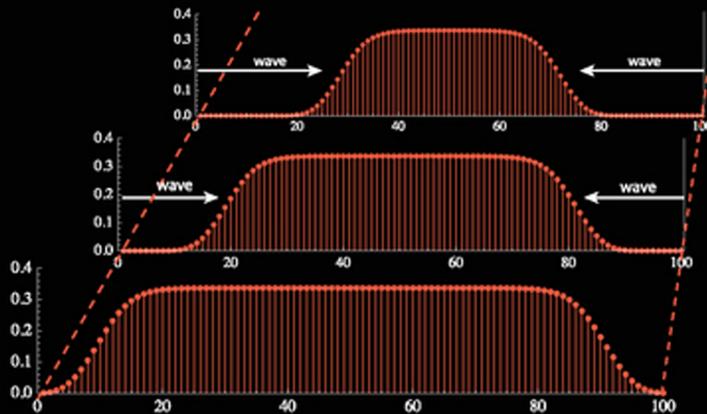
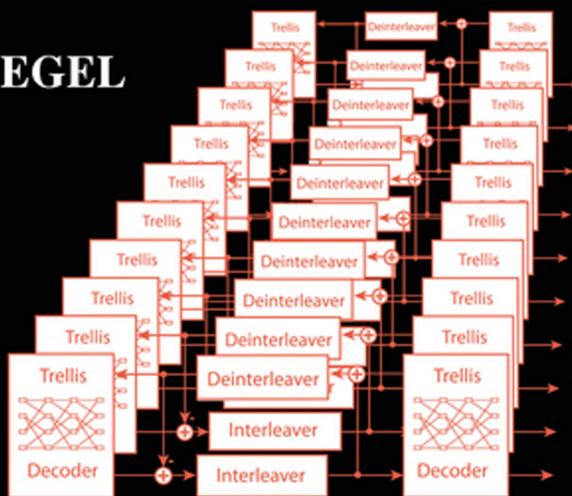


Trellis and Turbo Coding

Iterative and Graph-Based Error Control Coding
SECOND EDITION

CHRISTIAN B. SCHLEGEL
LANCE C. PÉREZ



IEEE SERIES ON
DIGITAL
& MOBILE
COMMUNICATION

John B. Anderson, Series Editor

IEEE
IEEE PRESS

WILEY

TRELLIS AND TURBO CODING

IEEE Press
445 Hoes Lane
Piscataway, NJ 08854

IEEE Press Editorial Board
Tariq Samad, *Editor in Chief*

George W. Arnold
Dmitry Goldgof
Ekram Hossain
Mary Lanzerotti

Vladimir Lumelsky
Pui-In Mak
Jeffrey Nanzer
Ray Perez

Linda Shafer
Zidong Wang
MengChu Zhou
George Zobrist

Kenneth Moore, *Director of IEEE Book and Information Services (BIS)*

Technical Reviewers

Todd Moon, *Utah State University*
Claudio Sacchi, *Utah State University*

TRELLIS AND TURBO CODING

Iterative and Graph-Based Error Control Coding

Second Edition

**CHRISTIAN B. SCHLEGEL
LANCE C. PÉREZ**



WILEY

Copyright © 2015 by The Institute of Electrical and Electronics Engineers, Inc.

Published by John Wiley & Sons, Inc., Hoboken, New Jersey. All rights reserved.

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) 750-4470, 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, or online at <http://www.wiley.com/go/permission>.

Limit of Liability/Disclaimer of Warranty: While the publisher and author have used their best efforts in preparing this book, they make no representations 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 or for technical support, please contact our Customer Care Department within the United States at (800) 762-2974, outside the United States 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 may not be available in electronic format. For information about Wiley products, visit our web site at www.wiley.com.

Library of Congress Cataloging-in-Publication is available.

ISBN 978-1-118-083161

Printed in the United States of America.

10 9 8 7 6 5 4 3 2 1

Contents

1	Introduction	1
1.1	Modern Digital Communications	1
1.2	The Rise of Digital Communications	4
1.3	Communication Systems	4
1.4	Error Control Coding	7
1.5	Bandwidth, Power, and Complexity	12
1.6	A Brief History—The Drive Towards Capacity	20
2	Communications Basics	27
2.1	The Probabilistic Viewpoint	27
2.2	Vector Communication Channels	29
2.3	Optimum Receivers	31
2.4	Matched Filters	33
2.5	Message Sequences	35
2.6	The Complex Equivalent Baseband Model	39
2.7	Spectral Behavior	44
2.8	Advanced Modulation Methods	46
2.8.1	OFDM	46
2.8.2	Multiple Antenna Channels (MIMO Channels)	48
2.9	A Communications System Case Study	53
2.10	Appendix 2.A	61
3	Trellis-Coded Modulation	67
3.1	An Introductory Example	67
3.2	Construction of Codes	71
3.3	Lattices	80
3.4	Lattice Formulation of Trellis Codes	86
3.5	Rotational Invariance	92
3.6	V.fast	99

3.7	The IEEE 802.3an Standard	101
3.8	Historical Notes	106
4	Trellis Representations	111
4.1	Preliminaries	111
4.2	The Parity-Check Matrix	112
4.3	Parity-Check Trellis Representations	113
4.4	Convolutional Codes and Their Trellis	115
4.5	Minimal Trellises	120
4.6	Minimum-Span Generator Matrices	124
4.7	Systematic Construction of the PC-Trellis	127
4.8	Tail-Biting Trellises	129
4.9	The Minimal Trellis of Convolutional Codes	133
4.10	Fundamental Theorems from Basic Algebra	139
4.11	Systematic Encoders	149
4.12	Maximum Free-Distance Convolutional Codes	151
4.13	The Squaring Construction and the Trellis of Lattices	154
4.14	The Construction of Reed–Muller Codes	161
4.15	A Decoding Example	163
4.16	Polar Codes and Their Relationship to RM Codes	166
	Appendix 4.A	171
5	Trellis and Tree Decoding	179
5.1	Background and Introduction	179
5.2	Tree Decoders	181
5.3	The Stack Algorithm	183
5.4	The Fano Algorithm	185
5.5	The M -Algorithm	186
5.6	Maximum Likelihood Decoding	197
5.7	A Posteriori Probability Symbol Decoding	200
5.8	Log-APP and Approximations	207
5.9	Error Analysis and Distance Spectrum	211
5.10	Random Coding Analysis of Optimal Decoding	222
5.11	Random Coding Analysis of Sequential Decoding	232
5.12	Some Final Remarks	238
6	Low-Density Parity-Check Codes	249
6.1	Introduction	249
6.2	LDPC Codes and Graphs	251
6.3	LDPC Decoding via Message Passing	255

6.4	Analysis Techniques	259
6.4.1	(Error) Probability Evolution for Binary Erasure Channels	259
6.4.2	Error Mechanism of LDPCs on BECs	265
6.4.3	Binary Symmetric Channels and the Gallager Algorithms	266
6.4.4	The AWGN Channel	270
6.5	Code Families and Construction	281
6.5.1	Constructions with Permutation Matrices	281
6.5.2	Cycle Reduction Design	286
6.5.3	RS-based Construction	287
6.5.4	Repeat-Accumulate Codes	289
6.6	Encoding of LDPC Codes	291
6.6.1	Triangular LDPC Codes	292
6.6.2	Specialized LDPC Codes	295
6.6.3	Approximate Triangularization	296
	Appendix 6.A	298
7	Error Floors	319
7.1	The Error Floor Problem	319
7.2	Dynamics of the Absorption Sets	323
7.3	Code Design for Low Error Floors	331
7.4	Impact of the Decoding Algorithm	335
7.5	Importance Sampling (IS)	336
7.6	Computing Error Rates via Importance Sampling	340
8	Turbo Coding: Basic Principles	351
8.1	Introduction	351
8.2	Parallel Concatenated Convolutional Codes	353
8.3	Distance Spectrum Analysis of Turbo Codes	356
8.4	The Free Distance of a Turbo Code	358
8.5	Weight Enumerator Analysis of Turbo Codes	364
8.6	Iterative Decoding of Turbo Codes	371
8.7	EXIT Analysis	376
8.8	Serial Concatenation	383
8.9	Cascaded Convolutional Codes	383
8.10	Weight Enumerator Analysis of SCCC _s	385
8.11	Iterative Decoding and Performance of SCCC _s	394
8.12	EXIT Analysis of Serially Concatenated Codes	397
8.13	Viewpoint	401
8.14	Turbo-Trellis-Coded Modulation	402
8.15	Serial Concatenation	406

8.16 EXIT Analysis of Serial TTCM	408
8.17 Differential-Coded Modulation	409
8.18 Concatenated Space-Time Coding	414
8.19 Bit-Interleaved Coded and Generalized Modulation	418
9 Turbo Coding: Applications	431
9.1 Interleavers	431
9.2 Turbo Codes in Telecommunication Standards	439
9.2.1 The Space Data System Standard	439
9.2.2 3G Wireless Standards	440
9.2.3 Digital Video Broadcast Standards	443
9.3 Product Codes and Block Turbo Decoding	446
9.4 Approximate APP Decoding	448
9.5 Product Codes with High-Order Modulations	451
9.6 The IEEE 802.16 Standard	453
9.7 Decoding of Polar Codes	454
9.8 Polar Code Performance and Outlook	458
10 Convolutional LDPC Codes and Spatial Coupling	465
10.1 Capacity: The Ultimate Limit	465
10.2 Low-Density Parity-Check Convolutional Codes	467
10.2.1 New LDPC Codes from Old	467
10.2.2 Decoding Convolutional LDPC Codes	472
10.3 Spatial Coupling: A General View	474
10.4 Spatial Coupling: Convergence Analysis	482
10.4.1 Problem Setup	482
10.4.2 Lyapunov Approach	483

Chapter 1

Introduction

1.1 Modern Digital Communications

With the advent of high-speed logic circuits and very large scale integration (VLSI), data processing and storage equipment has inexorably moved towards employing digital techniques. In digital systems, data is encoded into strings of zeros and ones, corresponding to the on and off states of semiconductor switches. This has brought about fundamental changes in how information is processed. While real-world data is primarily in “analog form” of one type or another, the revolution in digital processing means that this analog information needs to be encoded into a digital representation, e.g., into a string of ones and zeros. The conversion from analog to digital and back are processes which have become ubiquitous. Examples are the digital encoding of speech, picture, and video encoding and rendering, as well as the large variety of capturing and representing data encountered in our modern internet-based lifestyles.

The migration from analog communications of the first half of the 20-th century to the now ubiquitous digital forms of communications were enabled primarily by the fast-paced advances in high-density device integration. This has been the engine behind much of the technological progress over the last half century, initiated by the creation of the first integrated circuit (IC) by Kilby at Texas Instruments in 1958. Following Moore’s informal law, device sizes, primarily CMOS (Complementary Metal-Oxide Semiconductors), shrink by a factor two every two years, and computational power doubles accordingly. An impression for this exponential growth in computing capability can be gained from Figure 1.1, which shows the number of transistors integrated in a single circuit and the minimum device size for progressive fabrication processes – known as *implementation nodes*.

While straightforward miniaturization of the CMOS devices is becoming increasingly more difficult, transistor designers have been very creative in modifying the designs to stay on the Moore trajectory. As of 2015 we now see the introduction of 3-dimensional

transistor structures such as thin FETs, double-gated FETs, and tunnel FETs, and it is expected that carbon nanotube devices may continue miniaturization well into the sub-10 nm range. In any case, the future for highly complex computational devices is bright.

CPU Transistor Counts 1970-2020

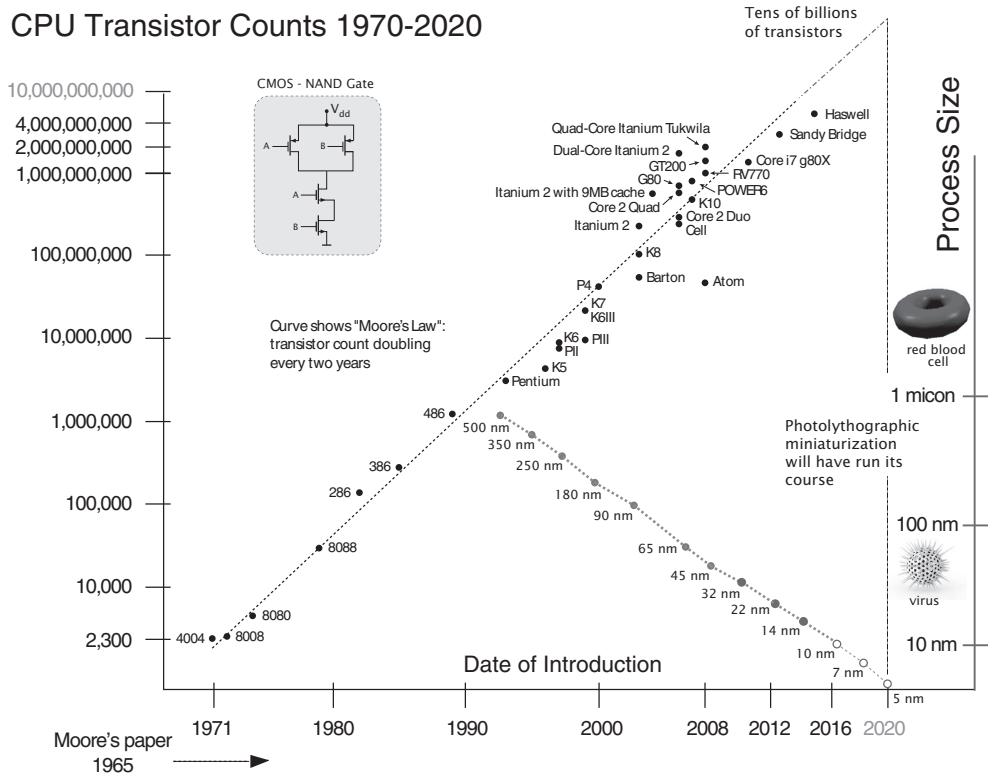


Figure 1.1: Moore's law is driving progress in electronic devices. Top left: A basic CMOS switching structure. Bottom left: Moore observed his “doubling law” in 1965 and predicted that it would continue “at least another 10 years.”

One such computational challenge is data communications: in particular data integrity, as discussed in this book. The migration from analog to digital information processing has opened the door for many sophisticated algorithmic methods. Digital information is

treated differently in communications than analog information. Signal estimation becomes signal detection; that is, a communications receiver need not look for an analog signal and make a “best” estimate, it only needs to make a decision between a finite number of discrete signals, say a one or a zero in the most basic case. Digital signals are more reliable in a noisy communications environment; they can usually be detected perfectly, as long as the noise levels are below a certain threshold. This allows us to restore digital data, and, through error correcting techniques, correct errors made during transmission. Digital data can easily be encoded in such a way as to introduce dependency among a large number of symbols, thus enabling a receiver to make a more accurate detection of the symbols. This is the essence of *error control coding*.

Finally, there are also strong theoretical reasons behind the migration to digital processing. Nyquist’s sampling theorem, discussed in Section 1.3, tells us that, fundamentally, it is sufficient to know an analog signal at a number of discrete points in time. This opens the door for the discrete time treatment of signals. Then, Shannon’s fundamental channel coding theorem states that the values of these discrete time samples themselves, can contain only a finite amount of information. Therefore, only a finite amount of discrete levels are required to capture the full information content of a signal.

The digitization of data is convenient for a number of other reasons too. The design of signal processing algorithms for digital data is much easier than designing analog signal processing algorithms, albeit not typically less complex. However, the abundance of such digital algorithms, including the error control and correction techniques discussed in this book, combined with their ease of implementation in *very large scale integrated* (VLSI) circuits has led to the plethora of successful applications of error control coding we see in practice today.

Error control coding was first applied in deep-space communications where we are confronted with low-power communications channels with virtually unlimited bandwidth. On these data links, convolutional codes (Chapter 4) are used with sequential and Viterbi decoding (Chapter 5), and the future will see the application of turbo coding. The next successful application of error control coding was to storage devices, most notably the compact disk player, which employs powerful Reed-Solomon codes [21] to handle the raw error probability from the optical readout device which is too large for high-fidelity sound reproduction without error correction. Another hurdle taken was the successful application of error control to bandwidth-limited telephone channels, where trellis-coded modulation (Chapter 3) was used to produce impressive improvements and push transmission rates towards the theoretical limit of the channel. Nowadays, coding is routinely applied to satellite communications [41, 49], teletext broadcasting, computer storage devices, logic circuits, semiconductor memory systems, magnetic recording systems, audio-video, and WiFi systems. Modern mobile communications systems like the pan-European TDMA digital telephony standard GSM [35], IS 95 [47], CDMA2000, IMT2000, and the new 4-th generation LTE and LTE-A standards [63, 64] all use error control coding.

1.2 The Rise of Digital Communications

Modern digital communication theory started in 1928 with Nyquist's seminal work on telegraph transmission theory [36]. The message from Nyquist's theory is that finite bandwidth implies discrete time. That is, a signal whose bandwidth is limited can always be represented by sample values taken at discrete time intervals. The sampling theorem of this theory then asserts that the band-limited signal can always be reconstructed *exactly* from these discrete-time samples.¹ Only these discrete samples need to be processed by a receiver since they contain all the necessary information of the entire waveform.

The second pillar to establish the supremacy of digital information processing came precisely from Shannon's 1948 theory. Shannon's theory essentially establishes that the discrete-time samples which are used to represent a bandlimited signal, could be adequately described by a finite number of amplitude samples, the number of which depended on the level of the channel noise. These two theories combined state that a finite number of levels taken at discrete time intervals are completely sufficient to characterize any bandlimited signal in the presence of noise, that is, in any communication system.

With these results, technology has moved towards a complete digitization of communications systems, with error control coding being the key to realize the sufficiency of discrete amplitude levels. We will study Shannon's theorem in more detail in Section 1.5.

1.3 Communication Systems

Figure 1.2 shows the basic configuration of a point-to-point digital communications link. The data to be transmitted over this link can either come from some analog source, in which case it must first be converted into digital format (digitized), or it can be a digital information source. If this data is a speech signal, for example, the digitizer is a speech codec [22]. Usually the digital data is source encoded to remove unnecessary redundancy from the data, i.e., the source data is compressed [14]. Source encoding has the effect that the digital data which enters the encoder has statistics which resemble that of a random symbol source with maximum entropy, i.e., all the different digital symbols occur with equal likelihood, and are statistically independent. The channel encoder operates on this compressed data and introduces controlled redundancy for transmission over the channel. The modulator converts the discrete channel symbols into waveforms which are transmitted through the waveform channel. The demodulator reconverts the waveforms

¹Since it is not shown elsewhere in this book, we present Nyquist's sampling theorem here. It is given by the following exact series expansion of the function $s(t)$ which is bandlimited to $[-1/2T, 1/2T]$:

$$s(t) = \sum_{i=-\infty}^{\infty} s(iT) \operatorname{sinc}\left(\frac{\pi}{T}(t - iT)\right); \quad \operatorname{sinc}(x) = \frac{\sin(x)}{x}.$$

back into a discrete sequence of received symbols, and the decoder reproduces an estimate of the compressed input data sequence, which is subsequently reconverted into the original signal or data sequence.

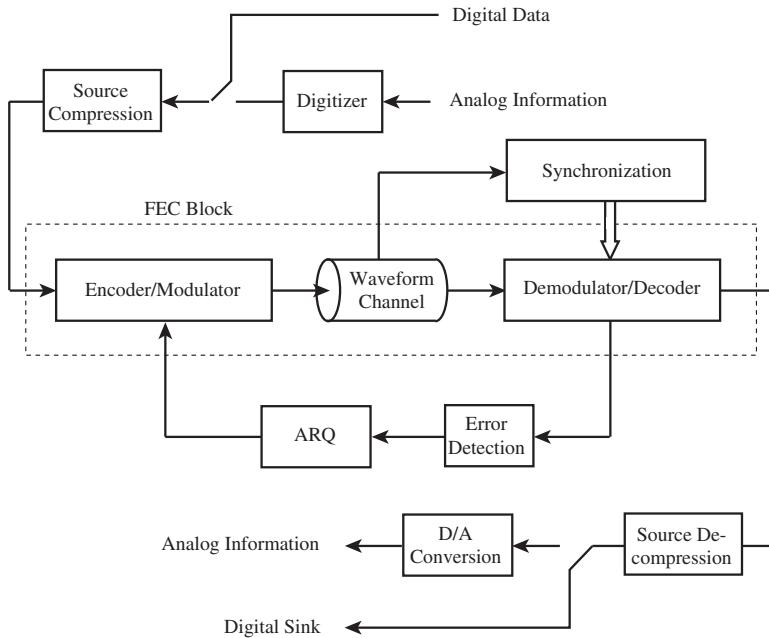


Figure 1.2: System diagram of a complete point-to-point communication system for digital data. The forward error control (FEC) block is the topic of this book.

An important ancillary function at the receiver is the synchronization process. We usually need to acquire carrier frequency and phase synchronization, as well as symbol timing synchronization in order for the receiver to be able to operate. Synchronization is not a topic of this book, and we will assume in most of our discussion that synchronization has been established (with the exception of phase synchronization in the case of rotationally invariant codes in Chapters 4 and 8). References [32] and [33] treat synchronization issues in detail. Since synchronization is a relatively slow estimation process, and data detection is a fast process, we usually have those two operations separated in real receiver implementations as indicated in Figure 1.2. However, we would like to note at this point that novel, iterative receiver designs are increasingly integrating these auxillary functions, for an example of phase synchronization see Chapter 8.

Another important feature in some communication systems is *automatic repeat request* (ARQ). In systems with ARQ the receiver also performs error detection, and, through a return channel, requests retransmission of erroneous data blocks, or data blocks which cannot be reconstructed with sufficient confidence [25]. ARQ can usually improve the data transmission quality substantially, but the return channel needed for ARQ is not always available, or may be impractical. For a deep-space probe on its way to the outer rim of our solar system, ARQ is infeasible since the return path takes too long (several hours!). For speech-encoded signals ARQ is usually impossible for an analogous reason, since only a maximum speech delay of about 200 ms is acceptable. In broadcast systems, ARQ is ruled out for obvious reasons. Error control coding without ARQ is termed *forward error correction* or *control* (FEC). FEC is more difficult to perform than simple error detection and ARQ, but dispenses with the return channel. Oftentimes, FEC and ARQ are both integrated into hybrid error control systems [25, 24] for data communications.

This book deals primarily with FEC—the dashed block in Figure 1.2. The reason why we can do this relatively easily is due to different functionalities of the various blocks just discussed, and the fact that they operate largely autonomously from each other. Each of them represents a separate entity with its own optimization strategy, and data is simply passed between the different blocks, sometimes with little extra mutual interaction. A notable point in Figure 1.2 is that the encoder/modulator and the demodulator/decoder are combined operations. This is done to reflect the fact that error protection and modulation—in the sense of choosing the discrete signal points that represent the digital data—is a process best addressed jointly. This view of joint encoding/modulation was first proposed by Wozencraft and Kennedy [61] and Massey [30] and then realized with stunning results by Ungerböck [50, 51, 52] in the methods of *trellis-coded modulation* of the 1980’s.

Since we will assume the encoder input data to be a sequence of independent, identically and uniformly distributed symbols (courtesy of the source compression), the single most important parameter to optimize for the FEC block is arguably the bit and/or symbol error rate, and we will adopt this as our criterion for the quality of an FEC system. Note that this is not necessarily the most meaningful measure in all cases. Consider, for example, pulse-code-modulated (PCM) speech, where an error in the most significant bit is significantly more detrimental than an error in the least significant bit. Researchers have looked at schemes with unequal error protection for such applications (e.g., [18]). However, such methods usually are a variation of the basic theme of obtaining a minimum error rate. Occasionally we may have need for the frame-, or block error rate (FER), which describes the probability that an entire block of data is incorrectly received. Most communications protocols operate on the basis of frame errors, and frame error control is exercised at the upper layers of a communications protocol. While of course tightly connected, the frame-, and bit error rates do sometimes follow different tendencies. This will become important when we discuss error floors for very-low error rate applications, such as those using cable modem or optical fiber communications.

1.4 Error Control Coding

The modern approach to error control in digital communications started with the ground-breaking work of Shannon [45], Hamming [19], and Golay [16]. While Shannon presented a theory which explained the fundamental limits on the efficiency of communications systems, Hamming and Golay were the first to develop practical error control schemes. A new paradigm was born, one in which errors are not synonymous with data which is irretrievably lost; but by clever design, errors could be corrected, or avoided altogether. This new thinking was revolutionary. Even though Shannon's theory promised that large improvements in the performance of communication systems could be achieved, practical improvements had to be excavated by laborious work over half a century of intensive research. One reason for this lies in a fundamental shortcoming of Shannon's theory. While it clearly states theoretical limits on communication efficiency, its methodology provides no insight on how to actually achieve these limits, since it is based on sophisticated averaging arguments which eliminate all detailed system structure. Coding theory, on the other hand, evolved from Hamming and Golay's work into a flourishing branch of applied mathematics [27].

Let us see where it all started. The most famous formula from Shannon's work is arguably the channel capacity of an ideal band-limited Gaussian channel,² which is given by

$$C = W \log_2(1 + S/N) \text{ [bits/second].} \quad (1.1)$$

In this formula, C is the channel capacity, that is, the maximum number of bits which can be transmitted through this channel per unit time (second), W is the bandwidth of the channel, and S/N is the signal-to-noise power ratio at the receiver. Shannon's main theorem, which accompanies (1.1), asserts that error probabilities as small as desired can be achieved as long as the transmission rate R through the channel (in bits/second) is smaller than the channel capacity C . This can be achieved by using an appropriate encoding and decoding operation. However, Shannon's theory is silent about the structure of these encoders and decoders.

This new view was in marked contrast to early practices, which embodied the opinion that in order to reduce error probabilities, the signal energy had to be increased, i.e., the S/N had to be improved. Figure 1.3 shows the error performance of QPSK, a popular modulation method for satellite channels (see Chapter 2) which allows data transmission of rates up to 2 bits/symbol. The bit error probability (BER) of QPSK is shown as a function of the signal-to-noise ratio S/N per dimension normalized per bit (see Section 1.3), henceforth called SNR. It is evident that an increased SNR provides a gradual decrease in error probability. This contrasts markedly with Shannon's theory which promises zero(!) error probability at a spectral efficiency of 2 bits/s/Hz, which is the maximum that QPSK

²The exact definitions of these basic communications concepts are given in Chapter 2.

can achieve, as long as $\text{SNR} > 1.5$ (1.76 dB), shattering conventional wisdom. The limit on SNR is calculated using (1.1)—see Section 1.5.

Bit Error Probability (BER)

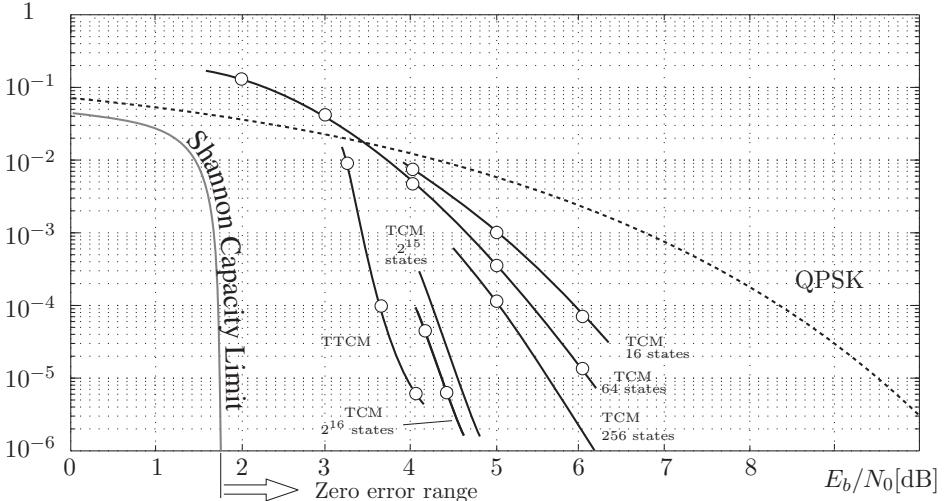


Figure 1.3: Bit error probability of quadrature phase-shift keying (QPSK) and selected 8-PSK trellis-coded modulation (TCM), trellis-turbo-coded (TTCM), and block-turbo-coded (BTC) systems as a function of the normalized signal-to-noise ratio.

Also shown in Figure 1.3 is the performance of several trellis-coded modulation (TCM) and trellis-turbo-coded (TTCM) schemes using 8-ary phase-shift keying (8-PSK) (Chapter 4), and the improvement made possible by coding becomes evident. The difference in SNR for an objective target bit error rate between a coded system and an uncoded system is termed the *coding gain*. Note that the coding schemes shown in Figure 1.3 achieves these gains without requiring more bandwidth than the uncoded QPSK system.

As we will discuss in Chapter 4, a trellis code is generated by a circuit with a finite number of internal states. The number of these states is a direct measure of its decoding complexity if maximum-likelihood decoding is used. Note that the two very large codes are not maximum-likelihood decoded, they are sequentially decoded [55]. The turbo-trellis-coded modulation (TTCM) system is based on turbo coding principles (Chapter 9) using two small concatenated trellis codes. Coding then helps to realize the promise of Shannon's theory which states that for a desired error rate of $P_b = 10^{-6}$ we can gain almost 9 dB in expended signal energy over QPSK. This gain can be achieved by converting required

signal power into decoder complexity, as is done by the TCM and TTCM coding methods.

Incidentally, 1948, the year Shannon published his work, is also the birth year³ of the transistor, arguably the 20-th century's most fundamental invention, one which allowed the construction of very powerful, very small computing devices. Only this made the conversion from signal energy requirements to (system) complexity possible, giving coding and information theory a platform for practical realizations [21].

Figure 1.4 shows an early feasibility experiment, comparing the performance of a 16-state 8-PSK TCM code used in an experimental implementation of a single channel per carrier (SCPC) modem operating at 64 kbits/second [49] against QPSK and the theoretical performance established via simulations (Figure 1.3). This illustrates the viability of trellis coding for satellite channels. Interestingly, the 8-PSK TCM modem performance comes much closer to the theoretical performance than the original QPSK modem, achieving a practical coding gain of 5 dB. This is due to an effect where system inaccuracies, acting similar to noise, are handled better by a coded system.

