 & a_3 & \cdots & a_{n-1} & a_n & 0 \\ 0 & a_1 & a_2 & a_3 & \cdots & a_{n-1} & a_n \end{bmatrix}.$$

The two zeros in the first and last columns ensure that the trellis begins and ends in the 0 state. By viewing this as a space-time block code, the rank-criterion may be employed. The matrix $B(\mathbf{a}, \mathbf{e}) =$

$A(\mathbf{a}) - A(\mathbf{e})$ is, by the linearity of the coding mechanism, equal to $A(\mathbf{a}')$, where $\mathbf{a}' = \mathbf{a} - \mathbf{e}$. The columns containing the first and last elements of $\mathbf{a}'$ are linearly independent, so B has full rank: This code provides a diversity of $m = 2$ at 2 bits/second/Hz.

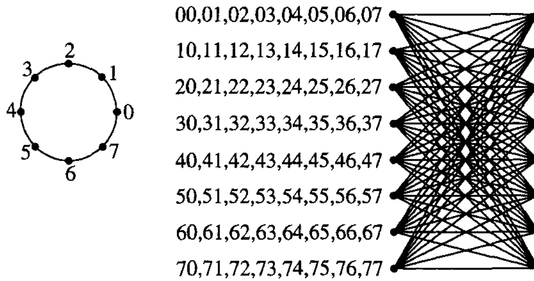


Figure 17.8: 8-PSK constellation and the trellis for a delay-diversity encoder.

□

Example 17.7 In the trellis of Figure 17.8, replace the output mappings with the sequences

state 0:	00, 01, 02, 03, 04, 05, 06, 07
state 1:	50, 51, 52, 53, 54, 55, 56, 57
state 2:	20, 21, 22, 23, 24, 25, 26, 27
state 3:	70, 71, 72, 73, 74, 75, 76, 77
state 4:	40, 41, 42, 43, 44, 45, 46, 47
state 5:	10, 11, 12, 13, 14, 15, 16, 17
state 6:	60, 61, 62, 63, 64, 65, 66, 67
state 7:	30, 31, 32, 33, 34, 35, 36, 37

This corresponds to a delay diversity code, with the additional modification that the delayed symbol is multiplied by -1 if it is odd $\{1, 3, 5, 7\}$. It has been observed [249] that this simple modification provides 2.5 dB of gain compared to simple delay diversity. □

Example 17.8 Figure 17.9 shows space-time codes for 4-PSK (transmitting 2 bits/second/Hz) using 8, 16, and 32 states. Each of these codes provides a diversity of 2. Figure 17.11(a) shows the probability of error performance (obtained via simulation) for these codes when used with two transmit antennas and two receive antennas. Figure 17.11(b) shows the performance with two transmit antennas and one receive antenna. □

Example 17.9 Figure 17.10 shows space-time codes for 8-PSK (transmitting 3 bits/second/Hz) using 16 and 32 states (with the code in Example 17.7 being an 8-state code). Each of these codes provides a diversity of 2. Figure 17.12(a) shows the probability of error performance (obtained via simulation) for these codes when used with two transmit antennas and two receive antennas. Figure 17.12(b) show the performance with two transmit antennas and one receive antenna.

□

17.5.1 Concatenation

Concatenation is also frequently employed in space-time coded systems. In this case, the outer code is frequently a TCM system whose symbols are transmitted via an inner space-time coded system.

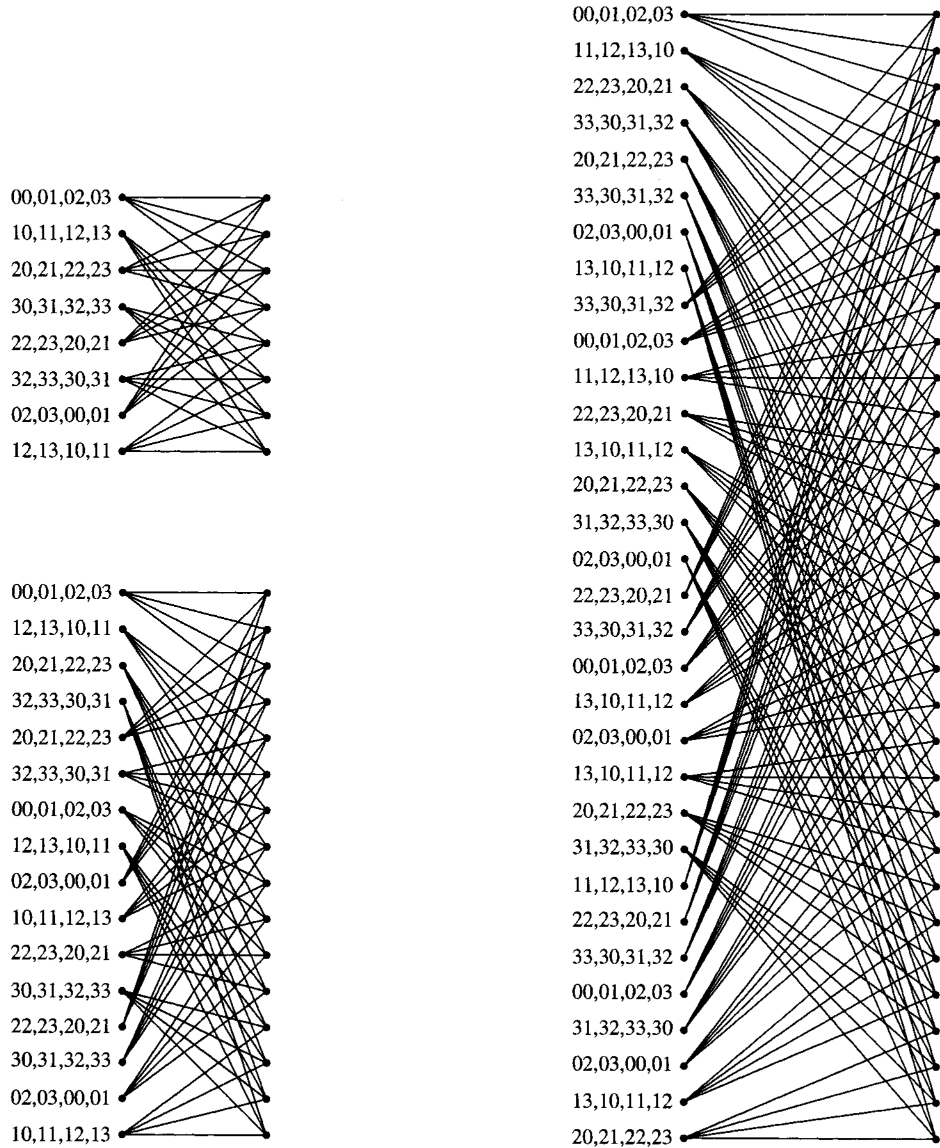


Figure 17.9: Space-time codes with diversity 2 for 4-PSK having 8, 16, and 32 states [332].

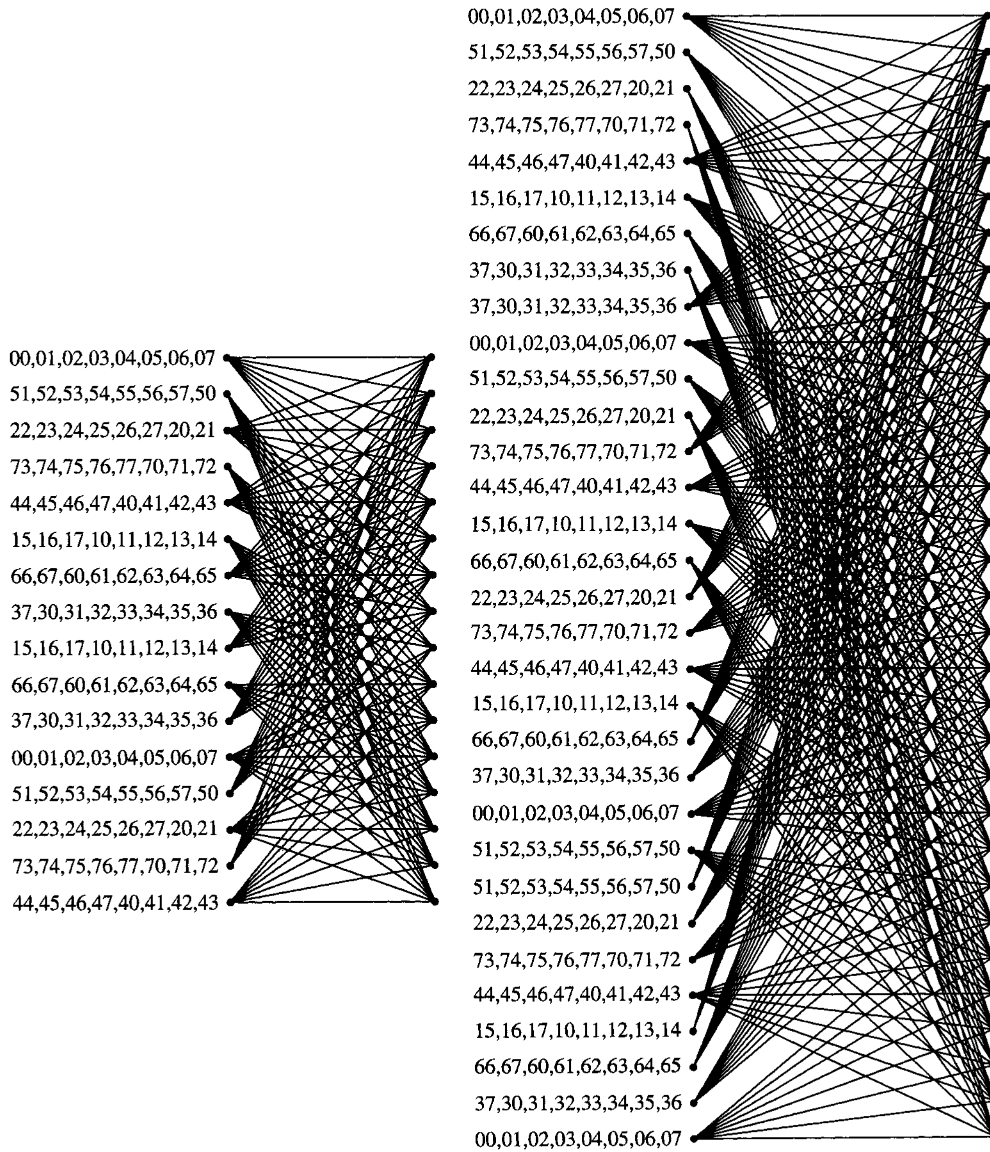
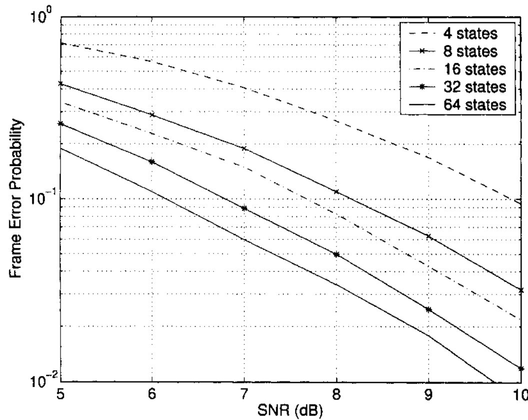
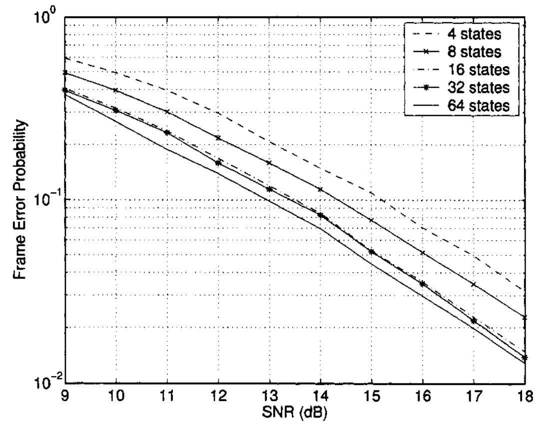


Figure 17.10: Space-time codes with diversity 2 for 8-PSK having 16 and 32 states [332].

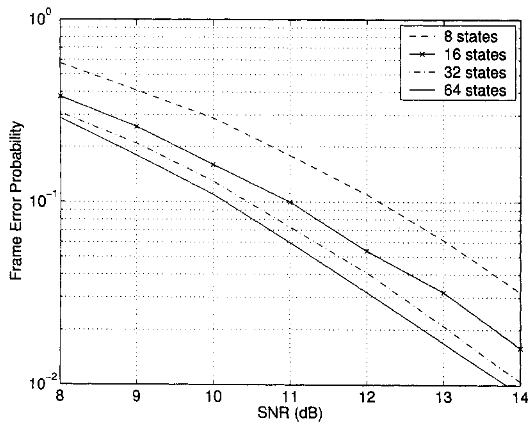


(a) 2 receive and 2 transmit antennas.

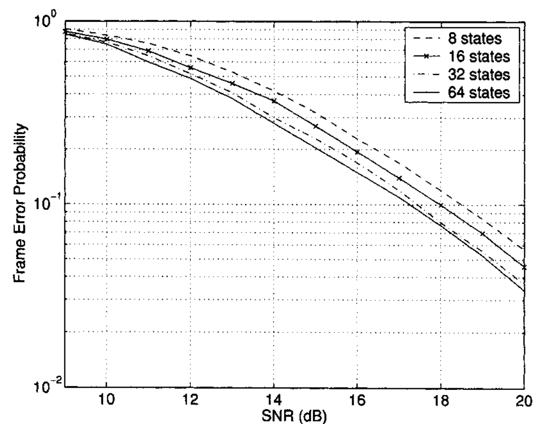


(b) 1 receive and 2 transmit antennas.

Figure 17.11: Performance of codes with 4-PSK that achieve diversity 2 [332].



(a) 2 receive and 2 transmit antennas.



(b) 1 receive and 2 transmit antennas.

Figure 17.12: Performance of codes with 8-PSK that achieve diversity 2 [332].

17.6 How Many Antennas?

We cannot cannot continue to add antennas without reaching a point of diminishing returns. One argument for the number of antennas is based on channel capacity. It has been proved [101, 333] that the capacity of a multiantenna system with a single receive antenna is a random variable of the form $\log_2(1 + (\chi^2_{2n}/2n)\text{SNR})$, where χ^2_{2n} is a χ^2 random variable with $2n$ degrees of freedom (e.g., formed by summing the squares of $2n$ independent zero-

mean, unit-variance Gaussian random variables). By the law of large numbers, as $n \to \infty$,

$$\frac{1}{2n}\chi^2_{2n} \to \frac{1}{2}(E[X^2] + E[X^2]) = 1 \text{ where } X \sim \mathcal{N}(0, 1).$$

In practice, the limit begins to be apparent for $n \geq 4$, suggesting that more than four transmit antennas will provide little additional improvement over four antennas. It can also be argued that with two receive antennas, $n = 6$ transmit antennas provides almost all the benefit possible.

However, there are systems for which it is of interest to employ both interference suppression and diversity. In these cases, having additional antennas is of benefit. A "conservation theorem" [15, p. 546] says that

$$\text{diversity order} + \text{number of interferers} = \text{number of receive antennas}.$$

As the number of interferers to be suppressed increases, having additional antennas is of value.

17.7 Estimating Channel Information

All of the decoders described in this chapter assume that the channel parameters in the form of the $h_{j,i}$ is known at the receiver. Estimating these is viewed for our purposes as signal processing beyond the scope of this book, so we say only a few words regarding the problem.

It is possible to send a "pilot" signal which is known by the receiver, and from this pilot to estimate the channel parameters if the channel is sufficiently static. However, a pilot signal consumes bandwidth and transmitter power and reduces the overall throughput. Another approach is to use differential space-time codes [154, 150], where information is coded in the *change* of symbols, but at the expense of a 3 dB penalty.

Another approach is to blindly estimate the channel parameters, without using the transmitted symbols. This is developed in [326].

17.8 Exercises

17.1 Consider a transmission scheme in which n transmit antennas are used. The vector $\mathbf{a} = [a_1, a_2, \ldots, a_n]^T$ is transmitted to a single receiver through a channel with coefficients h_i^* so that $r = \mathbf{h}^H\mathbf{a} + n$, where $\mathbf{h} = [h_1, h_2, \ldots, h_n]^T$ and where n is zero-mean AWGN with variance σ^2. Diversity can be obtained by sending the same symbol a from each antenna, so that $\mathbf{a} = a\mathbf{w}$, for some "steering vector" $\mathbf{w}$. The received signal is thus $r = \mathbf{h}^H\mathbf{w}a + n$. Show that the weight vector $\mathbf{w}$ of length $\|\mathbf{w}\| = 1$ which maximizes the SNR at the receiver is $\mathbf{w} = \mathbf{h}/\|\mathbf{h}\|$ and determine the maximum SNR.

17.2 Show that if $G = X + jY$, where $X \sim \mathcal{N}(0, \sigma_f^2)$ and $Y \sim \mathcal{N}(0, \sigma_f^2)$, then $Z = |G| = \sqrt{X^2 + Y^2}$ has density

$$f_Z(z) = \begin{cases} \frac{z}{\sigma_f^2} e^{-z^2/2\sigma_f^2} & z \geq 0 \\ 0 & \text{otherwise.} \end{cases}$$

17.3 Show that if $G = X + jY$, where $X \sim \mathcal{N}(0, \sigma_f^2)$ and $Y \sim \mathcal{N}(0, \sigma_f^2)$, then $A = |G|^2 = X^2 + Y^2$ has density

$$f_A(a) = \begin{cases} \frac{1}{2\sigma_f^2} e^{-a/2\sigma_f^2} & a \geq 0 \\ 0 & \text{otherwise.} \end{cases}$$

A is said to be a *chi-squared* (χ^2) random variable with two degrees of freedom.

17.4 Show that (17.5) is correct.

17.5 Using the 4×4 orthogonal design $\mathcal{O}_4$, show that the matrix $H_{j,\text{eff}}$ of (17.17) is proportional to an orthogonal matrix.

17.6 Let $\mathcal{U}$ be a complex orthogonal design of size n. Show that by replacing each complex variable $x_i = x_i^1 + jx_i^2$ in the matrix with the 2×2 matrix $\begin{bmatrix} x_i^1 & x_i^2 \\ x_i^2 & x_i^1 \end{bmatrix}$, that a $2n \times 2n$ real matrix $\mathcal{O}$ is formed that is a real orthogonal design of size $2n$.

Conclude that complex orthogonal designs of size n exist only if $n = 2$ or $n = 4$.

17.7 Show that a modified delay diversity scheme which uses codewords formed by

$$A = \begin{bmatrix} a_1 & a_2 & \cdots & a_{n-1} & a_n \\ a_n & a_1 & a_2 & \cdots & a_{n-1} \end{bmatrix},$$

which is a tail-biting code, does not satisfy the rank criterion and hence does not achieve full diversity.

17.8 [15] For the set of space-time codes

$$A_1 = \begin{bmatrix} x_1 & x_2^* \\ x_2 & x_1^* \end{bmatrix} \quad A_2 = \begin{bmatrix} x_1 & -x_2 \\ x_2 & x_1 \end{bmatrix} \quad A_3 = \begin{bmatrix} x_1^* & -x_2 \\ x_2^* & x_1 \end{bmatrix} \quad A_4 = \begin{bmatrix} x_1 & x_2 \\ x_2 & x_1 \end{bmatrix}$$

(a) Find the diversity order of each code, assuming that the transmitted symbols are selected independently and uniformly from a 4-QAM alphabet and that the receiver has a single antenna.

(b) Determine which of these codes allows for scalar detection at the receiver.

(c) For those codes having full diversity, determine the coding gain.

17.9 References

Propagation modeling, leading to the Rayleigh channel model, is described in [163]; see also [322], [15] and [275]. The Jakes model which produced Figure 17.2 is described in [163]. Our discussion of MIMO channels, and the discussion of diversity following from it, was drawn from [15]. Additional coding-related discussions relating to multiple-receiver diversity appear in [71, 325, 307, 361, 376].

The Alamouti scheme is described in [3]. The generalization to orthogonal designs is described in [331]. The rank criterion is presented following [332]. This paper also presents a thorough discussion of space-time trellis codes and hybrid codes capable of dealing with either slow or fast fading channels.

The paper [332] presents many designs of space-time trellis codes, as does [249]. Combined interference suppression and space-time coding is also discussed in the latter.

Appendix A

Log Likelihood Algebra

In this appendix we present an algebra for the log likelihood of binary variables, leading to a rule for combining log likelihoods called the tanh rule [134, 131]. This algebra pertains to many aspects of soft decision decoding in the book, aspects which may be covered with varying order of presentation. Rather than duplicate this material in each pertinent context, it has been placed here.

Let $\tilde{x}$ be a binary-valued random variable taking on values in the set $\{1, -1\}$. We may think of $\tilde{x}$ as being a mapping from a variable x taking on values in $\{0, 1\}$, with $\tilde{x} = 1 - 2x$. (Note that this is a different convention than used throughout most of the book.) The log likelihood ratio of $\tilde{x}$ is

$$\lambda(\tilde{x}) = \log \frac{P(\tilde{x} = 1)}{P(\tilde{x} = -1)}.$$

The logarithm is the natural logarithm. From the log likelihood ratio, the probability can be easily recovered as

$$P(\tilde{x} = 1) = \frac{e^{\lambda(\tilde{x})}}{1 + e^{\lambda(\tilde{x})}}.$$

Figure A.1 shows $\lambda(x)$ as a function of $P(x = 1)$. The *sign* of $\lambda(\tilde{x})$ is the hard decision of the value of $\tilde{x}$. We can take $|\lambda(\tilde{x})|$ as a measure of the *reliability*. Values of $\tilde{x}$ for which there is certainty, $P(\tilde{x} = 1) = 1$ or $P(\tilde{x} = 1) = 0$, have $|\lambda(\tilde{x})| = \infty$.

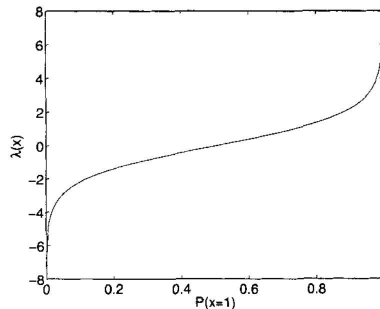


Figure A.1: Log likelihood ratio.

Suppose now that we regard the $\tilde{x}$ as elements in $GF(2)$, with the identity being 1 and let $\oplus$ denote the addition operation on these elements:

$$1 \oplus 1 = 1 \qquad 1 \oplus -1 = -1 \qquad -1 \oplus 1 = -1 \qquad -1 \oplus -1 = 1.$$

Lemma A.1 *If $\tilde{x}_1$ and $\tilde{x}_2$ are statistically independent, then*

$$\lambda(\tilde{x}_1 \oplus \tilde{x}_2) = \log \frac{1 + e^{\lambda(\tilde{x}_1)} e^{\lambda(\tilde{x}_2)}}{e^{\lambda(\tilde{x}_1)} + e^{\lambda(\tilde{x}_2)}}.$$

The proof is Exercise 1. Using the relations

$$\tanh(x/2) = \frac{e^x - 1}{e^x + 1} \quad \text{and} \quad \log \frac{1+x}{1-x} = 2\tanh^{-1} x$$

we have the following result, which can be verified by straightforward expansion:

Lemma A.2

$$\begin{aligned}
\lambda(\tilde{x}_1 \oplus \tilde{x}_2) &= \log \frac{(e^{\lambda(\tilde{x}_1)} + 1)(e^{\lambda(\tilde{x}_2)} + 1) + (e^{\lambda(\tilde{x}_1)} - 1)(e^{\lambda(\tilde{x}_2)} - 1)}{(e^{\lambda(\tilde{x}_1)} + 1)(e^{\lambda(\tilde{x}_2)} + 1) - (e^{\lambda(\tilde{x}_1)} - 1)(e^{\lambda(\tilde{x}_2)} - 1)} \\
&= \log \frac{1 + \tanh(\lambda(\tilde{x}_1/2)) \tanh(\lambda(\tilde{x}_2/2))}{1 - \tanh(\lambda(\tilde{x}_1/2)) \tanh(\lambda(\tilde{x}_2/2))} \\
&= 2\tanh^{-1}\left(\tanh(\lambda(\tilde{x}_1/2)) \tanh(\lambda(\tilde{x}_2/2))\right).
\end{aligned}$$

This can be re-expressed in a form which is interestingly symmetric:

$$\tanh\left(\frac{1}{2}\lambda(\tilde{x}_1 \oplus \tilde{x}_2)\right) = \tanh(\lambda(\tilde{x}_1/2)) \tanh(\lambda(\tilde{x}_2/2)). \tag{A.1}$$

Equation (A.1) is referred to as the tanh rule.

The log likelihood ratio of the sum has the following important approximation (when $\tilde{x}_1$ and $\tilde{x}_2$ are statistically independent):

$$\lambda(\tilde{x}_1 \oplus \tilde{x}_2) \approx \operatorname{sign}(\lambda(\tilde{x}_1)) \operatorname{sign}(\lambda(\tilde{x}_2)) \min(|\lambda(\tilde{x}_1)|, |\lambda(\tilde{x}_2)|). \tag{A.2}$$

We now define a special algebra for log likelihood ratios. We define the operator $\boxplus$ by

$$\boxed{\lambda(\tilde{x}_1) \boxplus \lambda(\tilde{x}_2) \triangleq \lambda(\tilde{x}_1 \oplus \tilde{x}_2)} \tag{A.3}$$

with

$$\lambda(\tilde{x}) \boxplus \infty = \lambda(\tilde{x}) \qquad \lambda(\tilde{x}) \boxplus -\infty = -\lambda(\tilde{x}) \qquad \lambda(\tilde{x}) \boxplus 0 = 0. \tag{A.4}$$

These conditions have the following interpretations: The reliability of $\tilde{x}\oplus$ (some other element with infinite reliability) is the same reliability as $\tilde{x}$. The reliability of $\tilde{x}\oplus$ (some other element which is totally unreliable) is the negative of the reliability of $\tilde{x}$. The reliability of $\tilde{x}\oplus$ (some other element of completely ambiguous reliability), $\lambda = 0$, is completely ambiguous.

The $\boxplus$ operator has an identity and is commutative and associative, but no inverse exists[1]: two unreliable elements cannot add up to the reliable element ∞.

We use $\sum\!\oplus$ to denote the $GF(2)$ series such as

$$\sum_{i=p}^{q}\!\oplus x_i = x_p \oplus x_{p+1} \oplus \cdots \oplus x_q$$

[1]The $\boxplus$ operator thus defines operations for a commutative *monoid*.

and $\sum\boxplus$ to denote a $\boxplus$-series:

$$\sum_{i=p}^{q}\boxplus\lambda(x_i) = \lambda(x_p)\boxplus\lambda(x_{p+1})\boxplus\cdots\boxplus\lambda(x_q).$$

By extending (A.3) by induction,

$$\sum_{i=p}^{q}\boxplus\lambda(x_i) = \lambda\left(\sum_{i=p}^{q}\oplus x_i\right).$$

If $\tilde{x}_1, \tilde{x}_2, \ldots, \tilde{x}_J$ are independent binary-valued random variables, applying Lemma A.2 inductively we obtain

$$\sum_{j=1}^{J}\boxplus\lambda(\tilde{x}_j) = \lambda\left(\sum_{j=1}^{J}\oplus\tilde{x}_j\right) = \log\frac{\prod_{j=1}^{J}(e^{\lambda(\tilde{x}_j)}+1)+\prod_{j=1}^{J}(e^{\lambda(\tilde{x}_j)}-1)}{\prod_{j=1}^{J}(e^{\lambda(\tilde{x}_j)}+1)-\prod_{j=1}^{J}(e^{\lambda(\tilde{x}_j)}-1)}$$
$$= \log\frac{1+\prod_{j=1}^{J}\tanh(\lambda(\tilde{x}_j)/2)}{1-\prod_{j=1}^{J}\tanh(\lambda(\tilde{x}_j)/2)} = 2\tanh^{-1}\left(\prod_{j=1}^{J}\tanh(\lambda(\tilde{x}_j)/2)\right)$$

and

$$\tanh\left(\frac{1}{2}\sum_{j=1}^{J}\boxplus\lambda(\tilde{x}_j)\right) = \prod_{j=1}^{J}\tanh(\lambda(\tilde{x}_j)/2).$$

The reliability can be approximated, as in (A.2), by

$$\sum_{j=1}^{J}\boxplus\lambda(\tilde{x}_j) \approx \left(\prod_{j=1}^{J}\operatorname{sign}(\lambda(\tilde{x}_j))\right)\min_{j\in\{1,2,\ldots,J\}}|\lambda(\tilde{x}_j)|. \tag{A.5}$$

The reliability of the sum $\boxplus$ is therefore determined by the smallest reliability of the terms in the sum.

A.1 Exercises

A.1 Show that Lemma A.1 is true.

A.2 In some cases, it is convenient to deal with $\{0, 1\}$-valued random variables. Let x_1 and x_2 be $\{0, 1\}$-valued variables. Define

$$\lambda(x) = \log\frac{P(x=1)}{P(x=0)}.$$

Addition operations, denoted here as $+$, are now over $GF(2)$, with identity 0.

Analogous to Lemmas A.1 and A.2, show that

$$\lambda(x_1+x_2) = \log\frac{e^{\lambda(x_1)}+e^{\lambda(x_2)}}{1+e^{\lambda(x_1)+\lambda(x_2)}}$$

and

$$\lambda(x_1+x_2) = \log\frac{(e^{\lambda(x_1)}+1)(e^{\lambda(x_2)}+1)-(e^{\lambda(x_1)}-1)(e^{\lambda(x_1)}-1)}{(e^{\lambda(x_1)}+1)(e^{\lambda(x_2)}+1)+(e^{\lambda(x_1)}-1)(e^{\lambda(x_1)}-1)}$$

and

$$\lambda(x_1 + x_2) = -\log \frac{1 + \tanh(-\lambda(x_1)/2)\tanh(-\lambda(x_2)/2)}{1 - \tanh(-\lambda(x_1)/2)\tanh(-\lambda(x_2)/2)}$$
$$= -2\tanh^{-1}(\tanh(-\lambda(x_1)/2)\tanh(-\lambda(x_2)/2))$$

so that

$$\tanh(-\lambda(x_1 + x_2)/2) = \tanh(-\lambda(x_1)/2)\tanh(-\lambda(x_2)/2).$$

A.3 Show that the following *tanh rule* is true: For independent $\{0, 1\}$-valued random variables $x_1, x_2, \ldots, x_n$,

$$\lambda\left(\sum_{i=1}^{n} x_i\right) = -2\tanh^{-1}\left(\prod_{i=1}^{n} \tanh\left(-\frac{\lambda(x_i)}{2}\right)\right) \tag{A.6}$$

or

$$\tanh\left(-\frac{\lambda\left(\sum_{i=1}^{n} x_i\right)}{2}\right) = \prod_{i=1}^{n} \tanh\left(-\frac{\lambda(x_i)}{2}\right). \tag{A.7}$$

A.4 In conjunction with the tanh rule of (A.6), we define the function

$$f(x) = \log \frac{e^x + 1}{e^x - 1} = -\log(\tanh(x/2)).$$

(a) Show that $f(f(x)) = 1$. That is, $f(x)$ is its own inverse.

(b) Show that the tanh rule (A.1) can be expressed as

$$\lambda\left(\sum_{i=1}^{n} x_i\right) = \left[-\prod_{i=1}^{n} \operatorname{sign}(\lambda(c_i))\right] f\left(\sum_{i=1}^{n} f(|\lambda(c_i)|)\right).$$

References

[1] D. Agarwal and A. Vardy, "The Turbo Decoding Algorithm and its Phase Trajectories," *IEEE Trans. Information Theory*, vol. 47, no. 2, pp. 699–722, Feb. 2001.

[2] S. M. Aji and R. J. McEliece, "The Generalized Distributive Law," *IEEE Trans. Info. Theory*, vol. 46, no. 2, pp. 325–343, Mar. 2000.

[3] S. Alamouti, "A Simple Transmit Diversity Technique for Wireless Communications," *IEEE J. on Selected Areas in Comm.*, vol. 16, no. 8, pp. 1451–1458, Oct. 1998.

[4] J. Anderson and M. Hladik, "Tailbiting MAP Decoders," *IEEE Journal on Selected Areas in Communications*, vol. 16, no. 2, pp. 297–302, Feb. 1998.

[5] J. B. Anderson and A. Svensson, *Coded Modulation Systems*. New York: Kluwer Academic/Plenum, 2003.

[6] K. Andrews, C. Heegard, and D. Kozen, "A Theory of Interleavers," Cornell University Department of Computer Science," TR97-1634, June 1997.

[7] ——, "Interleaver Design Methods for Turbo Codes," in *Proc. 1998 Intl. Symposium on Info. Theory*, 1998, p. 420.

[8] S. Ar, R. Lipton, R. Rubinfeld, and M. Sudan, "Reconstructing Algebraic Functions from Mixed Data," in *Proceedings of the 33rd Annual IEEE Symposium on Foundations of Computer Science*, 1992, pp. 503–512.

[9] D. Augot and L. Pecquet, "A Hensel Lifting to Replace Factorization In List-Decoding of Algebraic-Geometric and Reed-Solomon Codes," *IEEE Trans. Information Theory*, vol. 46, pp. 2605–2614, Nov. 2000.

[10] L. R. Bahl, C. D. Cullum, W. D. Frazer, and F. Jelinek, "An efficient algorithm for computing free distance," *IEEE Trans. Info. Theory*, pp. 437–439, May 1972.

[11] L. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal Decoding of Linear Codes for Minimizing Symbol Error Rate," *IEEE Trans. Info. Theory*, vol. 20, pp. 284–287, Mar. 1974.

[12] A. Barbulescu and S. Pietrobon, "Interleaver Design for Turbo Codes," *Electron. Lett.*, vol. 30, no. 25, p. 2107, 1994.

[13] ——, "Terminating the Trellis of Turbo-Codes in the Same State," *Electron. Lett.*, vol. 31, no. 1, pp. 22–23, 1995.