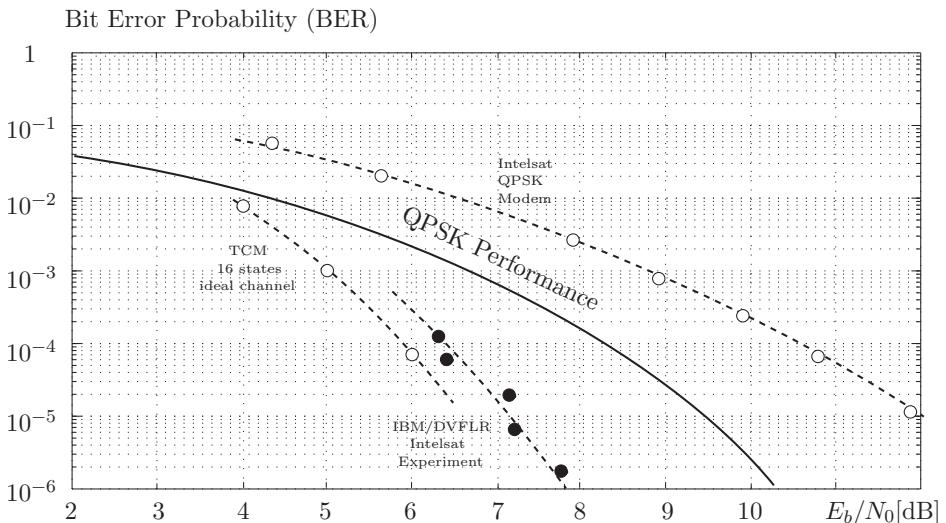


Figure 1.4: Measured bit error probability of (QPSK) and a 16-state 8-PSK (TCM) modem over a 64 kbit/s satellite channel [49]. The discrepancy between the theoretical curves and the implemented BER results are due to non-ideal behaviors of other components in the signaling chain, collectively known as *implementation loss*.

³Transistor action was first observed on December 15, 1947, but the news of the invention was not made public until June 30, 1948.

Figure 1.5 shows the performance of selected rate $R = 1/2$ bits/symbol (binary) convolutional and turbo codes codes on an additive white Gaussian noise channel (see also [24, 20, 37, 38, 39]). Contrary to TCM, these codes do not preserve bandwidth and the gains in power efficiency in Figure 1.5 are partly obtained by a power bandwidth trade-off, i.e., the rate 1/2 convolutional codes require twice as much bandwidth as uncoded transmission. This bandwidth expansion may not be an issue in deep-space communications or the application of error control to spread spectrum systems [53, 54]. As a consequence, for the same complexity, a higher coding gain is achieved than with TCM. Note that the convolutional codes reach a point of diminishing returns.

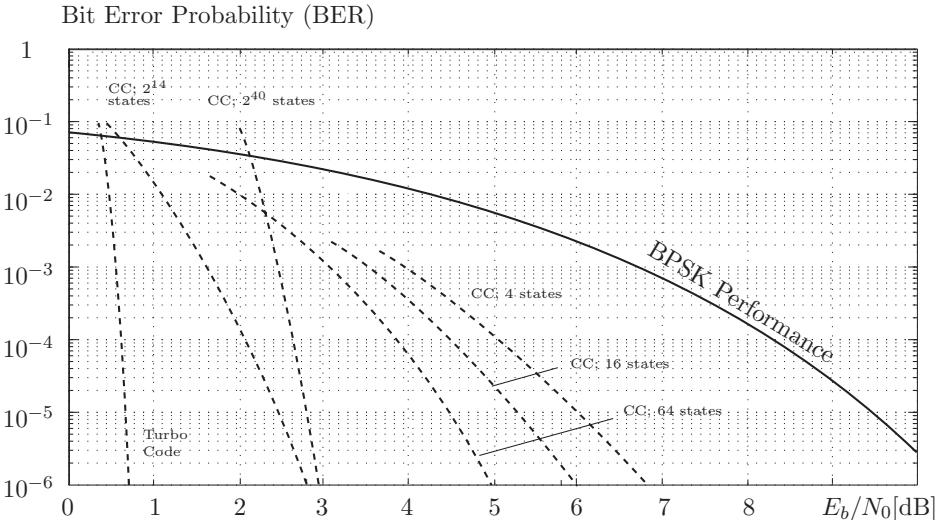


Figure 1.5: Bit error probability of selected rate $R = 1/2$ convolutional and turbo codes as a function of the normalized signal-to-noise ratio. The large-state space code is decoded sequentially, while the performance of all the other convolutional codes is for maximum-likelihood decoding. Simulation results are taken from [20], [38], and [4]. The Shannon limit is at $E_b/N_0=0$ dB, and that for BPSK is at $E_b/N_0 = 0.19$ dB.

For very low target error probabilities, tandem coding methods, called *concatenated coding*, have become very popular [24, 10, 23, 4]. In *classic concatenation*, as illustrated in Figure 1.6, the FEC codec is broken up into an inner and an outer code. The inner code is most often a trellis code which performs the channel error control, and the outer code is typically a high-rate Reed-Solomon (RS) block code. Its function it is to clean up the residual output error of the inner code. This combination has proven very powerful, since the error mechanism of the inner code is well matched to the error correcting capabilities

of the outer system. Via this tandem construction, very low error rates are achievable.

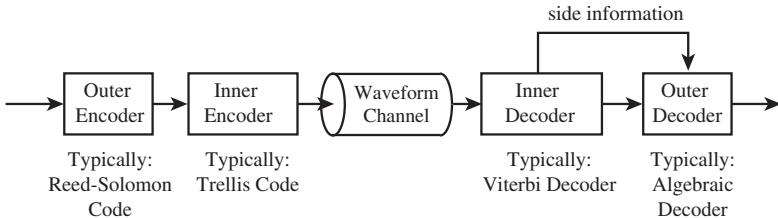


Figure 1.6: Classic concatenated FEC coding using inner and outer codecs.

With the discovery of turbo codes [4] new concatenated structures have appeared, shown in Figure 1.7. *Serial concatenation* is very similar to classic concatenation; however, the interleaver Π and deinterleaver Π^{-1} are fundamental structures, discussed later in Chapters 6 and 8. Furthermore, the outer code in these new structures is usually a very weak error control code such as a simple parity check code. Its function is not to clean up residual errors from the inner decoder, but, through interaction with the inner codes to form a strong error control system, a.k.a. a turbo code.

The original turbo codes, however, used a parallel arrangement of two codes, known as *parallel concatenation*, where both encoders have identical roles. Both arrangements, parallel and serial concatenation, are decoded with the same iterative decoding procedure which alternately invokes soft-output decoders for the two component codes. The structure and workings of these decoders is explored in detail in Chapters 5, 6, 8, and 9.

The field of error control and error correction coding naturally breaks into two disciplines, named somewhat inappropriately block coding and trellis coding. While block coding, which traditionally was approached as applied mathematics, has produced the bulk of publications in error control coding, trellis and turbo coding is favored in most practical applications. One reason for this is the ease with which soft-decision decoding can be implemented for trellis and turbo codes. Soft-decision is the operation whereby the demodulator does not make hard final decisions on the transmitted symbols or bits. Rather, it passes the received signal values directly on to the decoder which derives probabilistic information from those signals, which are then used to generate a final estimate of the decoded bits. There are no “errors” to be corrected; the decoder operates on reliability information obtained by comparing the received signals with the possible set of transmitted signals to arrive at a decision for an entire codeword. This soft-decision processing yields a 2 dB advantage on *additive white Gaussian noise* (AWGN) channels. In many applications the trellis decoders act as “ S/N transformers” (e.g., [17]), improving the channel behavior as in concatenated coding. Ironically, many block codes can be de-

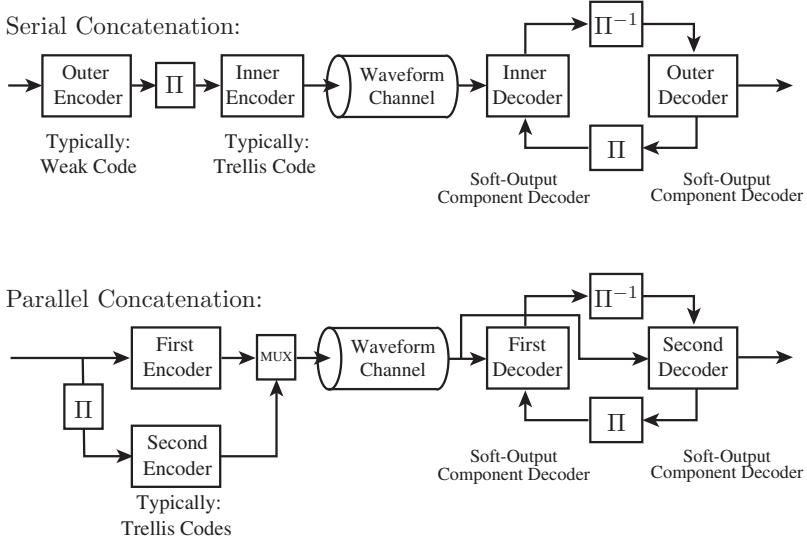


Figure 1.7: Modern concatenated FEC coding using two component codes.

coded very successfully using decoding methods developed for trellis codes (Chapter 4), smearing the boundaries between these two branches of error control coding.

1.5 Bandwidth, Power, and Complexity

Nyquist showed in 1928 [36] that a channel of bandwidth W (in Hz) is capable of supporting PAM signals at a rate of $2W$ samples/second without causing intersymbol interference. In other words, using Nyquist's method of interpolating band-limited functions, there are $2W$ independent *signal dimensions* per second⁴. If two carriers ($\sin(2\pi f_c)$ and $\cos(2\pi f_c)$) are used in quadrature, as in double side-band suppressed carrier amplitude modulation (DSB-SC), we have W pairs of dimensions (or complex dimensions) per second. This is the ubiquitous QAM format popular in digital radio systems (Chapter 2).

The parameter which characterizes how efficiently a system uses its allotted bandwidth is the bandwidth efficiency η , defined as

$$\eta = \frac{\text{Transmission Rate}}{\text{Channel Bandwidth } W} \text{ [bits/s/Hz].} \quad (1.2)$$

⁴A more general analysis using more sophisticated interpolation functions and a more general definition of bandwidth reveals that there are approximately $2.4W$ independent signal dimensions per second [62].

Using Shannon's capacity formula (1.1) and dividing by W we obtain the maximum bandwidth efficiency for an additive white Gaussian noise channel, the *Shannon limit*, as

$$\eta_{\max} = \log_2 \left(1 + \frac{S}{N} \right) \text{ [bits/s/Hz].} \quad (1.3)$$

In order to calculate η , we must suitably define the channel bandwidth W . This is obvious for some signaling schemes, like Nyquist signaling, which have a rather sharply defined bandwidth (see Chapter 2), but becomes more arbitrary for modulation schemes with infinite spectral occupancy. One commonly used definition is the 99% bandwidth definition, i.e., W is defined such that 99% of the transmitted signal power falls within the band of width W . This 99% bandwidth corresponds to an out-of-band power of -20 dB.

The average signal power S can be expressed as

$$S = \frac{kE_b}{T} = RE_b, \quad (1.4)$$

where E_b is the energy per bit, k is the number of bits transmitted per symbol, and T is the duration of a symbol. The parameter $R = k/T$ is the transmission rate of the system in bits/s. Rewriting the signal-to-noise power ratio S/N , where $N = WN_0$, i.e., the total noise power equals the one-sided noise power spectral density (N_0) multiplied by the width of the transmission band, we obtain the Shannon limit in terms of the bit energy and noise power spectral density, given by

$$\eta_{\max} = \log_2 \left(1 + \frac{RE_b}{WN_0} \right). \quad (1.5)$$

Since $R/W = \eta_{\max}$ is the limiting spectral efficiency, we obtain a bound from (1.5) on the minimum bit energy required for reliable transmission at a given spectral efficiency:

$$\frac{E_b}{N_0} \geq \frac{2^{\eta_{\max}} - 1}{\eta_{\max}}, \quad (1.6)$$

also called the *Shannon bound*.

If spectral efficiency is not at a premium, and a large amount of bandwidth is available for transmission, we may choose to use bandwidth rather than power to increase the channel capacity (1.1). In the limit as the signal is allowed to occupy an infinite amount of bandwidth, i.e., $\eta_{\max} \rightarrow 0$, we obtain

$$\frac{E_b}{N_0} \geq \lim_{\eta_{\max} \rightarrow 0} \frac{2^{\eta_{\max}} - 1}{\eta_{\max}} = \ln(2), \quad (1.7)$$

the absolute minimum bit energy to noise power spectral density required for reliable transmission. This minimum $E_b/N_0 = \ln(2) = -1.59$ dB.

We can cheat on the Shannon limit by allowing a certain number of errors in the following way: Assume that the original information source of rate R is being compressed into a rate $R' < R$. According to source coding theory, this introduces a distorting in the sense that original information can no longer be reconstructed perfectly [14]. If the source is binary, this compression results in a non-zero reconstruction bit error rate and satisfies $R' = R(1 - h(\text{BER}))$, where $h(p) = -p \log(p) - (1 - p) \log(1 - p)$ is the *binary entropy function*. As long as $R' < C$, the channel coding system will add no extra errors, and the only errors are due to the lossy compression. The source encoder has a rate of $1/(1 - h(\text{BER}))$, and consequently the average power is

$$S = \frac{RE_b}{1 - h(\text{BER})} \quad (1.8)$$

since less energy is used to transport a bit. The actual rate over the channel is R' , from which we obtain a modified Shannon bound for non-zero bit error rates, given by

$$\frac{E_b}{N_0} \geq \frac{2^{(1-h(\text{BER}))\eta_{\max}} - 1}{\eta_{\max}} (1 - h(\text{BER})). \quad (1.9)$$

This is the bound plotted in Figure 1.3 for a spectral efficiency of $\eta_{\max} = 2$ bits/s/Hz.

The implicit dependence of our formulas on the somewhat arbitrary definition of the bandwidth W is not completely satisfactory, and we prefer to normalize these formulas per signal dimension. Let R_d be the rate in bits/dimension, then the capacity of an additive white Gaussian noise channel per dimension is the maximum rate/dimension at which reliable transmission is possible. It is given by [62]

$$C_d = \frac{1}{2} \log_2 \left(1 + 2 \frac{R_d E_b}{N_0} \right) \text{ [bits/dimension]}, \quad (1.10)$$

or as

$$C_c = \log_2 \left(1 + \frac{RE_b}{N_0} \right) \text{ [bits/complex dimension]}, \quad (1.11)$$

if we normalize to complex dimensions, in which case R is the rate per complex dimension. Both (1.10) and (1.11) can easily be derived from (1.1).

Applying similar manipulations as above, we obtain the Shannon bound normalized per dimension as

$$\frac{E_b}{N_0} \geq \frac{2^{2C_d} - 1}{2C_d}; \quad \frac{E_b}{N_0} \geq \frac{2^{C_c} - 1}{C_c}. \quad (1.12)$$

Equations (1.12) are useful when the question of waveforms and pulse shaping is not a central issue, since it allows one to eliminate these considerations by working with signal dimensions, rather than the signals itself (see also Chapter 2). We will use (1.12) for our comparisons.

The Shannon bounds (1.3) and (1.12) relate the spectral efficiency η to the power efficiency E_b/N_0 , and establish fundamental limits in the trade-off between the two primary resources of data communications; power and spectrum. These limits hold regardless of the signal constellation or coding method that is used for transmission.

Figure 1.8 shows (1.12)—as a solid line—as well as similar limits calculated for the cases where the transmitted signal constellations are restricted to BPSK, QPSK, 8-PSK, 16-PSK, and 16-QAM (see Chapter 2). As can be seen, these restrictions lead to various degrees of loss that has to be accepted when a particular constellation is chosen. The figure also shows the power and bandwidth efficiencies of some popular uncoded quadrature constellations as well as that of a number of error control coded transmission schemes. The trellis-coded modulation schemes used in practice, for example, achieve a power gain of up to 6 dB without loss in spectral efficiency, while the more powerful coded modulation methods such as trellis-turbo coding and block turbo coding provide even further gain. The binary coding methods achieve a gain in power efficiency, but at the expense of spectral efficiency with respect to the original signal constellation. The turbo-coded methods come extremely close to capacity for $\eta \leq 1$; we present such methods which can be made to approach the Shannon bound arbitrarily closely in Chapter 8.

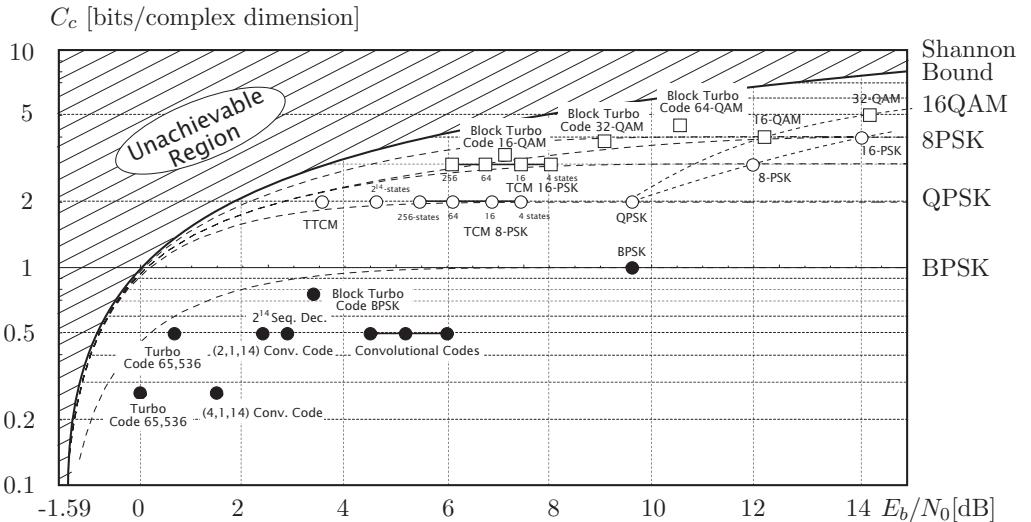


Figure 1.8: Theoretical spectral and power efficiency limits for various signal constellations and spectral efficiencies achieved by coded and uncoded transmission methods.

In all cases a rate reduction (traditional coding) or a signal set expansion (coded modulation) is required in order to give the coding system the required redundancy. This is also clearly obvious from the capacity curves for the different signal constellations.

In order to compare these different communications systems, we also need a parameter expressing the performance level of a system. This parameter is the information bit error probability P_b . For practical systems it typically falls into the range $10^{-3} \geq P_b \geq 10^{-6}$, though it is sometimes required to be lower, e.g., for digital TV, or much lower as for optical and cable modem systems where BER on the order of $P_b \leq 10^{-6}$ are required. The performance points in Figure 1.8 are drawn for $P_b = 10^{-5}$.

Other methods which combine coding with signal shaping exist and exhibit similar gains. For example, coded overlapped quadrature modulation (OC-QPSK) is a combined coding and controlled intersymbol interference method [43] which causes smaller amplitude variations than Nyquist signaling, and is therefore useful for systems with amplifier non-linearities, like satellite traveling wave tube (TWT) amplifiers. Continuous-phase modulation (CPM) is a constant-amplitude modulation format [1], which also has many similarities with TCM, and derives from frequency modulation aiming at improving the bandwidth efficiency by smoothing phase transitions between symbols. CPM has gone through an evolution similar to TCM, and the reader is referred to the book by Anderson, Aulin, and Sundberg [1], the standard reference on the subject. However, with the advent of highly linear amplifiers, CPM has lost one of its main selling points and has seen significantly less deployment than the linear modulation techniques.

The last, and somewhat hidden, player in the application of coding is complexity. While we have shown that power and bandwidth can be captured elegantly by Shannon's theory, measures of complexity are much more difficult to define. First there is what we might term *code complexity*. In order to approach the Shannon bound, larger and larger codes are required. In fact, Shannon et al. [46] proved the following *lower bound* on the codeword error probability P_B :

$$P_B > 2^{-n(E_{\text{sp}}(R)+o(N))}, \quad E_{\text{sp}}(R) = \max_{\mathbf{q}} \max_{\rho>1} (E_0(\mathbf{q}, \rho) - \rho R). \quad (1.13)$$

The exponent $E_0(\mathbf{q}, \rho)$ is called the Gallager exponent and depends on the symbol probability distribution \mathbf{q} and the optimization parameter ρ , it is discussed in more detail in Chapter 5, as well as in Gallager's book on information theory [13]. The most important point to notice is that this bound is exponential in the codelength n .

The bound is plotted for rate $R = 1/2$ in Figure 1.9 for BPSK modulation [44], together with selected turbo-coding schemes and classic concatenated methods. The performance of codes for various lengths follows the tendency of the bound, and we see a diminishing return as the code size exceeds $n \approx 10^4 - 10^5$, beyond which only small gains are possible. This is the reason why most practical applications of large codes target block sizes no larger than this. On the other hand, codes cannot be shortened much below $n = 10^4$ without

a measurable loss in performance, which must be balanced against the overall system performance. Implementations of near-capacity error control systems therefore have to process blocks of 10,000 symbols or more, requiring appropriate storage and memory management.

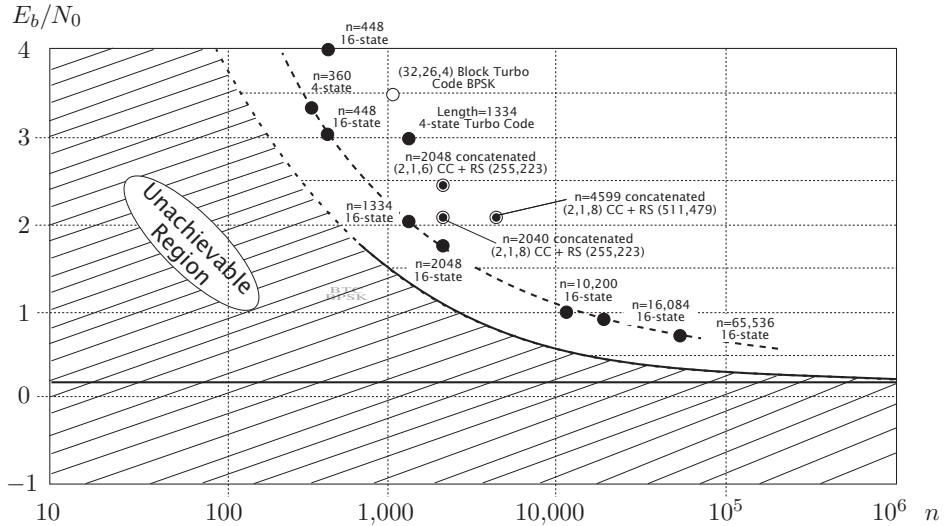


Figure 1.9: Block error rate P_B and sphere-packing lower bound for rate $R = 1/2$ coded example coding systems using BPSK. Turbo codes and selected classic concatenated coding schemes are compared. The solid line is the Shannon limit at $E_b/N_0 = 0.19$ dB at $R = 0.5$.

The other component of our complexity consideration is the *computational complexity*, that is, the amount of processing that has to be performed to decode a codeword. This is notoriously difficult to measure, and parameters like the code state space or number of multiplications may not be relevant. What ultimately counts is the size and power consumption of a VLSI implementation, which is very much technology, architecture, and design dependent. It suffices to say that what a coding theorist considers complex may not be complex for a VLSI circuit designer. An example of this is the “folk theorem” that an APP trellis decoder (Chapter 5) is about four times as complex as a Viterbi decoder for the same code. Some circuit designers would argue that the trace-back implementation of the Viterbi decoder easily compensates for the more complex arithmetic of the APP decoder and that there is no real difference between the two in terms of implementation complexity.

The situation of turbo codes is even more complicated to evaluate, since most of the complexity of the decoder resides in the storage requirements of the large blocks that need to be processed and the efficient management of the this memory. The computational units of a turbo decoder take up less than 10% of the VLSI area. Processing large blocks, however, is inherent in effective error control coding as discussed above.

More recently decoding circuits based on analog processing have emerged [26, 34, 57], using subthreshold CMOS and bipolar multiplier circuits, both based on variations of the well-known *Gilbert cell* [31]. These circuits are ideally matched to the arithmetic requirements of soft-decision decoders, making use of the fundamental exponential transfer characteristics of the transistors. Digital information is processed by small currents in the transistor's off-region, representing probabilities of the various discrete random variables. Contrary to mainstream digital VLSI implementations where the component transistors are used as on-off switches, and are optimized for that operation, analog processing operates the transistors in their off-position, and all computations are accomplished by what digital technology calls leakage currents. Analog decoding is also different from conventional analog processing in that digital information is processed. This leads to a surprising robustness of the analog decoder to typical analog impairments such as current mismatch, body effects, etc.

Several such decoders have already been successfully fabricated and tested, or extensively simulated by a few research groups around the world, see, e.g., [58, 59]. The initial idea was that subthreshold operation would result in substantial power savings of the decoding operation. While operating the transistor in the subthreshold region has indeed a strong potential to save power (see, e.g., [6, 7]), the story regarding analog decoders is more complex.

Zargham et al. [65] for example show that the current mirror circuit which is the base of the Gilbert multiplier used in most of the analog designs is increasingly susceptible to gate threshold voltage variations in the transistors that are paired to make the current mirror. As the fabrication process shrinks, small variations in the gate size translate into exponentially varying current errors in the Gilbert cells. This ultimately leads to a breakdown of the functionality of the decoder as a whole. Zargham et al. [65] argue that analog decoders in sub-100 nm processes will require such large oversizing of the transistors that going to smaller processes is not productive.

In an investigation by Winstead and Schlegel [42], the computational and energy requirements of message passing decoders as discussed in Chapters 6 and 9 are examined from fundamental viewpoints of computational theory and minimum-energy switching principles. Since advanced message-passing decoders have a computational complexity which is linear in the number of bits decoded, a direct energy cost measure per decoded bit can be computed, quite irrespective of the actual code or the code family being used. This leads to predictions of the energy cost per decoded message bit as miniaturization processes continue to advance to ever smaller scales. Figure 1.10 shows the energy per-

formance attained by a number of message-passing decoders that were built by several research and industrial groups. What is evident is that the process miniaturization is the major driver for a reduction in the energy per bit. The ultimate limit for charged-based computation is based on a conceptual single-charge device [66], and would attain a decoding energy consumption anywhere between sub-fempto-joule to about 10 fJ/decoded bit, depending on code and implementation parameters.

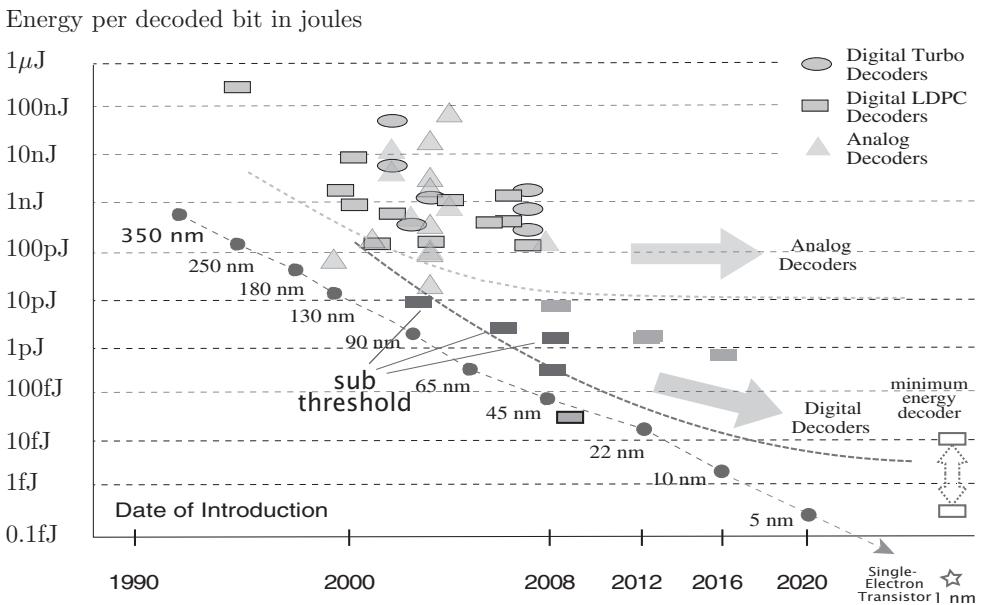


Figure 1.10: Energy consumption of various message-passing decoders and predictions for future developments in joules per decoded bit.

While predictions are notoriously difficult to make with accuracy, it appears that analog decoders based on Gilbert-cell technology would flatten out at around 10 pJ/decoded bit, primarily due to the gate-threshold variations discussed above [65]. Digital decoders, on the other hand, can benefit from the miniaturization, and the calculations in [42] indicate that a minimal energy cost for a digital implementation would be around $30,000kT$, where k is Boltzmann's constant, and T is the temperature. This translates to about 0.1 femto-joules. Any further reductions in power consumption would need to rely on speculative technologies such as adiabatic computation and/or quantum computing.

1.6 A Brief History—The Drive Towards Capacity

Forward error control (FEC) coding celebrated its first success in the application of convolutional codes to deep-space probes in the 1960's and 1970's, and for quite a while afterwards, FEC was considered an intellectual curiosity with deep-space communications as its only viable practical application. Deep space communications is a classical case of power-limited communications, and it serves as a picture book success story of error control coding.

If we start with uncoded binary phase-shift keying (BPSK) as our baseline transmission method (see Chapter 2) and assume coherent detection, we can achieve a bit error rate of $P_b = 10^{-5}$ at a bit energy-to-noise power ratio of $E_b/N_0 = 9.6$ dB and at a spectral efficiency of ideally 1 bit/dimension. From the Shannon limit in Figure 1.11, it can be seen that 1 bit/dimension is theoretically achievable with $E_b/N_0 = 1.76$ dB, indicating that a power savings of nearly 8 dB is possible by applying proper coding. 8 dB is an over 6-fold savings in transmit power, antenna size, or other aspect directly linked to the received signal power, as, for example, 2.5 times the transmission distance.

Spectral Efficiency [bits/dimension]

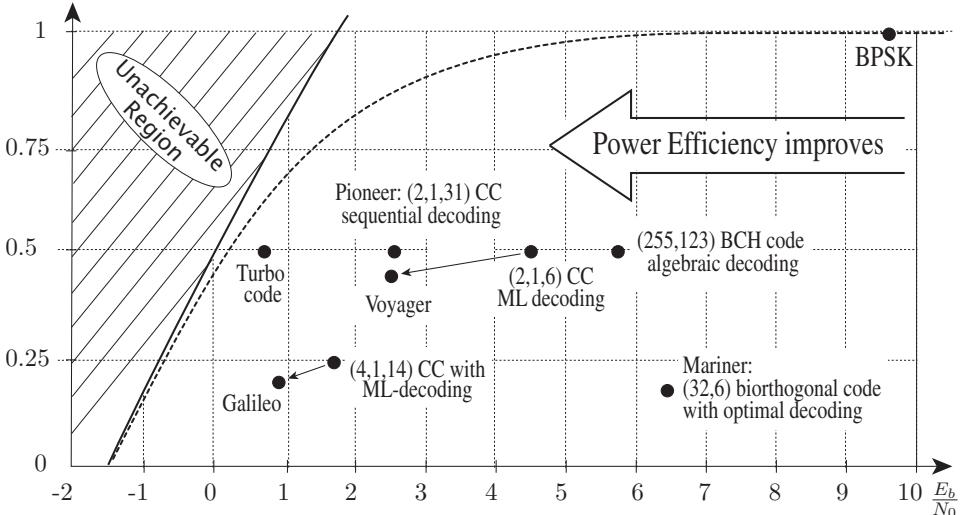


Figure 1.11: Some milestones in the drive towards channel capacity achieved by the space systems which evolved over the past 50 years as answer to the Shannon capacity challenge.