[14] J. R. Barry, "The BCJR Algorithm for Optimial Equalization," `http://users.ece.gatech.edu/~barry/6603/handouts/Notes on the BCJR Algorithm`.

[15] J. R. Barry, E. A. Lee, and D. G. Messerschmitt, *Digital Communication*, 3rd ed. Boston: Kluwer Academic, 2004.

[16] G. Battail and J. Fang, "D'ecodage pondéré optimal des codes linéaires en blocs," *Annales des Télécommunications*, vol. 41, pp. 580–604, Nov. 1986.

[17] R. E. Bellman and S. E. Dreyfus, *Applied Dynamic Programming*. Princeton, NJ: Princeton University Press, 1962.

[18] S. Benedetto, R. Garello, M. Mondin, and M. Trott, "Rotational Invariance of Trellis Codes Part II: Group Codes and Decoders," *IEEE Trans. Information Theory*, vol. 42, pp. 766–778, May 1996.

[19] S. Benedetto and G. Montorsi, "Average Performance of Parallel Concatenated Block Codes," *Electron. Lett.*, vol. 31, no. 3, pp. 156–158, 1995.

[20] ——, "Design of Parallel Concatenated Convolutional Codes," *IEEE Trans. Comm.*, vol. 44, no. 5, pp. 591–600, 1996.

[21] ——, "Unveiling Turbo Codes: Some Results on Parallel Concatenated Coding Schemes," *IEEE Trans. Info. Theory*, vol. 42, no. 2, pp. 409–428, Mar. 1996.

[22] E. Berlekamp, "Nonbinary BCH decoding," in *Proc. Int'l. Symp. on Info. Th.*, San Remo, Italy, 1967.

[23] ——, "Bounded Distance +1 Soft-Decision Reed-Solomon Decoding," *IEEE Trans. Info. Theory*, vol. 42, no. 3, pp. 704–720, May 1996.

[24] E. R. Berlekamp, R. J. McEliece, and H. C. van Tilborg, "On the Inherent Intractability of Certain Coding Problems," *IEEE Trans. Info. Theory*, vol. 24, no. 3, pp. 384–386, May 1978.

[25] E. Berlekamp, *Algebraic Coding Theory*. New York: McGraw-Hill, 1968.

[26] C. Berrou, *Codes, Graphs, and Systems*. Boston: Kluwer Academic, 2002, ch. The Mutations of Convolutional Coding (Around the Trellis), p. 4.

[27] C. Berrou and A. Gaviuex, "Near Optimum Error Correcting Coding and Decoding: Turbo Codes," *IEEE Trans. Comm.*, vol. 44, no. 10, pp. 1261–1271, Oct. 1996.

[28] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon Limit Error-Correcting Coding and Decoding: Turbo Codes," in *Proc. 1993 IEEE International Conference on Communications*, Geneva, Switzerland, 1993, pp. 1064–1070.

[29] E. Biglieri, D. Divsalar, P. J. McLane, and M. K. Simon, *Introduction to Trellis-Coded Modulation, with Applications*. New York: Macmillan, 1991.

[30] P. Billingsley, *Probability and Measure*. New York: Wiley, 1986.

[31] G. Birkhoff and S. MacLane, *A Survey of Modern Algebra*, 3rd ed. New York: Macmillan, 1965.

[32] U. Black, *The V Series Recommendations, Protocols for Data Communications*. New York: McGraw-Hill, 1991.

[33] R. E. Blahut, *Theory and Practice of Error Control Codes*. Reading, MA: Addison-Wesley, 1983.

[34] ——, *Fast Algorithms for Digital Signal Processing*. Reading, MA: Addison-Wesley, 1985.

[35] I. Blake and K. Kith, "On the Complete Weight Enumerator of Reed-Solomon Codes," *SIAM J. Disc. Math*, vol. 4, no. 2, pp. 164–171, May 1991.

[36] R. Bose and D. Ray-Chaudhuri, "On a Class of Error-Correcting Binary Codes," *Inf. and Control*, vol. 3, pp. 68–79, 1960.

[37] A. E. Brouwer, "Bounds on the minimum distance of q-ary linear codes," `http://www.win.tue.nl/~aeb/voorlincod.html`, 2004.

[38] H. Burton, "Inversionless decoding of BCH codes," *IEEE Trans. Information Theory*, vol. 17, pp. 464–466, 1971.

[39] H. Burton and E. Weldon, "Cylic Product Codes," *IEEE Trans. Information Theory*, vol. 11, pp. 433–440, July 1965.

[40] J. B. Cain, G. C. Clark, and J. M. Geist, "Punctured Convolutional Codes of Rate $(n-1)/n$ and Simplified Maximum Likelihood Decoding," *IEEE Trans. Info. Theory*, vol. 25, no. 1, pp. 97–100, Jan. 1979.

[41] A. R. Calderbank and N. J. A. Sloane, "An Eight-Dimensional Trellis Code," *IEEE Trans. Info. Theory*, vol. 74, no. 5, pp. 757–759, May 1986.

[42] ——, "New Trellis Codes Based on Lattices and Cosets," *IEEE Trans. Info. Theory*, vol. 33, no. 2, pp. 177–195, March 1987.

[43] M. Cedervall and R. Johannesson, "A Fast Algorithm for Computing Distance Spectrum of Convolutional Codes," *IEEE Trans. Information Theory*, vol. 35, pp. 1146–1159, 1989.

[44] W. Chambers, "Solution of Welch-Berlekamp Key Equation by Euclidean Algorithm," *Electron. Lett.*, vol. 29, no. 11, p. 1031, 1993.

[45] W. Chambers, R. Peile, K. Tsie, and N. Zein, "Algorithm for Solving the Welch-Berlekamp Key-Equation with a Simplified Proof," *Electron. Lett.*, vol. 29, no. 18, pp. 1620–1621, 1993.

[46] D. Chase, "A Class of Algorithms for Decoding Block Codes With Channel Measurement Information," *IEEE Trans. Info. Theory*, vol. 18, no. 1, pp. 168–182, Jan. 1972.

[47] O. Chauhan, T. Moon, and J. Gunther, "Accelerating the Convergence of Message Passing on Loopy Graphs Using Eigenmessages," in *Proc. 37th Annual Asilomar Conference on Signals, Systems, and Computers*, 2003, pp. 79–83.

[48] P. Chevillat and J. D.J. Costello, "A Multiple Stack Algorithm for Erasurefree Decoding Of Convolutional Codes," *IEEE Trans. Comm.*, vol. 25, pp. 1460–1470, Dec. 1977.

[49] R. Chien, "Cyclic Decoding Procedures for Bose-Chaudhuri-Hocquenghem Codes," *IEEE Trans. Information Theory*, vol. 10, pp. 357–363, 1964.

[50] S.-Y. Chung, G. D. Forney, Jr., T. J. Richardson, and R. Urbanke, "On the Design of Low-Density Parity-Check Codes within 0.0045 dB of the Shannon Limit," *IEEE Commun. Letters*, vol. 5, no. 2, pp. 58–60, Feb. 2001.

[51] S.-Y. Chung, T. J. Richardson, and Rüdiger, "Analysis of Sum-Product Decoding of Low-Density Parity-Check Codes Using Gaussian Approximation," *IEEE Trans. Information Theory*, vol. 47, no. 2, pp. 657–670, Feb. 2001.

[52] T. K. Citron, "Algorithms and architectures for error correcting codes," Ph.D. dissertation, Stanford, Aug. 1986.

[53] J. H. Conway and N. J. A. Sloane, "Fast Quantizing and Decoding Algorithms for Lattice Quantizers and Codes," *IEEE Trans. Info. Theory*, vol. 28, no. 2, pp. 227–232, March 1982.

[54] ——, "Voronoi Regions of Lattices, Second Moments of Polytopes, and Quantization," *IEEE Trans. Info. Theory*, vol. 28, no. 2, pp. 211–226, March 1982.

[55] ——, "On the Voronoi Regions of Certain Lattices," *SIAM J. Alg. and Discrete Methods*, vol. 5, no. 3, pp. 294–305, September 1984.

[56] ——, *Sphere Packings, Lattices, and Groups*, 2nd ed. New York: Springer-Verlag, 1993.

[57] ——, "A Fast Encoding Method for Lattice Codes and Quantizers," *IEEE Trans. Info. Theory*, vol. 29, no. 6, pp. 820–824, November 1983.

[58] ——, "Soft Decoding Techniques for Codes and Lattices, Including the Golay Code and the Leech Lattice," *IEEE Trans. Info. Theory*, vol. 32, no. 1, pp. 41–50, January 1986.

[59] T. M. Cover and J. A. Thomas, *Elements of Information Theory*. New York: Wiley, 1991.

[60] D. Cox, J. Little, and D. O'Shea, *Ideals, Varieties, and Algorithms*. New York: Springer-Verlag, 1992.

[61] ——, *Using Algebraic Geometry*. New York: Springer-Verlag, 1998.

[62] J. Crockett, T. K. Moon, O. Chauhan, and J. Gunther, "Decoding LDPC using multiple-cycle eigenmessages," in *Asilomar Conf. on Signals, Systems, and Comp.*, Monterey, CA, Nov. 2004.

[63] D. Dabiri and I. F. Blake, "Fast Parallel Algorithms for Decoding Reed-Solomon Codes based on Remainder Polynomials," *IEEE Trans. Info. Theory*, vol. 41, no. 4, pp. 873–885, July 1995.

[64] F. Daneshgaran, M. Laddomada, and M. Mondin, "Interleaver Design for Serially Concatenated Convolutional Codes: Theory and Applications," *IEEE Trans. Information Theory*, vol. 50, no. 6, pp. 1177–1188, June 2004.

[65] D. G. Daut, J. W. Modestino, and L. D. Wismer, "New Short Constraint Length Convolutional Code Constructions for Selected Rational Rates," *IEEE Trans. Info. Theory*, vol. 28, no. 5, pp. 794–800, Sept. 1982.

[66] M. C. Davey and D. J. MacKay, "Low Density Parity Check Codes Over $GF(q)$," *IEEE Com. Letters*, vol. 2, no. 6, 1998.

[67] J. R. Deller, J. G. Proakis, and J. H. L. Hansen, *Discrete-Time Processing of Speech Signals*. New York: Macmillan, 1993.

[68] P. Delsarte, "An Algebraic Approach to Coding Theory," Phillips, Research Reports Supplements 10, 1973.

[69] D. Divsalar and R. McEliece, "Effective Free Distance of Turbo Codes," *Electron. Lett.*, vol. 32, no. 5, pp. 445–446, Feb. 1996.

[70] D. Divsalar and F. Pollara, "Turbo Codes for Deep-Space Communications," Jet Propulsion Laboratory," JPL TDA Progress Report 42-122, pp. 42–120, Feb. 1995.

[71] D. Divsalar and M. Simon, "The Design of Coded MPSK for Fading Channel: Performance Criteria," *IEEE Trans. Comm.*, vol. 36, pp. 1013–1021, Sept. 1988.

[72] S. Dolinar and D. Divsalar, "Weight Distributions for Turbo Codes Using Random and Nonrandom Permutations," Jet Propulsion Laboratory," JPL TDA Progress Report 42-121, pp. 42–120, Aug. 1995.

[73] B. Dorsch, "A Decoding Algorithm for Binary Block Codes and J-ary Output Channels," *IEEE Trans. Information Theory*, vol. 20, pp. 391–394, May 1974.

[74] H. El Gamal and J. A. Roger Hammons, "Analyzing the Turbo Decoder Using the Gaussian Approximation," *IEEE Trans. Information Theory*, vol. 47, no. 2, pp. 671–686, Feb. 2001.

[75] M. Elia, "Algebraic Decoding of the (23,12,7) Golay Code," *IEEE Trans. Info. Theory*, vol. 33, no. 1, pp. 150–151, Jan. 1987.

[76] P. Elias, "Coding for Noisy Channels," *IRE Conv. Rept. Pt. 4*, pp. 37–47, 1955.

[77] ——, "List Decoding For Noisy Channels," Res. Lab. Electron., MIT, Cambridge, MA, Tech. Rep. 335, 1957.

[78] M. Eyuboglu and G. D. Forney, Jr., "Trellis Precoding: Combined Coding, Precoding and Shaping for Intersymbol Interference Channels," *IEEE Trans. Information Theory*, vol. 36, pp. 301–314, Mar. 1992.

[79] D. Falconer, "A Hybrid Sequential and Algebraic Decoding Scheme," Ph.D. dissertation, MIT, Cambridge, MA, 1967.

[80] R. M. Fano, "A heuristic discussion of probabilistic decoding," *IEEE Trans. Info. Theory*, vol. 9, pp. 64–73, Apr 1963.

[81] G.-L. Feng, "A Fast Special Factorization Algorithm in the Sudan Decoding Procedure for Reed-Solomon Codes," in *Proc. 31st Allerton Conf. Communications, Control, and Computing*, 2000, pp. 593–602.

[82] G.-L. Feng and K. K. Tzeng, "A Generalized Euclidean Algorithm for Multisequence Shift-Register Synthesis," *IEEE Trans. Info. Theory*, vol. 35, no. 3, pp. 584–594, May 1989.

[83] ——, "A Generalization of the Berlekamp-Massey Algorithm for Multisequence Shift-Register Synthesis With Applications To Decoding Cyclic Codes," *IEEE Trans. Info. Theory*, vol. 37, no. 5, pp. 1274–1287, Sept. 1991.

[84] M. Ferrari and S. Bellini, "Importance Sampling Simulation of Concatenated Block Codes," *Proc. of the IEE*, vol. 147, pp. 245–251, Oct. 2000.

[85] P. Fire, *A Class of Multiple-Error-Correcting Binary Codes For Non-Independent Errors*, Sylvania Report No. RSL-E-2, Sylvania Electronic Defense Laboratory, Reconnaissance Systems Division, Mountain View, CA, Mar. 1959.

[86] G. D. Forney, Jr., "On Decoding BCH Codes," *IEEE Trans. Info. Theory*, vol. 11, no. 4, pp. 549–557, Oct 1965.

[87] ——, *Concatenated Codes*. Cambridge, MA: MIT Press, 1966.

[88] ——, "Generalized Minimum Distance Decoding," *IEEE Trans. Info. Theory*, vol. 12, no. 2, pp. 125–131, Apr. 1966.

[89] ——, "The Viterbi Algorithm," *Proc. IEEE*, pp. 268–278, Mar. 1973.

[90] ——, "Convolutional Codes III: Sequential Decoding," *Inform. Control*, vol. 25, pp. 267–297, July 1974.

[91] ——, "Coset Codes — Part I: Introduction and Geometrical Classification," *IEEE Trans. Info. Theory*, vol. 34, no. 5, pp. 1123–1151, September 1988.

[92] ——, "Coset Codes — Part II: Binary Lattices and Related Codes," *IEEE Trans. Info. Theory*, vol. 34, no. 5, pp. 1152–1187, September 1988.

[93] ——, "Geometrically Uniform Codes," *IEEE Trans. Information Theory*, vol. 37, pp. 1241–1260, 1991.

[94] ——, "Trellis Shaping," *IEEE Trans. Info. Theory*, vol. 38, no. 2, pp. 281–300, Mar. 1992.

[95] ——, "Codes on Graphs: Normal Realizations," *IEEE Trans. Info. Theory*, vol. 47, no. 2, pp. 520–548, Feb. 2001.

[96] ——, "Maximum-likelihood Sequence Estimation of Digital Sequences in the Presence of Intersymbol Interference," *IEEE Trans. Info. Theory*, vol. IT-18, no. 3, pp. 363–378, May 1972.

[97] ——, "Convolutional Codes I: Algebraic Structure," *IEEE Trans. Info. Theory*, vol. 16, no. 6, pp. 720–738, Nov. 1970.

[98] G. D. Forney, Jr., L. Brown, M. V. Eyuboglu, and I. John L. Moran, "The V.34 High-Speed Modem Standard," *IEEE Communications Magazine*, vol. 34, pp. 28–33, Dec. 1996.

[99] G. D. Forney, Jr. and M. Eyuboglu, "Combined Equalization and Coding Using Precoding," *IEEE Commun. Mag.*, vol. 29, no. 12, pp. 25–34, Dec. 1991.

[100] G. D. Forney, Jr., R. G. Gallager, G. R. Lang, F. M. Longstaff, and S. U. Qureshi, "Efficient Modulation for Band-Limited Channels," *IEEE J. on Selected Areas in Comm.*, vol. 2, no. 5, pp. 632–647, September 1984.

[101] G. Foschini and M. Gans, "On Limits of Wireless Communications in a Fading Environment," *Wireless Personal Comm.*, vol. 6, no. 3, pp. 311–355, Mar. 1998.

[102] M. P. Fossorier, F. Burkert, S. Lin, and J. Hagenauer, "On the Equivalence Between SOVA and Max-Log-MAP Decodings," *IEEE Comm. Letters*, vol. 2, no. 5, pp. 137–139, May 1998.

[103] M. P. Fossorier and S. Lin, "Chase-Type and GMD-Type Coset Decoding," *IEEE Trans. Comm.*, vol. 48, pp. 345–350, Mar. 2000.

[104] M. Fossorier and S. Lin, "Soft-Decision Decoding of Linear Block Codes Based on Ordered Statistics," *IEEE Trans. Information Theory*, vol. 41, pp. 1379–1396, Sept. 1995.

[105] ——, "Differential Trellis Decoding of Convolutional Codes," *IEEE Trans. Information Theory*, vol. 46, pp. 1046–1053, May 2000.

[106] J. B. Fraleigh, *A First Course in Abstract Algebra*. Reading, MA: Addison-Wesley, 1982.

[107] B. Frey, *Graphical Models for Machine Learning and Digital Communication*. Cambridge, MA: MIT Press, 1998.

[108] B. J. Frey and D. J. MacKay, "A Revolution: Belief Propagation in Graphs With Cycles," in *Advances in Neural Information Processing Systems*. MIT Press, 1998.

[109] B. Friedland, *Control System Design: An Introduction to State-Space Design*. New York: McGraw-Hill, 1986.

[110] G. Szegö, *Orthogonal Polynomials*, 3rd ed. Providence, RI: American Mathematical Society, 1967.

[111] R. G. Gallager, *Information Theory and Reliable Communication*. New York: Wiley, 1968.

[112] ——, "Low-Density Parity-Check Codes," *IRE Trans. on Info. Theory*, vol. IT-8, pp. 21–28, Jan. 1962.

[113] ——, *Low-Density Parity-Check Codes*. Cambridge, MA: M.I.T. Press, 1963.

[114] S. Gao and M. A. Shokrollahi, "Computing Roots of Polynomials over Function Fields of Curves," in *Coding Theory and Cryptography*, D. Joyner, Ed. Springer-Verlag, 1999, pp. 214–228.

[115] J. Geist, "An Empirical Comparison of Two Sequential Decoding Algorithms," *IEEE Trans. Comm. Tech.*, vol. 19, pp. 415–419, Aug. 1971.

[116] ——, "Search Properties for Some Sequential Decoding Algorithms," *IEEE Trans. Information Theory*, vol. 19, pp. 519–526, July 1973.

[117] A. Geramita and J. Seberry, *Orthogonal Designs, Quadratic Forms and Hadamard Matrices*, ser. Lecture Notes in Pure and Applied Mathematics, v. 43. New York and Basel: Marcel Dekker, 1979.

[118] R. Gold, "Maximal Recursive Sequences with 3-Valued Recursive Cross-Correlation Functions," *IEEE Trans. Info. Theory*, pp. 154–156, 1968.

[119] ——, "Optimal Binary Sequences for Spread Spectrum Multiplexing," *IEEE Trans. Info. Theory*, pp. 619–621, October 1967.

[120] S. W. Golomb, *Shift Register Sequences*. San Francisco: Holden–Day, 1967.

[121] G. H. Golub and C. F. Van Loan, *Matrix Computations*, 3rd ed. Baltimore, MD: Johns Hopkins University Press, 1996.

[122] M. Gonzàlez-Lopez, L. Castedo, and J. García-Frías, "BICM for MIMO Systems Using Low-Density Generator Matrix (LDGM) Codes," in *Proc. International Conference on Acoustics, Speech, and Signal Processing*. IEEE, May 2004.

[123] V. Goppa, "Codes on Algebraic Curves," *Soviet Math. Dokl.*, vol. 24, pp. 170–172, 1981.

[124] ——, *Geometry and Codes*. Dordrecht: Kluwer Academic, 1988.

[125] D. Gorenstein and N. Zierler, "A Class of Error Correcting Codes in p^m Symbols," *J. Society of Indust. Appl. Math.*, vol. 9, pp. 207–214, June 1961.

[126] R. Graham, D. Knuth, and O. Patashnik, *Concrete Mathematics*. Reading, MA: Addison-Wesley, 1989.

[127] J. Gunther, M. Ankapura, and T. Moon, "Blind Turbo Equalization Using a Generalized LDPC Decoder," in *Proc. 11th Digital Signal Processing Workshop*, Taos Ski Valley, NM, Aug. 2004, pp. 206–210.

[128] V. Guruswami and M. Sudan, "Improved Decoding of Reed-Solomon Codes and Algebraic Geometry Codes," *IEEE Trans. Info. Theory*, vol. 45, no. 6, pp. 1757–1767, Sept. 1999.

[129] D. Haccoun and M. Ferguson, "Generalized Stack Algorithms for Decoding Convolutional Codes," *IEEE Trans. Information Theory*, vol. 21, pp. 638–651, Apr. 1975.

[130] J. Hagenauer, "Rate Compatible Punctured Convolutional Codes and Their Applications," *IEEE Trans. Comm.*, vol. 36, pp. 389–400, Apr. 1988.

[131] ——, "Source-Controlled Channel Decoding," *IEEE Trans. Comm.*, vol. 43, no. 9, pp. 2449–2457, Sept. 1995.

[132] J. Hagenauer and P. Hoeher, "A Viterbi Algorithm with Soft-Decision Outputs and Its Applications," in *Proc. Globecom*, Dallas, TX, Nov. 1989, pp. 1680–1686.

[133] J. Hagenauer, E. Offer, and L. Papke, *Reed Solomon Codes and Their Applications*. New York: IEEE Press, 1994, ch. Matching Viterbi Decoders and Reed-Solomon Decoders in a Concatenated System.

[134] ——, "Iterative Decoding of Binary Block and Convolutional Codes," *IEEE Trans. Info. Theory*, vol. 42, no. 2, pp. 429–445, Mar. 1996.

[135] D. Haley, A. Grant, and J. Buetefuer, "Iterative encoding of low-density parity-check codes," in *Global Communication Conference (GLOBECOM)*. IEEE, Nov. 2002, pp. 1289–1293.

[136] R. W. Hamming, *Coding and Information Theory*, 2nd ed. Englewood Cliffs, NJ: Prentice-Hall, 1986.

[137] R. Hamming, "Error detecting and error correcting codes," *Bell Syst. Tech. Journal*, vol. 29, pp. 41–56, 1950.

[138] A. R. Hammonds, Jr., P. V. Kummar, A. Calderbank, N. Sloane, and P. Solè, "The $\mathbb{Z}_4$-Linearity of Kerdock, Preparata, Goethals, and Related Codes," *IEEE Trans. Info. Theory*, vol. 40, no. 2, pp. 301–319, Mar. 1994.

[139] Y. Han, C. Hartmann, and C. Chen, "Efficient Priority-First Search Maximum-Likelihood Soft-Decision Decoding of Linear Block Codes," *IEEE Trans. Information Theory*, vol. 39, pp. 1514–1523, Sept. 1993.

[140] Y. Han, C. Hartmann, and K. Mehrota, "Decoding Linear Block Codes Using a Priority-First Search: Performance Analysis and Suboptimal Version," *IEEE Trans. Information Theory*, vol. 44, pp. 1233–1246, May 1998.

[141] L. Hanzo, T. Liew, and B. Yeap, *Turbo Coding, Turbo Equalization and Space-Time Coding for Transmission Over Fading Channels*. West Sussex, England: Wiley, 2002.

[142] H. Harashima and H. Miyakawa, "Matched-transmission Technique for Channels with Intersymbol Interference," *IEEE Trans. Comm.*, vol. 20, pp. 774–780, Aug. 1972.

[143] C. Hartmann and K. Tzeng, "Generalizations of the BCH Bound," *Inform. Contr.*, vol. 20, no. 5, pp. 489–498, June 1972.

[144] C. Hartmann, K. Tzeng, and R. Chen, "Some Results on the Minimum Distance of Cyclic Codes," *IEEE Trans. Information Theory*, vol. 18, no. 3, pp. 402–409, May 1972.

[145] H. Hasse, "Theorie der h oen Differentiale in einem algebraishen Funcktionenk oper mit volkommenem Kostantenk oerp bei Charakteristic," *J. Reine. Ang. Math.*, pp. 50–54, 175.

[146] C. Heegard and S. B. Wicker, *Turbo Coding*. Boston: Kluwer Academic, 1999.

[147] J. Heller, "Short Constraint Length Convolutional Codes," Jet Propulsion Labs," Space Programs Summary 37–54, v. III, pp. 171–177, 1968.

[148] J. Heller and I. M. Jacobs, "Viterbi Decoding for Satellite and Space Communications," *IEEE Trans. Com. Tech.*, vol. 19, no. 5, pp. 835–848, Oct. 1971.

[149] F. Hemmati and D. Costello, "Truncation Error Probability in Viterbi Decoding," *IEEE Trans. Comm.*, vol. 25, no. 5, pp. 530–532, May 1977.

[150] B. Hochwald and W. Sweldons, "Differential Unitary Space-Time Modulation," Bell Labs," Lucent Technology Technical Report, 1999.

[151] A. Hocquenghem, "Codes Correcteurs D'erreurs," *Chiffres*, vol. 2, pp. 147–156, 1959.

[152] J. K. Holmes and C. C. Chen, "Acquisition time performance of PN spread-spectrum systems," *IEEE Transactions on Communications*, vol. COM-25, no. 8, pp. 778–783, August 1977.

[153] R. A. Horn and C. A. Johnson, *Matrix Analysis*. Cambridge: Cambridge University Press, 1985.

[154] B. Hughes, "Differential Space-Time Modulation," in *Proc. IEEE Wireless Commun. Networking Conf.*, New Orleans, LA, Sept. 1999.

[155] T. W. Hungerford, *Algebra*. New York: Springer-Verlag, 1974.

[156] S. Ideda, T. Tanaka, and S. Amari, "Information Geometry of Turbo and Low-Density Parity-Check Codes," *IEEE Trans. Info. Theory*, vol. 50, no. 6, pp. 1097–1114, June 2004.

[157] K. A. S. Immink, "Runlength-Limited Sequences," *Proceedings of the IEEE*, vol. 78, pp. 1745–1759, 1990.

[158] ——, *Coding Techniques for Digital Recorders*. Englewood Cliffs, NJ: Prentice-Hall, 1991.

[159] K. Immink, *Reed Solomon Codes and Their Applications*. New York: IEEE Press, 1994, ch. RS Code and the Compact Disc.

[160] International Telegraph and Telephone Consultive Committee (CCITT), "Recommendation v.29: 9600 bits per second modem standardized for use on point-to-point 4-wire leased telephone-type circuits," in *Data Communication over the Telephone Network "Blue Book"*. Geneva: International Telecommunications Union, 1988, vol. VIII, pp. 215–227.

[161] I. Jacobs and E. Berlekamp, "A Lower Bound to the Distribution of Computation for Sequential Decoding," *IEEE Trans. Information Theory*, vol. 13, pp. 167–174, Apr. 1967.

[162] N. Jacobson, *Basic Algebra I*. New York: Freeman, 1985.

[163] W. Jakes, *Microwave Mobile Communication*. New York, NY: IEEE Press, 1993.

[164] F. Jelinek, "An Upper Bound on Moments of Sequential Decoding Effort," *IEEE Trans. Information Theory*, vol. 15, pp. 140–149, Jan. 1969.

[165] ——, "Fast Sequential Decoding Algorithm Using a Stack," *IBM J. Res. Develop.*, pp. 675–685, Nov. 1969.

[166] F. Jelinek and J. Cocke, "Bootstrap Hybrid Decoding for Symmetric Binary Input Channels," *Inform. Control.*, vol. 18, pp. 261–281, Apr. 1971.

[167] H. Jin, A. Khandekar, and R. McEliece, "Irregular Repeat-Accumulate Codes," in *Proceedings 2nd International Symposium on Turbo Codes and Related Topics*, Brest, France, Sept. 2000, pp. 1–8.

[168] O. Joerssen and H. Meyr, "Terminating the Trellis of Turbo-Codes," *Electron. Lett.*, vol. 30, no. 16, pp. 1285–1286, 1994.

[169] R. Johannesson, "Robustly Optimal Rate One-Half Binary Convolutional Codes," *IEEE Trans. Information Theory*, vol. 21, pp. 464–468, July 1975.

[170] ——, "Some Long Rate One-Half Binary Convolutional Codes with an Optimum Distance Profile," *IEEE Trans. Information Theory*, vol. 22, pp. 629–631, Sept. 1976.

[171] ——, "Some Rate 1/3 and 1/4 Binary Convolutional Codes with an Optimum Distance Profile," *IEEE Trans. Information Theory*, vol. 23, pp. 281–283, Mar. 1977.

[172] R. Johannesson and E. Paaske, "Further Results on Binary Convolutional Codes with an Optimum Distance Profile," *IEEE Trans. Information Theory*, vol. 24, pp. 264–268, Mar. 1978.

[173] R. Johannesson and P. Stahl, "New Rate 1/2, 1/3, and 1/4 Binary Convolutional Encoders With Optimum Distance Profile," *IEEE Trans. Information Theory*, vol. 45, pp. 1653–1658, July 1999.

[174] R. Johannesson and K. Zigangirov, *Fundamentals of Convolutional Coding*. Piscataway, NJ: IEEE Press, 1999.

[175] R. Johannesson and Z.-X. Wan, "A Linear Algebra Approach to Minimal Convolutional Encoders," *IEEE Trans. Information Theory*, vol. 39, no. 4, pp. 1219–1233, July 1993.

[176] S. Johnson and S. Weller, "Construction of Low-Density Parity-Check Codes from Kirkman Triple Systems," in *Global Communications Conference (GLOBECOM)*, Nov 25–29, 2001, pp. 970–974.

[177] ——, "Regular Low-Density Parity-Check Codes from Combinatorial Designs," in *Information Theory Workshop*, 2001, pp. 90–92.

[178] ——, "Higher-Rate LDPC Codes from Unital Designs," in *Global Telecommunications Conference (GLOBECOM)*. IEEE, 2003, pp. 2036–2040.

[179] ——, "Resolvable 2-designs for Regular Low-Density Parity Check Codes," *IEEE Trans. Comm.*, vol. 51, no. 9, pp. 1413–1419, Sept. 2003.

[180] P. Jung and M. Nasshan, "Dependence of the Error Performance of Turbo-Codes on the Interleaver Structure in Short Frame Transmission Systems," *Electron. Lett.*, vol. 30, no. 4, pp. 287–288, Feb. 1994.

[181] T. Kailath, *Linear Systems*. Englewood Cliffs, NJ: Prentice-Hall, 1980.

[182] T. Kaneko, T. Nishijima, and S. Hirasawa, "An Improvement of Soft-Decision Maximum-Likelihood Decoding Algorithm Using Hard-Decision Bounded-Distance Decoding," *IEEE Trans. Information Theory*, vol. 43, pp. 1314–1319, July 1997.

[183] T. Kaneko, T. Nishijima, H. Inazumi, and S. Hirasawa, "An Efficient Maximum Likelihood Decoding of Linear Block Codes with Algebraic Decoder," *IEEE Trans. Information Theory*, vol. 40, pp. 320–327, Mar. 1994.

[184] T. Kasami, "A Decoding Method for Multiple-Error-Correcting Cyclic Codes by Using Threshold Logics," in *Conf. Rec. Inf. Process. Soc. of Japan (in Japanese)*, Tokyo, Nov. 1961.

[185] ——, "A Decoding Procedure for Multiple-Error-Correction Cyclic Codes," *IEEE Trans. Information Theory*, vol. 10, pp. 134–139, Apr. 1964.

[186] T. Kasami, S. Lin, and W. Peterson, "Some Results on the Weight Distributions of BCH Codes," *IEEE Trans. Information Theory*, vol. 12, no. 2, p. 274, Apr. 1966.

[187] D. E. Knuth, *The Art of Computer Programming*. Reading, MA: Addison-Wesley, 1997, vol. 1.

[188] R. Koetter, "On Algebraic Decoding of Algebraic-Geometric and Cyclic Codes," Ph.D. dissertation, University of Linkoping, 1996.

[189] R. Koetter and A.Vardy, "The Structure of Tail-Biting Trellises: Minimality and Basic Principles," *IEEE Trans. Information Theory*, vol. 49, no. 9, pp. 2081–2105, Sept. 2003.

[190] R. Koetter, A. C. Singer, and M. Tüchler, "Turbo Equalization," *IEEE Signal Processing Magazine*, vol. 21, no. 1, pp. 67–80, Jan. 2004.

[191] R. Koetter and A. Vardy, "Algebraic Soft-Decision Decoding of Reed-Solomon Codes," *IEEE Trans. Info. Theory*, vol. 49, no. 11, pp. 2809–2825, Nov. 2003.

[192] V. Kolesnik, "Probability decoding of majority codes," *Probl. Peredachi Inform.*, vol. 7, pp. 3–12, July 1971.

[193] R. Kötter, "Fast Generalized Minimum Distance Decoding of Algebraic-Geometry and Reed-Solomon Codes," *IEEE Trans. Info. Theory*, vol. 42, no. 3, pp. 721–737, May 1996.

[194] Y. Kou, S. Lin, and M. P. Fossorier, "Low-Density Parity-Check Codes Based on Finite Geometries: A Rediscovery and New Results," *IEEE Trans. Info. Theory*, vol. 47, no. 7, pp. 2711–2736, Nov. 2001.