One of the earliest attempts to close the signal energy gap to the Shannon limit was the use of a rate 6/32 biorthogonal (Reed-Muller) block code [27]. This code was used

on the Mariner Mars and Viking missions in conjunction with BPSK and soft-decision maximum-likelihood decoding. This system had a spectral efficiency of 0.1875 bits/symbol and achieved a bit error rate of $P_b = 10^{-5}$ with an $E_b/N_0 = 6.4$ dB. Thus, the (32,6) biorthogonal code required 3.2 dB less power than BPSK at the cost of a five-fold increase in the bandwidth. The performance of the 6/32 biorthogonal code is plotted in Figure 1.11, as are all the other systems discussed below.

In 1967, a new algebraic decoding technique was discovered for the popular Bose-Chaudhuri-Hocquenghem (BCH) codes [2, 28]. This new algorithm enabled the efficient hard-decision decoding of an entire class of block codes—for example, the (255,123) BCH code, which has a code rate of $R_d \approx 0.5$ bits/symbol and achieves a BER of $P_b = 10^{-5}$ at $E_b/N_0 = 5.7$ dB using algebraic decoding.

The next step was taken with the introduction of sequential decoding (see Chapter 5), which could make use of soft-decision decoding. Sequential decoding allowed the decoding of long constraint-length convolutional codes, and it was first used on the Pioneer 9 mission [11]. The Pioneer 10 and 11 missions in 1972 and 1973 both used a long constraint-length (2,1,31), nonsystematic convolutional code (Chapter 4) [29]. A sequential decoder was used which achieved $P_b = 10^{-5}$ with $E_b/N_0 = 2.5$ dB, and $R_d = 0.5$. This is only 2.5 dB away from the capacity of the channel.

Sequential decoding has the disadvantage that the computational load is variable, and this load grows exponentially the closer the operation point moves towards capacity (see Chapter 4). For this and other reasons, the next generation of space systems employed maximum-likelihood decoding. The Voyager spacecraft, launched in 1977, used a short constraint-length (2,1,6) convolutional code in conjunction with a soft-decision Viterbi decoder achieving $P_b = 10^{-5}$ at $E_b/N_0 = 4.5$ dB and a spectral efficiency of $R_d = 0.5$ bits/symbol. The biggest Viterbi decoder built to date [8] found application in the Galileo mission, where a (4,1,14) convolutional code is used, yielding a spectral efficiency of $R_d = 0.25$ bits/symbol at $P_b = 10^{-5}$ and $E_b/N_0 = 1.75$ dB. The performance of this system is 2.5 dB away from the capacity limit. The systems for Voyager and Galileo are further enhanced by the use of concatenation in addition to the convolutional inner code. An outer (255,223) Reed-Solomon code [27] is used to reduce the required signal-to-noise ratio by 2.0 dB for the Voyager system and by 0.8 dB for the Galileo system.

More recently, Turbo-codes [3] using iterative decoding have virtually closed the capacity gap by achieving $P_b = 10^{-5}$ at a spectacularly low E_b/N_0 of 0.7 dB with $R_d = 0.5$ bits/symbol, and longer turbo codes come even closer to capacity, e.g., [48]. It is probably appropriate to say that the half-century effort to reach capacity has been achieved with this latest invention, in particular in the regime of lower spectral efficiencies. With turbo coding then, another quite unexpected step of about 2 dB right to the capacity limit was made possible. Newer space communications systems virtually all use turbo or low-density parity-check codes discussed in this book—see also Chapter 9.

Space applications of error control coding have met with spectacular success, and

for a long time the belief that coding was useful only for improving power efficiency of digital transmission was popular. This attitude was thoroughly overturned by another spectacular success of error control coding, this time for applications of data transmission over voiceband telephone channels. Here it was not the power efficiency which was the issue, but rather spectral efficiency, i.e., given a standard telephone channel with a fixed bandwidth and SNR, what was the maximum practical rate of reliable transmission?

The first commercially available voiceband modem in 1962 achieved a transmission rate of 2400 bits/s. Over the next 10 to 15 years these rates improved to 9600 bits/s, which was then considered to be the maximum achievable rate, and efforts to push the rate higher were frustrated. Ungerböck's invention of trellis-coded modulation in the late 1970's, however, opened the door to further, unexpected improvements. The modem rates jumped to 14,400 bits/s and then to 19,200 bits/s using sophisticated TCM schemes [12]. The latest chapter in voiceband data modems is the establishment of the CCITT V.34 modem standard [15, 9]. The modems specified therein achieve a maximum transmission rate of 28,800 bits/s, and extensions to V.34 to cover two new rates at 31,200 bits/s and 33,600 bits/s have been specified. However, at these high rates, modems operate successfully only on a small percentage of the connections. It seems that the limits of the voiceband telephone channel have been reached (according to [56]). This needs to be compared to estimates of the channel capacity for a voiceband telephone channel, which are somewhere around 30,000 bits/s. The application of TCM was one of the fastest migrations of an experimental laboratory system to an international standard (V.32 - V.34) [5, 15, 9]. The trellis codes used in these advanced modems are discussed in treated in detail in Chapter 3.

In many ways the telephone voiceband channel is an ideal playground for the application of error control coding. Its limited bandwidth of about 3 kHz (400 Hz - 3400 Hz) implies low data rates by modern standards. It therefore provides an ideal experimental field for high-complexity error control methods, which can be implemented without much difficulty using current DSP technology. It is thus not surprising that coding for voiceband channels was the first successful application of bandwidth efficient error control.

Nowadays, trellis coding in the form of bandwidth-efficient TCM as well as more conventional convolutional coding and higher-order modulation turbo-coding systems are used for satellite communications, both geostationary and low-earth orbiting satellites, for land-mobile and satellite-mobile services, for cellular communications networks, personal communications services (PCS), high-frequency (HF) tropospheric long-range communications, and cable modems, and increasingly also for high-speed optical communications systems. The Shannon capacity is now routinely the goal that is targeted for high-efficiency communications systems. As an example thereof, the IEEE 802.3an standard for Ethernet communications over twisted pair copper cables is discussed in more detail in Chapter 6. To us it seems clear that the age of widespread application of error control coding is upon us and every efficient communications systems will entail some form of FEC.

Bibliography

- [1] J.B. Anderson, T. Aulin, and C-E. Sundberg, *Digital Phase Modulation*, Plenum Press, New York, 1986.
- [2] E.R. Berlekamp, *Algebraic Coding Theory*, McGraw-Hill, New York, 1968.
- [3] C. Berrou, A. Glavieux, and P. Thitimajshima, “Near Shannon limit error-correcting coding and decoding: Turbo-codes,” *Proc. 1993 IEEE Int. Conf. on Commun.*, Geneva, Switzerland, pp. 1064–1070, 1993.
- [4] C. Berrou and A. Galvieux, “Near optimal error-correcting coding and decoding: Turbo codes,” *IEEE Trans. Commun.*, vol. COM-44, no. 10, pp. 1261-1271, Oct. 1996.
- [5] U. Black, *The V Series Recommendations, Protocols for Data Communications Over the Telephone Network*, McGraw-Hill, New York, 1991.
- [6] D. Bol, R. Ambroise, D. Flandre, and J.-D. Legat, “Interests and limitations of technology scaling for subthreshold logic,” *IEEE Very Large Scale Integration (VLSI) Syst.*, vol. 17, no. 10, pp. 1508-1519, Oct. 2009.
- [7] D. Bol, “Pushing ultra-low-power digital circuits into the nanometer era,” Ph.D. thesis, Université catholique de Louvain, École Polytechnique de Louvain, Département d’Électricité, Dec. 2008.
- [8] O.M. Collins, “The subtleties and intricacies of building a constraint length 15 convolutional decoder,” *IEEE Trans. Commun.*, vol. COM-40, pp. 1810–1819, 1992.
- [9] G.D. Forney, L. Brown, M.V. Eyuboglu, J.L. Moran III, “The V.34 high-speed modem standard,” *IEEE Commun. Mag.*, pp. 28–33, Dec. 1996.
- [10] G.D. Forney, *Concatenated Codes*, MIT Press, Cambridge, Mass., 1966.
- [11] G.D. Forney, “Final report on a study of a sample sequential decoder,” Appendix A, Codex Corp., Watertown, MA, U.S. Army Satellite Communication Agency Contract DAA B 07-68-C-0093, April 1968.
- [12] G.D. Forney, “Coded modulation for bandlimited channels,” *IEEE Information Theory Society Newsletter*, Dec. 1990.
- [13] R.G. Gallager, *Information Theory and Reliable Communications*, John Wiley & Sons, New York, 1968.

- [14] R.M. Gray, *Source Coding Theory*, Kluwer Academic Publishers, Dordrecht, fourth printing, 1997.
- [15] CCITT Recommendations V.34.
- [16] M.J.E. Golay, "Notes on digital coding," *Proc. IEEE*, vol. 37, p. 657, 1949.
- [17] J. Hagenauer and P. Höher, "A Viterbi algorithm with soft-decision outputs and its applications," *Proc. IEEE Globecom'89*, 1989.
- [18] J. Hagenauer, "Rate compatible punctured convolutional codes (RCPC-codes) and their application," *IEEE Trans. Commun.*, vol. COM-36, pp. 389–400, April 1988.
- [19] R.W. Hamming, "Error detecting and error correcting codes," *Bell Syst. Tech. J.*, vol. 29, pp. 147–160, 1950.
- [20] J.A. Heller and J.M. Jacobs, "Viterbi detection for satellite and space communications," *IEEE Trans. Commun. Technol.*, COM-19, pp. 835–848, Oct 1971.
- [21] H. Imai et al. *Essentials of Error-Control Coding Techniques*, Academic Press, New York, 1990.
- [22] N.S. Jayant and P. Noll, *Digital Coding of Waveforms*, Prentice Hall, Englewood Cliffs, NJ, 1984.
- [23] K.Y. Lin and J. Lee, "Recent results on the use of concatenated Reed-Solomon/Viterbi channel coding and data compression for space communications," *IEEE Trans. Commun.*, vol. COM-32, pp. 518–523, 1984.
- [24] S. Lin and D.J. Costello, Jr., *Error Control Coding: Fundamentals and Applications*, Prentice Hall, Englewood Cliffs, NJ, 1983.
- [25] S. Lin, D.J. Costello, Jr., and M.J. Miller, "Automatic-repeat-request error-control schemes," *IEEE Commun. Mag.*, vol. 22, pp. 5–17, 1984.
- [26] H.-A. Loeliger, F. Lustenberger, M. Helfenstein, and F. Tarköy, "Probability propagation and decoding in analog VLSI," *IEEE Trans. Inform. Theory*, pp. 837–843, Feb. 2001.
- [27] F.J. MacWilliams and N.J.A. Sloane, *The Theory of Error Correcting Codes*, North Holland, New York, 1988.
- [28] J.L. Massey, "Shift register synthesis and BCH decoding," *IEEE Trans. Inform. Theory*, vol. IT-15, pp. 122–127, 1969.
- [29] J.L. Massey and D.J. Costello, Jr., "Nonsystematic convolutional codes for sequential decoding in space applications," *IEEE Trans. Commun. Technol.*, vol. COM-19, pp. 806–813, 1971.
- [30] J.L. Massey, "Coding and modulation in digital communications," *Proc. Int. Zürich Sem. Digital Commun.*, Zürich, Switzerland, March 1974, pp. E2(1)–E2(4).
- [31] C. Mead, *Analog VLSI and Neural Systems*, Addison-Wesley, Reading, MA, 1989.
- [32] H. Meyr and G. Ascheid, *Synchronization in Digital Communications*, Vol. 1, John Wiley & Sons, New York, 1990.

- [33] H. Meyr, M. Moeneclaey, and S.A. Fechtel, *Digital Communication Receivers*, John Wiley & Sons, New York, 1998.
- [34] M. Moerz, T. Gabara, R. Yan, and J. Hagenauer, "An analog .25 μm BiCMOS tailbiting MAP decoder," *IEEE Proc. International Solid-State Circuits Conf.*, pp. 356–357, San Francisco, Feb. 2000.
- [35] M. Mouly and M-B. Pautet, *The GSM System for Mobile Communications*, sold by the authors, ISBN 2-9507190-0-7. 1993.
- [36] H. Nyquist, "Certain topics in telegraph transmission theory," *AIEE Trans.*, pp. 617 ff., 1946.
- [37] J.P. Odenwalder, "Optimal decoding of convolutional codes," Ph.D. thesis, University of California, LA, 1970.
- [38] J.K. Omura and B.K. Levitt, "Coded error probability evaluation for antijam communication systems," *IEEE Trans. Commun.*, vol. COM-30, pp. 896–903, May 1982.
- [39] J.G. Proakis, *Digital Communications*, McGraw-Hill, New York, 1989.
- [40] S. Ramseier and C. Schlegel, "On the bandwidth/power tradeoff of trellis coded modulation schemes," *Proc. IEEE Globecom'93* (1993).
- [41] S.A. Rhodes, R.J. Fang, and P.Y. Chang, "Coded octal phase shift keying in TDMA satellite communications," *COMSAT Tech. Rev.*, vol. 13, pp. 221–258, 1983.
- [42] C. Schlegel and C. Winstead, "From mathematics to physics: Building efficient iterative error control decoders," *International Symposium on Turbo Coding and Iterative Information Processing*, Sweden, Aug. 2012.
- [43] C. Schlegel, "Coded overlapped quadrature modulation," *Proc. Global Conf. Commun. GLOBECOM'91*, Phoenix, AZ, Dec. 1991.
- [44] C. Schlegel and L.C. Perez, "On error bounds and turbo codes," *IEEE Commun. Lett.*, vol. 3, no. 7, July 1999.
- [45] C.E. Shannon, "A mathematical theory of communications," *Bell Syst. Tech. J.*, vol. 27, pp. 379–423, July 1948.
- [46] C.E. Shannon, R.G. Gallager, and E.R. Berlekamp, "Lower bounds to error probabilities for coding on discrete memoryless channels," *Inform. Contr.*, vol. 10, pt. I, pp. 65–103, 1967, Also, *Inform. Contr.*, vol. 10, pt. II, pp. 522–552, 1967.
- [47] TIA/EIA/IS-95 interim standard, mobile station–base station compatibility standard for dual-mode wideband spread spectrum cellular systems, Telecommunications Industry Association, Washington, D.C., July 1993.
- [48] S. tenBrink, "A rate one-half code for approaching the Shannon limit by 0.1 dB," *Electron. Lett.*, vol. 36, no. 15, pp. 1293–1294, July 2000.
- [49] G. Ungerboeck, J. Hagenauer, and T. Abdel-Nabi, "Coded 8-PSK experimental modem for the INTELSAT SCPC system," *Proc. ICDS, 7th*, pp. 299–304, 1986.

- [50] G. Ungerboeck, "Channel coding with multilevel/phase signals," *IEEE Trans. Inform. Theory*, vol. IT-28, no. 1, pp. 55-67, Jan. 1982.
- [51] G. Ungerboeck, "Trellis-coded modulation with redundant signal sets part I: Introduction," *IEEE Commun. Mag.*, vol. 25, no. 2, pp. 5-11, Feb. 1987.
- [52] G. Ungerboeck, "Trellis-coded modulation with redundant signal sets part II: State of the art," *IEEE Commun. Mag.*, vol. 25, no. 2, pp. 12-21, Feb. 1987.
- [53] A.J. Viterbi, "Spread spectrum communications—myths and realities," *IEEE Commun. Mag.*, vol. 17, pp. 11-18, May 1979.
- [54] A.J. Viterbi, "When not to spread spectrum—a sequel," *IEEE Commun. Mag.*, vol. 23, pp. 12-17, April 1985.
- [55] F.-Q. Wang and D.J. Costello, "Probabilistic construction of large constraint length trellis codes for sequential decoding," *IEEE Trans. Commun.*, vol. 43, no. 9, Sept. 1995.
- [56] R. Wilson, "Outer limits," *Electronic News*, May 1996.
- [57] C. Winstead, J. Dai, C. Meyers, C. Schlegel, Y.-B. Kim, W.-J. Kim, "Analog MAP decoder for (8,4) Hamming code in Subthreshold CMOS," *Advanced Research in VLSI Conference ARVLSI*, Salt Lake City, March, 2000.
- [58] C. Winstead, J. Dai, S. Yu, C. Myers, R. Harrison, and C. Schlegel, "CMOS analog MAP decoder for (8,4) Hamming code," *IEEE Journal of Solid State Circuits*, vol. 29, no. 1, Jan. 2004.
- [59] C. Winstead and C. Schlegel, "Importance Sampling for SPICE-level verification of analog decoders", *International Symposium on Information Theory (ISIT'03)*, Yokohama, June, 2003.
- [60] A.P. Worthen, S. Hong, R. Gupta, W.E. Stark, "Performance optimization of VLSI transceivers for low-energy communications systems," *IEEE Military Communications Conference Proceedings, MILCOM 1999*, vol. 2, pp. 1434-1438, 1999.
- [61] J.M. Wozencraft and R.S. Kennedy, "Modulation and demodulation for probabilistic coding", *IEEE Trans. Inform. Theory*, vol. IT-12, no. 3, pp. 291-297, July, 1966.
- [62] J.M. Wozencraft and I.M. Jacobs, *Principles of Communication Engineering*, John Wiley & Sons, New York, 1965, reprinted by Waveland Press, 1993.
- [63] 3GPP Group Website, Releases, <http://www.3gpp.org/releases>.
- [64] 3GPP2 Group Website, <http://www.3gpp2.org>.
- [65] M. Zargham, C. Schlegel, J. P. Chamorroc, C. Lahuecc, F. Seguinc, M. Jézéquel, and V. Gaudet, "Scaling of analog LDPC decoders in sub-100 nm CMOS processes," *Integration—the VLSI J.*, vol. 43, no. 4, Sept. 2010, pp. 365-377.
- [66] V. Zhirnov R. Cavin, III, J. Hutchby, and G. Bourianoff, "Limits to binary logic switch scaling: A gedanken model," *Proceedings of the IEEE*, vol. 91, no. 11, Nov. 2003.

Chapter 2

Communications Basics

2.1 The Probabilistic Viewpoint

Communication, the transmission of information from a sender to a receiver (destination), is fundamentally a random experiment, particularly evident if the information is in digital form. The sender selects one of a number of possible messages which is transmitted to the receiver, which has to determine which message was chosen by the sender. In doing so the receiver uses the fact that the transmitted message is chosen from a set of possible messages known to both sender and receiver. In doing so it applies probabilistic principles to determine which message was the most likely to have been sent. This is the fundamental concept of the decoding process.

The signal carrying the message is typically subject to random distortions, primarily noise, which adds a further complication to the transmission process. The noise in a communications link is a fundamental property of the link, arising from a variety of physical processes. It is often used to describe any process whose precise form is inherently unknown to the receiver, irrespective of whether it is really random, or only appears so to the receiver. Noise is not categorically a *bad* thing, for without noise, the concept of communications would not truly exist, since all information could be made available anywhere with little effort. Consider the hypothetical situation where we want to move an arbitrary amount of information virtually instantaneously. We could do this, conceptually, by transmitting a chosen voltage signal whose precise numerical representation could carry an unlimited amount of data, for example as the infinite expansion of that numerical representation into a binary representation, whose bits are our information to transport. All this is, of course, only thinkable if there is no noise that limits our ability to precisely define and transmit such a signal. Noise then, in a fundamental sense, prevents information from being omnipresent and enables the modern concepts of communications. Since noise is a random process, it should not surprise us that communication theory draws

heavily from probability theory.

Now Figure 2.1 shows a simplified system diagram of such a sender/receiver communication system. The transmitter performs the random experiment of selecting one of the M messages in the message set $\{m^{(i)}\}$, say $m^{(i)}$ and transmits a corresponding signal $s^{(i)}(t)$, chosen from a set of signals $\{s^{(i)}(t)\}$. In many cases this signal is a continuous function of time, such as a voltage level, a current, or ultimately an electromagnetic signal that propagates from transmitter to receiver typical for modern telecommunications. The transmission medium, irrespective of its physical representation, is generically called *the channel*. In reality it may be a telephone twisted wire pair, a radio link, an underwater acoustic link, or any other suitable arrangement, as elaborated in Chapter 1. One of the major impairments in all communication systems is thermal noise, the quintessential of all noise sources. Thermal noise is generated by the random motion of charged particles either inside the receiver itself, at the signal source, or anywhere in transit. Thermal noise has the property that it adds linearly to the received signal, which is due to the superposition principles of electromagnetic waves,¹ hence the channel model in Figure 2.1.

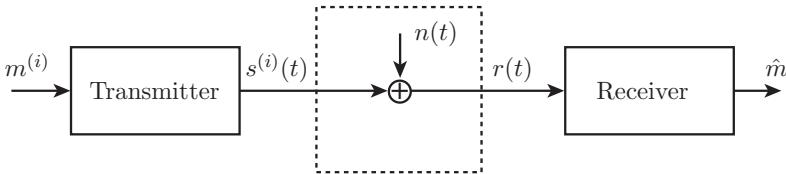


Figure 2.1: Block diagram of a sender/receiver communication system used on a channel with additive noise.

A large body of literature is available on modeling channel impairments on the transmitted signal. In this book we concentrate on the almost ubiquitous case of additive white Gaussian noise (AWGN), which is reviewed briefly in Appendix 2 A. Noise is a random quantity as discussed above, and, together with the random choice of messages, our communication system becomes a joint random experiment. More precisely, using Figure 2.1, these quantities are random processes (i.e., random functions). It is now the task of the receiver to detect the selected message $m^{(i)}$ with the best possible reliability, observing the received signal $r(t)$. In the context of a discrete set of messages, one speaks of message detection and the probabilities of correct or false detection of a message. The probability of error, which is defined as the probability that the message \hat{m} identified by the receiver is not the one originally chosen by the transmitter, i.e., $P_e = \Pr(\hat{m} \neq m^{(i)})$ is the almost

¹Actually, more fundamentally it is due to the fact that the solutions to the wave equation form a linear set, and the wave equation models most of the dominant physical channels.

universally accepted and widely used measure of the quality of the detection process. The solution to the detection problem is essentially completely known for many cases and we will subsequently present an overview of optimum receiver principles in additive white Gaussian noise.

2.2 Vector Communication Channels

A very popular way to generate signals at the transmitter which carry our messages is to synthesize them as a linear combination of N basis waveforms $\phi_j(t)$, i.e., the transmitter selects

$$s^{(i)}(t) = \sum_{j=1}^N s_j^{(i)} \phi_j(t) \quad (2.1)$$

as the transmitted signal for the i th message. Often the basis waveforms are chosen to be orthonormal, that is, they fulfill the condition

$$\int_{-\infty}^{\infty} \phi_j(t) \phi_l(t) dt = \delta_{jl} = \begin{cases} 1, & j = l, \\ 0, & \text{otherwise.} \end{cases} \quad (2.2)$$

This leads to a vector interpretation of the transmitted signals, since, once the basis waveforms are specified, $s^{(i)}(t)$ is completely determined by the N -dimensional vector of numbers

$$\mathbf{s}^{(i)} = (s_1^{(i)}, s_2^{(i)}, \dots, s_N^{(i)}). \quad (2.3)$$

We can now visualize the signals geometrically by viewing the signal vectors $\mathbf{s}^{(i)}$ in Euclidean N -space, spanned by the usual orthonormal basis vectors, where each basis vector is associated with a basis function. This geometric representation of signals is called the signal space representation, and the vectors are called a signal constellation. The idea is illustrated for $N = 2$ in Figure 2.2 for the signals $s^{(1)}(t) = \sin(2\pi f_1 t)w(t)$, $s^{(2)}(t) = \cos(2\pi f_1 t)w(t)$, $s^{(3)}(t) = -\sin(2\pi f_1 t)w(t)$, and $s^{(4)}(t) = -\cos(2\pi f_1 t)w(t)$, where

$$w(t) = \begin{cases} \sqrt{\frac{2E_s}{T}}, & 0 \leq t \leq T, \\ 0 & \text{otherwise,} \end{cases} \quad (2.4)$$

and $f_1 = k/T$ is an integer multiple of the symbol time $1/T$. The first basis function is $\phi_1(t) = \sqrt{2/T} \sin(2\pi f_1 t)$ and the second basis function is $\phi_2(t) = \sqrt{2/T} \cos(2\pi f_1 t)$. The signal constellation in Figure 2.2 is called quadrature-phase shift-keying (QPSK).

Note that while we have lost information about the actual waveform that is used to generate the signals $s_i(t)$, we have gained a higher level of abstraction, which will make it much easier to discuss subsequent concepts of coding and modulation. Knowledge of the signal vector $\mathbf{s}^{(i)}$ implies knowledge of the transmitted message $m^{(i)}$, since, in a sensible

system, there is a one-to-one mapping between the two. The problem of decoding a received waveform is therefore equivalent to recovering the signal vector $\mathbf{s}^{(i)}$.

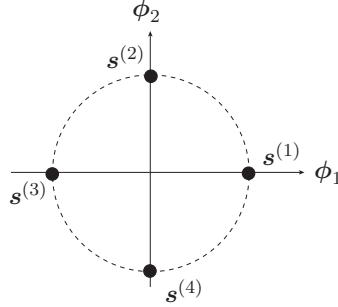


Figure 2.2: Illustration of four signals using two orthogonal basis functions.

This can be accomplished by passing the received signal waveform through a bank of correlators, each of which correlates $s^{(i)}(t)$ with one of the basis functions, performing the operation

$$\int_{-\infty}^{\infty} s^{(i)}(t) \phi_l(t) dt = \sum_{j=1}^N s_j^{(i)} \int_{-\infty}^{\infty} \phi_j(t) \phi_l(t) dt = s_l^{(i)}, \quad (2.5)$$

that is, the l th correlator recovers the l th component $s_l^{(i)}$ of the signal vector $\mathbf{s}^{(i)}$.

Later we will need the *squared Euclidean distance* between two signals $\mathbf{s}^{(i)}$ and $\mathbf{s}^{(j)}$, given as $d_{ij}^2 = |\mathbf{s}^{(i)} - \mathbf{s}^{(j)}|^2$, which is a measure of the noise resistance of these two signals. Note that

$$d_{ij}^2 = \sum_{l=1}^N \left(s_l^{(i)} - s_l^{(j)} \right)^2 \quad (2.6)$$

$$= \sum_{l=1}^N \left(\int_{-\infty}^{\infty} \left(s_l^{(i)} - s_l^{(j)} \right) \phi_l(t) dt \right)^2 \quad (2.7)$$

$$= \int_{-\infty}^{\infty} \left(s^{(i)}(t) - s^{(j)}(t) \right)^2 dt \quad (2.8)$$

is in fact the energy of the difference signal $(s^{(i)}(t) - s^{(j)}(t))$. (It is easier to derive (2.8) in the reverse direction, i.e., from bottom to top.)

It can be shown [28, 16] that this correlator receiver is optimal in the sense that no relevant information is discarded and minimum error probability can be attained, even

when the received signal contains additive white Gaussian noise. In this case the received signal $r(t) = s^{(i)}(t) + n_w(t)$ produces the received vector $\mathbf{r} = \mathbf{s}^{(i)} + \mathbf{n}$ at the output of the bank of correlators. The statistics of the noise vector \mathbf{n} can easily be evaluated, using the orthogonality of the basis waveforms and the noise autocorrelation function $E[n_w(t)n_w(t+\tau)] = \delta(\tau)N_0/2$, where $\delta(t)$ is *Dirac's delta function* and N_0 is the *one-sided noise power spectral density* (see Appendix 2.A). We obtain

$$\begin{aligned} E[n_l n_j] &= \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} E[n_w(\alpha) n_w(\beta)] \phi_l(\alpha) \phi_j(\beta) d\alpha d\beta \\ &= \frac{N_0}{2} \int_{-\infty}^{\infty} \phi_l(\alpha) \phi_j(\alpha) d\alpha = \begin{cases} N_0/2, & l = j, \\ 0, & \text{otherwise.} \end{cases} \end{aligned} \quad (2.9)$$

We are pleased to see that, courtesy of the orthonormal basis waveforms, the components of the random noise vector \mathbf{n} are all uncorrelated. Since $n_w(t)$ is a Gaussian random process, the sample values n_l, n_j are necessarily also Gaussian (For more detail see, e.g., [2]). From the above we conclude that the components of \mathbf{n} are independent Gaussian random variables with common variance $N_0/2$ and mean zero.

We have thus achieved a completely equivalent vector view of a communication system. The advantages of this point of view are manifold. Firstly, we need not concern ourselves with the actual choices of signal waveforms when discussing receiver algorithms, and, secondly, the difficult problem of waveform communication involving stochastic processes and continuous signal functions has been transformed into the much more manageable vector communication system involving only random vectors and signal vectors, and thirdly, we have gained a geometric view of communications. In this context, linear algebra, the tool for geometric operations, plays an important role in modern communication theory.

It is interesting to note that the vector representation is finite-dimensional, with the number of dimensions given by the dimensionality of the signal functions. The random function space is infinite-dimensional, however. The optimality of the correlator receiver [28] shows then that we need only be concerned with that part of the noise which is projected onto the finite-dimensional signal space. All other noise components are irrelevant. In fact, this is the way the optimality of the correlator receiver is typically proven: First it is shown that the noise components which are not part of the finite-dimensional signal space are irrelevant and need not be considered by the receiver. This leads directly to the finite-dimensional geometric signal space interpretation discussed above.

2.3 Optimum Receivers

If our bank of correlators produces a received vector $\mathbf{r} = \mathbf{s}^{(i)} + \mathbf{n}$, then an optimal detector will chose as message hypothesis, $\hat{m} = m^{(j)}$, the one which maximizes the conditional

probability

$$P[m^{(j)}|\mathbf{r}], \quad (2.10)$$

because this maximizes the overall probability of being correct, P_c , which can be seen from

$$P_c = \int_{\mathbf{r}} P[\text{correct}|\mathbf{r}]p(\mathbf{r})d\mathbf{r}, \quad (2.11)$$

that is, since $p(\mathbf{r})$ is non-negative everywhere, maximizing P_c can be achieved by maximizing $P[\text{correct}|\mathbf{r}]$ for each received \mathbf{r} . This is known as a maximum a posteriori (MAP) detection.

Using Bayes' rule, we obtain

$$P[m^{(j)}|\mathbf{r}] = \frac{P[m^{(j)}]p(\mathbf{r}|m^{(j)})}{p(\mathbf{r})}, \quad (2.12)$$

and, postulating that the signals are all used equally likely, it suffices to select $\hat{m} = m^{(j)}$ such that $p(\mathbf{r}|m^{(j)})$ is maximized. This, in turn, is the *maximum likelihood* (ML) receiver principle. It minimizes the signal error probability, but only for equally likely signals, in which case MAP and ML are equivalent.