[195] F. R. Kschischang, B. J. Frey, and H.-A. Loeliger, "Factor Graphs and the Sum-Product Algorithm," *IEEE Trans. Info. Theory*, vol. 47, no. 2, pp. 498–519, Feb. 2001.

[196] R. Laroia, N. Farvardin, and S. A. Tretter, "On Optimal Shaping of Multidimensional Constellations," *IEEE Trans. Info. Theory*, vol. 40, no. 4, pp. 1044–1056, July 1994.

[197] K. Larsen, "Short Convolutional Codes with Maximal Free Distance for Rates 1/2, 1/3 and 1/4," *IEEE Trans. Info. Theory*, vol. 19, pp. 371–372, May 1973.

[198] L. Lee, *Convolutional Coding: Fundamentals and Applications*. Boston, MA: Artech House, 1997.

[199] N. Levanon, *Radar Principles*. New York: Wiley Interscience, 1988.

[200] R. Lidl and H. Niederreiter, *Finite Fields*. Reading, MA: Addison-Wesley, 1983.

[201] ——, *Introduction to Finite Fields and their Applications*. Cambridge: Cambridge University Press, 1986.

[202] S. Lin and E. Weldon, "Further Results on Cyclic Product Codes," *IEEE Trans. Information Theory*, vol. 6, no. 4, pp. 452–459, July 1970.

[203] S. Lin and D. J. Costello, Jr., *Error Control Coding: Fundamentals and Applications*. Englewood Cliffs, NJ: Prentice-Hall, 1983.

[204] ——, *Error Control Coding: Fundamentals and Applications*, 2nd ed. Englewood Cliffs, NJ: Prentice-Hall, 2004.

[205] S. Lin, T. Kasami, T. Fujiwara, and M. Forrorier, *Trellises and Trellis-based Decoding Algorithms for Linear Block Codes*. Boston: Kluwer Academic Publishers, 1998.

[206] D. Lind and B. Marcus, *Symbolic Dynamics and Coding*. Cambridge, England: Cambridge University Press, 1995.

[207] J. Lodge, P. Hoeher, and J. Hagenauer, "The Decoding of Multidimensional Codes Using Separable MAP 'Filters'," in *Proc. 16th Biennial Symp. on Comm.*, Queen's University, Kingston, Ont. Canada, May 1992, pp. 343–346.

[208] J. Lodge, R. Young, P. Hoeher, and J. Hagenauer, "Separable MAP 'Filters' for the Decoding of Product and Concatenated Codes," in *Proc. IEEE Int. Conf. on Comm.*, Geneva, Switzerland, May 1993, pp. 1740–1745.

[209] H.-A. Loeliger, "An Introduction to Factor Graphs," *IEEE Signal Processing Mag.*, vol. 21, no. 1, pp. 28–41, Jan. 2004.

[210] T. D. Lookabaugh and R. M. Gray, "High–Resolution Quantization Theory and the Vector Quantizer Advantage," *IEEE Trans. Info. Theory*, vol. 35, no. 5, pp. 1020–1033, September 1989.

[211] D. Lu and K. Yao, "Improved Importance Sampling Technique for Efficient Simulation of Digital Communication Systems," *IEEE Journal on Selected Areas in Communications*, vol. 6, no. 1, pp. 67–75, Jan. 1988.

[212] M. G. Luby, M. Miteznmacher, M. A. Shokrollahi, and D. A. Spielman, "Improved Low-Density Parity-Check Codes Using Irregular Graphs," *IEEE Trans. Information Theory*, vol. 47, no. 2, pp. 585–598, Feb. 2001.

[213] H. Ma and J. Wolf, "On Tailbiting Codes," *IEEE Trans. Comm.*, vol. 34, pp. 104–111, Feb. 1986.

[214] X. Ma and X.-M. Wang, "On the Minimal Interpolation Problem and Decoding RS Codes," *IEEE Trans. Info. Theory*, vol. 46, no. 4, pp. 1573–1580, July 2000.

[215] D. J. MacKay, `http://www.inference.phy.cam.ac.uk/mackay/CodesFiles.html`.

[216] ——, "Near Shannon Limit Performance of Low Density Parity Check Codes," *Electron. Lett.*, vol. 33, no. 6, pp. 457–458, Mar 1997.

[217] ——, "Good Error-Correcting Codes Based on Very Sparse Matrices," *IEEE Trans. Info. Theory*, vol. 45, no. 2, pp. 399–431, March 1999.

[218] D. J. MacKay and R. M. Neal, "Good Codes Based on Very Sparse Matrices," in *Cryptography and Coding 5th IMA Conference*, ser. Lecture Notes in Computer Science, C. Boyd, Ed. Springer, 1995, vol. 1025, pp. 100–111.

[219] F. MacWilliams, "A Theorem on the Distribution of Weights in a Systematic Code," *Bell Syst. Tech. Journal*, vol. 42, pp. 79–94, 1963.

[220] F. MacWilliams and N. Sloane, *The Theory of Error-Correcting Codes*. Amsterdam: North-Holland, 1977.

[221] S. J. Mason, "Feedback Theory–Some Properties of Signal Flow Graphs," *Proceedings IRE*, pp. 1144–1156, September 1953.

[222] J. L. Massey, "Shift-Register Synthesis and BCH Decoding," *IEEE Trans. Info. Theory*, vol. IT-15, no. 1, pp. 122–127, 1969.

[223] ——, "Variable-Length Codes and the Fano Metric," *IEEE Trans. Info. Theory*, vol. IT-18, no. 1, pp. 196–198, Jan. 1972.

[224] J. Massey, *Threshold Decoding*. Cambridge, MA: MIT Press, 1963.

[225] ——, "Coding and Modulation in Digital Communications," in *Proc. Int. Zurich Seminar on Dig. Comm.*, Zurich, Switzerland, 1974, pp. E2(1)–E2(4).

[226] J. Massey and M. Sain, "Inverses of Linear Sequential Circuits," *IEEE Trans. Comp.*, vol. 17, pp. 330–337, Apr. 1968.

[227] R. J. McEliece, E. R. Rodemich, H. Rumsey, Jr., and L. R. Welch, "New Upper Bounds on the Rate of a Code via the Delsarte-MacWilliams Inequalities," *IEEE Trans. Info. Theory*, vol. 23, no. 2, pp. 157–166, Mar. 1977.

[228] R. McEliece, *The Theory of Information and Coding*, ser. Encyclopedia of Mathematics and its Applications. Reading, MA: Addison-Wesley, 1977.

[229] ——, "A Public-Key Cryptosystem Based On Algebraic Coding Theory," JPL, DSN Progress Report 42–44, January and February 1978.

[230] ——, "The Guruswami-Sudan Algorithm for Decoding Reed-Solomon Codes," JPL, IPN Progress Report 42-153, May 15, 2003 2003, available at `http://iprn.jpl.nasa.gov/progress_report/42-153/`.

[231] R. McEliece and L. Swanson, *Reed-Solomon Codes and Their Applications*. New York: IEEE Press, 1994, ch. Reed-Solomon Codes and the Exploration of the Solar System, pp. 25–40.

[232] R. J. McEliece, D. J. MacKay, and J.-F. Cheng, "Turbo Decoding as an Instance of Pearl's Belief Propagation Algorithm," *IEEE J. on Selected Areas in Comm.*, vol. 16, no. 2, pp. 140–152, Feb. 1998.

[233] R. J. McEliece, E. R. Rodemich, and J.-F. Cheng, "The Turbo Decision Algorithm," in *Proc. 33rd Annual Allerton Conference on Communication, Control, and Computing*, 1995, pp. 366–379.

[234] R. J. McEliece and J. B. Shearer, "A Property of Euclid's Algorithm and an Application to Padè Approximation," *SIAM J. Appl. Math*, vol. 34, no. 4, pp. 611–615, June 1978.

[235] R. J. McEliece and M. Yildrim, "Belief Propagation on Partially Ordered Sets," in *Mathematical Systems Theory in Biology, Communications, Computation, and Finance*, D. Gilliam and J. Rosenthal, Eds. IMA, 2003, available at `http://www.systems.caltech.edu/EE/Faculty/rjm/`.

[236] G. V. Meerbergen, M. Moonen, and H. D. Man, "Critically Subsampled Filterbanks Implementing Reed-Solomon Codes," in *Proc. International Conference on Acoustics, Speech, and Signal Processing*. Montreal, Canada: IEEE, May 2004, pp. II–989–992.

[237] J. Meggitt, "Error Correcting Codes and their Implementations," *IRE Trans. Info. Theory*, vol. 7, pp. 232–244, Oct. 1961.

[238] A. Michelson and A. Levesque, *Error Control Techniques for Digital Communication*. New York: Wiley, 1985.

[239] W. Mills, "Continued Fractions and Linear Recurrences," *Mathematics of Computation*, vol. 29, no. 129, pp. 173–180, Jan. 1975.

[240] M. Mitchell, "Coding and Decoding Operation Research," G.E. Advanced Electronics Final Report on Contract AF 19 (604)-6183, Air Force Cambridge Research Labs, Cambridge, MA, Tech. Rep., 1961.

[241] ——, "Error-Trap Decoding of Cyclic Codes," G.E. Report No. 62MCD3, General Electric Military Communications Dept., Oklahoma City, OK, Tech. Rep., Dec. 1962.

[242] T. K. Moon, "Wavelets and Orthogonal (Lattice) Spaces," *International Symposium on Information Theory*, p. 250, 1995.

[243] T. K. Moon and S. Budge, "Bit-Level Erasure Decoding of Reed-Solomon Codes Over $GF(2^n)$," in *Proc. Asilomar Conference on Signals and Systems*, 2004, pp. 1783–1787.

[244] T. K. Moon and J. Gunther, "On the Equivalence of Two Welch-Berlekamp Key Equations and their Error Evaluators," *IEEE Trans. Information Theory*, vol. 51, no. 1, pp. 399–401, 2004.

[245] T. Moon, "On General Linear Block Code Decoding Using the Sum-Product Iterative Decoder," *IEEE Comm. Lett.*, 2004, (accepted for publication).

[246] T. K. Moon and W. C. Stirling, *Mathematical Methods and Algorithms for Signal Processing*. Upper Saddle River, NJ: Prentice-Hall, 2000.

[247] M. Morii and M. Kasahara, "Generalized Key-Equation of Remainder Decoding Algorithm for Reed-Solomon Codes," *IEEE Trans. Info. Theory*, vol. 38, no. 6, pp. 1801–1807, Nov. 1992.

[248] D. Muller, "Application of Boolean Switching Algebra to Switching Circuit Design," *IEEE Trans. on Computers*, vol. 3, pp. 6–12, Sept. 1954.

[249] A. Naguib, N. Seshadri, and A. Calderbank, "Increasing Data Rate over Wireless Channels," *IEEE Signal Processing Magazine*, pp. 76–92, May 2000.

[250] I. Niven, H. S. Zuckerman, and H. L. Montgomery, *An Introduction to the Theory of Numbers*, 5th ed. New York: Wiley, 1991.

[251] J. Odenwalder, "Optimal Decoding of Convolutional Codes," Ph.D. dissertation, Department of Systems Sciences, School of Engineering and Applied Sciences, University of California, Los Angeles, 1970.

[252] V. Olshevesky and A. Shokrollahi, "A Displacement Structure Approach to Efficient Decoding of Reed-Solomon and Algebraic-Geometric Codes," in *Proc. 31st ACM Symp. Theory of Computing*, Atlanta, GA, May 1999.

[253] A. V. Oppenheim and R. W. Schafer, *Discrete-Time Signal Processing*. Englewood Cliffs, NJ: Prentice-Hall, 1989.

[254] E. Paaske, "Short Binary Convolutional Codes with Maximal Free Distance For Rate 2/3 and 3/4," *IEEE Trans. Information Theory*, vol. 20, pp. 683–689, Sept. 1974.

[255] A. Papoulis, *Probability, Random Variables, and Stochastic Processes*, 2nd ed. New York: McGraw Hill, 1984.

[256] N. Patterson, "The Algebraic Decoding of Goppa Codes," *IEEE Trans. Info. Theory*, vol. 21, no. 2, pp. 203–207, Mar. 1975.

[257] P. Pazkad and V. Anantharam, "A New Look at the Generalized Distributive Law," *IEEE Trans. Info. Theory*, vol. 50, no. 6, pp. 1132–1155, June 2004.

[258] J. Pearl, *Probabilistic Reasoning in Intelligent Systems*. San Mateo, CA: Morgan Kaufmann, 1988.

[259] L. Perez, J. Seghers, and D. Costello, "A Distance Spectrum Interpretation of Turbo Codes," *IEEE Trans. Information Theory*, vol. 42, pp. 1698–1709, Nov. 1996.

[260] W. W. Peterson, *Error-correcting Codes*. Cambridge, MA and New York: MIT Press and Wiley, 1961.

[261] W. Peterson, "Encoding and Error-Correction Procedures for the Bose-Chaudhuri Codes," *IEEE Trans. Information Theory*, vol. 6, pp. 459–470, 1960.

[262] W. Peterson and E. Weldon, *Error Correction Codes*. Cambridge, MA: MIT Press, 1972.

[263] S. Pietrobon, G. Ungerböck, L. Perez, and D. Costello, "Rotationally Invariant Nonlinear Trellis Codes for Two-Dimensional Modulation," *IEEE Trans. Information Theory*, vol. 40, no. 6, pp. 1773–1791, 1994.

[264] S. S. Pietrobon and D. J. Costello, "Trellis Coding with Multidimensional QAM Signal Sets," *IEEE Trans. Info. Theory*, vol. 29, no. 2, pp. 325–336, Mar. 1993.

[265] S. S. Pietrobon, R. H. Deng, A. Lafanechère, G. Ungerboeck, and D. J. Costello, "Trellis-Coded Multidimensional Phase Modulation," *IEEE Trans. Info. Theory*, vol. 36, no. 1, pp. 63–89, Jan. 1990.

[266] M. Plotkin, "Binary Codes With Specified Minimum Distances," *IEEE Trans. Information Theory*, vol. 6, pp. 445–450, 1960.

[267] H. V. Poor, *An Introduction to Signal Detection and Estimation*. New York: Springer-Verlag, 1988.

[268] J. Porath and T. Aulin, "Algorithm Construction of Trellis Codes," *IEEE Trans. Comm.*, vol. 41, no. 5, pp. 649–654, 1993.

[269] A. B. Poritz, "Hidden Markov Models: A Guided Tour," in *Proceedings of ICASSP*, 1988.

[270] G. Pottie and D. Taylor, "A Comparison of Reduced Complexity Decoding Algorithms for Trellis Codes," *IEEE Journal on Selected Areas in Communications*, vol. 7, no. 9, pp. 1369–1380, 1989.

[271] E. Prange, "Cyclic Error-Correcting Codes in Two Symbols," Air Force Cambridge Research Center, Cambridge, MA, Tech. Rep. TN-57-103, Sept. 1957.

[272] ——, "Some Cyclic Error-Correcting Codes with Simple Decoding Algorithms," Air Force Cambridge Research Center, Cambridge, MA, Tech. Rep. TN-58-156, Apr. 1958.

[273] ——, "The Use of Coset Equivalence in the Analysis and Decoding of Group Codes," Air Force Cambridge Research Center, Cambridge, MA, Tech. Rep. TN-59-164, 1959.

[274] O. Pretzel, *Codes and Algebraic Curves*. Oxford: Clarendon Press, 1998.

[275] J. G. Proakis, *Digital Communications*, 3rd ed. New York: McGraw-Hill, 1995.

[276] J. G. Proakis and M. Salehi, *Communication Systems Engineering*. Upper Saddle River, NJ: Prentice-Hall, 1994.

[277] M. Püschel and J. Moura, "The Algebraic Approach to the Discrete Cosine and Sine Transforms and Their Fast Algorithms," *SIAM J. of Computing*, vol. 32, no. 5, pp. 1280–1316, 2003.

[278] R. Pyndiah, A. Glavieux, A. Picart, and S. Jacq, "Near Optimum Decoding of Product Codes," in *IEEE Global Telecommunications Conference*. San Francisco: IEEE, Nov. 1994, pp. 339–343.

[279] R. M. Pyndiah, "Near-Optimum Decoding of Product Codes: Block Turbo Codes," *IEEE Trans. Comm.*, vol. 46, no. 8, pp. 1003–1010, Aug. 1998.

[280] L. R. Rabiner, "A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition," *Proc. IEEE*, vol. 77, no. 2, pp. 257–286, February 1989.

[281] T. V. Ramabadran and S. S. Gaitonde, "A Tutorial On CRC Computations," *IEEE Micro*, pp. 62–75, Aug. 1988.

[282] J. Ramsey, "Realization of Optimum Interleavers," *IEEE Trans. Information Theory*, vol. 16, pp. 338–345, May 1970.

[283] I. Reed, T.-K. Truong, X. Chen, and X. Yin, "Algebraic Decoding of the (41,21,9) Quadratic Residue Code," *IEEE Trans. Information Theory*, vol. 38, no. 3, pp. 974–986, May 1992.

[284] I. Reed, "A Class of Multiple-Error-Correcting Codes and a Decoding Scheme," *IEEE Trans. Information Theory*, vol. 4, pp. 38–49, Sept. 1954.

[285] I. Reed, R. Scholtz, T. Truong, and L. Welch, "The Fast Decoding of Reed-Solomon Codes Using Fermat Theoretic Transforms and Continued Fractions," *IEEE Trans. Info. Theory*, vol. 24, no. 1, pp. 100–106, Jan. 1978.

[286] I. Reed and G. Solomon, "Polynomial Codes over Certain Finite Fields," *J. Soc. Indust. Appl. Math*, vol. 8, pp. 300–304, 1960.

[287] I. Reed, X. Yin, and T.-K. Truong, "Algebraic Decoding of (32,16,8) Quadratic Residue Code," *IEEE Trans. Information Theory*, vol. 36, no. 4, pp. 876–880, July 1990.

[288] T. J. Richardson and R. Urbanke, *Iterative Coding Systems*. (available online), March 30, 2003.

[289] T. J. Richardson and R. L. Urbanke, "Efficient encoding of low-density parity-check codes," *IEEE Trans. Information Theory*, vol. 47, no. 2, pp. 638–656, Feb. 2001.

[290] T. J. Richardson and R. Urbanke, "The Capacity of Low-Density Parity-Check Codes Under Message-Passing Decoding," *IEEE Trans. Info. Theory*, vol. 47, no. 2, pp. 599–618, Feb. 2001.

[291] T. J. Richardson, "Error Floors of LDPC Codes," `www.ldpc-codes.com/papers/ErrorFloors.pdf`.

[292] T. J. Richardson, M. A. Shokrollahi, and R. L. Urbanke, "Design of Capacity-Approaching Irregular Low-Density Parity-Check Codes," *IEEE Trans. Information Theory*, vol. 47, no. 2, pp. 619–637, Feb. 2001.

[293] R. Rivest, A. Shamir, and L. Adleman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems," *Comm. of the ACM*, vol. 21, no. 2, pp. 120–126, 1978.

[294] P. Robertson, "Illuminating the Structure of Parallel Concatenated Recursive (TURBO) Codes," in *Proc. GLOBECOM*, vol. 3, San Francisco, CA, Nov. 1994, pp. 1298–1303.

[295] P. Robertson, E. Villebrun, and P. Hoher, "A Comparison of Optimal and Sub-Optimal MAP Decoding Algorithms Operating in the Log Domain," in *Proc. of the Int'l Conf. on Comm. (ICC)*, Seattle, WA, June 1995, pp. 1009–1013.

[296] C. Roos, "A New Lower Bound for the Minimum Distance of a Cyclic Code," *IEEE Trans. Information Theory*, vol. 29, no. 3, pp. 330–332, May 1983.

[297] R. M. Roth and G. Ruckenstein, "Efficient Decoding of Reed-Solomon Codes Beyond Half the Minimum Distance," *IEEE Trans. Info. Theory*, vol. 46, no. 1, pp. 246–256, Jan. 2000.

[298] L. Rudolph, "Easily Implemented Error-Correction Encoding-Decoding," G.E. Report 62MCD2, General Electric Corporation, Oklahoma City, OK, Tech. Rep., Dec. 1962.

[299] ——, "A Class of Majority Logic Decodable Codes," *IEEE Trans. Information Theory*, vol. 13, pp. 305–307, Apr. 1967.

[300] S. Sankaranarayanan, A. Cvetkoviĉ, and B. Vasiĉ, "Unequal Error Protection for Joint Source-Channel Coding Schemes," in *Proceedings of the International Telemetering Conference (ITC)*, 2003, p. 1272.

[301] D. V. Sarwate and M. B. Pursley, "Crosscorrelation Properties of Pseudorandom and Related Sequences," *Proc. IEEE*, vol. 68, no. 5, pp. 593–619, May 1980.

[302] J. Savage, "Sequential Decoding — The Computational Problem," *Bell Syst. Tech. Journal*, vol. 45, pp. 149–175, Jan. 1966.

[303] C. Schlegel, *Trellis Coding*. New York: IEEE Press, 1997.

[304] M. Schroeder, *Number Theory in Science and Communication*, 2nd ed. New York: Springer-Verlag, 1986.

[305] R. Sedgewick, *Algorithms*. Reading, MA: Addison-Wesley, 1983.

[306] J. Seghers, "On the Free Distance of TURBO Codes and Related Product Codes," Swiss Federal Institute of Technology, Zurich, Switzerland, Final Report, Diploma Project SS 1995 6613, Aug. 1995.

[307] N. Seshadri and C.-E. W. Sundberg, "Multi-level Trellis Coded Modulation for the Rayleigh Fading Channel," *IEEE Trans. Comm.*, vol. 41, pp. 1300–1310, Sept. 1993.

[308] K. Shanmugan and P. Balaban, "A Modified Monte Carlo Simulation Technique for Evaluation of Error Rate in Digital Communication Systems," *IEEE Trans. Comm.*, vol. 28, no. 11, pp. 1916–1924, Nov. 1980.

[309] C. Shannon, "A Mathematical Theory of Communication," *Bell Syst. Tech. Journal*, vol. 27, pp. 623–656, 1948, (see also *Collected Papers of Claude Shannon*, IEEE Press, 1993).

[310] M. Shao and C. L. Nikias, "An ML/MMSE Estimation Approach to Blind Equalization," in *Proc. ICASSP*, vol. 4. IEEE, 1994, pp. 569–572.

[311] R. Y. Shao, S. Lin, and M. P. Fossorier, "Two simple stopping criteria for turbo decoding," *IEEE Trans. Comm.*, vol. 47, no. 8, pp. 1117–1120, Aug. 1999.

[312] R. Shao, M. Fossorier, and S. Lin, "Two Simple Stopping Criteria for Iterative Decoding," in *Proc. IEEE Symp. Info. Theory*, Cambridge, MA, Aug. 1998, p. 279.

[313] A. Shibutani, H. Suda, and F. Adachi, "Reducing Average Number of Turbo Decoding Iterations," *Electron. Lett.*, vol. 35, no. 9, pp. 70–71, Apr. 1999.

[314] R. Singleton, "Maximum Distance q-ary Codes," *IEEE Trans. Information Theory*, vol. 10, pp. 116–118, 1964.

[315] B. Sklar, "A Primer on Turbo Code Concepts," *IEEE Comm. Magazine*, pp. 94–101, Dec. 1997.

[316] P. Smith, M. Shafi, and H. Gao, "Quick Simulation: A Review of Importance Sampling Techniques in Communications Systems," *IEEE Journal on Selected Areas in Communications*, vol. 15, no. 4, pp. 597–613, May 1997.

[317] J. Snyders, "Reduced Lists of Error Patterns for Maximum Likelihood Soft Decoding," *IEEE Trans. Information Theory*, vol. 37, pp. 1194–1200, July 1991.

[318] J. Snyders and Y. Be'ery, "Maximum Likelihood Soft Decoding of Binary Block Codes and Decoders for the Golay Codes," *IEEE Trans. Information Theory*, vol. 35, pp. 963–975, Sept. 1989.

[319] H. Song and B. V. Kumar, "Low-Density Parity-Check Codes For Partial Response Channels," *IEEE Signal Processing Magazine*, vol. 21, no. 1, pp. 56–66, Jan. 2004.

[320] P. Staahl, J. B. Anderson, and R. Johannesson, "Optimal and Near-Optimal Encoders for Short and Moderate-Length Tail-Biting Trellises," *IEEE Trans. Information Theory*, vol. 45, no. 7, pp. 2562–2571, Nov. 1999.

[321] H. Stichtenoth, *Algebraic Function Fields and Codes*. Berlin: Springer-Verlag, 1993.

[322] G. Stüber, *Principles of Mobile Communication*, 2nd ed. Boston: Kluwer Academic Press, 2001.

[323] M. Sudan, "Decoding of Reed-Solomon Codes Beyond the Error-Correction Bound," *J. Complexity*, vol. 13, pp. 180–193, 1997.

[324] Y. Sugiyama, M. Kasahara, S. Hirasawa, and T. Namekawa, "A Method for Solving Key Equation for Goppa Codes," *Inf. and Control*, vol. 27, pp. 87–99, 1975.

[325] C.-E. W. Sundberg and N. Seshadri, "Coded Modulation for Fading Channels: An Overview," *European Trans. Telecommun. and Related Technol.*, pp. 309–324, May 1993, special issue on Application of Coded Modulation Techniques.

[326] A. Swindlehurst and G. Leus, "Blind and Semi-Blind Equalization for Generalized Space-Time Block Codes," *IEEE Trans. Signal Processing*, vol. 50, no. 10, pp. 2489–2498, Oct. 2002.

[327] D. Taipale and M. Pursley, "An Improvement to Generalized-Minimum-Distance Decoding," *IEEE Trans. Information Theory*, vol. 37, pp. 167–172, Jan. 1991.

[328] O. Takeshita and J. D.J. Costello, "New Classes of Algebraic Interleavers for Turbo-Codes," in *Int. Conf. on Info. Theory (Abstracts)*, Cambridge, MA, Aug. 1998.

[329] ——, "New Deterministic Interleaver Designs for Turbo Codes," *IEEE Trans. Information Theory*, vol. 46, pp. 1988–2006, Sept. 2000.

[330] R. Tanner, "A Recursive Approach To Low Complexity Codes," *IEEE Trans. Info. Theory*, vol. 27, no. 5, pp. 533–547, Sept. 1981.

[331] V. Tarokh, H. Jararkhami, and A. Calderbank, "Space-Time Block Codes from Orthogonal Designs," *IEEE Trans. Info. Theory*, vol. 45, no. 5, pp. 1456–1467, July 1999.

[332] V. Tarokh, N. Seshadri, and A. Calderbank, "Space-Time Codes for High Data Rate Wireless Communication: Performance Criterion and Code Construction," *IEEE Trans. Info. Theory*, vol. 44, no. 2, pp. 744–765, Mar. 1998.

[333] E. Telatar, "Capacity of Multi-Antenna Gaussian Channels," AT&T Bell Labs Internal Tech. Memo, June 1995.

[334] S. ten Brink, "Iterative Decoding Trajectories of Parallel Concatenated Codes," in *Third IEEE ITG Conf. on Source and Channel Coding*, Munich, Germany, Jan. 2000.

[335] ——, "Rate One-Half Code for Approaching the Shannon Limit by 0.1 dB," *Electron. Lett.*, vol. 36, pp. 1293–1294, July 2000.

[336] ——, "Convergence Behavior of Iteratively Decoded Parallel Concatenated Codes," *IEEE Trans. Comm.*, vol. 49, no. 10, pp. 1727–1737, Oct. 2001.

[337] A. Tietäväinen, "On the nonexistence of perfect codes over finite fields," *SIAM J. Appl. Math.*, vol. 24, pp. 88–96, 1973.

[338] O. Tirkkonen and A. Hottinen, "Square-Matrix Embeddable Space-Time Block Codes for Communication: Performance Criterion and Code Construction," *IEEE Trans. Information Theory*, vol. 44, no. 2, pp. 744–765, Mar. 2002.

[339] M. Tomlinson, "New Automatic Equalizer Employing Modulo Arithmetic," *Electron. Lett.*, vol. 7, pp. 138–139, Mar. 1971.

[340] M. Trott, S. Benedetto, R. Garello, and M. Mondin, "Rotational Invariance of Trellis Codes Part I: Encoders and Precoders," *IEEE Trans. Information Theory*, vol. 42, pp. 751–765, May 1996.

[341] M. Tsfasman and S. Vlăduţ, *Algebraic-Geometric Codes*. Dordrecht, The Netherlands: Kluwer Academic Publishers, 1991.

[342] M. Tsfasman, S. Vlăduţ, and T. Zink, "On Goppa Codes Which Are Better than the Varshamov-Gilbert Bound," *Math. Nachr.*, vol. 109, pp. 21–28, 1982.

[343] M. Tüchler, R. Koetter, and A. C. Singer, "Turbo Equalization: Principles and New Results," *IEEE Trans. Comm.*, vol. 50, no. 5, pp. 754–767, May 2002.

[344] G. Ungerböck and I. Csajka, "On Improving Data-Link Performance by Increasing Channel Alphabet and Introducing Sequence Coding," in *IEEE Int. Symp. on Inform. Theory*, Ronneby, Sweden, June 1976, p. 53.

[345] G. Ungerboeck, "Channel Coding with Multilevel/Phase Signals," *IEEE Trans. Info. Theory*, vol. 28, no. 1, pp. 55–67, Jan. 1982.

[346] ——, "Trellis-Coded Modulation with Redundant Signal Sets. Part I: Introduction," *IEEE Comm. Mag.*, vol. 25, no. 2, pp. 5–11, Feb. 1987.

[347] ——, "Trellis-Coded Modulation with Redundant Signal Sets. Part II: State of the Art," *IEEE Comm. Mag.*, vol. 25, no. 2, pp. 12–21, Feb. 1987.

[348] A. Valembois and M. Fossorier, "An Improved Method to Compute Lists of Binary Vectors that Optimize a Given Weight Function with Application to Soft Decision Decoding," *IEEE Comm. Lett.*, vol. 5, pp. 456–458, Nov. 2001.

[349] G. van der Geer and H. van Lint, *Introduction to Coding Theory and Algebraic Geometry*. Basel: Birkhäuser, 1988.

[350] J. van Lint, *Introduction to Coding Theory*, 2nd ed. Berlin: Springer-Verlag, 1992.

[351] B. Vasic, "Structured Iteratively Decodable Codes based on Steiner Systems and their Application in Magnetic Recording," in *Global Telecommunications Conference (GLOBECOM)*. San Antonio, TX: IEEE, Nov. 2001, pp. 2954–2960.

[352] B. Vasic, E. Kurtas, and A. Kuznetsov, "LDPC Codes Based on Mutually Orthogonal Latin Rectangles and Their Application In Perpendicular Magnetic Recording," *IEEE Trans. Magnetics*, vol. 38, no. 5, pp. 2346–2348, Sept. 2002.

[353] B. Vasic and O. Milenkovic, "Combinatorial Constructions of Low-Density Parity-Check Codes for Iterative Decoding," *IEEE Trans. Information Theory*, vol. 50, no. 6, pp. 1156–1176, June 2004.

[354] S. Verdu, "Optimum multi-user signal detection," Ph.D. dissertation, University of Illinois, 1984.

[355] A. J. Viterbi, "Approaching the Shannon Limit: Theorist's Dream and Practitioner's Challenge," in *Proc. of the Int. Conf. on Millimeter Wave and Far Infrared Science and Tech.*, 1996, pp. 1–11.

[356] ——, "An Intuitive Justification and Simplified Implementation of the MAP decoder for Convolutional Codes," *IEEE J. Special Areas in Comm.*, pp. 260–264, Feb. 1997.

[357] A. J. Viterbi and J. Omura, *Principles of Digital Communication and Coding*. New York: McGraw-Hill, 1979.

[358] A. J. Viterbi, "Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm," *IEEE Trans. Information Theory*, vol. 13, pp. 260–269, Apr. 1967.

[359] ——, "Convolutional Codes and Their Performance in Communication Systems," *IEEE Trans. Com. Techn.*, vol. 19, no. 5, pp. 75–772, Oct. 1971.

[360] J. von zur Gathen and J. Gerhard, *Modern Computer Algebra*. Cambridge: Cambridge University Press, 1999.

[361] L.-F. Wei, "Coded M-DPSK with Built-in Time Diversity for Fading Channels," *IEEE Trans. Information Theory*, vol. 39, pp. 1820–1839, Nov. 1993.

[362] ——, "Trellis-Coded Modulation with Multidimensional Constellations," *IEEE Trans. Info. Theory*, vol. 33, no. 4, pp. 483–501, July 1987.

[363] L. Wei, "Rotationally Invariant Convolutional Channel Coding with Expanded Signal Space I: 180 Degrees," *IEEE Journal on Selected Areas in Communications*, vol. 2, pp. 659–672, 1984.

[364] ——, "Rotationally Invariant Convolutional Channel Coding with Expanded Signal Space II: Nonlinear Codes," *IEEE Journal on Selected Areas in Communications*, vol. 2, pp. 672–686, 1984.

[365] ——, "Trellis-Coded Modulation with Multidimensional Constellations," *IEEE Trans. Information Theory*, vol. 33, pp. 483–501, 1987.

[366] ——, "Rotationally Invariant Trellis-Coded Modulation with Multidimensional M-PSK," *IEEE Journal on Selected Areas in Communications*, vol. 7, no. 9, pp. 1281–1295, 1989.

[367] Y. Weiss, "Correctness of Local Probability Propagation in Graphical Models with Loops," *Neural Computation*, vol. 12, pp. 1–41, 2000.

[368] Y. Weiss and W. T. Freeman, "Correctness of Belief Propagation in Gaussian Graphical Models of Arbitrary Topology," *Neural Computation*, vol. 13, pp. 2173–2200, 2001.

[369] L. R. Welch and E. R. Berlekamp, "Error Correction for Algebraic Block Codes," U.S. Patent Number 4,633,470, Dec. 30, 1986.

[370] N. Wiberg, "Codes and Decoding on General Graphs," Ph.D. dissertation, Linköping University, 1996.

[371] N. Wiberg, H.-A. Loeliger, and R. Kötter, "Codes and Iterative Decoding on General Graphs," *Euro. Trans. Telecommun.*, vol. 6, pp. 513–525, 1995.