Since $\mathbf{r} = \mathbf{s}^{(i)} + \mathbf{n}$ and \mathbf{n} is an additive Gaussian random vector independent of the signal $\mathbf{s}^{(i)}$, we may further develop the optimum receiver using $p(\mathbf{r}|m^{(j)}) = p_n(\mathbf{r} - \mathbf{s}^{(j)})$, which is an N -dimensional Gaussian density function given by

$$p_n(\mathbf{r} - \mathbf{s}^{(j)}) = \frac{1}{(\pi N_0)^{N/2}} \exp\left(-\frac{|\mathbf{r} - \mathbf{s}^{(j)}|^2}{N_0}\right). \quad (2.13)$$

Maximizing (2.10) is now seen to be equivalent to minimizing the Euclidean distance

$$|\mathbf{r} - \mathbf{s}^{(j)}|^2 \quad (2.14)$$

between the received vector and the hypothesized signal vector $\mathbf{s}^{(j)}$.

The decision rule (2.14) implies *decision regions* $\mathcal{D}^{(j)}$ for each signal point which consist of all the points in Euclidean N -space which are closer to $\mathbf{s}^{(j)}$ than any other signal point. These regions are also known as *Voronoi regions*, named after Georgy Feodosevich Voronoy, a Ukrainian mathematician of the late 19th century. These decision regions are illustrated for QPSK in Figure 2.3.

The probability of error given a particular transmitted signal $\mathbf{s}^{(i)}$ can now be interpreted as the probability that the additive noise \mathbf{n} carries the signal $\mathbf{s}^{(i)}$ outside its decision region $\mathcal{D}^{(i)}$. This probability can be calculated as

$$P_e(\mathbf{s}^{(i)}) = \int_{\mathbf{r} \notin \mathcal{D}^{(i)}} p_n(\mathbf{r} - \mathbf{s}^{(i)}) d\mathbf{r}. \quad (2.15)$$

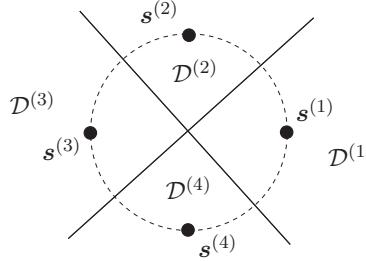


Figure 2.3: Decision regions for ML detection of equiprobable QPSK signals.

Equation (2.15) is, in general, very difficult to calculate, and simple expressions exist only for some special cases. The most important such special case is the two-signal error probability, which is defined as the probability that signal $\mathbf{s}^{(i)}$ is decoded as signal $\mathbf{s}^{(j)}$ assuming that there are only these two signals. In order to calculate the two-signal error probability we may disregard all signals except $\mathbf{s}^{(i)}, \mathbf{s}^{(j)}$. Take, for example, $\mathbf{s}^{(i)} = \mathbf{s}^{(1)}$ and $\mathbf{s}^{(j)} = \mathbf{s}^{(2)}$ in Figure 2.3. The new decision regions are $\mathcal{D}^{(i)} = \mathcal{D}^{(1)} \cup \mathcal{D}^{(4)}$ and $\mathcal{D}^{(j)} = \mathcal{D}^{(2)} \cup \mathcal{D}^{(3)}$. The decision region $\mathcal{D}^{(i)}$ is expanded to a half-plane and the probability of deciding on message $m^{(j)}$ when message $m^{(i)}$ was actually transmitted, known as the *pair-wise error probability*, is given by

$$P_{\mathbf{s}^{(i)} \rightarrow \mathbf{s}^{(j)}} = Q\left(\sqrt{\frac{d_{ij}^2}{2N_0}}\right), \quad (2.16)$$

where $d_{ij}^2 = |\mathbf{s}^{(i)} - \mathbf{s}^{(j)}|^2$ is the energy of the difference signal, and

$$Q(\alpha) = \frac{1}{\sqrt{2\pi}} \int_{\alpha}^{\infty} \exp\left(-\frac{\beta^2}{2}\right) d\beta, \quad (2.17)$$

is a non-elementary integral, referred to as the (Gaussian) *Q*-function. For a more detailed discussion, see [28]. In any case, Equation (2.16) states that the probability of error between two signals decreases exponentially with their squared Euclidean distance d_{ij} , due to the well-known overbound [16]

$$Q\left(\sqrt{d_{ij}^2/2N_0}\right) \leq (1/2) \exp(-d_{ij}^2/4N_0). \quad (2.18)$$

2.4 Matched Filters

The correlation operation (2.5) used to recover the signal vector components can be implemented as a filtering operation. The signal $s^{(i)}(t)$ is passed through a filter with impulse

response $\phi_l(-t)$ to obtain

$$u(t) = s^{(i)}(t) \star \phi_l(-t) = \int_{-\infty}^{\infty} s^{(i)}(\alpha) \phi_l(\alpha - t) d\alpha. \quad (2.19)$$

If the output of the filter $\phi_l(-t)$ is sampled at time $t = 0$, Equations (2.19) and (2.5) are identical, i.e.,

$$u(t = 0) = s_l^{(i)} = \int_{-\infty}^{\infty} s^{(i)}(\alpha) \phi_l(\alpha) d\alpha. \quad (2.20)$$

Of course some appropriate delay needs to be built into the system to guarantee that $\phi_l(-t)$ is causal. We shall not be concerned with this delay in our treatise.

The maximum-likelihood receiver now minimizes $|\mathbf{r} - \mathbf{s}^{(i)}|^2$, or equivalently maximizes

$$2 \cdot \mathbf{r} \cdot \mathbf{s}^{(i)} - |\mathbf{s}^{(i)}|^2, \quad (2.21)$$

where we have neglected the term $|\mathbf{r}|^2$, which is common to all the hypotheses. The correlation $\mathbf{r} \cdot \mathbf{s}^{(i)}$ is the central part of (2.21) and can be implemented in two different ways: as the basis-function-matched filter receiver, performing a summation after correlation to combine the different signal components, i.e.,

$$\mathbf{r} \cdot \mathbf{s}^{(i)} = \sum_{j=1}^N r_j s_j^{(i)} = \sum_{j=1}^N s_j^{(i)} \int_{-\infty}^{\infty} r(t) \phi_j(t) dt, \quad (2.22)$$

or as the signal matched filter receiver directly computing

$$\mathbf{r} \cdot \mathbf{s}^{(i)} = \int_{-\infty}^{\infty} r(t) \sum_{j=1}^N s_j^{(i)} \phi_j(t) dt = \int_{-\infty}^{\infty} r(t) s^{(i)}(t) dt. \quad (2.23)$$

The two different receiver implementations are illustrated in Figure 2.4. Usually, if the number of basis functions is much smaller than the number of signals, the basis-function-matched filter implementation is preferred. Spread spectrum systems [29], where it is more expedient to use (2.23), are notable exceptions.

The optimality of the correlation receiver (2.5) implies that both receiver structures of Figure 2.4 are optimal also. The sampled outputs of the matched filters are therefore sufficient for optimal detection of the transmitted message, and hence form what is known as *sufficient statistics* [5].

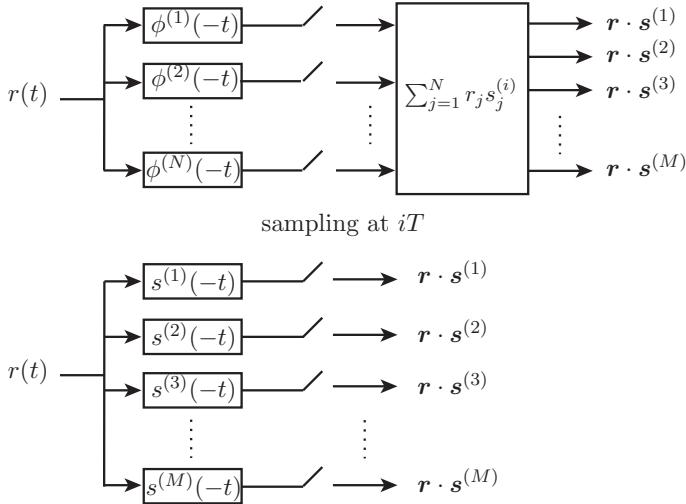


Figure 2.4: The basis-function matched filter receiver shown on top and the signal matched filter receiver shown in the bottom. M is the number of messages in the message set.

2.5 Message Sequences

In practice, our signals $s^{(i)}(t)$ will not be limited to a small set of messages, but consist of large data transfer that requires the transmission of sequences of messages, analogous to how sentences are made up of sequences of letters. The sequences of messages are then represented by identical, time-displaced waveforms, called signal pulses. The complete “phrase” of our message is now described by

$$s^{(i)}(t) = \sum_{r=-l}^l a_r^{(i)} p(t - rT), \quad (2.24)$$

where $p(t)$ is some convenient signal pulse waveform, the $a_r^{(i)}$ are the discrete symbol values from our signal alphabet, which is finite, for example, binary signaling: $a_r^{(i)} \in \{-1, 1\}$, and $2l + 1$ is the length of the sequence in symbols. The parameter T is the time delay between successive pulses, also called the *symbol period* or *symbol rate*. The output of the filter matched to the entire signal $s^{(i)}(t)$ is what we need to compute in principle and is

given as

$$\begin{aligned} y(t) &= \int_{-\infty}^{\infty} r(\alpha) \sum_{r=-l}^l a_r^{(i)} p(\alpha - rT - t) d\alpha \\ &= \sum_{r=-l}^l a_r^{(i)} \int_{-\infty}^{\infty} r(\alpha) p(\alpha - rT - t) d\alpha, \end{aligned} \quad (2.25)$$

and the sampled value of $v(t)$ at $t = 0$ is given by

$$y(t = 0) = \sum_{r=-l}^l a_r^{(i)} \int_{-\infty}^{\infty} r(\alpha) p(\alpha - rT) d\alpha = \sum_{r=-l}^l a_r^{(i)} y_r, \quad (2.26)$$

where $y_r = \int_{-\infty}^{\infty} r(\alpha) p(\alpha - rT) d\alpha$ is the output of a filter matched to the pulse $p(t)$ sampled at time $t = rT$. We see that the signal-matched filter can be implemented conveniently by a single pulse matched filter, whose output $y(t)$ is simply sampled at multiples of the symbol time T , a much more economical approach than a brute-force implementation of (2.25).

The time-shifted waveforms $p(t - rT)$ may serve as orthonormal basis functions, if they fulfill the orthogonality condition

$$\int_{-\infty}^{\infty} p(t - rT) p(t - hT) dt = \delta_{rh}. \quad (2.27)$$

The output waveform of the pulse-matched filter $p(-t)$ in the absence of noise is given by

$$z(t) = \sum_{r=-l}^l a_r^{(i)} p(t - rT) \star p(-t) = \sum_{r=-l}^l a_r^{(i)} g(t - rT), \quad (2.28)$$

where $g(t - rT) = \int_{-\infty}^{\infty} p(\alpha - rT) p(\alpha - t) d\alpha$ is the composite pulse/pulse-matched filter waveform. If (2.27) holds, $z(t)$ is completely separable and the r th sample value $z(rT) = z_r$ depends only on $a_r^{(i)}$, i.e., $z_r = a_r^{(i)}$, even if the pulses $p(t - rT)$ overlap in time. If successive symbols $a_r^{(i)}$ are chosen independently the system may then be viewed as independently using a 1-dimensional signal constellation system $2l + 1$ times. This results in tremendous savings in complexity and represents the state of the art of digital signaling.

Condition (2.27) ensures that symbols transmitted at different times do not interfere with each other, that is, we have *intersymbol interference*-free signaling. Condition (2.27) is known as the Nyquist criterion, which requires that the composite waveform $g(t)$ passes through zero at all multiples of the sampling time T , except $t = 0$, i.e.,

$$\int_{-\infty}^{\infty} p(t - rT) p(t - hT) dt = \delta_{rh} \Rightarrow g(rT) = \begin{cases} 1, & \text{if } r = 0, \\ 0, & \text{otherwise.} \end{cases} \quad (2.29)$$

An example of such a composite pulse $g(t)$ together with its Fourier transform $G(f)$ is shown in Figure 2.5. Note that $G(f)$ also happens to be the energy spectrum of the transmitted pulse $p(t)$ since $G(f) = P(f)P^*(f) = |P(f)|^2$.

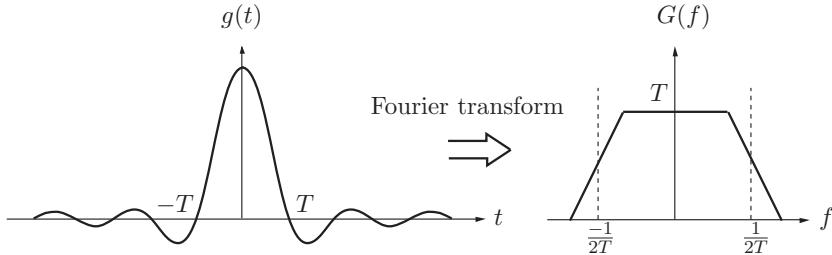


Figure 2.5: Example of a Nyquist pulse with trapezoid frequency spectrum.

The Nyquist criterion can be translated into the frequency domain via the Fourier transform by observing that

$$\begin{aligned}
 g(rT) &= \int_{-\infty}^{\infty} G(f) e^{2\pi j f rT} df \\
 &= \sum_{n=-\infty}^{\infty} \int_{(2n-1)/2T}^{(2n+1)/2T} G(f) e^{2\pi j f rT} df \\
 &= \int_{-1/2T}^{1/2T} \sum_{n=-\infty}^{\infty} G\left(f + \frac{n}{T}\right) e^{2\pi j f rT} df. \tag{2.30}
 \end{aligned}$$

If the folded spectrum

$$\sum_{n=-\infty}^{\infty} G\left(f + \frac{n}{T}\right) = T, \quad -\frac{1}{2T} \leq f \leq \frac{1}{2T} \tag{2.31}$$

equals a constant, T , for normalization reasons, the integral in (2.30) evaluates to $g(rT) = \sin(\pi r)/(\pi r) = \delta_{0r}$, i.e., the sample values of $g(t)$ are zero at all non-zero multiples of T , as required. Equation (2.31) is the spectral form of the Nyquist criterion for no intersymbol interference (compare Figure 2.5).

A very popular Nyquist pulse is the spectral raised-cosine pulse whose Fourier trans-

form is given by

$$G(f) = \begin{cases} T, & |f| \leq \frac{1-\beta}{2T}, \\ \frac{T}{2} \left(1 - \sin\left(\frac{\pi(2T|f|-1)}{2\beta}\right) \right), & \frac{1-\beta}{2T} \leq |f| \leq \frac{1+\beta}{2T}, \\ 0, & \frac{1+\beta}{2T} \leq |f|, \end{cases} \quad (2.32)$$

where $\beta \in [0, 1]$ is the roll-off factor of the pulse, whose bandwidth is $(1 + \beta)/(2T)$. In practical systems, values of β around 0.3 are routinely used, and pulses with β as low as 0.1 can be approximated by realizable filters, producing spectrally compact signals.

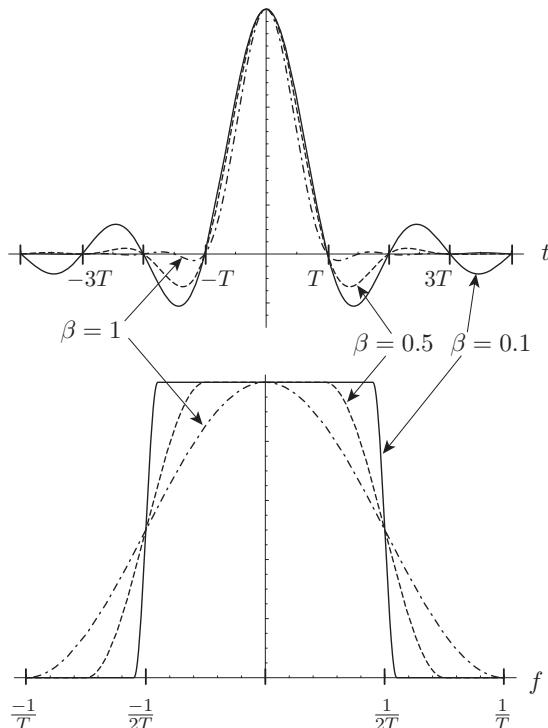


Figure 2.6: Spectral raised-cosine Nyquist pulses and their Fourier transforms.

The impulse response corresponding to $G(f)$ is

$$g(t, \beta) = \frac{\sin(\pi t/T)}{\pi t/T} \frac{\cos(\beta\pi t/T)}{1 - (2\beta t/T)^2}. \quad (2.33)$$

Pulses $g(t, \beta)$ and their spectra are shown in Figure 2.6 for various the roll-off factors.

These observations lead to the extremely important and popular root-Nyquist signaling concept shown in Figure 2.7. This name stems from the fact that the actual pulse shape used for transmission, $p(t)$, is the inverse Fourier transform of $\sqrt{G(f)}$, the Nyquist pulse. In the case of a rectangular brick-wall frequency response, however, $p(t) = g(t) = \frac{\sin(\pi t/T)}{\pi t/T}$.

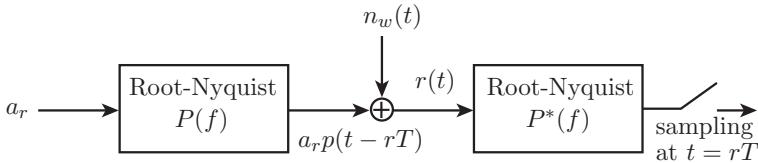


Figure 2.7: Communications with root-Nyquist signaling for optimal noise suppression.

While Nyquist signaling achieves excellent spectral efficiencies, there are some implementation related difficulties associated with it. Since the pulses are not time duration limited, they need to be approximated over some finite time duration, which causes some spectral spill-over. A more severe problem occurs with timing errors in the sampling. Inaccurate timing will generate a possibly large number of adjacent symbols to interfere with each other's detection, making timing very crucial. Transmission over non-flat frequency channels also causes more severe intersymbol interference than with some other pulse shapes. This interference needs to be compensated or equalized. Equalization is discussed in standard textbooks on digital communications, e.g., [16, 21].

2.6 The Complex Equivalent Baseband Model

In practical applications, one often needs to shift a narrow-band signal into a higher frequency band for purposes of transmission. The reason for that may lie in the transmission properties of the physical channel, which only permits the passage of signals in certain, usually high, frequency bands. This occurs, for example, in radio transmission. The process of shifting a signal in frequency is called modulation with a carrier frequency. Modulation is also important for wire-bound transmission, since it allows the coexistence of several signals on the same physical medium, all residing in different frequency bands; this is known as *frequency division multiplexing* (FDM). Probably the most popular modulation method for digital signals is *quadrature double side-band suppressed carrier* (DSB-SC) modulation.

DSB-SC modulation is a simple linear shift in frequency of a signal $x(t)$ with low-frequency content, called a *baseband signal*, into a higher frequency band by multiplying $x(t)$ with a cosine or sine waveform with carrier frequency f_0 , as shown in Figure 2.8, giving a carrier signal

$$s_0(t) = x(t)\sqrt{2} \cos(2\pi f_0 t), \quad (2.34)$$

where the factor $\sqrt{2}$ is used to make the powers of $s_0(t)$ and $x(t)$ equal.

If our baseband signal $x(t)$ occupies frequencies from 0 to W Hz, $s_0(t)$ occupies frequencies from $f_0 - W$ to $f_0 + W$ Hz, an expansion of the bandwidth by a factor of 2. But we quickly note that we can put another orthogonal signal, namely $y(t)\sqrt{2} \sin(2\pi f_0 t)$, into the same frequency band and that both baseband signals $x(t)$ and $y(t)$ can be recovered by the demodulation operation shown in Figure 2.8, known as a *product demodulator*, where the low-pass filters $W(f)$ serve to reject unwanted out-of-band noise and signals. It can be shown (see, for example, [28]) that this arrangement is optimal; i.e., no information or optimality is lost by using the product demodulator for DSB-SC modulated signals.

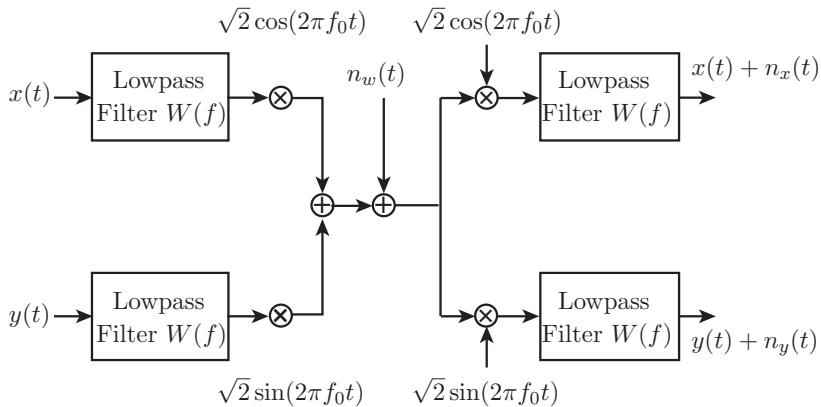


Figure 2.8: Quadrature DSB-SC modulation/demodulation stages.

If the synchronization between modulator and demodulator is perfect, the two signals, $x(t)$, the *in-phase* signal, and $y(t)$, the *quadrature* signal, are recovered independently without affecting each other. The DSB-SC modulated bandpass channel is then, in essence, a dual channel for two independent signals, each of which may carry an independent data stream.

In view of our earlier approach using *basis functions*, we may want to view pairs of inputs to the two channels as a 2-dimensional signal. Since these two dimensions are intimately linked through the carrier modulation, and since bandpass signals are so ubiquitous in digital communications, a complex notation for bandpass signals has been

adopted widely. In this notation, the in-phase signal $x(t)$ is real, the quadrature signal $jy(t)$ is an imaginary signal, and the bandpass signal $s_0(t)$ can be expressed as

$$\begin{aligned} s_0(t) &= x(t)\sqrt{2}\cos(2\pi f_0 t) - y(t)\sqrt{2}\sin(2\pi f_0 t) \\ &= \operatorname{Re}\left[(x(t) + jy(t))\sqrt{2}e^{2\pi j f_0 t}\right], \end{aligned} \quad (2.35)$$

where $s(t) = (x(t) + jy(t))$ is called the *complex envelope* of $s_0(t)$.

If both signals are sequences of identical pulses $p(t)$, the complex envelope becomes

$$s(t) = \sum_{r=-l}^l (a_r + jb_r) p(t - rT) = \sum_{r=-l}^l c_r p(t - rT), \quad (2.36)$$

where c_r is a complex (2-dimensional) number, representing both the in-phase and the quadrature information symbol.

The noise entering the system is demodulated and produces the two low-pass noise waveforms $n_x(t)$ and $n_y(t)$, given by

$$n_x(t) = \sqrt{2} \int_{-\infty}^{\infty} n_w(\alpha) \cos(2\pi f_0 \alpha) w(t - \alpha) d\alpha; \quad (2.37)$$

$$n_y(t) = \sqrt{2} \int_{-\infty}^{\infty} n_w(\alpha) \sin(2\pi f_0 \alpha) w(t - \alpha) d\alpha, \quad (2.38)$$

where $w(t) = \sin(2\pi tW)/2\pi tW$ is the impulse response of an ideal low-pass filter with cutoff frequency W . The autocorrelation function of $n_x(t)$ (and $n_y(t)$ analogously) is given by

$$\begin{aligned} \operatorname{E}[n_x(t)n_x(t+\tau)] &= 2 \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \cos(2\pi f_0 \alpha) \cos(2\pi f_0 \beta) w(t - \alpha) w(t + \tau - \beta) \\ &\quad \times \operatorname{E}[n_w(\alpha) n_w(\beta)] d\alpha d\beta \\ &= \frac{N_0}{2} \int_{-\infty}^{\infty} (1 + \cos(4\pi f_0 \alpha)) w(t - \alpha) w(t + \tau - \alpha) d\alpha \\ &= \frac{N_0}{2} \int_{-\infty}^{\infty} w(t - \alpha) w(t + \tau - \alpha) d\alpha \\ &= \frac{N_0}{2} \int_{-W}^W e^{-2\pi j f \tau} df = N_0 W \frac{\sin(2\pi W \tau)}{2\pi W \tau}, \end{aligned} \quad (2.39)$$

where we have used Parseval's relationships ([28], pp. 237–238) in the third step, i.e., the power of $n_x(t)$ equals $\operatorname{E}[n_x^2(t)] = WN_0$. Equation (2.39) is the correlation function of

white noise after passage through a low-pass filter of bandwidth W , i.e., the multiplication with the demodulation carrier has no influence on the statistics of the output noise $n_x(t)$ and can be ignored. Similarly, the cross-correlation function between $n_x(t)$ and $n_y(t)$ is evaluated as

$$\begin{aligned} \text{E}[n_x(t)n_y(t + \tau)] &= 2 \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \cos(2\pi f_0 \alpha) \sin(2\pi f_0 \beta) w(t - \alpha) w(t + \tau - \beta) \\ &\quad \times \text{E}[n_w(\alpha)n_w(\beta)] d\alpha d\beta \\ &= \frac{N_0}{2} \int_{-\infty}^{\infty} \sin(4\pi f_0 \alpha) w(t - \alpha) w(t + \tau - \alpha) d\alpha = 0. \end{aligned} \quad (2.40)$$

This system can be modeled as two parallel channels affected by two independent Gaussian noise processes $n_w^{(x)}(t)$ and $n_w^{(y)}(t)$ as illustrated in Figure 2.9.

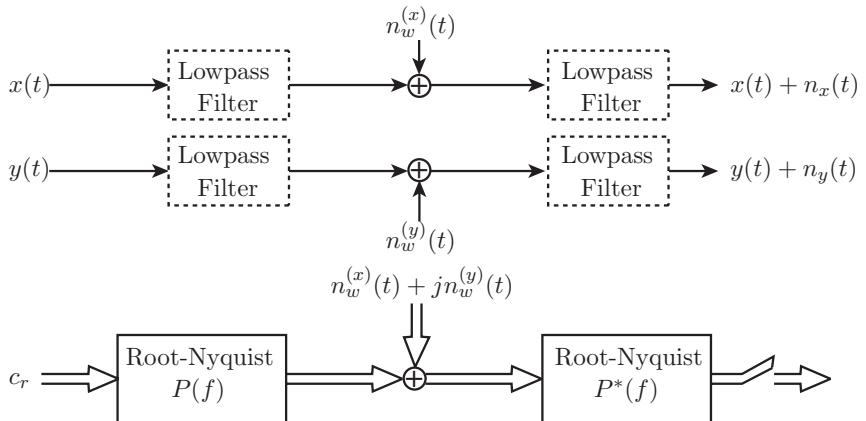


Figure 2.9: Modeling of a DSB-SC system in additive white Gaussian noise by two independent AWGN channels. These two channels can be represented as one complex channel model, shown in the lower part of the figure, and referred to as the equivalent complex baseband model for DSB-SC modulation.

Equivalently we can use a complex model with complex noise $n(t) = n_x(t) + j n_y(t)$, whose correlation is given by

$$\text{E}[n(t)n^*(t)] = 2N_0 W \frac{\sin(2\pi W \tau)}{2\pi W \tau}, \quad (2.41)$$

which is a shorthand version of (2.39) and (2.40). We thus have arrived at the complex equivalent baseband model also shown in Figure 2.9, which takes at its input complex numbers c_r , passes them through complex (dual) modulator and demodulator filters, and feeds the sampled values y_r into a complex receiver. Note that as long as we adopt the convention always to use a receiver filter ($P(f)$ in Figure 2.9), we may omit the low-pass filter in the model, since it can be subsumed into the receiver filter, and the noise source can be made an ideal white Gaussian noise source ($n_x(t) = n_w^{(x)}(t)$, $n_y(t) = n_w^{(y)}(t)$) with correlation function

$$\mathbb{E}[n(t)n^*(t)] = N_0\delta(t). \quad (2.42)$$

If Nyquist pulses are used with independent complex data symbols c_r , each sample is in fact an independent, 2-dimensional signal constellation, hence the ubiquity of 2-dimensional signal constellations. Figure 2.10 shows some of the more popular 2-dimensional signal constellations. The signal points correspond to sets of possible complex values which c_r can assume.

We now assume for the remainder of this book that all signal constellations are normalized so that their average energy is unity. This requires that the complex signal point c_r needs to be multiplied with the amplification factor $\sqrt{E_s}$ in order to generate the average signal energy E_s .

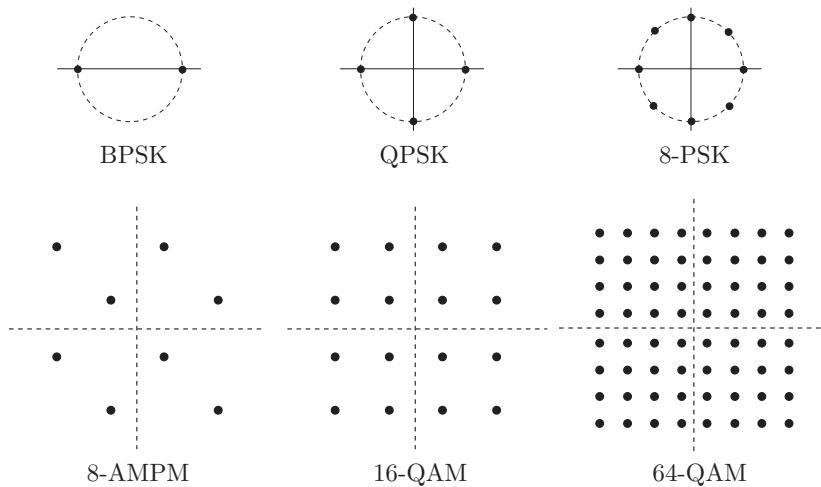


Figure 2.10: Popular 2-dimensional signal constellations.

2.7 Spectral Behavior

Since bandwidth has become an increasingly treasured resource, an important parameter of how efficiently a system uses its allotted bandwidth is the bandwidth efficiency η , defined in (1.2) as

$$\eta = \frac{\text{Bit Rate}}{\text{Channel Bandwidth } W} [\text{bits/s/Hz}]. \quad (2.43)$$

If raised-cosine Nyquist signaling is used with a roll-off factor of 0.3, BPSK achieves a bandwidth efficiency of $\eta = 0.884$ bits/s/Hz at an E_b/N_0 of 8.4 dB as shown in Figure 1.8. The bandwidth efficiency of QPSK is twice that of BPSK for the same of E_b/N_0 , because QPSK uses the complex dimension of the signal space. Also, 8-PSK is less power efficient than 8-AMPM (also called 8-cross) due to the equal energy constraint of the different signals, and the resulting smaller Euclidean distances between signal points. It can be noted from Figure 1.8 how bandwidth can be traded for power efficiency and vice versa, even without applying any coding. All performance points for uncoded signaling lie on a line parallel to the Shannon bound.

In order to evaluate the spectral efficiency in (2.43), we must first find the power spectrum of the complex pulse train

$$s(t - \delta) = \sum_{r=-l}^l c_r p(t - rT - \delta), \quad (2.44)$$

where we have introduced a random delay $\delta \in [0, T[$, and we further assume that the distribution of δ is uniform. This will simplify our mathematics and has no influence on the power spectrum, since, surely, knowledge of the delay of $s(t)$ can have no influence on its spectral power distribution.