[372] S. Wicker and V. Bhargava, *Reed Solomon Codes and Their Applications*. New York: IEEE Press, 1994.

[373] S. B. Wicker, *Error Control Systems for Digital Communications and Storage*. Englewood Cliffs, NJ: Prentice-Hall, 1995.

[374] S. B. Wicker and S. Kim, *Fundamentals of Codes, Graphs, and Iterative Decoding*. Boston: Kluwer Academic, 2003.

[375] R. J. Wilson, *Introduction to Graph Theory*. Essex, England: Longman Scientific, 1985.

[376] S. Wilson and Y. Leung, "Trellis Coded Phase Modulation on Rayleigh Fading Channels," in *Proc. IEEE ICC*, June 1997.

[377] J. K. Wolf, "Efficient Maximum Likelihood Decoding of Linear Block Codes Using a Trellis," *IEEE Trans. Info. Theory*, vol. 24, no. 1, pp. 76–80, Jan. 1978.

[378] J. Wozencraft and B. Reiffen, *Sequential Decoding*. Cambridge, MA: MIT Press, 1961.

[379] X.-W. Wu and P. Siegel, "Efficient Root-Finding Algorithm with Application to List Decoding of Algebraic-Geometric Codes," *IEEE Trans. Information Theory*, vol. 47, pp. 2579–2587, Sept. 2001.

[380] Y. Wu, B. D. Woerner, and W. J. Ebel, "A simple stopping criterion for turbo decoding," *IEEE Comm. Lett.*, vol. 4, no. 8, pp. 258–260, Aug. 2000.

[381] J. Yeddida, W. Freeman, and Y. Weiss, "Generalized Belief Propagation," in *Advances in Neural Information Processing Systems*, T. Leen, T. Dietterich, and V. Tresp, Eds., 2000, vol. 13, pp. 689–695, tR-2000-26.

[382] R. W. Yeung, *A First Course in Information Theory*. New York: Kluwer Academic, 2002.

[383] E. Zehavi and J. Wolf, "On the Performance Evaluation of Trellis Codes," *IEEE Trans. Information Theory*, vol. 32, pp. 196–202, Mar. 1987.

[384] F. Zhai and I. J. Fair, "New Error Detection Techniques and Stopping Criteria for Turbo Decoding," in *IEEE Canadian Conference on Electronics and Computer Engineering (CCECE)*, 2000, pp. 58–60.

[385] W. Zhang, "Finite-State Machines in Communications," Ph.D. dissertation, University of South Australia, 1995.

[386] R. E. Ziemer and R. L. Peterson, *Digital Communications and Spread Spectrum Systems*. New York: Macmillan, 1985.

[387] N. Zierler, "Linear Recurring Sequences," *J. Soc. Indust. Appl. Math.*, vol. 7, pp. 31–48, 1959.

[388] K. S. Zigangirov, "Some Sequential Decoding Procedures," *Prob. Pederachi Inform.*, vol. 2, pp. 13–25, 1966.

Index

Symbols

(n, k, d) code, 84
$K_k(x; n, q)$ (Krawtchouk polynomial), 415
O notation, 408
$R_g[\cdot]$, 124
$R_n = GF(2)[x]/(x^n - 1)$, 118
$R_{n,q} = \mathbb{F}_q[x]/(x^n - 1)$, 118
$\mathbb{C}$ and $\mathbb{R}$, 194
$\mathbb{F}$ (field), 73
$\mathbb{F}_q$, 74
α (for extension field), 198
α_t (forward probability, MAP algorithm), 591
β_t (backward probability, MAP algorithm), 591
$\boxplus$, 736
χ (character), 415
χ^2 distribution, 718, 734
χ_p (Legendre symbol), 372
$\equiv$, 184
    properties, 185
γ_t (transition probability), 591
$\triangleq$, 15
μ Moebius function, 222
$\oplus$ (direct sum), 392
$\otimes$ (Kronecker product), 370
$\perp$, 79
ϕ function (Euler ϕ), 185, 229
$\sim$ (summary notation), 682
$\sum\oplus$, 737
$\sum\boxplus$, 737
$\times$ (Cartesian product), 64, 682
o notation, 408
$x^n - 1$ factorization, 215
(a, b) (greatest common divisor), 176
(n, k), 83
$\binom{n}{k}$, 58
$[n, k]$, 83
$[\mathbf{u}|\mathbf{u} + \mathbf{v}]$ construction, 112, 404
    Reed-Muller code, 391
$\nmid$, 175
$|$, 69, 175
$|G|$, 63
802 wireless standard, 634, 721

A

`A1-2.txt`, 675
`A1-4.txt`, 675
Abelian, 63
add-compare-select, 481
add-compare-select (ACS), 481
adjacent, 457
AEP, 46
`Agall.m`, 637
`Agall.txt`, 637
Alamouti code, 719
α (BCJR algorithm), 591
α_q, 407
alternant code, 277
annihilator, 157, 160
$A_q(n, d)$, 407
arithmetic coding, 6
arithmetic-geometric inequality, 334
`Asmall.txt`, 675
associative, 62
asymptotic coding gain, 103
    convolutional code, 504
    TCM code, 541
asymptotic equipartition property, 46
asymptotic rate, 407
augmented block code, 106
automorphism, 81
AWGNC (additive white Gaussian noise channel), 44

B

backward pass BCJR algorithm, 593
Barker code, 170
base field, 196
basic transfer function matrix, 463
basis of vector space, 77
BAWGNC (binary AWGNC), 44
Bayes' rule, 17
BCH bound, 237, 238
BCH code, 235
    decoder programming, 283
    design, 235
    design distance, 237
    narrow sense, 235
    primitive, 235
    weight distribution, 239
`BCHdec.cc`, 283
`BCHdec.h`, 283
`bchdesigner`, 241
`bchweight.m`, 240
BCJR algorithm, 469, 588
    matrix formulation, 597
`BCJR.cc`, 629
`BCJR.h`, 629
belief propagation, 682
Bellman, 474
Berlekamp-Massey algorithm, 253
    programming, 281
Bernoulli random process, 24
best known code, 107
β (BCJR algorithm), 591
Bhattacharya bound, 502
bijective, 70
binary detection, 18
binary erasure channel, 532
binary operation, 62
binary phase-shift keying (BPSK), 10
binary symmetric channel (BSC), 23
`BinConv.h`, 526
`BinConvdec01.h`, 528
`BinConvdecBPSK.cc`, 528
`BinConvdecBPSK.h`, 528
`BinConvFIR.cc`, 526
`BinConvFIR.h`, 526
`BinConvIIR.cc`, 526
`BinConvIIR.h`, 526
`BinLFSR.cc`, 162
`BinLFSR.h`, 162
binomial theorem, 201
`BinPolyDiv.cc`, 162
`BinPolyDiv.h`, 162
bipartite graph, 456, 457, 638, 686
bit error rate, 98, 491
block code, 83
    definition, 83
    trellis representation, 38, 523
block turbo coding, 623
Bluestein chirp algorithm, 290
Boolean functions, 375
bound
    comparison of, 407
    Elias, 420
    Gilbert-Varshamov, 409
    Griesmer, 411
    Hamming, 89, 406
    linear programming, 414
    McEliece-Rodemich-Rumsey-Welch, 418
    Plotkin, 111, 410
    Singleton, 88, 406
    Varshamov-Gilbert, 111
bounded distance decoding, 30, 93, 101, 322, 450
BPSK (binary phase-shift keying), 10
`bpskprob.m`, 21
`bpskprobplot.m`, 21
branch metric, 473
BSC (binary symmetric channel), 23
`bsc.c`, 285
bucket, 515
burst error, 425
    detection,cyclic codes, 149

C

canonical homomorphism, 73
capacity, 9
    BSC, 59
Cartesian product, 64, 682
catastrophic code, 461, 462, 530
Cauchy-Schwartz inequality, 411
causal codeword, 354
`cawgnc.m`, 51
`cawgnc2.m`, 45
Cayley-Hamilton theorem, 169
`cbawgnc.m`, 51
`cbawgnc2.m`, 45
CD, 41
central limit theorem, 712

channel capacity, 42
  BSC, 43
  of MIMO channel, 732
channel coding theorem, 9, 45
  LDPC codes and, 634
  turbo codes and, 582
channel reliability, 19, 605
character (of a group), 415
characteristic polynomial, 169
Chase decoding algorithms, 445
Chauhan, Ojas, 533
checksum, 56
Chernoff bound, 502, 532
`chernoff1.m`, 502
$\chi_p(x)$, 372
chi-squared random variable, 734
Chien search, 248
`Chiensearch.cc`, 283
`Chiensearch.h`, 283
Chinese remainder theorem, 188
chirp algorithm, 290
χ^2 distribution, 718, 734
Christoffel-Darboux formula, 419
circuits, *see* realizations
clustering in factor graphs, 700
code
  Alamouti, 719
  alternant, 277
  BCH, 235
  best known, 107
  convolutional, 452
  CRC, 147
  dual, 86
  Fire, 433
  generalized RS, 277
  Golay, 398
  Goppa, 278
  Hadamard, 374
  Hamming, 34, 53
  Justeson, 290
  LDPC, 635
  maximal-length, 97
  MDS, 245
  parity check, 107
  quadratic residue, 396
  Reed-Muller, 376
  Reed-Solomon, 242
  repeat accumulate (RA), 671, 672
  repetition, 28
  self-dual, 109, 399
  simplex, 97, 374
  space-time block, 719
  space-time trellis, 728
  TCM, 535
  turbo, 584
coding gain, 36
  asymptotic, 103
column distance function, 521
column space, 77
commutative, 63
compact disc, 427
companion matrix, 169
compare-select-add, 481
complete decoder, 93
`comptut.pdf`, x
`computekm.m`, 333
`computeLbar.m`, 357
`computeLm.cc`, 337
`computeLm.m`, 337, 357
`computetm.m`, 357
concatenated codes, 432
concentration principle, 637
`concodequant.m`, 486
conditional entropy, 41
congruence, 184
conjugacy class, 210
conjugate elements, 209
conjugate of field element, 210
connection polynomial, 130
consistent random variables, 620, 656
constellation expansion factor, 541
constituent encoder, 584
constraint length, 465
continued fraction, 228
`Convdec.cc`, 528
`Convdec.h`, 528
convolutional code, 452
  equivalent, 461
  feedforward/feedback encoder, 454
  Markov property, 590
  tables of codes, 506
Cook-Toom algorithm, 699
correction distance
  of GS decoder, 333
correlation, 162
correlation discrepancy, 450
coset, 67
  of standard array, 92
coset leader weight distribution, 101
CRC (cyclic redundancy check) code, 147
  byte oriented algorithm, 150
cross entropy, 41
  stopping criteria, 606
`crtgamma.m`, 189
`crtgammapoly.m`, 189
cryptography, 7
  McEliece public key, 280
  RSA public key, 186
cyclic code, 38
  burst detection, 149
  definition, 113
  encoding, 133
cyclic group, 66
cyclic redundancy check (CRC), 147
cyclic shift, 113
`cyclomin`, 217
cyclotomic coset, 217

D

D-transform, 127, 452
dB scale, 21
DBV-RS2, 634
decibel, 21
decoder failure, 30, 93, 249
  LDPC decoder, 648
decoding depth, 482
`densev1.m`, 658
`densevtest.m`, 658
density evolution, 655
  irregular codes, 664
derivative
  formal, 263, 289
  Hasse, 329, 330
derivative, formal, 219
design distance, 237
design rate, 636
detected bit error rate, 99
detection
  binary, 18
d_{free}, 495
difference sets, 669
differential encoder, 558
differential Viterbi algorithm, 481
digraph, 457
Dijkstra's algorithm, 472
dimension
  of linear code, 83
  of vector space, 77
direct product, 64
direct sum, 392
  matrix, 393
directed graph, 457
discrete Fourier transform (DFT), 192, 269, 271, 683
  factor graph, 687, 703
displacement, 368
distance distribution, 414
distance spectrum, 547, 614
distributive law, 74, 76, 115
  generalized, 680
diversity, 710, 712
  delay, 728
  frequency, 712
  spatial, 712
  time, 712
diversity order, 718
  space-time code, 723
divides, 69, 175
divisible, 175
division algorithm, 175
  polynomial, 114
`doexitchart.m`, 660
`dotrajectory.m`, 660
double error pattern, 168
double-adjacent-error pattern, 167
$D(P\|Q)$, 42
dual code, 86
  cyclic generator, 167
dual space, 79

E

E_b, 10
E_c, 26
edge, 457
eigenmessage, 679
eight-to-fourteen code, 428
Eisenstein integer, 566
elementary symmetric functions, 250
Elias bound, 420
encryption
  RSA, 187
entropy, 4, 40
  differential, 43
  function, q-ary, 407
  function, binary, 4, 420
equivalence relation, 68
equivalent
  block codes, 85
  convolutional code, 461
`erase.mag`, 268
erasure decoding, 104
  binary, 105
  Reed-Solomon codes, 267

Welch-Berlekamp decoding, 321
error detection, 90
error floor, 584, 653, 654, 671
error locator polynomial, 247, 248
error rate, 98
error trapping decoder, 435
Euclidean algorithm, 177, 368
continued fractions and, 228
extended, 181
LFSR and, 182
matrix formulation, 226, 227
Padé approximations and, 228
properties, 227
Reed-Solomon decoding, 266
to find error locator, 266
Euclidean distance, 12
Euclidean domain, 180
Euclidean function, 180
Euler ϕ function, 185, 229
Euler integration formula, 367
Euler's theorem, 186
evaluation homomorphism, 190, 191
exact sequence, 312
EXIT chart, 619
LDPC code, 660
exit1.m, 660
exit2.m, 660
exit3.m, 660
expurgated block code, 106
extended code, 106
Reed-Solomon, 276
extension field, 196
extrinsic information, 603
extrinsic probability, 582, 600
LDPC decoder, 643

F

factor graph
definition, 686
Forney, 708
normal, 708
factor group, 71
factorization step, GS decoder, 324, 330
factorization theorem, 332
fadepbplot.m, 714
fadeplot.m, 712
fading channel, 710
flat, 712
quasistatic, 713
Rayleigh, 712, 713
family (set), 457
Fano algorithm, 511, 517
Fano metric, 511, 513
fanoalg.m, 517
fanomet.m, 515
fast Hadamard transform, 382
feedback encoder, 454
feedforward encoder, 454
Feng-Tzeng algorithm, 338
fengtzeng.m, 341
Fermat's little theorem, 186
fht.cc, 383
fht.m, 383
field, 73, 193
finite, 193
filter bank, 699
finddfree, 506
finite field, 193
finite geometry, 668
finite impulse response, 129
Fire code, 433
flow graph simplification, 494
formal derivative, 232, 263, 289
formal series, 494
Forney factor graph, 708
Forney's algorithm, 262
forward pass
BCJR algorithm, 593
forward-backward algorithm, 593
factor graph representation, 696
free distance, 495
free Euclidean distance, 541
free module, 303
frequency domain decoding, 275
frequency domain vector, 272
freshman exponentiation, 201
fromcrt.m, 189
fromcrtpolym., 189
fundamental parallelotope, 564
fundamental theorem of algebra, 196
fundamental volume, 564
fyx0.m, 695

G

galdec.cc, 675
galdec.h, 675
galdecode.m, 648, 675
Gallager codes, 634
Galois biography, 197
Galois Field
example, 196
Galois field Fourier transform, 269
galtest.cc, 675
galtest2.cc, 675
γ (BCJR algorithm), 593
gap, x
Gaussian integers, 232
gaussj2, 86
gaussj2.m, 635
GCD (greatest common divisor), 176
gcd.c, 181
gcdpoly.cc, 224
gdl.m, 695
generalized minimum distance, 368, 441
generalized Reed-Solomon code, 277
generating function, 218
generator
matrix, 84
matrix, lattice, 563
of cyclic group, 66
of principal ideal, 119
polynomial, 121
genrm.cc, 376
genstdarray.c, 91
getinf.m, 660
getinfs.m, 660
GF(16) table, 198
$GF(q)$, 197
GF2.h, 224
GFFT, 269
GFNUM2m.cc, 224
GFNUM2m.h, 224
Gilbert-Varshamov bound, 111, 409
girth
of a graph, 457, 678
GMD (generalized minimum distance) decoding, 368
Golay code, 398
algebraic decoder, 400
arithmetic decoder, 401
golayrith.m, 402
golaysimp.m, 401
good codes, 637
Goppa code, 278
Gröbner basis, 368
Gram matrix, 564
graph
algorithms, 680
bipartite, 456, 457, 638
definitions, 456
simple, 457
Tanner, 638
tree, 457
trellis, 456
Gray code, 23
order, 13
greatest common divisor, 176
Green machine, 383
Griesmer bound, 411
ground field, 201
group
cyclic, 66
definition, 62
Guruswami-Sudan algorithm, 322