The advantage of (2.44) is that, if we let $l \rightarrow \infty$, $s(t - \delta)$ can be made stationary. Starting with the autocorrelation function

$$\begin{aligned} R_s(t, t + \tau) &= E[s^*(t)s(t + \tau)] = E[s^*(t - \delta)s(t + \tau - \delta)] \\ &= \sum_{p=-\infty}^{\infty} \sum_{q=-\infty}^{\infty} E[c_p^* c_q] E_{\delta}[p(t-pT-\delta)p(t+\tau-qT-\delta)], \end{aligned} \quad (2.45)$$

let us assume that the discrete autocorrelation of the symbols c_r is stationary, i.e.,

$$R_{cc}(r) = E[c_q^* c_{q+r}] \quad (2.46)$$

depends only on r . This lets us rewrite (2.45) as

$$\begin{aligned}
 R_s(t, t + \tau) &= \sum_{r=-\infty}^{\infty} R_{cc}(r) \sum_{q=-\infty}^{\infty} E_{\delta}[p(t - qT - \delta)p(t + \tau - (q + r)T - \delta)] \\
 &= \sum_{r=-\infty}^{\infty} R_{cc}(r) \sum_{q=-\infty}^{\infty} \frac{1}{T} \int_0^T p(t - qT - \delta)p(t + \tau - (q + r)T - \delta) d\delta \\
 &= \sum_{r=-\infty}^{\infty} R_{cc}(r) \frac{1}{T} \int_0^T p(\delta)p(\delta + \tau - rT) d\delta,
 \end{aligned} \tag{2.47}$$

where the last term, $\int_0^T p(\delta)p(\delta + \tau - rT) d\delta = R_{pp}(\tau - rT)$, is the pulse autocorrelation function of $p(t)$, which depends only on τ and r , making $R_s(t, t + \tau) = R_s(\tau)$ stationary.

In order to find the power spectral density of $s(t)$, we now merely need the Fourier transform of $R_s(\tau)$, i.e.,

$$\begin{aligned}
 \Phi_{cc}(f) &= \frac{1}{T} \int_{-\infty}^{\infty} \sum_{r=-\infty}^{\infty} R_{cc}(r) R_{pp}(\tau - rT) e^{-j2\pi f\tau} d\tau \\
 &= \frac{1}{T} \sum_{r=-\infty}^{\infty} R_{cc}(r) e^{-j2\pi frT} \int_{-\infty}^{\infty} R_{pp}(\tau) e^{-j2\pi f\tau} d\tau \\
 &= \frac{1}{T} C(f) G(f),
 \end{aligned}$$

where

$$C(f) = \sum_{r=-\infty}^{\infty} R_{cc}(r) e^{-j2\pi frT} \tag{2.48}$$

is the spectrum-shaping component resulting from the correlation of the complex symbol sequences c_r , and $G(f) = |P(f)|^2$ is the energy spectrum of the symbol pulse $p(t)$ (page 37).

While the spectral factor due to the correlation of the pulses, $C(f)$, can be used to help shape the signal spectrum as in partial-response signaling ([16], pp. 548 ff) or correlative coding [18], $R_{cc}(r)$ is often an impulse δ_{0r} , corresponding to choosing uncorrelated, zero-mean-valued symbols c_r , and $C(f)$ is flat. If this is the case, the spectrum of the transmitted signal is exclusively shaped by the symbol pulse $p(t)$. This holds for example for QPSK, if the symbols are chosen independently and with equal probability. In general it holds for any constellation and symbol probabilities whose discrete autocorrelation $R_{cc}(r) = \delta_{0r}$, and whose symbol mean² $E[c_q] = 0$. Constellations for which this is achieved by the uniform probability distribution for the symbols are called *symmetric constellations*

²If the mean of the symbols c_r is not equal to zero, discrete frequency components appear at multiples of $1/T$ (see [16], Section 3.4).

(e.g., M-PSK, 16-QAM, 64-QAM, etc.). We will see in Chapter 3 that the uncorrelated nature of the complex symbols c_r is the basis for the fact that TCM does not shape the signal spectrum (also discussed in [2]).

2.8 Advanced Modulation Methods

2.8.1 OFDM

Orthogonal Frequency-Division Multiplexing (OFDM) is a modulation method where a potentially large number of narrowband carrier signals are aggregated into a high-dimensional modulation format. It is based on the observation that two frequency bursts

$$s_1(t) = \cos 2\pi f_1 t \text{ and } s_2(t) = \cos 2\pi(f_1 + \Delta f)t, \quad 0 \leq t \leq T \quad (2.49)$$

are orthogonal if $\Delta f = 1/T$. OFDM is now simply the aggregation of N pairs of such cosine and sine frequency burst, spaced at frequencies $f_i = f_0 + i/T$. While the system with two signals may not be spectrally efficient, if we aggregate a large number of carriers, spectral efficiencies that exceed those achievable even with narrow Nyquist signaling are possible. An illustration of OFDM is shown in Figure 2.11.

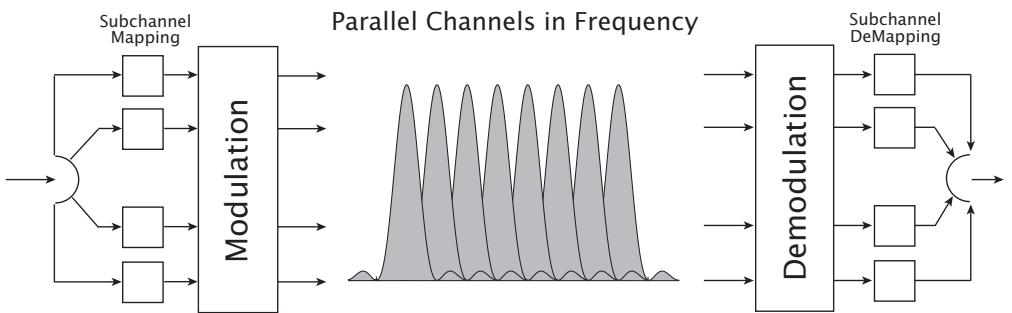


Figure 2.11: Principle of orthogonal frequency-division multiplexing (OFDM).

The spectrum of OFDM is made up of $\sin(x)/x$ carriers that overlap in frequency, but are orthogonal. The spectral efficiency of OFDM is given by

$$\eta_{\text{OFDM}} \approx \frac{N+10}{N\Delta f} \times \frac{2}{T} \approx 2 \quad [\text{symbols/s/Hz}],$$

that is, very close to ideal Nyquist signaling. A number of modern telecommunications standards have adopted OFDM as the modulation format of choice. Apart from efficient

spectrum utilization, OFDM offers unprecedented flexibility in the assignment of frequencies and fine tuning of data rates.

In practice, the generation of OFDM via discrete multiple carriers is very complex and inefficient, and a computationally much more efficient way to generate the OFDM signal is via the discrete Fourier transform (DFT) using the inverse DFT to transmit, along with the DFT at the receiver. The inverse DFT computes the complex time samples s_k in $\mathbf{s} = [s_0, \dots, s_{N-1}]$, as

$$s_k = \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} c_n e^{-j2\pi nk/N}, \quad \forall k : 0 \leq k < N$$

each time sample is then transmitted using a root-Nyquist transmission system as shown in Figure 2.12, and the original symbols c_n can be recovered with the DFT as

$$\begin{aligned} c_n &= \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} s_k e^{j2\pi kn/N}, \quad \forall n : 0 \leq n < N \\ &= \frac{1}{N} \sum_{k=0}^{N-1} \sum_{n'=0}^{N-1} c_{n'} e^{j2\pi k(n-n')/N} \\ &= \frac{1}{N} \sum_{n'=0}^{N-1} c_{n'} \underbrace{\sum_{k=0}^{N-1} e^{j2\pi k(n-n')/N}}_{=0 \text{ for } n \neq n'} = c_n; \text{ for } n = n' \end{aligned}$$

The transmitted symbols are thus recovered in an ideal system, and since the DFT can be computed as a fast-fourier transform with a complexity of order $\mathcal{O}(N \log(N))$, substantial complexity can be saved in modulating and demodulating.

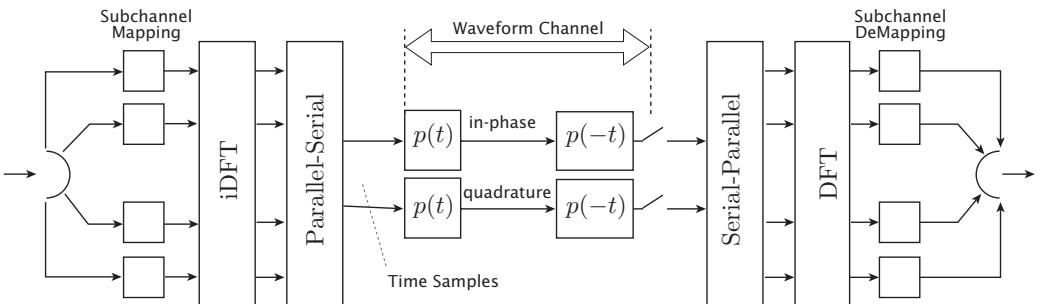


Figure 2.12: Implementation of an OFDM system using the discrete Fourier transform.

2.8.2 Multiple Antenna Channels (MIMO Channels)

As we have seen in Chapter 1, there exists a sharp limit on the achievable data rate on a given channel, the channel capacity, and much of this book deals with coding techniques to approach this limit. Further increases in the data carrying capacity of a channel can only be achieved by modifying the channel itself. One method is to use many parallel channels, rather than a single channel. For wireless systems this is achieved by using an array of transmit and an array of receive antennas as illustrated in Figure 2.13, which illustrates a MIMO system for a wireless router application. This methodology has rapidly been integrated into some modern high-capacity wireless systems, such as the IEEE WiFi standards, and the next generation cellular systems LTE [30, 31], which, incidentally, also use OFDM as the modulation method.

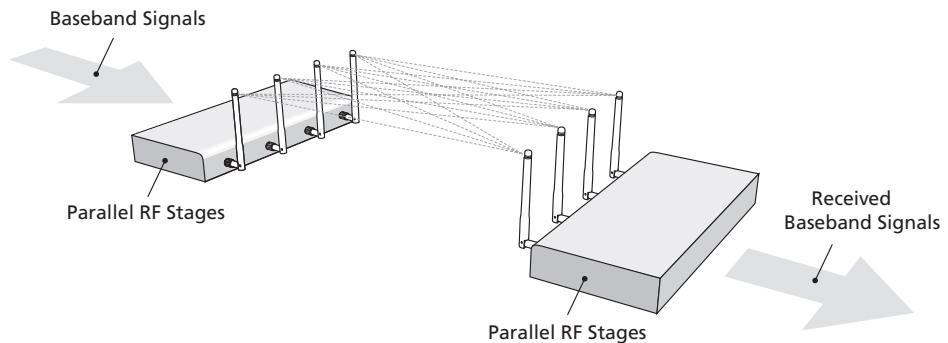


Figure 2.13: Multiple antenna communications system.

Each transmit antenna is operated by a separate DSB-SC modulated signal, and therefore each channel from transmit antenna i to receive antenna j is described by a complex path gain h_{ij} . This constitutes the most simple model which does not suffer from inter-symbol interference, given that the symbol rate is slower than the channel dispersion. The composite channel can be described by the complex vector equation

$$\mathbf{y}_r = \mathbf{H}\mathbf{c}_r + \mathbf{n}_r, \quad (2.50)$$

where $\mathbf{y} = (y_{1r}, \dots, y_{Nr})$ is the N_r -dimensional received complex vector at time r , $\mathbf{c}_r = (c_{1r}, \dots, c_{Nr})$ is the N_t -dimensional vector of complex transmit symbols, \mathbf{H} is the $N_r \times N_t$ matrix of complex path gains, and \mathbf{n} is a N_r -dimensional noise vector. Note that each receive antenna is demodulated independently, hence there are N_r independent noise sources. The mathematical model of this channel is illustrated in Figure 2.14.

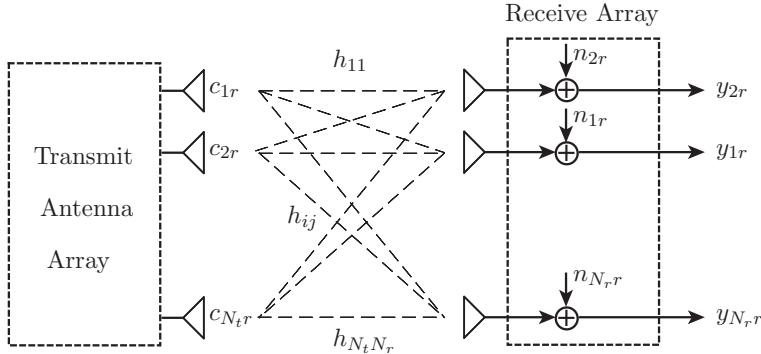


Figure 2.14: Multiple-input multiple-output (MIMO) channel for increased capacity in modern wireless systems.

The Shannon capacity of this channel can be calculated by applying the multi-dimensional capacity formula [4, 6] and is given by

$$C = \log_2 \left[\det \left(\mathbf{I}_{N_r} + \frac{E_s}{N_0 N_{N_t}} \mathbf{H} \mathbf{H}^+ \right) \right]; \quad [\text{bits/channel use}] \quad (2.51)$$

which is a generalization of the capacity formula (1.11) discussed in Chapter 1. E_s is the energy of the entire $2N_t$ -dimensional space-time symbol, which is spread over the N_t transmit antennas. Reducing (2.51) to a single (real) dimension leads back to (1.11).

We now apply the singular-value decomposition (SVD) to the channel matrix \mathbf{H} to obtain a decomposition of the channel equation. The SVD [10] of this matrix is given by

$$\mathbf{H} = \mathbf{U} \mathbf{D} \mathbf{V}^+, \quad (2.52)$$

where both \mathbf{U} and \mathbf{V} are unitary matrices of size $N_r \times N_r$ and $N_t \times N_t$, and therefore invertible, i.e., $\mathbf{U}^{-1} = \mathbf{U}^+$. The beauty of the SVD is that the matrix \mathbf{D} is diagonal and contains the *singular values* $d_1 \geq \dots \geq d_n \geq 0$; $n = \min(N_t, N_r)$ of \mathbf{H} , which are the square roots of the non-zero eigenvalues of $\mathbf{H} \mathbf{H}^+$. (If \mathbf{H} is a square and hermitian matrix, the singular values are the eigenvalues.)

Simple manipulations now lead to the following decomposed equivalent channel form:

$$\begin{aligned} \mathbf{y} &= \mathbf{H} \mathbf{c} + \mathbf{n} \\ &= \mathbf{U} \mathbf{D} \mathbf{V}^+ \mathbf{c} + \mathbf{n}, \\ \tilde{\mathbf{y}} = \mathbf{U}^+ \mathbf{y} &= \mathbf{D} \tilde{\mathbf{c}} + \tilde{\mathbf{n}}. \end{aligned} \quad (2.53)$$

Due to the unitary nature of \mathbf{U} , \mathbf{n} and $\tilde{\mathbf{n}}$ have the same statistics.

Equation (2.53) is now simply an aggregate of $\min(N_r, N_t)$ parallel channels as illustrated in Figure 2.15. Each channel has power d_j , corresponding to the j th singular value of \mathbf{H} . That is, simple linear pre-, and postprocessing by multiplying the transmitted and received symbol vectors by unitary matrices decomposes the channel into parallel channels. Alas, this is only possible if the channel is known at the transmitter. Without that knowledge, efficient transmission over the MIMO channel is much more complicated.

Compensating for the variable gain of these parallel channels as shown by the dashed box in Figure 2.15 has no effect on capacity and turns the channels into n parallel channels affected by additive noise sources with variances $N_0/d_1^2, \dots, N_0/d_n^2$.

Then the capacity of this set of parallel channels is given by the following [8, Section 7.5, pp. 343 ff]

Theorem 2.1 *The information theoretic capacity of n parallel additive white Gaussian noise channels with noise variances $N_0/d_1^2, \dots, N_0/d_n^2$ is given by*

$$C = \sum_{j=1}^n \log\left(1 + \frac{d_j^2 E_n}{N_0}\right) = \sum_{j=1}^n \log\left(\frac{d_j^2 \mu}{N_0}\right) \quad (2.54)$$

and is achieved by the energy allocation

$$\begin{aligned} \frac{2\sigma_n^2}{d_n^2} + E_n &= \mu, & \sigma_n^2 < \mu \\ E_n &= 0, & \sigma_n^2 \geq \mu. \end{aligned} \quad (2.55)$$

This theorem is known as the *Waterfilling Theorem*. It says that the available energy should be distributed such that low-noise channels receive more energy than high noise

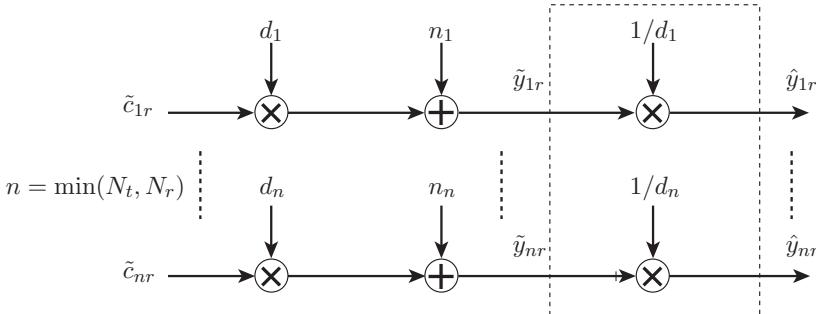


Figure 2.15: Decomposition of a MIMO channel into parallel channels via the SVD.

channels. It can be visualized in Figure 2.16, where power levels are shown in black, by thinking of the total available power as liquid which is poured into connected containers whose base level is at the height of the noise power—think of the noise as sediment.

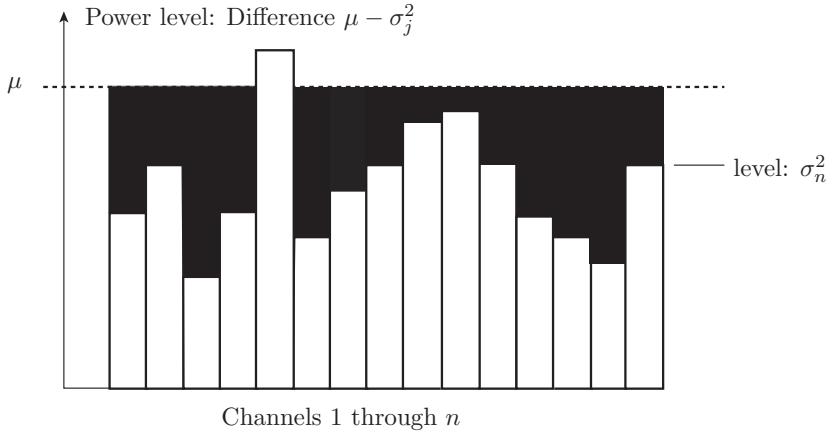


Figure 2.16: Illustration of the water-filling capacity theorem.

Proof: Consider

$$I(\mathbf{x}; \hat{\mathbf{y}}) \stackrel{(1)}{\leq} \sum_{j=1}^n I(x_j; \hat{y}_j) \quad (1) \text{ independent } x_n \quad (2.56)$$

$$\stackrel{(2)}{\leq} \underbrace{\sum_{n=1}^n \frac{1}{2} \log \left(1 + \frac{E_n}{\sigma_n^2} \right)}_{f(\mathbf{E})} \quad (2) \text{ Gaussian } x_n \quad (2.57)$$

$$f(\mathbf{E})$$

Since equality can be achieved in both inequalities above, the next step is to find the maximizing energy distribution $\mathbf{E} = [E_1, \dots, E_N]$. We identify the following sufficient and necessary conditions via constrained maximization using a Lagrange multiplier:

$$\begin{aligned} \frac{\partial f(\mathbf{E})}{\partial E_n} &\leq \lambda, \\ \frac{1}{2(\sigma_n^2 + E_n)} &\leq \lambda, \\ \sigma_n^2 + E_n &\leq \frac{1}{2\lambda} = \mu. \end{aligned} \quad (2.58)$$

A more detailed version of this proof is given in [8].

Q.E.D.

The capacity of MIMO wireless channels has been studied in a number of papers [6, 24, 14] and analyzed as a function of the transmission environment [26, 9, 12]. Under favorable conditions it is possible that this channel increases the Shannon capacity by a factor of n .

Figure 2.17 shows cumulative probability distribution of the channel capacity for indoor measurements [11] for a 4×4 antenna system using quarter-length whip antennas over a ground plane. The measurements are compared to the theoretical capacity computed above assuming independent Rayleigh channels between antenna pairs. While a large number of factors impact the capacity, the measurements show that antenna spacing, among other aspects, has a strong effect. The capacity of this indoor channel in the 900-MHz ISM band is about 3.5 times that of a single antenna channel, verifying the claims made by theory. The results in Figure 2.17 were with an FPGA-based MIMO testbed using orthogonal Walsh sequences to measure channel gains. The measurements were carried out in an indoor office building environment [12, 11].

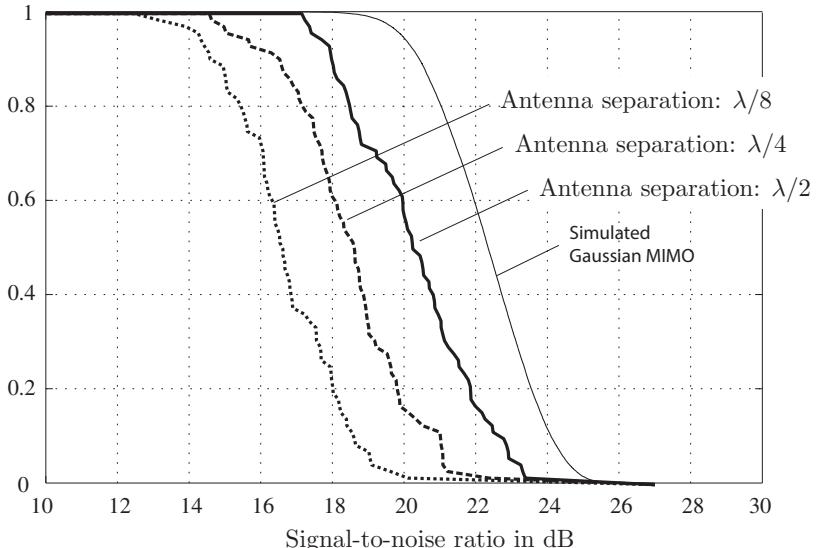


Figure 2.17: Measured verification of the capacity gain of MIMO systems for a 4×4 antenna array system using quarter-length whip antennas.

Communications over MIMO channels typically takes two approaches, space-time coding [1, 22, 23, 13] or space-time layering [6, 7]. In space-time coding the transmitted signal is coded over space (antennas) and time to achieve maximal diversity gain. In space-time

layering the signal is separated into decomposed channels by linear and/or non-linear processing techniques. We will discuss applications of turbo coding principle to space-time coding for MIMO channels in Chapter 8.

2.9 A Communications System Case Study

While we generally focus our discussion on error control coding only in this book, it is important to realize that in the vast majority of applications, error control is only one important component of a complete system. Many other pieces have to fit together to enable a physical system to transmit digital data from a transmitter to a receiver. In this sense, the abstraction of a communications system in Figure 2.1 is just that, a maximally reduced model used for the purpose of understanding fundamental concepts. In reality, a communications system will look more like the one shown in Figure 2.18.

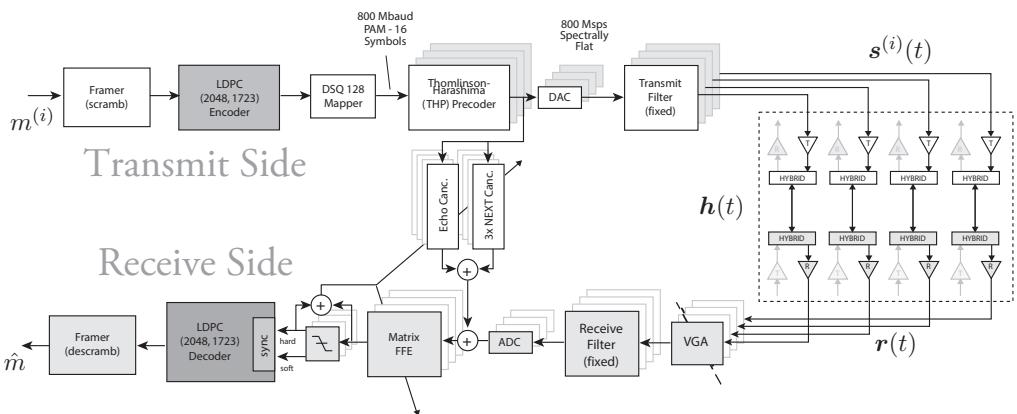


Figure 2.18: Block diagram of the IEEE 802.3an transceiver system for communications over 4 twisted copper cable pairs, used primarily for cabling for computer systems and data centers.

The dark shaded blocks are the encoder and decoder systems, here a low-density parity-check code discussed in detail in Chapter 6. Among the other systems we find basic memory and latches which order bits into transmission frames and insert control signals, the digital mapper function, precoding used to counter channel intersymbol interference distortion, various cancellation filters, and the analog signal modulation components.

We start with the signal constellation. In the IEEE 802.3an standard, a special constellation is used, called the dithered square constellation (DSQ). It is in essence a basic 2-dimensional constellation as in Figure 2.10, where a small amount of signal space coding is introduced by using only every second constellation point, i.e., only using points with an even coordinate sum. Technically, the constellation is shaped from the rotated Z_2 lattice discussed in detail in Chapter 3.

The constellation is shown in Figure 2.19, which shows the 128-point constellation used in the standard. The bit assignment follows a set partitioned approach as we will discuss in detail in Chapter 3, and it is arranged into coded and uncoded bits analogous to Figure 3.16, following a lattice partition approach also discussed in Chapter 3. Here we simply wish to present the signal constellation as an applied example of a high-density signal set.

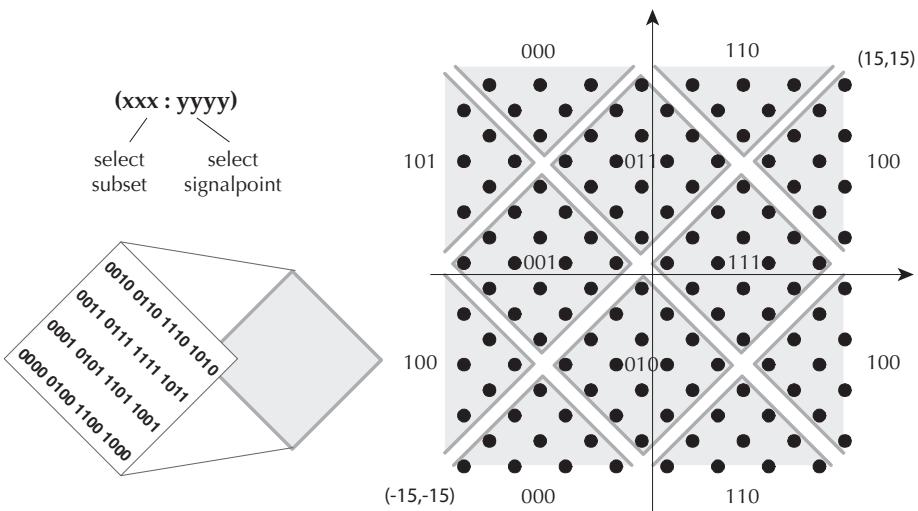


Figure 2.19: 128 DSQ signal set used in the IEEE 802.3an cable modem standard. The four signal point selection bits are encoded with the LDPC code, while the subset selection bits are uncoded, as in Figure 3.16.

The next major processing block in the system is the Tomlinson-Harashima precoder (THP). Before we summarize its functionality, we need to take a look at the physical channel that the twisted copper cables present to the transceiver. In Figure 2.1 we assumed that the only relevant channel distortion was additive noise, actually additive white Gaussian noise as discussed earlier in this chapter. In most practical situations, the channel first

has to be conditioned into a state where the model in Figure 2.1 applies. With channels with an uneven frequency response this requires some form of channel equalization.

Let us start by looking at the frequency response of 100 m of category 6 (Cat 6) Ethernet cabling, shown in Figure 2.20. As is typical with wireline channels, the frequency response drops off exponentially fast with frequency, and higher frequencies require more effort to be used. A similar situation presents itself for DSL and voiceband modems.

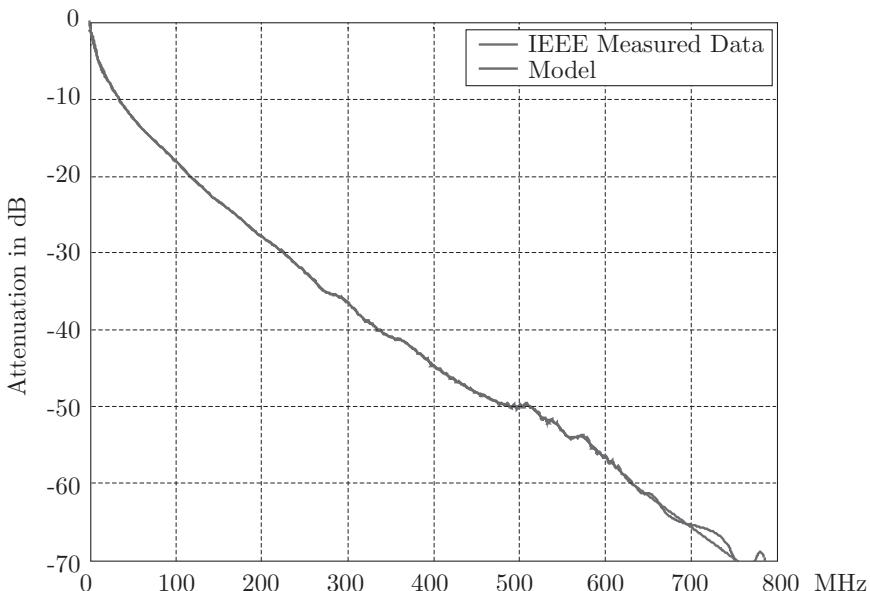


Figure 2.20: Frequency response $H(f)$ of 100 m of category 6 Ethernet cabling.

The channel, of course, is not used directly as analog transmission medium, but via a discrete sampled channel as discussed in Figure 2.7, where, under ideal conditions, a signal sample (or constellation point) a_r appears at the receiver distorted only by the addition of Gaussian noise. This is true only for frequency-flat transmission channels, and a discrete impulse over the channel in Figure 2.20 with a sampling rate of 800 Msamples/s will appear as the sample sequence shown in Figure 2.21.