H

$H(X)$ (entropy), 5, 40
$H_2(x)$, 4
h2.m, 45
Hadamard
code, 374
matrix, 369
transform, 95, 369, 683
transform, decoding, 379
transform, fast, 382
Hadamard code, 374
hadex.m, 382
hamcode74pe.m, 36
Hamming bound, 89, 406
asymptotic, 422
Hamming code, 34, 53, 97
decoder, 141
dual, 97
encoder circuit, 135
extended, 377
message passing decoder, 654, 694
Tanner graph, 39, 694
Hamming distance, 24
Hamming sphere, 29, 89, 406
Hamming weight, 83
Hammsphere, 89
hard-decision decoding
convolution code, 484
Hartman-Tzeng bound, 239, 368
Hasse
derivative, 329, 330
theorem, 328

hill climbing, 667
historical milestones, 40
homomorphism, 72
horizontal step, 644
Horner's rule, 283
Huffman code, 6

I

$I(X, Y)$, 42
i.i.d., 24, 46
ideal of ring
  definition, 118
  principal, 119
identity, 62
IEEE wireless standard, 721
importance sampling, 60
increased distance factor, 541
induced operation, 69
inequality
  arithmetic-geometric, 334
  Cauchy-Schwartz, 411
  information, 59
information inequality, 59
information theory, 40
information theory inequality, 42
injective, 70
inner product, 11, 78
input redundancy weight enumerating function (IRWEF), 615
interference suppression, 734
interleaver, 425
  block, 426
  convolutional, 427
  cross, 427
  turbo code, 584, 614
interpolating polynomial, 191
interpolation, 190
  step, GS decoder, 324, 330
interpolation theorem, 331
intersection of kernels, 342
invariant factor decomposition, 464
inverse, 62
invmodp.m, 341
irreducible, 159
irreducible polynomial, 196, 207
  number of, 218
IRWEF, 615
ISBN, 3
isomorphism, 70
  ring, 118

J

Jacobian logarithm, 609
Jacobsthal matrix, 373
Jakes method, 712
jakes.m, 712
Justeson codes, 290

K

Kalman filter, 610
kernel, 304, 310, 312
  function, local, 682
  of homomorphism, 73
kernel function
  global, 682
key equation, 263, 266, 268
  Welch Berlekamp, 297
Kirkman triple, 669
Klein 4-group, 64, 80
Kötter algorithm, 342
kotter.cc, 346
kotter1.cc, 350
Krawtchouk polynomial, 415, 416
  properties, 423
krawtchouk.m, 415
Kronecker
  construction of Reed-Muller, 391
  product, 370
    properties, 370
Kronecker product theorem, 370
Kullback-Leibler distance, 41, 606, 621

L

Lagrange interpolation, 192, 303
Lagrange's theorem, 68
latency, 426
latin rectangle, 669
latta2.m, 568
lattice, 72, 563
  code, 567
lattstuff.m, 563
lattz2m, 568
Laurent series, 452
Lbarex.m, 357
LCM (least common multiple), 235
LDGM codes, 671
LDPC code, 634
  arbitrary alphabet, 647
  combinatoric constructions, 669
  concentration principle, 637
  decode threshold, 658
  definition, 635
  density evolution, 655
  difference set, 669
  eigenmessage, 679
  EXIT chart, 660
  fast encoding, 669
  finite geometry, 668
  irregular, 660
  iterative hard decoder, 677
  Kirkman triple, 669
  latin rectangle, 669
  Steiner designs, 669
  sum product decoding, 648, 678
  use of BCJR with, 646
ldpc.m, 648, 675
ldpclogdec.m, 652
ldpcsim.mat, 660
leading coefficient, 327
leading monomial, 327
leading term, 119
least common multiple, 226, 229
least-squares, 90
left inverse, 165
Legendre symbol, 372, 403
Lempel-Ziv coding, 6
lengthened block code, 106
lexicographic order, 326
LFSR, 154, 170, 234, 290
  for extension field, 199
likelihood
  function, 16
  ratio, 19
limsup, 407
linear code
  definition, 83
  dimension, 83
  generator matrix, 84
  rate, 83
linear combination, 76
linear feedback shift register, *see* LFSR
linear function, 232
linear programming, 413
  bound, 414
linearly dependent, 77
list decoder, 31, 293, 322
local kernel function, 682
log likelihood
  algebra, 735
  arithmetic, 611
  ratio, 19
loghist.m, 660
low density generator matrix codes, 671
low density parity check, *see* LDPC
lpboundex.m, 418

M

M algorithm, 521
MacWilliams identity, 95, 109
magma, x
majority logic decoding, 384
makeB.m, 549
MakeLFSR, 162
makgenfromA.m, 635
MAP
  algorithm, 588
  decoding, factor graph representation, 685
  detection, 17
marginalize product of functions (MPF), 682
marginalizing, 680
Markov property
  convolutional codes, 590
Markov source, 588
Mason's rule, 494, 498
masseymodM.m, 258
matched filter, 15
  matrix, 716
Mattson-Solomon polynomial, 289
max-log-MAP algorithm, 608
maximal ratio combiner, 717
maximal-length
  sequence, 155, 159
  shift register, 234
maximal-length code, 97, 167
maximum a posteriori detection, 17
maximum distance separable (MDS), 88
maximum likelihood
  decoder, 30, 322
  decoder, factor graph representation, 684
  detection, 18
  sequence estimator, 469
maximum-likelihood sequence estimator
  vector, 716
McEliece public key, 280
MDS code, 88, 245, 246, 287
  extended RS, 276
  generalized Reed-Solomon, 277
  weight distribution, 246
Meggitt decoder, 139

memoryless channel, 25
Mersenne prime, 234
message passing, 649, 689
message passing rule, 690
message-passing, 682
metric quantization, 484
milestones, historical, 40
MIMO channel, 714
  narrowband, 716
mindist.m, 34
minimal basic convolutional encoder, 465
minimal basic encoder, 465
minimal polynomial, 209, 212
minimum distance, 29
ML detection, *see* maximum likelihood detection
MLFSR code, 97
ModAr.cc, 223
ModAr.h, 223
ModArnew.cc, 223
module
  definition, 302
  free, 303
Moebius function, 222
moment generating function, 532
monic, 119
monoid, 681, 736
monomial ordering, 325
multiplicative order, 211
multiplicity matrix, 359
multiplicity of zero, 328
mutual information, 42, 619

N

narrow sense BCH, 235
narrowband MIMO channel, 716
natural homomorphism, 73
nchoosektest.m, 36
Newton identities, 250, 285
nilpotent, 165
node error, 491
normal factor graph, 708
normalization
  alpha and beta, 595
  probability, 592
nullspace, 79, 87
Nyquist sampling theorem, 50

O

one-to-one, 70
onto, 70
ord, 328
order
  multiplicative, 211
  of a field element, 201
  of a finite group, 63
  of a group element, 67
  of zero, 328
ordered statistic decoding, 447
orthogonal, 11, 79
  on a bit, 385
orthogonal complement, 79
orthogonal design
  complex, 727
  generalized complex, 727
  generalized real, 726
  real, 723
orthogonal matrix, 563
orthogonal polynomial, 419
orthonormal, 11
output transition matrix, 549

P

$P(E)$, 98
Padé approximation, 228, 234
Paley construction, 371
parallel concatenated code, 582, 584
parity, 85
  overall, 106
parity check
  code, 107
  equations, 87
  matrix, 34, 86
  polynomial, 123
  probability, 678
  symbols, 85
partial syndrome, 523
partition, 68
partition chain, 567
path
  algorithm, shortest, 472
  enumerator, 493
  in graph, 457
  merging, 474
  metric, 473
  survivor, 474
P_b, 98
$P_b(E)$, 98
$P_d(E)$, 99
P_{d_b}, 99
peak-to-average power ratio, 562
perfect code, 89, 93
permutation, 64
permuter, 584
perp, 79
Peterson's algorithm, 251
ϕ function, 185, 229
phifun.m, 51
philog.m, 51
pi2ml, 362
pivottableau.m, 413
plotbds.m, 407
plotcapcmp.m, 45
plotcbawn2.m, 51
plotconprob.m, 504
Plotkin bound, 111, 410
  asymptotic, 421
polyadd.m, 116
polyaddm.m, 116
polydiv.m, 116
polymult.m, 116
polymultm.m, 116
polynomial
  irreducible, 196
  Krawtchouk, 415
  minimal, 212
  orthogonal, 423
  primitive, 208
polynomial division
  circuits, 129
polynomial encoder, 463
polynomial multiplication
  circuits, 128
polynomial ring, 115
polynomialT.cc, 223
polynomialT.h, 223
polysub.m, 116
polysubm.m, 116
power sum symmetric functions, 250
primfind, 209
primfind, 209
primitive BCH code, 235
primitive element, 202, 396
primitive polynomial, 155, 160, 208
  table of, 209
primitive.txt, 155
principal character, 415
principal ideal, 119
probability of bit error, 98
probability of decoder error, 98
probability of decoder failure, 99
probability of undetected codeword error, 98
product code, 430
progdet.m, 100
progdetH15.m, 100
pseudonoise, 154
pseudorandom sequence, 8
Psi.m, 658
psifunc.m, 656
Psiinv.m, 658
$P_u(E)$, 98
P_{u_b}, 99
puncture
  block code, 106
  convolutional code, 507
  matrix, 508
  Reed-Solomon, 276

Q

Q function, 20
  bounds, 57, 503, 504
QAM (quadrature-amplitude modulation), 535
QAM constellation
  energy requirements, 536
qf.c, 20
qf.m, 20
quadratic residue, 371
  code, 396
quadrature-amplitude modulation (QAM), 535
quantization of metric, 484
quotient ring, 116

R

R-linear combination, 302
random code, 637
random error correcting
  capability, 30, 93
  codes, 425
rank criterion, 723
rank of polynomial, 327
rate, 28
  asymptotic, 407
  of convolutional code, 452
  of linear code, 83
rate compatible punctured codes, 510, 533
rate-distortion theory, 7, 51
rational encoder, 463
rational interpolation, 302
Rayleigh density, 713, 733
Rayleigh fading, 712
  channel, 713
real orthogonal design, 723
realizations
  controller form, 132, 453

division, 130
first-element first, 132
multiplication
first-element first, 129
last-element first, 128
multiplication and division
first-element first, 132
observability form, 132, 454
polynomial multiplication, 128
reciprocal polynomial, 166, 231
recursive systematic convolutional (RSC), 582, 584
reducefree.m, 413
reducible solution, 304
redundancy, 89
Reed-Muller code, 376
Reed-Solomon code, 242
burst correction, 431
decoder workload, 293
generalized, 277
Guruswami-Sudan Decoder, 322
pipelined decoder algorithm, 310
programming, 284
soft output decoder, 358
soft-input soft-output decoder, 699
weight distribution, 246
reedsolwt.m, 246
reflexive property, 68
register exchange, 482
relative distance, 406
relative entropy, 41, 606
relatively prime, 176
reliability, 440, 735
matrix, 359
reliability class, 442
remainder decoding, 293
repcodeprob.m, 32
repcodes.m, 33
repeat-accumulate (RA) code, 586, 671
repetition code, 28, 676
residue class, 71
restorefree.m, 413
restriction to $GF(q)$, 277
reversible code, 166, 289
right inverse
in a ring, 165
of a matrix, 462
ring
characteristic, 115
definition, 114
polynomial, 115
quotient, 116
rmdecex.m, 381
rmdecex2.m, 387
Roos bound, 239, 368
root of unity, 215
rotational invariance, 556, 562
TCM codes, 556
Roth-Ruckenstein algorithm, 350
rothruck.cc, 354
rothruck.h, 354
row space, 77
RSA encryption, 186, 187
RSdec.cc, 284
RSdec.h, 284
rsdecode.cc, 285
RSenc.cc, 284
RSenc.h, 284
rsencode.cc, 285
runlength-limited codes, 8

S

scaling, 592
self-dual code, 86, 399, 403, 404
semiring, 681
separation theorem, 7, 51
sequential decoding, 511
serially concatenated code, 586, 671
set partitioning, 545
Shannon sampling theorem, 50
shape gain, 568
shortened block code, 106
signal constellation, 10, 535
signal energy, 13
signal shape, 562
signal space, 10
signal-to-noise ratio (SNR), 21
simplex code, 97, 109, 374, 423
simplex1.m, 413
Singleton bound, 88, 406
SISO decoding
Reed Solomon, 699
turbo code, 587
S_n, 65
SNR (signal to noise ratio), 21
soft-decision decoding, 439
BCJR algorithm, 588
convolution code, 484
LDPC, 640
performance, 103
soft input, 32
soft output Viterbi algorithm, 610
turbo code, 582, 587
soft-input, hard-output (SIHO), 439
soft-input, soft-output (SISO), 439
sort
for soft detection, 441
source code, 6
source coding theorem, 6
source/channel coding, 7
SOVA, 469, 610
space-time code
block, 719
trellis, 728
spanning set, 77
spanning tree, 702
sparse matrix, 635
sparseHno4.m, 668
spectral efficiency, 536
TCM advantage, 562
spectral thinning, 614
spectrum, 272
sphere packing, 562
splitting field, 204
spread-spectrum, 8
squaring construction, 392
stack algorithm, 511, 515
stack bucket, 515
stackalg.m, 515
standard
802.11, 721
ISBN, 3
V.32, V.33, 557
V.34, 561, 571
standard array, 91
Steiner designs, 669
Stirling formula, 408, 570
derivation, 421
stretching, 701
subcode, 143
subfield, 206
subcode, 277, 288
subgroup, 65
proper, 65
sublattice, 566
subspace of vector space, 78
Sugiyama algorithm, 182, 266
sum-product algorithm, 690
LDPC codes, 648
summary notation, 682
support set, 242
surjective, 70
survivor path, 474
Sylvester construction, 371
symmetric group, 65
symmetric property, 68
synchronization
of convolutional decoders, 486
syndrome, 90
BCH, RS, 247
decoding, 94
polynomial, 123
systematic
convolutional code, 453, 469
definition, 85
encoding, cyclic codes, 124

T

T algorithm, 522
tail biting, 734
tail biting code, 522
tanh rule, 676, 696, 707, 735, 736
0-1 valued variables, 737, 738
Tanner graph, 38, 638
TCM
multidimensional, 561
Ungerboeck framework, 544
tcmrot2.cc, 557
tcmt1.cc, 549
TCP/IP protocol, 426
testBCH.cc, 283
testbcjr.cc, 629
testBinLFSR.cc, 162
testBinPolyDiv.cc, 162
testBM.cc, 282
testChien.cc, 283
testconvdec.cc, 529
testconvenc, 526
testcrp.m, 189
testcrt.m, 189
testfht.cc, 383
testft.m, 341
testgcdpoly.cc, 224
testgdl2.m, 695
testgfnum.cc, 224
testGolay.cc, 401
testGS1.cc, 346
testGS2.cc, 354
testGS3.cc, 347

testGS5.cc, 350
testmodar1.cc, 223
testmodarnew.cc, 223
testpoly1.cc, 223
testpxy.cc, 325
testQR.cc, 397
testrepcode.cc, 33
testRS.cc, 284
teststack.m, 515
testturbodec2.cc, 629
threshtab.m, 658
tier of parity checks, 640
time domain vector, 272
tocrt.m, 189
tocrtpoly.m, 189
Toeplitz matrix, 122, 251
total order, 325
totient function, 185
trace (of a field element), 232
traceback, 483
transfer function
  bound, 552
  matrix, 453
  of graph, 493
transitive property, 68
trellis, 456
  for block code, 38, 523
  time-varying, 696
trellis coded modulation, 535, *see* TCM
triangle inequality
  Hamming distance and, 57
truncation error, 482
turbo code, 582
  block coding, 623
  decoding, 601
  error floor, 612
  EXIT chart, 619
  extrinsic information, 603
  hard decision aided termination, 608
  interleaver, 584
  likelihood ratio decoding, 602
  parallel concatenated code, 582
  primitive polynomials and, 632
  sign change ratio, 607
  stopping criteria, 605, 606
  terminal state, 602
turbo equalization, 626
typical set, 46

U
UDP protocol, 104, 105, 426
undetected bit error rate, 99
unequal error protection, 489, 522
Ungerboeck coding framework, 544
unimodular matrix, 464, 563
union bound, 22
  block code performance and, 103
  convolutional code performance, 499
  TCM performance, 546, 547
unit of a ring, 115
up to isomorphism, 80
UPC, 3
utiltkm.cc, 440
utiltkm.h, 440

V
V.32 standard, 557
V.33 standard, 557
V.34 standard, 561, 571
valuation, 180
Vandermonde matrix, 237
variable-rate error control, 509
vector space definition, 75
vertex, 457
vertical step, 641
Viterbi algorithm, 469, 471
  hard decisions, 588
  soft metric, 485
  soft output, 610
voln.m, 563
$V_q(n,t)$, 57, 89

W
waterfall region, 584
weight, 635
weight distribution, 95
  BCH code, 239
  RS code, 245
weight enumerator, 95
weight profile, 446
weighted code, 3, 56
weighted degree, 325, 327
  order, 326
Welch-Berlekamp algorithm, 293, 303
well defined, 70
Wilson's theorem, 225
Wolf trellis, 38, 523
writesparse.m, 637

Y
y-root, 350

Z
Zech logarithm, 204
zero
  multiplicity of, 329
zero divisor, 165, 193
zero state forcing sequence, 588
ZJ algorithm, 511, 515