The capacity of such an *intersymbol interference* channel can be computed as an extension of the capacity formula for the white Gaussian noise channel, which we introduced in (1.1). If the spectrum of the noise is not flat, we need to compute the capacity of the

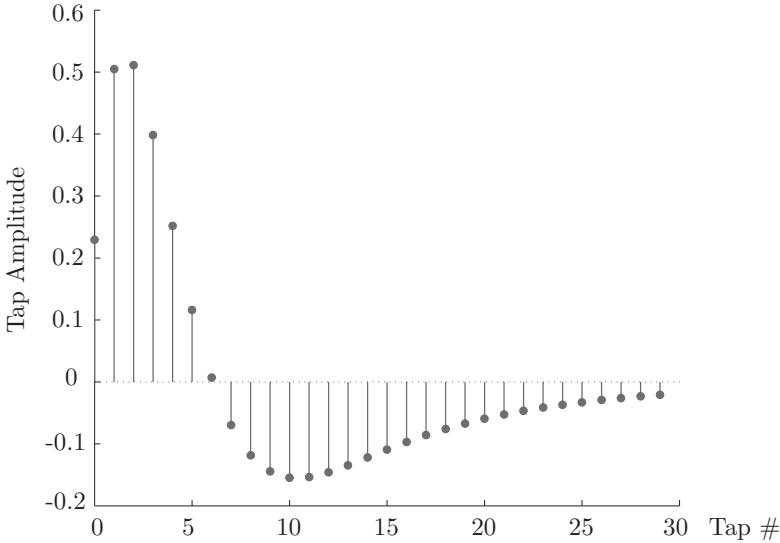


Figure 2.21: Discrete impulse response of the channel in Figure 2.20 received with a low-pass filter with a 3-dB bandwidth of 400 MHz, and sampled at 800 Msamples/s.

channel by adding over small frequency intervals, arriving at the integral equation

$$C = \int_0^W \log \left(1 + \frac{S(f)|H(f)|^2}{N(f)} \right) df \quad (2.59)$$

where $S(f)$ is the signal power spectral density, and $N(f)$ is the noise power spectral density, and $H(f)$ is the channel frequency response. If we let $N(f) = \sigma^2$ and $S(f) = S/W$, it is easy to see that the original Shannon capacity is obtained again.

In order to evaluate (2.59) we need to optimize over the different signal power distributions which have the same energy, i.e., those with

$$S = \int_0^W S(f) df$$

constant. This is accomplished by the so-called *water-filling* theorem, that is, the transmit power is distributed such that frequencies with a larger noise get less power allocated in a process identical to the one discussed in Section 2.8.2. We obtain

$$S(f) = \begin{cases} E - \frac{N(f)}{|H(f)|^2} & \text{if } S(f) \leq 0, \\ 0 & \text{otherwise.} \end{cases} \quad (2.60)$$

However, in practice the transmit power spectral density $S(f)$ is often approximately flat. This leads to a small, but insignificant, loss at the signal-to-noise power ratios of interest. Figure 2.22 below illustrates the per sample capacities of the IEEE 802.3an cable channel with frequency response from Figure 2.20. The following two cases are shown: (i) Formula (2.59) and (ii) the capacity of the channel which results from THP precoding, which suffers a loss of 13.26 dB w.r.t. the ideal flat channel with the same overall power S . Spectrally modulating the transmit power only has an effect at significantly lower SNR than those of the IEEE 802.3an operating point.

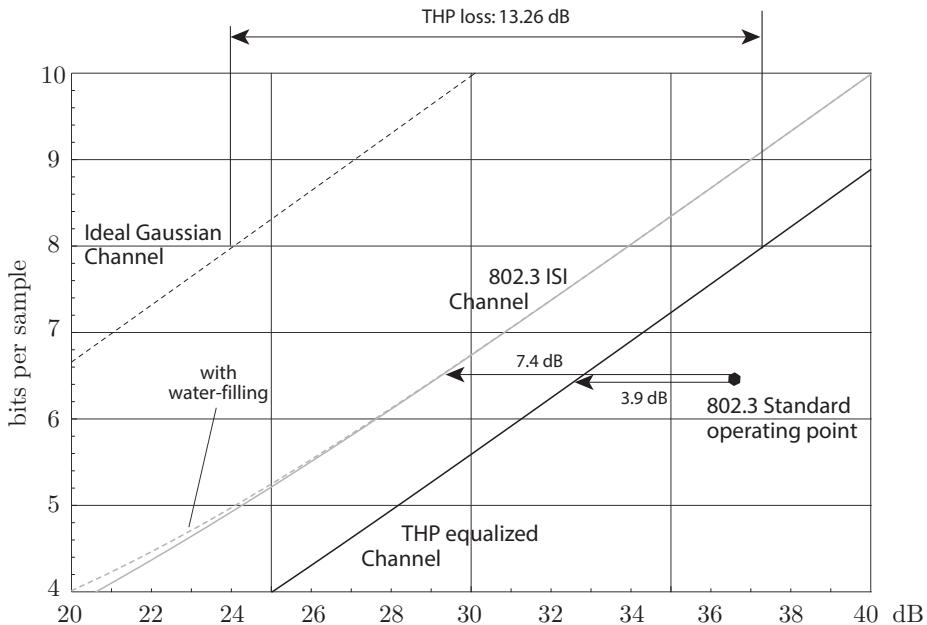


Figure 2.22: Various information theoretical capacities for the cable modem channel of Figure 2.20 as a function of the SNR S/N .

Also shown is the IEEE 802.3 Standard operating point at 23 dB slicer SNR (corresponding to a symbol-to-noise SNR of 37 dB). There is a gap of about 3.9 dB to the THP capacity, as well as a 7.4 dB gap to the ISI channel capacity. About 2 dB of that is due to the code loss, a combination of small size and the high rate. However, what is also apparent is that investing more complex signal processing than THP is probably not worth the effort given the small additional possible gain.

The principle of THP is quite simple and is illustrated in Figure 2.23. The main idea is to subtract the echoes that the channel is introducing via its impulse response (Figure 2.21) from the transmitted signal, such that the resulting signal at the receiver is free of interference.

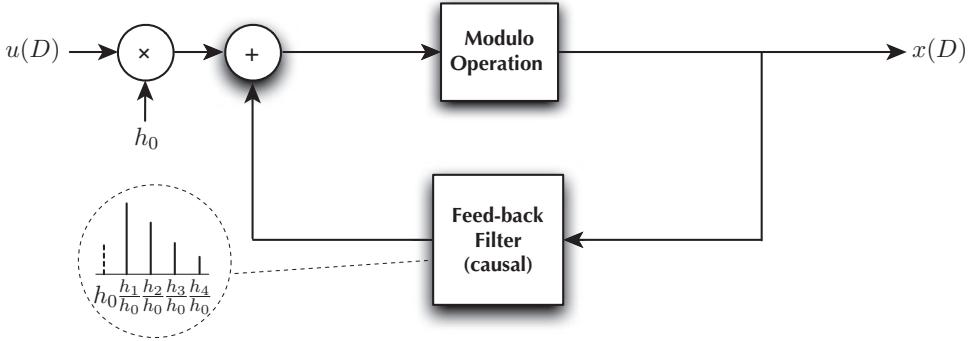


Figure 2.23: Principle of Tomlinson-Harashima precoding.

More precisely, if we denote the impulse response by $h(D) = h_0 + h_1D + \dots + h_LD^L$, where D is the discrete delay operator, then the THP transmitter first forms

$$x(D) = \frac{u(D)}{h(D)}, \quad (2.61)$$

where $u(D)$ is the sequence of input symbols u_i , which are drawn from the discrete constellation, here the one in Figure 2.19. The inverse above is realized by the feedback circuit in Figure 2.23. If the signal $x(D)$ is now transmitted through a channel with impulse response $h(D)$, the output of the channel is given by $y(D) = x(D) + n(D)$, i.e., distortion free.

The problem is that inverse filtering can generate possibly large amplitudes and the filter can become unstable. This is undesirable from circuit point of view. Consequently, Tomlinson [25] introduced a modulo reduction operation, where the amplitude of $x(D)$ is folded back into the original range of the signal constellation. The effect of this operation at the receiver is that the received signal is given by

$$y_i = x_i + n_i + k_i A, \quad (2.62)$$

where k_i is an integer and A is the range of the modulation, in our case $A = 30$ from Figure 2.19. In principle, THP now operates exactly like ideal decision-feedback equalization and

we obtain a channel where additive noise is the only distortion. THP was introduced by M. Tomlinson in 1971 [25].

As a final point, we want to mention that the Ethernet cabling, carrying 4 twisted strands of copper wire, also causes another kind of interference which needs to be controlled. These types of interference are the near-end crosstalk (NEXT), the far-end crosstalk (FEXT), and channel echo, caused by transmitted signals being reflected along the wires and traveling backward on the cable reach the receiver unintentionally. These effects are illustrated in Figure 2.24.

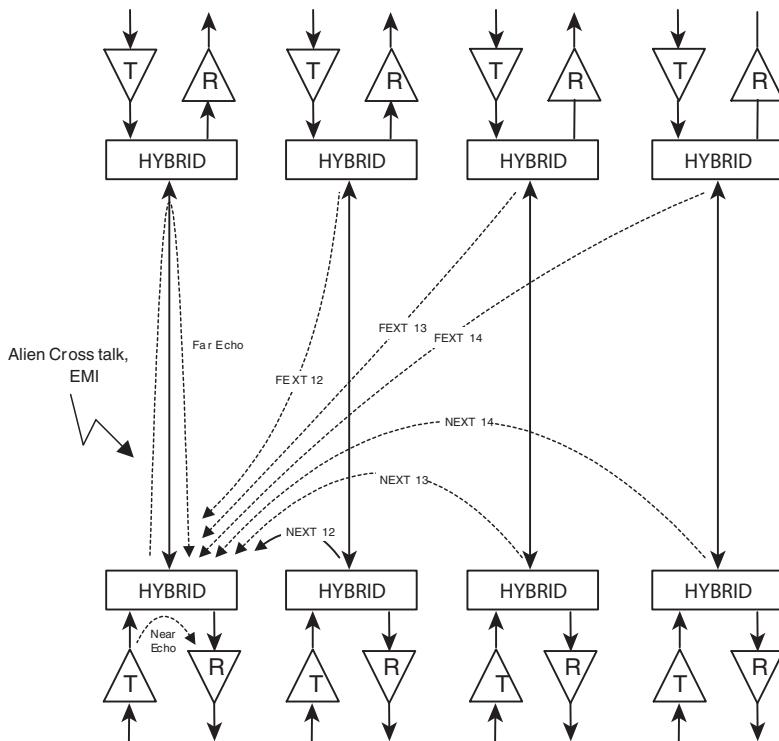


Figure 2.24: Signal distortion effects of the Ethernet cabling system, specifically the near- and far-end crosstalks, and the line echo distortions.

Due to the high frequencies used in the IEEE 802.3an signaling system, NEXT, FEXT, and echo distortions are very significant, and need to be addressed before demodulation and decoding can take place. The relative interference levels of these distortions are

illustrated in Figure 2.25, and compared with the useful signal levels that is received at the end of 100 m of cabling. Comparing with Figure 2.20, we see that the interference levels can be as much as 40 dB above the levels of the target signal.

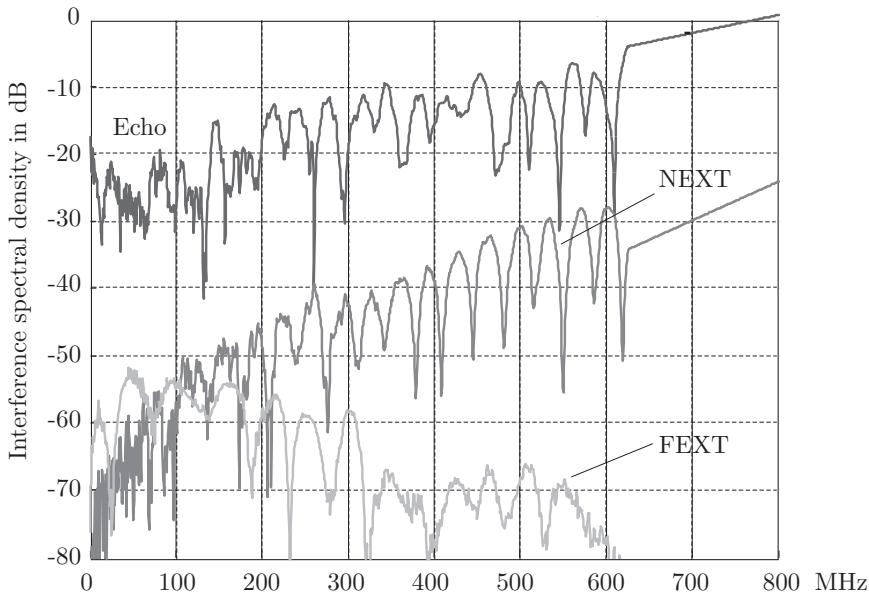


Figure 2.25: Measured interference levels of NEXT, FEXT, and echo compared to useful signal levels, which can be seen in Figure 2.20.

At the filter cutoff frequency at 400 MHz, for example, the echo signal level is 35 dB stronger than the target signal level. The NEXT levels also exceed the signal level, and the FEXT levels are high enough such that their cancellation is necessary in order to operate a high-density signal constellation, such as the DSQ constellation of IEEE 802.3an.

Going into further details is beyond the scope of this text, however, we wish to note that all of these interference sources are tracked at the receiver via signal estimators, usually employing least mean-squares (LMS) adaptive algorithms. The interfering signals are then subtracted from the received signal to isolate the target signal. This can be done since the original source of the interference, the transceiver's own transmitted signal, is available to the receiver side. As seen in Figure 2.18, these signals are fed through adaptive filters to the receiver side, where they are used to cancel the effects of NEXT, FEXT, and the channel echos.

As can be appreciated from Figure 2.24, this is fairly complex task since it requires 10 different cancelation filters per lane, that is, 40 separate cancelation filters, which all need to be trained and adapted continuously to ensure adequate cancelation. The fact that the IEEE 802.3an transceiver does operate to specification is a testimony to the high degree of maturity of such cancelation methods. It is therefore not surprising that nearly 50% of the hardware efforts lies in the various cancelation filters, which, although theoretically quite straightforward, constitute nonetheless a major engineering triumph.

The details of the coding system, bit assignment, and modulation using set partitioning principles will be discussed in detail in Section 3.7.

2.10 Appendix 2.A

The random waveform $n(t)$ which is added to the transmitted signal in the channel of Figure 2.1 can be modeled as a Gaussian random process in many important cases. Let us start by approximating this noise waveform by a sequence of random rectangular pulses $w(t)$ of duration T_s , i.e.,

$$n(t) = \sum_{i=-\infty}^{\infty} n_i w(t - iT_s - \delta), \quad (2.63)$$

where the weighing coefficients n_i are independent Gaussian random variables, T_s is the chosen discrete time increment and δ is some random delay, uniformly distributed in $[0, T_s[$. The random delay δ is needed to make (2.63) stationary. A sample function of $n(t)$ is illustrated in Figure 2.26.

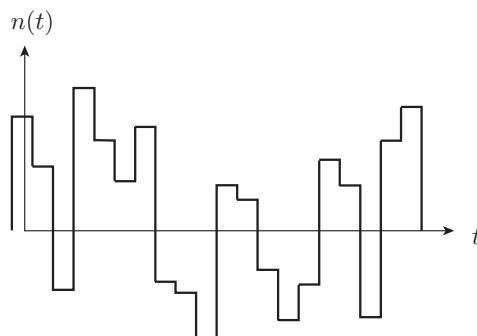


Figure 2.26: Sample function of the discrete approximation to the additive channel noise $n(t)$.

A stationary Gaussian process is completely determined by its mean and correlation function (see, e.g., [5, 19]). It is also straightforward to see that $n(t)$ as defined in (2.63) is stationary. The mean of the noise process is assumed to be zero, and the correlation function of $n(t)$ is then calculated as

$$\begin{aligned}
 R(\tau) &= E[n(t)n(t+\tau)] \\
 &= E\left[\sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} n_i n_j w(t-iT_s - \delta) w(t+\tau - jT_s - \delta)\right] \\
 &= \sigma^2 E_{\delta}\left[\sum_{i=-\infty}^{\infty} w(t-iT_s - \delta) w(t+\tau - iT_s - \delta)\right] \\
 &= \sigma^2 \Delta_{T_s}(\tau),
 \end{aligned} \tag{2.64}$$

where $\sigma^2 = E[n_i^2]$ is the variance of the discrete noise samples n_i and the expectation is now only over the variable delay δ . This expectation is easily evaluated and $\Delta_{T_s}(\tau)$ is a triangular function as shown in Figure 2.27 below.

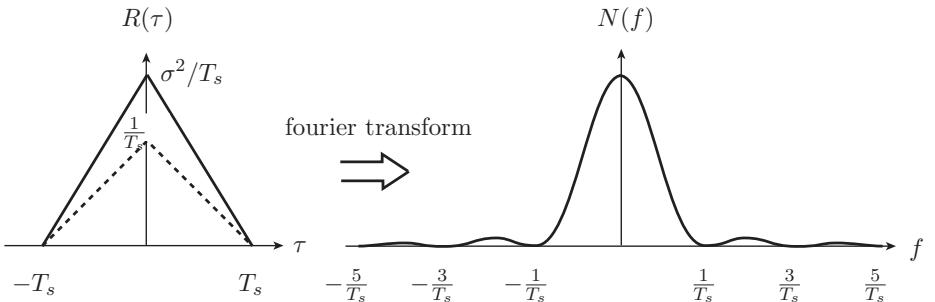


Figure 2.27: Correlation function $R(\tau)$ and power spectral density $N(f)$ of the discrete approximation to the noise function $n(t)$.

We note that the power of the noise function, which is given by $P = R(\tau = 0) = 1/T_s$, increases linearly with the inverse of the sample time T_s . To see why that is a reasonable effect, let us consider the power spectral density $N(f)$ of the random process $n(t)$, given by the Fourier transform of the correlation function $R(\tau)$ as

$$N(f) = \int_{-\infty}^{\infty} R(\tau) e^{-2\pi j \tau f} d\tau, \tag{2.65}$$

where f is the frequency variable. The power density spectrum $N(f)$ is also shown in Figure 2.27. Note that as $T_s \rightarrow 0$ in the limit, $N(f) \rightarrow \sigma^2$. In the limit approximation, (2.63) therefore models noise with an even distribution of power over all frequencies. Such noise is called *white noise* in analogy to the even frequency content of white light, and we will denote it by $n_w(t)$ henceforth. In the literature the constant $\sigma^2 = N_0/2$, and N_0 is known as the one-sided noise power spectral density [28].

Naturally, the model makes no physical sense in the limit, since it would imply infinite noise power. But we will be careful not to use the white noise $n_w(t)$ without some form of low-pass filtering. If $n_w(t)$ is filtered, the high noise frequencies are rejected in the stop-bands of the filter, and for the filter output it is irrelevant how we model the noise in its stop-bands.

As $n(t) \rightarrow n_w(t)$, the correlation function $R(\tau)$ will degenerate into a pulse of width zero and infinite height as $T_s \rightarrow 0$, i.e., $R(\tau) \rightarrow \sigma^2\delta(\tau)$, where $\delta(\tau)$ is known as *Dirac's impulse function*. $\delta(\tau)$ is technically not a function but a distribution. We will only need the *sifting property* of $\delta(t)$, i.e.,

$$\int_{-\infty}^{\infty} \delta(t - \alpha) f(t) dt = f(\alpha), \quad (2.66)$$

where $f(t)$ is an arbitrary function, which is continuous at $t = \alpha$. Property (2.66) is easily proven by using (2.64) and carrying out the limit operation in the integral (2.66). In fact, the relation

$$\int_a^b \delta(\tau - \alpha) f(\tau) d\tau = \lim_{T_s \rightarrow 0} \int_a^b \Delta_{T_s}(\tau - \alpha) f(\tau) d\tau = f(\alpha), \quad (2.67)$$

for any α in the interval (a, b) , can be used as a proper definition of the impulse function. If the limit in (2.67) could be taken inside the integral

$$\delta(\tau - \alpha) = \lim_{T_s \rightarrow 0} \Delta_{T_s}(\tau - \alpha). \quad (2.68)$$

However, any function which is zero everywhere except at one point equals zero when integrated in the Riemann sense, and hence Equation (2.68) is a symbolic equality only, to be understood in the sense of (2.67). An introductory discussion of distributions can be found, for example, in [15], pp. 269–282.

Bibliography

- [1] S. Alamouti, “A simple transmit diversity technique for wireless communications,” *IEEE J. Sel. Areas Commun.*, vol. 16, no. 8, pp. 1451–1458, Oct. 1998.
- [2] E. Biglieri, “Ungerboeck codes do not shape the signal power spectrum,” *IEEE Trans. Inform. Theory*, vol. IT-32, no. 4, pp. 595–596, July 1986.
- [3] R.E. Blahut, *Digital Transmission of Information*, Addison-Wesley, New York, 1990.
- [4] T.M. Cover and J.A. Thomas, *Elements of Information Theory*, John Wiley & Sons, New York, 1991.
- [5] W. Feller, *An Introduction to Probability Theory and Its Applications*, vol. 1 and 2, revised printing of the third edition, John Wiley & Sons, New-York, 1970.
- [6] G.J. Foschini, “Layered space-time architecture for wireless communication in a fading environment when using multi-element antennas,” *Bell Labs Tech. J.*, vol. 1, no. 2, pp. 41–59, Aug. 1996.
- [7] G.J. Foschini and M.J. Gans, “On limits of wireless communication in a fading environment when using multiple antennas,” *Wireless Personal Commun.*, vol. 6, no. 3, pp. 311–355, Mar. 1998.
- [8] R.G. Gallager, *Information Theory and Reliable Communication*, John Wiley & Sons, New York, 1968.
- [9] D. Gesbert, H. Bölcseki, D.A. Gore, and A.J. Paulraj, “Outdoor MIMO wireless channels: Models and performance prediction,” *IEEE Trans. Commun.*, vol. 50, no. 12, Dec. 2002.
- [10] G.H. Golub and C.F. Van Loan, *Matrix Computations*, third edition, John Hopkins University Press, Baltimore, 1996.
- [11] P. Goud, R. Hang, D. Truhachev, and C. Schlegel, “A portable MIMO testbed and selected channel measurements,” *EURASIP J. Applied Sig. Proc.*, Special Issue on Implementation Aspects and Testbeds for MIMO Systems, vol. 2006.
- [12] P. Goud Jr., C. Schlegel, W.A. Krzymien, R. Hang, “Multiple antenna communication systems—an emerging technology,” *Can. J. Electr. Comput. Eng.*, Special Issue Advances in Wireless Commun. and Networking, vol. 29, no. 1/2, Jan./Apr. 2004, pp. 51–59.

- [13] B.L. Hughes, "Differential space-time modulation," *IEEE Trans. Inform. Theory*, vol. 46, no. 7, pp. 2567–2578, Nov. 2000.
- [14] T.L. Marzetta and B.M. Hochwald, "Capacity of a mobile multiple-antenna communication link in Rayleigh flat-fading," *IEEE Trans. Inform. Theory*, vol. 45, no. 1, pp. 139–157, Jan. 1999.
- [15] A. Papoulis, *The Fourier Integral and its Applications*, McGraw-Hill, New York, 1962.
- [16] J.G. Proakis, *Digital Communications*, McGraw-Hill, New York, 5th edition, 2007.
- [17] S. Ramseier and C. Schlegel, "On the bandwidth/power tradeoff of trellis coded modulation schemes," *Proc. IEEE Globecom'93*, (1993).
- [18] S. Ramseier, "Bandwidth-efficient correlative trellis coded modulation schemes," *IEEE Trans. Commun.*, vol. COM-42, no. 2/3/4, pp. 1595–1605, 1994.
- [19] K.S. Shanmugan and A.M. Breipohl, *Random Signals: Detection, Estimation and Data Analysis*, John Wiley & Sons, New York, 1988.
- [20] C.E. Shannon, "A mathematical theory of communications," *Bell Syst. Tech. J.*, vol. 27, pp. 379-423, July 1948.
- [21] B. Sklar, *Digital Communications: Fundamentals and Applications*, Prentice Hall, Englewood Cliffs, NJ, 1988.
- [22] V. Takokh, N. Seshadri, and A.R. Calderbank, "Space-time codes for high data rate wireless communication: performance criterion and code construction," *IEEE Trans. Inform. Theory*, vol. 44, no. 2, pp. 744–765, March 1998.
- [23] V. Takokh, H. Jafarkhani, and A.R. Calderbank, "Space-time block codes from orthogonal designs," *IEEE Trans. Inform. Theory*, vol. 45, no. 4, pp. 1121–1128, May 1999.
- [24] E. Telatar, "Capacity of multi-antenna Gaussian channels," *Eur. Trans. Telecommun.*, vol. 10, no. 6, Nov.–Dec. 1999.
- [25] M. Tomlinson, "New automatic equaliser employing modulo arithmetic," *Electronics Lett.*, vol. 7, no. 5/6, March 1971.
- [26] A.L. Swindlehurst, G. German, J. Wallace, and M. Jensen, "Experimental measurements of capacity for MIMO indoor wireless channels," *Proc. Third IEEE Signal Processing Workshop*, Taoyuan, Taiwan, March 20–25, 2001.
- [27] A.J. Viterbi, "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm", *IEEE Trans. Inform. Theory*, vol. IT-13, pp. 260-269, 1967.
- [28] J.M. Wozencraft and I.M. Jacobs, *Principles of Communication Engineering*, John Wiley & Sons, New York, 1965, reprinted by Waveland Press, 1993.
- [29] R.L. Peterson, R.E. Ziemer, and D.E. Borth, *Introduction to Spread-Spectrum Communications*, Prentice Hall, Englewood Cliffs, NJ, 1995.
- [30] 3GPP Group Website, Releases, <http://www.3gpp.org/releases>
- [31] 3GPP2 Group Website, <http://www.3gpp2.org>

Chapter 3

Trellis-Coded Modulation

3.1 An Introductory Example

As we have seen in Chapter 1, power and bandwidth are limited resources in modern communications systems and their efficient exploitation fundamentally involves an increase in the complexity of a communication system. While there are strict limits on the power and bandwidth resources, the complexity of systems has steadily increased to obtain efficiencies ever closer to the these limits. One very successful method of reducing the power requirements without increase in the requirements on bandwidth was introduced by Gottfried Ungeröbeck [30, 31, 33, 34], subsequently termed *Trellis-Coded Modulation* (TCM). We start this chapter with an illustrative example of this novel method of combining coding and modulation.

Let us assume that we are using a standard QPSK modulation scheme which allows us to transmit two information bits/symbol. We know from Chapter 2 that the most likely error a decoder makes due to noise is to confuse two neighboring signals, say signal $\mathbf{s}^{(2)}$ and signal $\mathbf{s}^{(1)}$ (compare Figure 2.2). This will happen with probability

$$P_{\mathbf{s}^{(1)} \rightarrow \mathbf{s}^{(2)}} = Q\left(\sqrt{\frac{d^2 E_s}{2N_0}}\right), \quad (3.1)$$

where $d^2 = 2$ for a unit energy signal constellation, and E_s is the average transmit energy per symbol. Instead of using such a system to transmit one symbol at a time, the encoder shown in Figure 3.1 is used. This encoder consists of two parts, the first of which is a *Finite-State Machine* (FSM) with a total of eight states, where state \mathbf{s}_r at time r is defined by the contents of the delay cells, or shift-registers, i.e., $\mathbf{s}_r = (s_r^{(2)}, s_r^{(1)}, s_r^{(0)})$. The second part is called a *signal mapper* and its function is a memoryless mapping of the three bits $\mathbf{v}_r = (u_r^{(2)}, u_r^{(1)}, v_r^{(0)})$ into one of the eight symbols of an 8-PSK signal set. The FSM

accepts two input bits $\mathbf{u}_r = (u_r^{(2)}, u_r^{(1)})$ at each symbol time r , and it transitions from a state \mathbf{s}_r to one of 4 possible successor states \mathbf{s}_{r+1} . In this fashion the encoder generates a sequence of output symbols $\mathbf{x} = (x_0, x_1, x_2, \dots)$. Assuming that the encoder is operated in a continuous fashion, there are four choices at each time r , which allows us to transmit 2 information bits/symbol, the same as with QPSK.

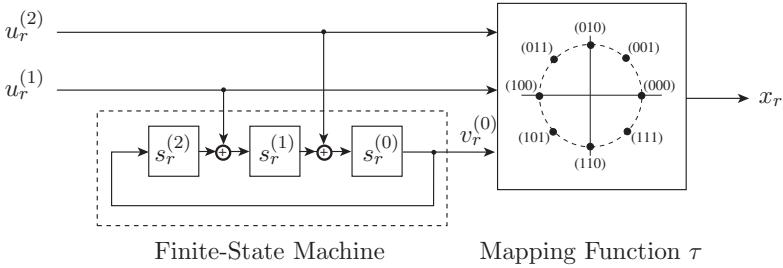


Figure 3.1: Trellis encoder with an 8-state finite-state machine (FSM) driving a 3-bit to 8-PSK signal mapper.

A graphical interpretation of the operation of this FSM, and in fact the entire encoder, will prove immensely useful. Since the FSM is time-invariant, it can be represented by a state transition diagram as shown in Figure 3.2. The nodes in this transition diagram are the states of the FSM, and the branches represent the possible transitions between states. Each branch can now be labeled by the pair of input bits $\mathbf{u} = (u^{(2)}, u^{(1)})$ which cause the transition, as well as by either the output triple $\mathbf{v} = (u^{(2)}, u^{(1)}, v^{(0)})$ or the output signal $x(\mathbf{v})$. (In Figure 3.2 we have used $x(\mathbf{v})$, represented in decimal notation, i.e., $x_{\text{dec}}(\mathbf{v}) = u^{(2)}2^2 + u^{(1)}2^1 + v^{(0)}2^0$.)

If we index the state transition diagram by both the states and the time index r , Figure 3.2 expands into the *trellis diagram*, or simply the *trellis* of Figure 3.3. It is the two-dimensional representation of the operation of the encoder, capturing all possible state transitions starting from an originating state (usually state 0), and terminating in a final state (usually also state 0). The trellis in Figure 3.3 is terminated to length $L = 5$. This requires that the FSM be driven back into state 0 at time $r = 5$. As a consequence, the branches at times $r = 3$ and 4 are predetermined, and no information is transmitted in those two time units. Contrary to the short trellis in Figure 3.3, in practice, the length of the trellis will be several hundred or thousands of time units, possibly even indeterminate, corresponding to continuous operation. When and where to terminate the trellis is a matter of practical considerations, related to the block and frame sizes used.

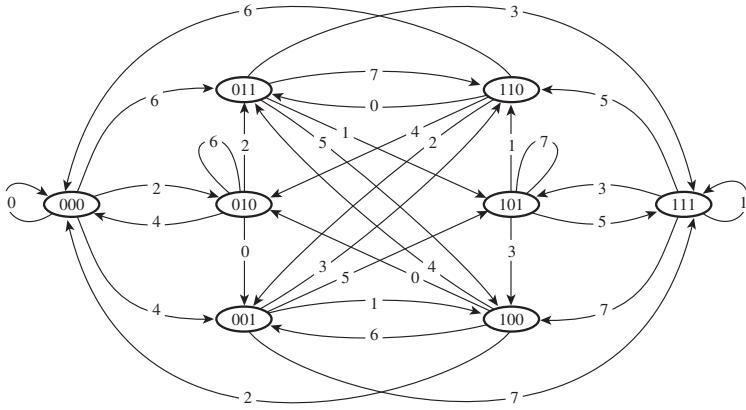


Figure 3.2: State transition diagram of the encoder from Figure 3.1. The labels on the branches are the encoder output signals $x(\mathbf{v})$, in decimal notation.

Now each path through the trellis corresponds to a unique message and is associated with a unique message sequence (compare Section 2.5). The term *trellis-coded modulation* originates from the fact that these encoded sequences consist of modulated symbols which are associated with the branches of the trellis. Colloquially we say the symbols are attached to the branch.

At first sight it is not clear at all what has been gained by this complicated encoding since the signals in the 8-PSK signal sets are much closer together than those in the QPSK signal set and have a higher symbol error rate. The FSM, however, puts restrictions on the symbols which can form any given valid sequence, and these restrictions can be exploited by a smart decoder. In fact, what counts is the distance between signal sequences \mathbf{x} , and not the distance between individual signals. Let us then assume that such a decoder can follow all possible sequences through the trellis, and it makes decisions between sequences. This is illustrated in Figure 3.4 for two sequences $\mathbf{x}^{(e)}$ (erroneous) and $\mathbf{x}^{(c)}$ (correct). These two sequences differ in the three symbols shown. An optimal decoder will make an error between these two sequences with probability $P_s = Q\left(\sqrt{d_{ec}^2 E_s / 2N_0}\right)$, where $d_{ec}^2 = 4.586 = 2 + 0.56 + 2$ is the Euclidean distance between $\mathbf{x}^{(e)}$ and $\mathbf{x}^{(c)}$, which, incidentally, is much larger than the QPSK distance of $d^2 = 2$. Going through all possible sequence pairs $\mathbf{x}^{(e)}$ and $\mathbf{x}^{(c)}$, one finds that those highlighted in Figure 3.4 have the smallest

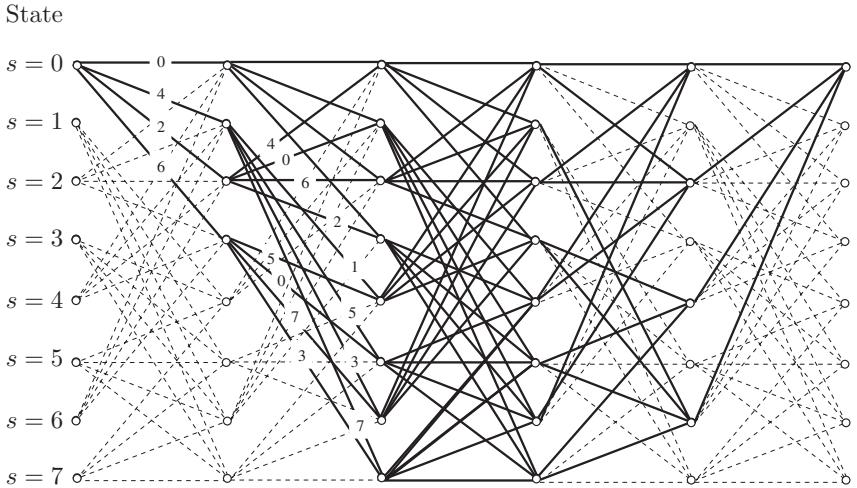


Figure 3.3: Trellis diagram for the encoder in Figure 3.1. The code and the trellis are terminated to length $L = 5$.

squared Euclidean distance, and hence the probability that the decoder makes an error between those two sequences is the most likely error.

We now see that by virtue of carrying out sequence decoding, rather than symbol decoding, the distances between alternatives can be increased, even though the signal constellation used for sequence coding has a smaller minimum distance between signal points. For this code we may decrease the symbol power by about 3.6 dB, i.e., we use less than half the power needed for QPSK.

A more precise error analysis is not quite so simple since the possible error paths in the trellis are highly correlated, which makes an exact analysis of the error probability impossible for all but the most simple cases. In fact, a lot of work has gone into analyzing the error behavior of trellis codes and we devote a large portion of Chapter 5 to this topic. This simple example should convince us, however, that some real coding gain can be achieved by this method with a relatively manageable effort. Having said that, we must stress that the lion share of the work is performed by the decoder, about which we will have to say more later in Chapter 5.

Figure 3.5 shows an early application of trellis coded modulation to improve the power efficiency on an Intelsat data link, which used to operate uncoded QPSK signaling. The performance of a 16-state 8-PSK TCM code used on a single channel per carrier (SCPC) modem operating at 64 kbits/second [32] outperforms QPSK by over 5 dB. This early

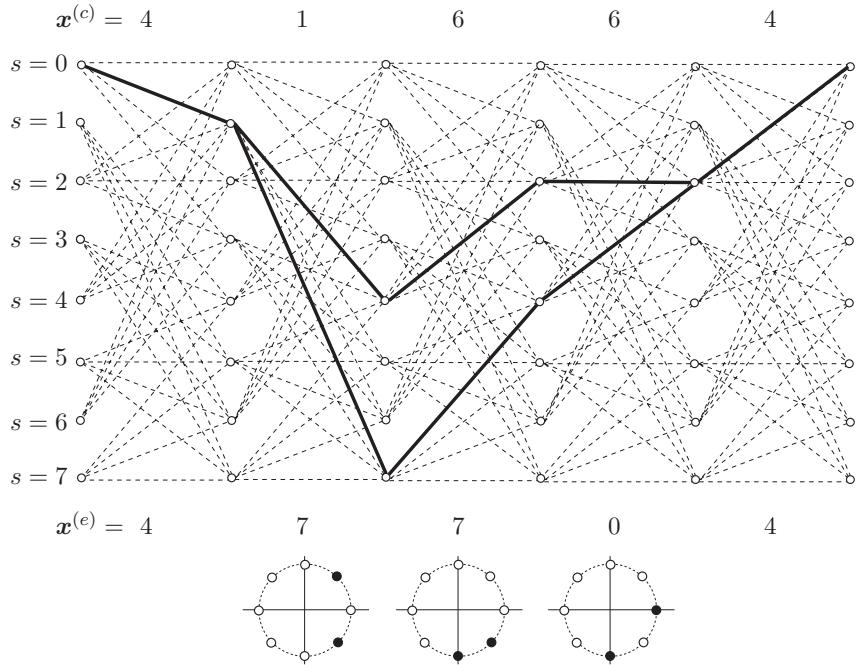


Figure 3.4: Section of the trellis of the encoder in Figure 3.1. The two solid lines depict two possible paths with their associated signal sequences through this trellis. The numbers on top are the signals transmitted if the encoder follows the upper path, and the numbers at the bottom are those on the lower path.

experiment proved the viability of trellis coding for satellite channels. Interestingly, the 8-PSK TCM modem performance comes much closer to its theoretical performance limit than does the original QPSK modem. This is due to the fact that system inaccuracies, acting similar to noise, are handled much better by a system incorporating error control coding.

3.2 Construction of Codes

To build a trellis code, we need an FSM to generate the trellis and a mapping of branches into symbols to be transmitted. This is customarily done by a structure like the one in

Bit Error Probability (BER)

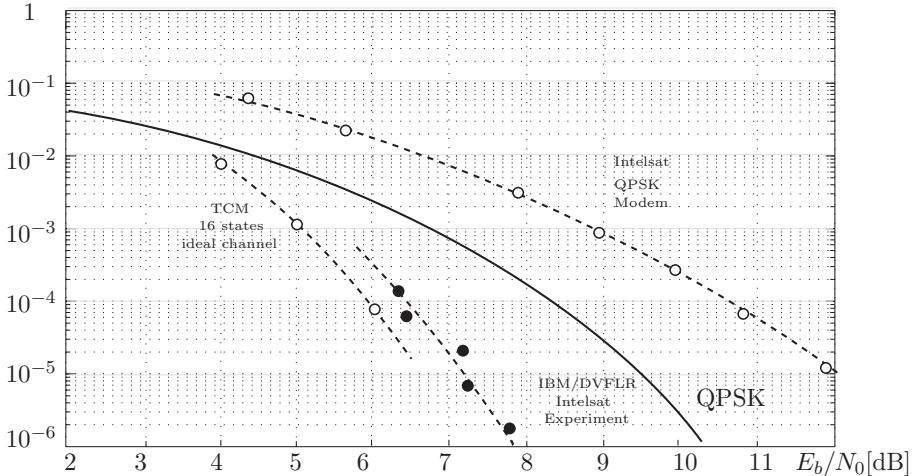


Figure 3.5: Measured bit error probability of QPSK and a 16-state 8-PSK TCM modem over a 64 kbit/s satellite channel [32].

Figure 3.6. The FSM part is identical to a traditional *convolutional code*,¹ which has 2^ν states and two k bits/symbol. The mapping of trellis branches to symbols is accomplished by the mapping function, which is an arbitrary memoryless function $x_r = \tau(\mathbf{u}_r, \mathbf{s}_r)$ of the current state of the FSM and the new input digits $\mathbf{u}_r = (u_r^{(k)}, \dots, u_r^{(1)})$, but typically only the output bits $(\mathbf{u}_r, \mathbf{s}_r) = \mathbf{v}_r = (v_r^{(n)}, v_r^{(n-1)}, \dots, v_r^{(0)})$ participate the mapping function $x_r = \tau(\mathbf{v}_r)$.

The second component of the encoder is the mapping function $x_r = \tau(\mathbf{u}_r, \mathbf{s}_r)$, which assigns a signal x_r to each transition of the FSM. The mapping function of the trellis code discussed in the beginning of this chapter maps three output bits from the FSM into one 8-PSK signal point as shown in Figure 3.7.

The important quantities of a signal set are the squared Euclidean distances between pairs of signal points, since they determine the squared Euclidean distances between sequences, which are the important parameters of a trellis code.

A signal on a branch in the trellis $x_i = \tau(\mathbf{v}_i)$ is generated by the mapping function τ ,

¹More precisely, the generating FSM is a convolutional encoder in the systematic feedback form (compare Chapter 4).

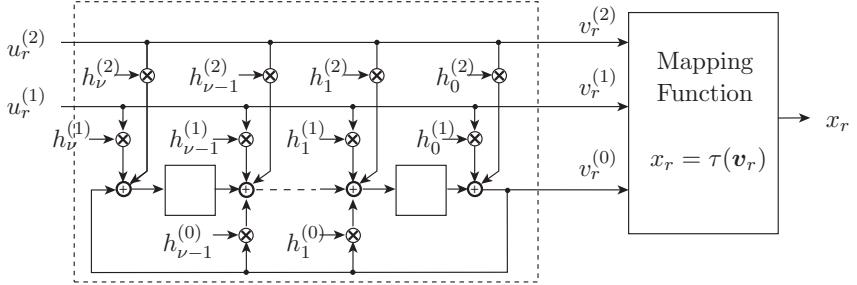


Figure 3.6: Structure of a generic trellis encoder.

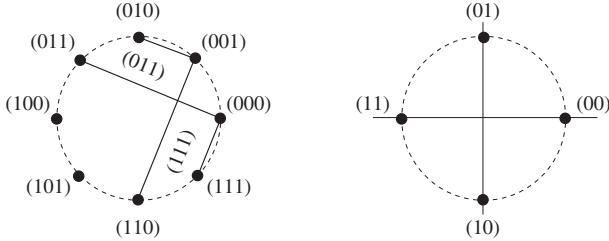


Figure 3.7: 8-PSK and QPSK signal sets with natural and Gray mapping.

given as input the branch label \mathbf{v}_i . In general, the distance between two branch signals $d^2 = \|x_1 - x_2\|^2$ is related to the branch labels in a complicated manner. However, if the squared Euclidean distance between x_1 and x_2 , given by

$$d^2 = \|\tau(\mathbf{v}_1) - \tau(\mathbf{v}_2)\|^2, \quad (3.2)$$

depends only on the difference between their branch labels, i.e.,

$$\begin{aligned} d^2 &= \|\tau(\mathbf{v}_1) - \tau(\mathbf{v}_2)\|^2 \\ &= \left\| \underbrace{\tau(\mathbf{v}_1 \oplus (-\mathbf{v}_2))}_{\mathbf{e}_v} - \tau(0, 0) \right\|^2 \\ &= \|\tau(\mathbf{e}_v) - \tau(0, 0)\|^2, \end{aligned} \quad (3.3)$$

where the binary operator \oplus denotes the bit-wise EXOR operation, then the signal $\tau(0, 0)$ can always be used as a reference signal. Signal mappings with this property are called

regular. An example of a regular mapping is the QPSK signal set in Figure 3.7 used with Gray mapping. It is easy to see that a binary difference of (01) and (10) will always produce a $d^2 = 2$. The binary difference of (11) produces $d^2 = 4$. The 8-PSK signal set shown in Figure 3.7 uses a mapping known as *natural mapping*, where the signal points are numbered sequentially in a counter clockwise fashion. This mapping is not regular, since the binary difference $v = (011)$ can produce the two distances $d^2 = 0.56$ and $d^2 = 3.14$. It can be shown that no regular mapping exists for an 8-PSK signal set. However, clever local averaging techniques can be used to arrive at error expression which still allow the use of the all-zero sequence as reference sequence, see [27, Chapter 3].

Now, from Figure 3.4 we see that an error path diverges from the correct path at some state and merges with the correct path again at a (possibly) different state. The task of designing a good trellis code crystallizes into designing a trellis code for which different symbol sequences are separated by large squared Euclidean distances. Of particular importance is the minimum squared Euclidean, defined as $d_{\text{free}}^2 = \min_{\mathbf{x}^{(i)}, \mathbf{x}^{(j)}} \|\mathbf{x}^{(i)} - \mathbf{x}^{(j)}\|^2$ and also called the *free distance* of the code. A code with a large d_{free}^2 is generally expected to perform well, and d_{free}^2 has become the major design criterion for trellis codes. (The free distance for the 8-PSK trellis code of Section 3.1 is $d_{\text{free}}^2 = 4.586$.)

One heuristic design rule [31], which was used successfully in designing codes with large d_{free}^2 , is based on the following observation: If we assign to the branches leaving a state signals from a subset where the distances between points are large, and likewise assign such signals to the branches merging into a state, we are assured that the total distance is at least the sum of the minimum distances between the signals in these subsets. For our 8-PSK code example we can choose these subsets to be QPSK signal subsets of the original 8-PSK signal set. This is done by partitioning the 8-PSK signal set into two QPSK sets as illustrated in Figure 3.8 below. The mapping function is now chosen such that the state information bit $v^{(0)}$ selects the subset and the input bits u select a signal within the subset. Since all branches leaving a state have the same state information bit $v^{(0)}$, all the branch signals are either in subset A or subset B , and the difference between two signal sequences picks up an incremental distance of $d^2 = 2$ over the first branch where they diverge. In order to achieve this, we need to make sure that the choice of u does not affect $v^{(0)}$, which is done by setting $h_0^{(2)} = 0$ and $h_0^{(1)} = 0$ in Figure 3.6.

To guarantee that the signal on branches merging into a state can also be chosen from one of these two subsets, we set $h_\nu^{(2)} = 0$ and $h_\nu^{(1)} = 0$. This again has the effect that merging branches have the same value of the state information bit $v^{(0)}$. These are Ungerböck's [31] original design rules.

We have now assured that the minimum distance between any two paths is at least twice that of the original QPSK signal set. The values of the remaining connector coefficients $h^{(2)} = h_1^{(2)}, \dots, h_{\nu-1}^{(2)}$, $h^{(1)} = h_1^{(1)}, \dots, h_{\nu-1}^{(1)}$ and $h^{(0)} = h_1^{(0)}, \dots, h_{\nu-1}^{(0)}$ are much harder to find, and one usually resorts to computer search programs or heuristic strategies.

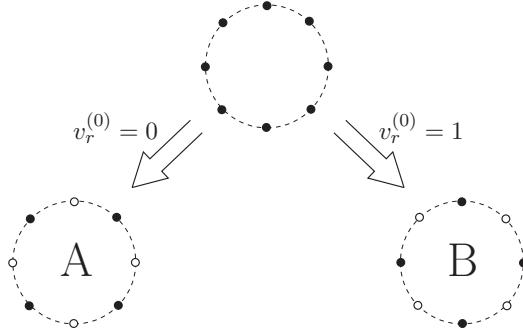


Figure 3.8: 8-PSK signal set partitioned into constituent QPSK signal sets.

Table 3.1 shows the best 8-PSK trellis codes found using 8-PSK with natural mapping. The figure gives the connector coefficients d_{free}^2 , the average path multiplicity $A_{d_{\text{free}}}$ of d_{free}^2 , and the average bit multiplicity $B_{d_{\text{free}}}$ of d_{free}^2 . $A_{d_{\text{free}}}$ is the average number of paths at distance d_{free}^2 , and $B_{d_{\text{free}}}$ is the average number of bit errors on those paths. Both, $A_{d_{\text{free}}}$ and $B_{d_{\text{free}}}$, as well as the higher-order multiplicities, are important parameters determining the error performance of a trellis code. This is discussed in detail in Chapter 5. (The connector coefficients are given in octal notation, e.g., $h^{(0)} = 23 = 10111$, where a 1 means connected and a 0 means no connection.)

From Table 3.1 one can see that an asymptotic coding gain (coding gain for $\text{SNR} \rightarrow \infty$ over the reference constellation which is used for uncoded transmission at the same rate) of about 6 dB can quickly be achieved with moderate effort. But, since for optimal decoding the complexity of the decoder grows proportionally to the number of states, it becomes very hard to go much beyond a 6 dB gain, leaving a significant gap to the Shannon bound. It will be up to turbo coded systems to close this gap. Since the asymptotic coding gain is a reasonable yardstick at the bit error rates of interest, codes with a maximum of about one thousand states seem to exploit most of what can be gained by this type of coding.

Some researchers have used different mapping functions in an effort to improve the bit error performance which can be improved by up to 0.5 dB using 8-PSK Gray mapping (label the 8-PSK symbols successively by (000), (001), (011), (010), (110), (111), (101), (100)) as done by Du and Kasahara [6] and Zhang [41, 42]. Zhang also used another mapping to improve the bit multiplicity, using ((000), (001), (010), (011), (110), (111), (100), (101)) as labeling. The search criterion employed involved minimizing the bit multiplicities of

Number of States	$h^{(0)}$	$h^{(1)}$	$h^{(2)}$	d_{free}^2	$A_{d_{\text{free}}}$	$B_{d_{\text{free}}}$	Asymptotic Coding Gain
4	5	2	-	4.00*	1	1	3.0 dB
8	11	2	4	4.59*	2	7	3.6 dB
16	23	4	16	5.17*	2.25	11.5	4.1 dB
32	45	16	34	5.76*	4	22.25	4.6 dB
64	103	30	66	6.34*	5.25	31.125	5.0 dB
128	277	54	122	6.59*	0.5	2.5	5.2 dB
256	435	72	130	7.52*	1.5	12.25	5.8 dB
512	1525	462	360	7.52*	0.313	2.75	5.8 dB
1024	2701	1216	574	8.10*	1.32	10.563	6.1 dB
2048	4041	1212	330	8.34	3.875	21.25	6.2 dB
4096	15201	6306	4112	8.68	1.406	11.758	6.4 dB
8192	20201	12746	304	8.68	0.617	2.711	6.4 dB
32768	143373	70002	47674	9.51	0.25	2.5	6.8 dB
131072	616273	340602	237374	9.85			6.9 dB

Table 3.1: Connectors, free squared Euclidean distance and asymptotic coding gains of some maximum free distance 8-PSK trellis codes. The codes with an * were found by exhaustive computer searches [34, 24], while the other codes were found by various heuristic search and construction methods [24, 25]. The connector polynomials are in octal notation.

several spectral lines in the distance spectrum of a code.² Table 3.2 gives the best 8-PSK codes found with respect to the bit error probability.

If we go to higher-order signal sets such as 16-QAM, 32-cross, 64-QAM, etc., there are, at some point, not enough states left such that each diverging branches lead to different states, and we have parallel transitions, i.e., two or more branches connect two states. Naturally we would want to assign signals with large distances to such parallel branches, since the probability of parallel path errors cannot be influenced by the code.

Parallel transitions actually occur also for the first 8-PSK code in Table 3.1, whose trellis is given in Figure 3.9. Here the parallel transitions are by choice though, not by necessity. Note that the minimum distance path pair through the trellis has $d^2 = 4.56$, but that is not the most likely error to happen. All signals on parallel branches are from a BPSK subset of the original 8-PSK set, and hence their distance is $d^2 = 4$, which gives

²The notion of distance spectrum of a trellis code will be introduced and discussed in Chapter 5.

Number of States	$h^{(0)}$	$h^{(1)}$	$h^{(2)}$	d_{free}^2	$A_{d_{\text{free}}}$	$B_{d_{\text{free}}}$
8	17	2	6	4.59*	2	5
16	27	4	12	5.17*	2.25	7.5
32	43	4	24	5.76*	2.375	7.375
64	147	12	66	6.34*	3.25	14.8755
128	277	54	176	6.59*	0.5	2
256	435	72	142	7.52*	1.5	7.813
512	1377	304	350	7.52*	0.0313	0.25
1024	2077	630	1132	8.10*	0.2813	1.688

Table 3.2: Table of 8-PSK codes using a different mapping function [41].

the 3 dB asymptotic coding gain of the code over QPSK ($d^2 = 2$).

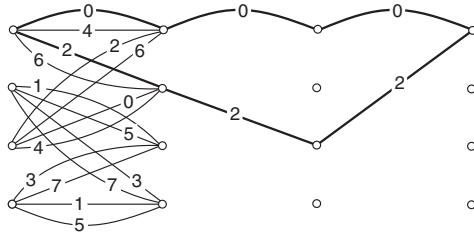


Figure 3.9: 4-state 8-PSK trellis code with parallel transitions.

In general we partition a signal set into a *partition chain* of subsets, such that the minimum distance between signal points in the new subsets is maximized at every level. This is illustrated with the 16-QAM signal set and a binary partition chain (split each set into two subsets at each level) in Figure 3.10.

Note that the partitioning can be continued until there is only one signal left in each subset. In such a way, by following the partition path, a “natural” binary label can be assigned to each signal point. The natural labeling of the 8-PSK signal set in Figure 3.7 (M-PSK in general) can also be generated in this way. This method of partitioning a signal set is called *set partitioning* with increasing intra-subset distances. The idea is to use these constellations for codes with parallel transitions.

An alternative way of generating parallel transitions is to choose not to encode all the input bits in u_r . Using the encoder in Figure 3.11 with a 16-QAM constellation,

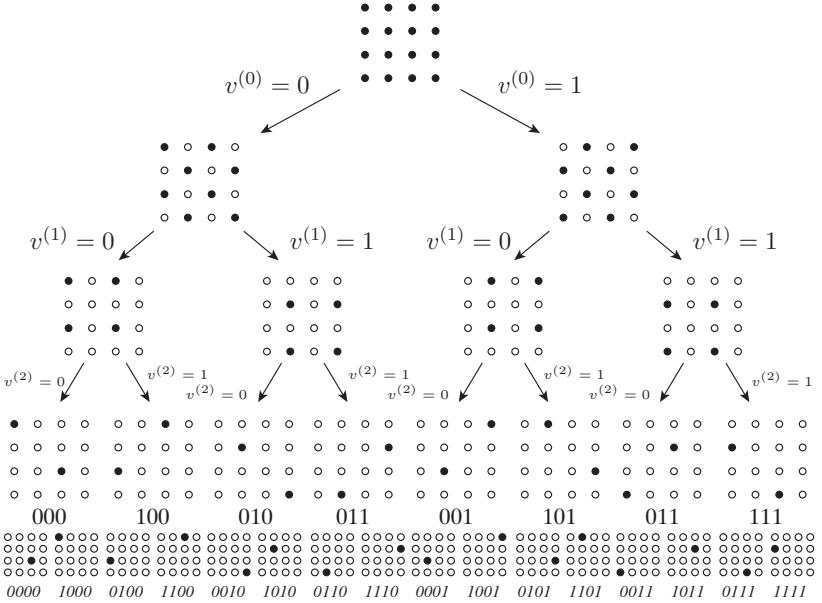


Figure 3.10: Set partitioning of a 16-QAM signal set into subsets with increasing minimum distance. The final partition level used by the encoder in Figure 3.11 is the fourth level, i.e., the subsets with two signal points each.

for example, the first information bit $u_r^{(3)}$ is not encoded and the output signal of the FSM selects now a subset rather than a signal point (here one of the subsets at the fourth partition level in Figure 3.10). The uncoded bit(s) select the actual signal point within the subset. Analogously then, the encoder now has to be designed to maximize the minimum interset distances of sequences, since it cannot influence the signal point selection within the subsets. The advantage of this strategy is that the same encoder can be used for all signal constellations with the same intraset distances at the final partition level, in particular for all signal constellation which are nested versions of each other, such as 16-QAM, 32-cross, 64-QAM, etc.

Figure 3.11 shows such a generic encoder which maximizes the minimum interset distance between sequences, and it can be used with all QAM-based signal constellations. Only the two least significant information bits affect the encoder FSM. All other information bits cause parallel transitions. Table 3.3 shows the coding gains achievable with such

an encoder structure. The gains when going from 8-PSK to 16-QAM are most marked since rectangular constellations have a somewhat better power efficiency than constant energy constellations.

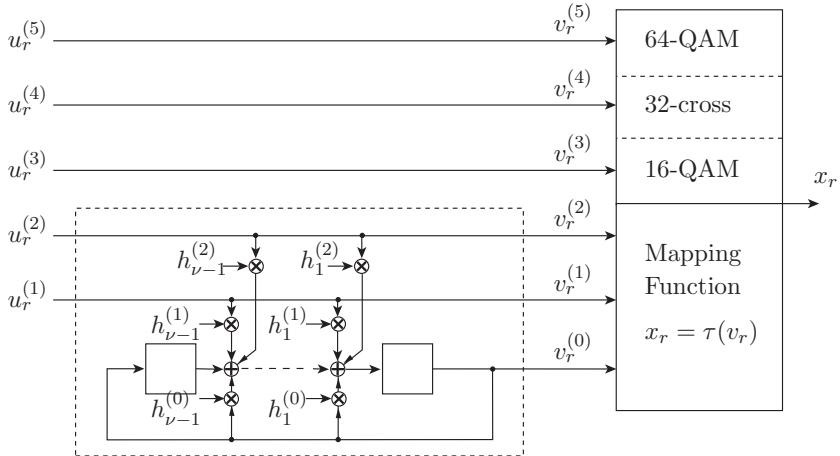


Figure 3.11: Generic encoder for QAM signal constellations.

Number of States	Connectors			d_{free}^2	Asymptotic Coding Gain				
	$h^{(0)}$	$h^{(1)}$	$h^{(2)}$		16- QAM/ 8-PSK	32-cross/ 16QAM	64- QAM/ 32-cross	$A_{d_{\text{free}}}$	$B_{d_{\text{free}}}$
4	5	2	-	4.0	4.4dB	3.0dB	2.8dB		
8	11	2	4	5.0	5.3 dB	4.0 dB	3.8 dB	3.656	18.313
16	23	4	16	6.0	6.1 dB	4.8 dB	4.6 dB	9.156	53.5
32	41	6	10	6.0	6.1 dB	4.8 dB	4.6 dB	2.641	16.063
64	101	16	64	7.0	6.8 dB	5.4 dB	5.2 dB	8.422	55.688
128	203	14	42	8.0	7.4 dB	6.0 dB	5.8 dB	20.328	100.031
256	401	56	304	8.0	7.4 dB	6.0 dB	5.8 dB	3.273	16.391
512	1001	346	510	8.0	7.4 dB	6.0 dB	5.8 dB		

Table 3.3: Connectors and gains of maximum free distance QAM trellis codes [34].

3.3 Lattices

Soon after the introduction of trellis codes and the idea of set partitioning, it was realized that certain signal constellations and their partitioning could be described elegantly by lattices. This formulation is a particularly convenient tool in the discussion of multidimensional trellis codes. We begin by defining a lattice:

Definition 3.1 *An N -dimensional lattice Λ is the set of all points*

$$\Lambda = \{x\} = \{i_1 b_1 + \cdots + i_N b_N\}, \quad (3.4)$$

where x is an m -dimensional row vector (point) in \mathbf{R}^m , b_1, \dots, b_N are N linearly independent basis vectors in \mathbf{R}^m , and i_1, \dots, i_N range through all integers.

A lattice is therefore something similar to a vector space, but the coefficients are restricted to be integers. Lattices have been studied in mathematics for many decades, in particular in addressing issues such as sphere packings, the coverings, or quantization [15, 5]. The sphere packing problem asks the question, “What is the densest way of packing together a large number of equal-sized spheres.” The covering problem asks for the least dense way to cover space with equal overlapping spheres and the quantization problem addresses the problem of placing points in space so that the average second moment of their Voronoi cells is as small as possible. A comprehensive treatise on lattices is given by Conway and Sloane in [5]. We will use only a few results from lattice theory in our study of mapping functions.

Not surprisingly, operations on lattices can conveniently be described by matrix operations. To this end, let

$$\mathbf{M} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_N \end{bmatrix} \quad (3.5)$$

be the *generator matrix* of the lattice. All the lattice points can then be generated by $i\mathbf{M}$, where $i = (i_1, \dots, i_N)$. Operations on lattices can now easily be described by operations on the generator matrix. An example is \mathbf{RM} , where

$$\mathbf{R} = \begin{bmatrix} 1 & -1 & & & & & \\ 1 & 1 & & & & & \\ & & 1 & -1 & & & \\ & & 1 & 1 & & & \\ & & & & \ddots & \ddots & \\ & & & & & 1 & -1 \\ & & & & & 1 & 1 \end{bmatrix} \quad (3.6)$$

is an operation which rotates and expands pairs of coordinates, called the *rotation operator*, and is important for our purpose.

Two parameters of lattices are important, the minimum distance between points d_{\min} and the number of nearest neighbors, lucidly described as the *kissing number* τ of the lattice. This is because if we fill the m -dimensional Euclidean space with m -dimensional spheres of radius $d_{\min}/2$ centered at the lattice points, there are exactly τ spheres which touch (kiss) any given sphere. The density Δ of a lattice is the proportion of the space that is occupied by these touching spheres. Also, the *fundamental volume* $V(\Lambda)$ is the N -dimensional volume per lattice point, i.e., if we partition the total space into regions of equal size $V(\Lambda)$, each such region is associated with one lattice point.

Lattices have long been used with attempts to pack equal-sized spheres in m dimensions, such that as much of the space as possible is covered by spheres. Lattices are being investigated for the problem, since if we find a “locally” dense packing, it is also “globally” dense due to the linearity of the lattice.

A popular lattice is Z^N , the *cubic lattice*, consisting of all N -tuples with integer coordinates. Its minimum distance is $d_{\min} = 1$ and its kissing number is $\tau = 2N$. Trivially, Z^1 is the densest lattice in one dimension.

Another interesting lattice³ is D_N , the *checkerboard lattice*, whose points consist of all points whose integer coordinates sum to an even number. It can be generated from Z^N by casting out that half of all the points which have an odd integer sum. In doing so we have increased d_{\min} to $\sqrt{2}$, and the kissing number to $\tau = 2N(N - 1)$, for $N \geq 2$, and have obtained a much denser lattice packing, which can be seen as follows. Since we need to normalize d_{\min} in order to compare packings, let us shrink D_N by $(1/\sqrt{2})^N$. This puts then $\sqrt{2}^N/2$ as many points as Z^N into the same volume. (The denominator equals 2 since we eliminated half the points.) Therefore D_N is $2^{N/2-1}$ times as dense as Z^N . Note that D_2 has the same density as Z^2 , and it is in fact a rotated version of the latter, i.e., $D_2 = \mathbf{R}Z^2$. D_4 , the *Schläfli lattice* is the densest lattice packing in 4 dimensions. We will return to D_4 in our discussion of multi-dimensional trellis codes.

In order to describe signal constellations by lattices, we have to shift and scale the lattice. To obtain the rectangular constellations Q from Z^2 , for example, we set $Q = c(Z^2 + \{\frac{1}{2}, \frac{1}{2}\})$, where c is an arbitrary scaling factor, usually chosen to normalize the average symbol energy to unity, as agreed upon in Chapter 2, and the shift by $(\frac{1}{2}, \frac{1}{2})$ centers the lattice. As can be seen from Figure 3.12, such a shifted lattice can be used as a template for all QAM constellations.

In order to extract a finite signal constellation from the lattice, a boundary is introduced and only points inside the boundary are chosen. This boundary shapes the signal constellation and affects the average power used in transmitting signal points from such a

³The reason for the different notation of Z^N and D_N is that $Z^N = Z \times Z \times \cdots \times Z$, the Cartesian product of 1-dimensional lattices Z , while D_N is not.

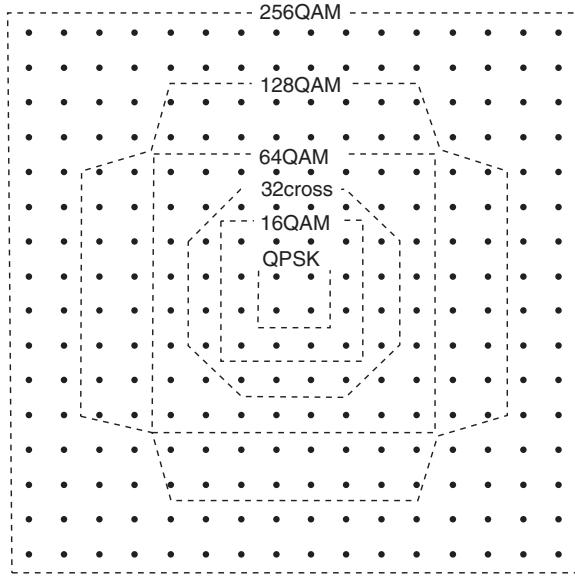


Figure 3.12: The integer lattice Z^2 as a template for the QAM constellations.

constellation. The effect of this shaping is summarized in the shaping gain γ_s .

If we consider the rectangular constellations (see Figure 3.12), the area of the square and the cross is given by $V(Z^2)2^n$, where 2^n is the number of signal points. The average energy E_s of these two constellations can be calculated by an integral approximation for large numbers of signal points and we obtain $E_s \approx \frac{2}{3}2^n$ for the square and $E_s \approx \frac{31}{48}2^n$ for the cross shape. That is, the cross constellation is $\frac{31}{32}$ or $\gamma_s = 0.14$ dB more energy efficient than the rectangular boundary.

The best enclosing boundary would be a circle, especially in higher dimensions. It is easy to see that the energy savings of an N -dimensional spherical constellation over an N -dimensional rectangular constellation is proportional to the inverse ratio of their volumes. This gain can be calculated as [9]

$$\gamma_s = \frac{\pi(N+2)}{12} \left(\left(\frac{N}{2} \right)! \right)^{-2/N}, \quad N \text{ even.} \quad (3.7)$$

In the limit, $\lim_{N \rightarrow \infty} \gamma_s = \pi e / 6$, or 1.53 dB. To obtain this gain, however, the constituent 2-dimensional signal points in our N -dimensional constellation will not be distributed uniformly. In fact, they will tend towards a Gaussian distribution.

The fundamental coding gain⁴

$$\gamma(\Lambda) = d_{\min}^2 / V(\Lambda)^{2/N} \quad (3.8)$$

relates the minimum squared Euclidean distance to the fundamental volume per 2 dimensions. This definition is meaningful since the volume per 2 dimensions is directly related to the signal energy of a QAM constellation, and thus $\gamma(\Lambda)$ expresses an asymptotic coding gain. For example, from the preceding discussion

$$\gamma(D_N) = \frac{2}{(2V(\Lambda))^{2/N}} = 2^{1-\frac{2}{N}} \gamma(Z^N), \quad (3.9)$$

and $\gamma(Z^N) = 1$ shall be used as reference henceforth. For example, $\gamma(D_4) = 2^{\frac{1}{2}}$, i.e., D_4 has a coding gain of $2^{\frac{1}{2}}$ or 1.51 dB over Z^4 . The definition (3.8) is also invariant to scaling as one would expect.

Partitioning of signal sets can now be discussed much more conveniently with the help of the lattice and is applicable to all signal constellations derived from that lattice. The binary set partitioning in Figure 3.10 can be derived from the binary lattice partition chain

$$Z^2 / \mathbf{R}Z^2 / 2Z^2 / 2\mathbf{R}Z^2 = Z^2 / D_2 / 2Z^2 / 2D_2. \quad (3.10)$$

Every level of the partition in (3.10) creates a sublattice Λ' of the original lattice Λ as shown in Figure 3.13. The remaining points, if we delete Λ' from Λ , are a shifted version Λ'_s of Λ' , called a coset. (There are p cosets in a p -ary partition.) Λ' and Λ'_s together make up Λ , i.e., $\Lambda = \Lambda' \cup \Lambda'_s$. (E.g. $Z^2 = \mathbf{R}Z^2 \cup (\mathbf{R}Z^2 + (1, 0))$.) The partition chain (3.10) generates $2^3 = 8$ cosets of $2\mathbf{R}Z^2$ and each coset is a translate of the final sublattice.

A coset of a lattice Λ is denoted by $\Lambda + \mathbf{c}$, where \mathbf{c} is some constant vector which specifies the coset. Note that if \mathbf{c} is in Λ , $\Lambda + \mathbf{c} = \Lambda$. Mathematically speaking, we have defined an equivalence relation of points, i.e., all points equivalent to \mathbf{c} modulo Λ are in the same coset $\Lambda + \mathbf{c}$.

If we now have a lattice partition Λ/Λ' , and we choose \mathbf{c} such that $\Lambda' + \mathbf{c} \in \Lambda$ (note: not Λ'), then every element in Λ can be expressed as

$$\Lambda = \Lambda' + [\Lambda/\Lambda'], \quad (3.11)$$

where $[\Lambda/\Lambda']$ is just a fancy way of writing the set of all such vectors \mathbf{c} . Equation (3.11) is called the coset decomposition of Λ in terms of cosets (translates) of the lattice Λ'

⁴The fundamental coding gain is related to the center density δ used in [5] by

$$\gamma(\Lambda) = 4\delta^{\frac{2}{N}}.$$

The center density of a lattice is defined as the number of points per unit volume if the touching spheres have unit radius.

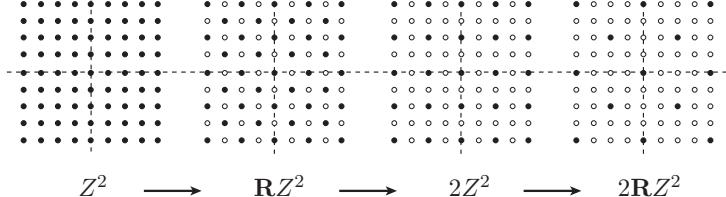


Figure 3.13: Illustration of the binary partition chain (3.10).

and the number of such different cosets is the order of the partition, denoted by $|\Lambda/\Lambda'|$. As an example consider the decomposition of Z^2 into $Z^2 = \mathbf{R}Z^2 + \{(0,0), (0,1)\}$. Analogously an entire partition chain can be defined, for example, for $\Lambda/\Lambda'/\Lambda''$ we may write $\Lambda = \Lambda'' + [\Lambda'/\Lambda''] + [\Lambda/\Lambda']$, that is, every element in Λ can be expressed as an element of Λ'' plus a shift vector from $[\Lambda'/\Lambda'']$ plus a shift vector from $[\Lambda/\Lambda']$. Again, for example, $Z^2 = 2Z^2 + \{(0,0), (1,1)\} + \{(0,0), (0,1)\}$; see Figure 3.13.

The lattice partitions discussed here are all binary partitions, that is, at each level the lattice is split into two cosets as in (3.10). The advantage of binary lattice partitions is that they naturally map into binary representations. Binary lattice partitions will be discussed more in Chapter 5.

Let us then consider a chain of K binary lattice partitions, i.e.,

$$\Lambda_1/\Lambda_2/\dots/\Lambda_K, \quad (3.12)$$

for which (3.10) may serve as an example of a chain of 4 lattices. There are then 2^K cosets of Λ_K whose union makes up the original lattice Λ_1 . Each such coset can now conveniently be identified by a K -ary binary vector $\mathbf{v} = (v^{(1)}, \dots, v^{(K)})$, that is, each coset is given by Λ_K shifted by

$$\mathbf{c}(\mathbf{v}) = \sum_{i=1}^K v^{(i)} \mathbf{c}^{(i)}, \quad (3.13)$$

where $\mathbf{c}^{(i)}$ is an element of Λ_i but not of Λ_{i+1} . The two vectors $\{0, \mathbf{c}^{(i)}\}$ are then two coset representatives for the cosets of Λ_{i+1} in the binary partition Λ_i/Λ_{i+1} . (Compare the example for the partition $Z^2/2Z^2$ above, where $\mathbf{c}^{(1)} = (0,1)$ and $\mathbf{c}^{(2)} = (1,1)$, see also Figure 3.14 below). Generalizing the above to the chain of length K , the cosets of Λ_K in the partition chain Λ/Λ_K are given by the linear sum (3.13).

In fact, the partition chain (3.10) is the basis for the generic encoder in Figure 3.11 for all QAM constellations. With the lattice formulation, we wish to describe our trellis encoder in the general form of Figure 3.15. The FSM encodes k bits and the mapping

function selects one of the cosets of the final sublattice of the partition chain, while the $n - k$ uncoded information bits select a signal point from that coset. The encoder can only affect the choice of cosets and therefore needs to be designed to maximize the minimum distance between sequences of cosets, where the minimum distance between two cosets Λ and Λ' is defined as $\min_{\substack{x \in \Lambda \\ x' \in \Lambda'}} |x, x'|^2 = \min_{x' \in \Lambda'} |x'|^2$ (taking the origin as reference, i.e., $x = 0$).

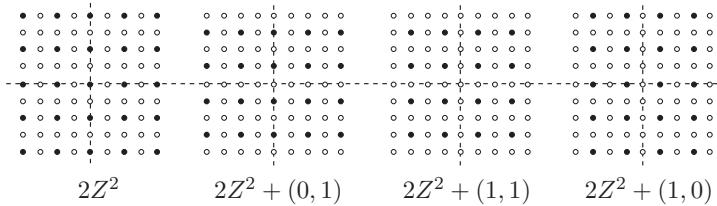


Figure 3.14: The 4 cosets of $2\mathbb{Z}^2$ in the partition $\mathbb{Z}^2/2\mathbb{Z}^2$.

As shown in Figure 3.15 the trellis code breaks up naturally into two components. The first is a called a *coset code* and is made up of the finite state machine and the mapping into cosets of a suitable lattice partition Λ/Λ' . The second part is the choice of the actual signal point within the chosen coset. This signal choice determines the constellation boundary and therefore shapes the signal set. It is referred to as *shaping*.

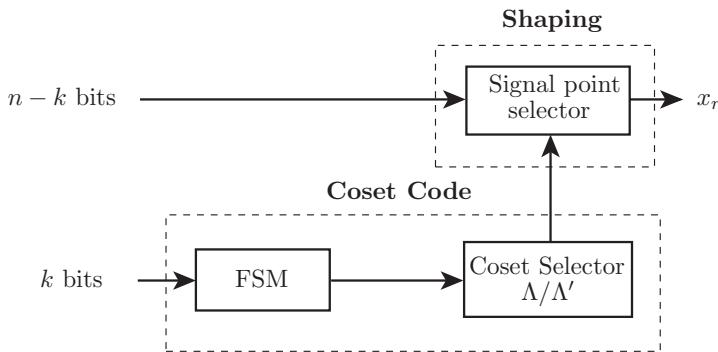


Figure 3.15: Generic trellis encoder block diagram using the lattice notation.

3.4 Lattice Formulation of Trellis Codes

A coset code as shown in Figure 3.15 will be denoted by $\mathcal{C}(\Lambda/\Lambda'; C)$, where C is the finite-state machine generating the branch labels. In most known cases, C is a convolutional code. The sequence of branch labels, the codewords of C , will be written as (v_r, v_{r+1}, \dots) , i.e., a sequence of binary $k + m$ -tuples, where m is the number of parity bits in the convolutional code (e.g., $m = 1$ for the codes generated by the encoder in Figure 3.6). Each v_j serves as a label which selects a coset of Λ' at time r , and the coded part of the encoder selects therefore a sequence of lattice cosets. The $n - k$ uncoded bits select one of 2^{n-k} signal points from the coset Λ'_r at time r . The sequence of branch labels v_r from the FSM are mapped by the coset selector into sequence of cosets \mathbf{c}_r .

Since the branch labels of our trellis code are now cosets containing more than one signal point, the minimum distance of the code is $\min(d_{\text{free}}^2, d_{\text{min}}^2)$, where d_{free}^2 is the minimum free squared Euclidean distance between any two output sequences and d_{min}^2 is the minimum squared Euclidean distance between members of Λ' , the final sublattice of Λ .

Figure 3.16 shows the generic QAM encoder in lattice formulation, where $v^{(0)}$ selects one of the two cosets of $\mathbf{R}\mathbf{Z}^2$ in the partition $\mathbf{Z}^2/\mathbf{R}\mathbf{Z}^2$, $v^{(1)}$ selects one of the two cosets of \mathbf{Z}^2 in the partition $\mathbf{R}\mathbf{Z}^2/2\mathbf{Z}^2$ and $v^{(2)}$ selects one of the two cosets in the partition $2\mathbf{Z}^2/2\mathbf{R}\mathbf{Z}^2$. (Compare also Figures 3.10 and 3.11.) The final selected coset in the $\mathbf{Z}^2/2\mathbf{R}\mathbf{Z}^2$ partition is given by

$$\mathbf{c} = v^{(0)}\mathbf{c}^{(0)} + v^{(1)}\mathbf{c}^{(1)} + v^{(2)}\mathbf{c}^{(2)}, \quad (3.14)$$

where $\mathbf{c}^{(0)}$ is the coset representative in the partition $\mathbf{Z}^2/\mathbf{R}\mathbf{Z}^2$, and $\mathbf{c}^{(1)}$ and $\mathbf{c}^{(2)}$ are the coset representatives in the partitions $\mathbf{R}\mathbf{Z}^2/2\mathbf{Z}^2$ and $2\mathbf{Z}^2/2\mathbf{R}\mathbf{Z}^2$, respectively.

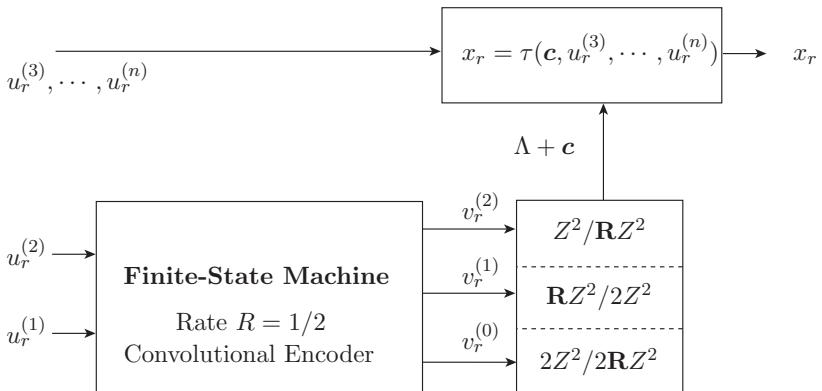


Figure 3.16: Lattice encoder for the 2-dimensional constellations grown from \mathbf{Z}^2 .

An important fact should be noted at this point. Contrary to the trellis code in Figure 3.11, the coset codes discussed in this section are group codes; i.e., two output coset sequences $c_1(D)$ and $c_2(D)$ may be added to produce another valid output sequence $c_3(D) = c_1(D) + c_2(D)$. The reason why this is the case lies precisely in the lattice formulation. With the lattice viewpoint we have also introduced an addition operator which allows us to add lattice points or entire lattices. We also note that the lattices themselves are infinite, and there are no boundary problems. The linearity of the lattice makes the entire trellis code regular, and one can use the all-zero sequence as reference sequence.

Using this new lattice description, trellis codes can now easily be classified. Ungerböck's original 1-, and 2-dimensional PAM codes are based on the four-way partition $Z/4Z$ and the eight-way partition $Z^2/2\mathbf{R}Z^2$, respectively. The 1-dimensional codes are used with a rate $R = 1/2$ convolutional code and the 2-dimensional codes with a rate $R = 2/3$ convolutional code with the exception of the 4-state code ($\nu = 2$) which uses a rate $R = 1/2$ code.

Table 3.4 shows these codes where the 2-dimensional codes are essentially a reproduction of Table 3.3. Also shown in the table is N_D , the number of nearest neighbors in the coset code. Note that the number of nearest neighbors is based on the infinite lattice and selecting a particular constellation from the lattice will reduce that number. As we will discuss in more detail later the number of nearest neighbors, and in fact the number of neighbors at a given distance in general, also affect performance and sometimes codes with a smaller d_{free}^2 can outperform codes with a larger d_{free}^2 but more nearest neighbors.

So far we have only concerned ourselves with 2-dimensional lattices as basis for our signal constellations, but in fact, this is an arbitrary restriction and multi-dimensional trellis codes, i.e., trellis codes using lattices of dimensions larger than 2 have been constructed and have a number of advantages. Multi-dimensional in this context is a theoretical concept, since, in practice, multi-dimensional signals are transmitted as sequences of 1- or 2-dimensional signals.

We have seen earlier that in order to introduce coding we need to expand the signal set from the original uncoded signal set. The most usual and convenient way is to double the constellation size, i.e., introduce one bit of redundancy. Now this doubling reduces the minimum distance within the constellation, and this reduction has to be compensated for by the code before any coding gain can be achieved. If we use, say a 4-dimensional signal set, this doubling causes the constituent 2-dimensional constellations to be expanded by a factor of only $\sqrt{2}$ (half a bit of redundancy per 2-D constellation). The advantage is that there is less initial loss in minimum distance within the signal set. If we consider rectangular signal sets derived from Z^N , we obtain the following numbers. For 2 dimensions the signal set expansion costs 3 dB in minimum distance loss, for 4-dimensional signals the loss is 1.5 dB and for 8-dimensional signal sets it is down to 0.75 dB. We see that the code itself has to overcome less and less signal set expansion loss. Another point in

Number of States	Λ	Λ'	d_{\min}^2	Asymptotic Coding Gain	N_D
1-dimensional codes					
4	Z	$4Z$	9	3.52 dB	8
8	Z	$4Z$	10	3.98 dB	8
16	Z	$4Z$	11	4.39 dB	16
32	Z	$4Z$	13	5.12 dB	24
64	Z	$4Z$	14	5.44 dB	72
128	Z	$4Z$	16	6.02 dB	132
256	Z	$4Z$	16	6.02 dB	4
512	Z	$4Z$	16	6.02 dB	4
2-dimensional codes					
4	Z^2	$2Z^2$	4	3.01 dB	4
8	Z^2	$2RZ^2$	5	3.98 dB	16
16	Z^2	$2RZ^2$	6	4.77 dB	56
32	Z^2	$2RZ^2$	6	5.77 dB	16
64	Z^2	$2RZ^2$	7	5.44 dB	56
128	Z^2	$2RZ^2$	8	6.02 dB	344
256	Z^2	$2RZ^2$	8	6.02 dB	44
512	Z^2	$2RZ^2$	8	6.02 dB	4

Table 3.4: Lattice partition, free distance, asymptotic coding gain and number of nearest neighbors for the original Ungerööck 1- and 2-dimensional trellis codes.

favor of multi-dimensional codes is that linear codes with 90-degree phase invariance can be constructed. This will be explored in more detail in Section 3.7.

Let us consider codes over 4-dimensional rectangular signal sets as an example. The binary partition chain we use is

$$Z^4/D_4/RZ^4/RD_4/2Z^4/2D_4/\dots \quad (3.15)$$

with minimum distances

$$1/\sqrt{2}/\sqrt{2}/2/2/2\sqrt{2}/\dots \quad (3.16)$$

The FSM generating the code trellis is now designed to maximize the interset distance of sequences. No design procedure for good codes is known to date and computer searches are usually carried out, either exhaustively or with heuristic selection and rejection rules. Note,

however, that the computer search needs to optimize only the coset code which is linear, and therefore we can choose $v(D) = 0$ as the reference sequence. Now we have a simple mapping of v_r into a distance increment d_i^2 , and $d_{\text{free}}^2 = \min_{\mathbf{c}(D), \mathbf{c}'(D)} \|\mathbf{c}(D) - \mathbf{c}'(D)\|^2 = \min_{v(D)} \sum_r d_r^2(v_r)$, where $d_r^2(\mathbf{v}_r) = \min_{x \in \Lambda_s(\mathbf{v}_r)} \|x\|^2$ is the distance from the origin to the closest point in the coset $\Lambda_s(\mathbf{v}_r)$ specified by \mathbf{v}_r .

Figure 3.17 illustrates the partitioning tree analogously to Figure 3.10 for the 4-dimensional 8-way partition $\Lambda/\Lambda' = Z^4/\mathbf{R}D_4$. The cosets at each partition are given as unions of two 2-D cosets, i.e., $D_2 \times D_2 \cup \overline{D}_2 \times \overline{D}_2$ is the union of the Cartesian product of D_2 with D_2 and \overline{D}_2 with \overline{D}_2 , where $\overline{D}_2 = D_2 + (0, 1)$ is the second coset.

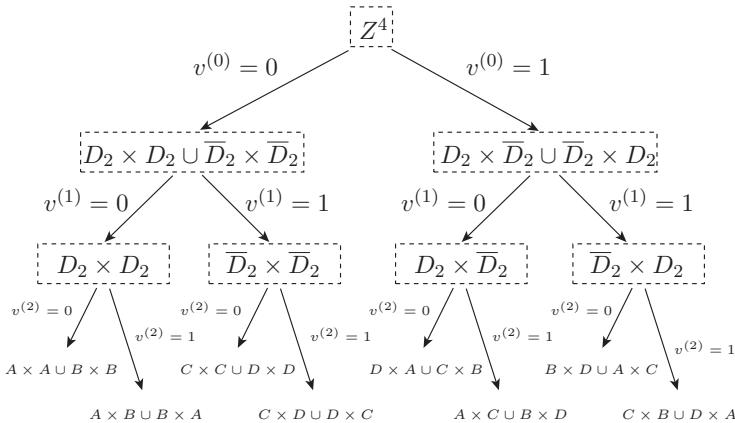


Figure 3.17: 8-way lattice partition tree of Z_4 . The constituent 2 dimensional cosets are $A = 2Z^2$, $B = 2Z^2 + (1, 1)$, $C = 2Z^2 + (0, 1)$ and $D = 2Z^2 + (1, 0)$ as shown in Figure 3.14.

Figure 3.18 shows a 16-state encoder using a rate $R = 2/3$ convolutional code to generate a trellis with $d_{\text{free}}^2 = d_{\text{min}}^2 = 4$, i.e., the parallel transitions are those producing the minimum distance. Since the final lattice $\Lambda' = \mathbf{R}D_4$ is a rotated version of the checkerboard lattice D_4 , we immediately also know the number of nearest neighbors at $d_{\text{min}}^2 = 4$, which is $N_D = 24$. The addition of the differential encoder allows the code to be made rotationally invariant to 90 degree phase rotations. This particular encoder will be discussed further in Section 3.7 on rotational invariance.

One philosophy, brought forth by Wei [36], is to choose a lattice partition Λ/Λ' where Λ' is a denser lattice than Λ . A dense lattice Λ' increases the minimum distance, and therefore the asymptotic coding gain, however this also increases the number of nearest neighbors. Another advantage is that this philosophy simplifies code construction since codes with fewer states may be used to achieve a given coding gain.

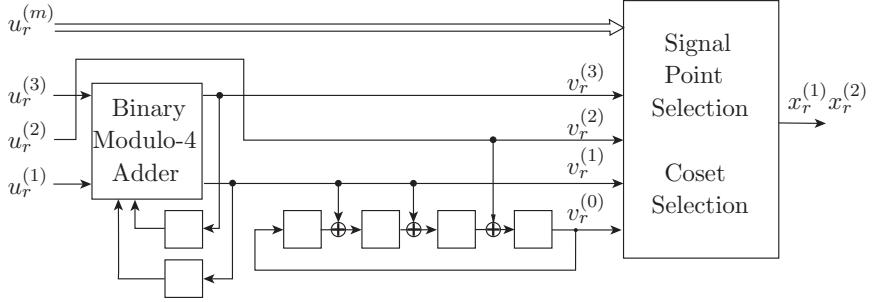


Figure 3.18: 16-state 4-dimensional trellis code using the 8-way partition shown in Figure 3.17 above. The differential encoder on the input makes the code rotationally invariant.

Tables 3.4 summarizes the best multi-dimensional trellis codes found to date. The index in the last column refers to the source of the code. Note that the table also includes 8-dimensional codes, mainly pioneered by Wei [36], and by Calderbank and Sloane [3, 4].

Lattice	Kissing Numbers (Nearest neighbors)	Neigh- bors)	d_{\min}^2	Fundamental Coding $\gamma(\lambda)$	Gain
D_N (Checkerboard)	$2N(N - 1)$		2	$2^{1 - \frac{2}{N}}$	
E_8 (Gosset)	240		4	2	
H_{16}			4		
Λ_{16} (Barnes-Wall)	4320		8	$2^{\frac{3}{2}}$	
Λ_{24} (Leech)	196560		8	4	
D_{32} (Barnes-Wall)	208320		16	4	

Table 3.5: Sublattices of Z^N , their minimum distance d_{\min}^2 , and their kissing number.

While we now have discussed the construction and classification of multi-dimensional codes, we have said little about the motivation to use more than 2 dimensions. 2-dimensional signal constellations seem to be a natural choice since bandpass double-sideband modulated signals naturally generate a pair of closely related dimensions, the inphase and quadrature channels. The motivation to go to higher dimensional signal sets is not quite so obvious, but there are two very important reasons. The first is the smaller constellation expansion factor discussed previously. The second is that multi-dimensional codes can be made invariant to phase rotations using linear FSMs.

All the lattices discussed, i.e., D_2 , D_4 , E_8 , and H_{16} , are sublattices of the Cartesian product lattice Z^N . Signal constellations derived from these lattices are particularly useful since a modem built for Z^2 , i.e., QAM constellations, can easily accommodate the lattices without much extra effort. We simply disallow the signal points in Z^N which are not points in the sublattice in question. Table 3.4 shows these useful sublattices.

Number of States	Λ	Λ'	d_{\min}^2	Asymptotic Coding Gain	N_D	Source
4-dimensional codes						
8	Z^4	$\mathbf{R}D_4$	4	4.52 dB	44	W
16	D_4	$2D_4$	6	4.77 dB	152	C-S
64	D_4	$2D_4$	8	5.27 dB	828	C-S
16	Z^4	$\mathbf{R}D_4$	4	4.52 dB	12	W
32	Z^4	$2Z^4$	4	4.52 dB	4	W
64	Z^4	$2D_4$	5	5.48 dB	72	W
128	Z^4	$2D_4$	6	6.28 dB	728	U
8-dimensional codes						
16	Z^8	E_8	4	5.27 dB	316	W
32	Z^8	E_8	4	5.27 dB	124	W
64	Z^8	E_8	4	5.27 dB	60	W
128	Z^8	$\mathbf{R}D_8$	4	5.27 dB	28	U
32	RD_8	$\mathbf{R}E_8$	8	6.02 dB		W
64	RD_8	$\mathbf{R}E_8$	8	6.02 dB	316	W
128	RD_8	$\mathbf{R}E_8$	8	6.02 dB	124	W
8	E_8	$\mathbf{R}E_8$	8	5.27 dB	764	C-S
16	E_8	$\mathbf{R}E_8$	8	5.27 dB	316	C-S
32	E_8	$\mathbf{R}E_8$	8	5.27 dB	124	C-S
64	E_8	$\mathbf{R}E_8$	8	5.27 dB	60	C-S
16-dimensional codes						
32	Z^{16}	H_{16}	4	5.64 dB		W
64	Z^{16}	H_{16}	4	5.64 dB	796	W
128	Z^{16}	H_{16}	4	5.64 dB	412	W

Table 3.6: Best multi-dimensional trellis codes based on binary lattice partitions of 4, 8, and 16 dimensions. The source column indicates the origin of the code, i.e., codes marked U were found in [34], those marked W in [36] and those marked C-S in [4]. The Gosset lattice E_8 and the lattice H_{16} are discussed in Chapter 5.

3.5 Rotational Invariance

With ideal coherent detection of a DSB-SC signal (Figure 2.8), the absolute carrier phase of the transmitted signal needs to be known at the receiver, in order to ensure the correct orientation of the signal constellation. Before synchronization is accomplished, the received signal has a phase offset by an angle θ ; i.e., the received signal, ignoring noise, is given by (see Equation (2.34))

$$s_0(t) = x(t)\sqrt{2} \cos(2\pi f_0 t + \phi). \quad (3.17)$$

This offset is typically by frequency drifts of the local oscillator, channel doppler effects, and phase noise in the system.

Let us assume for purposes of illustration that we are using an MPSK signal set. We can then write (3.17) as

$$s_0(t) = \sqrt{2} \cos(2\pi f_0 t + \phi_m(t) + \phi), \quad (3.18)$$

where $\phi_m(t)$ is the time-varying data phase. For example, for QPSK $\phi_m(t)$ can assume the angles $\pi/2, \pi, 3\pi/2$ and 2π . The carrier phase tracking loop, conventionally a *phase-locked loop* (PLL) circuit, first needs to eliminate the data dependent part of the phase. In the case of the MPSK signaling this can be accomplished by raising received signal to the M th power. This M th order power device generates a spectral line $\cos(2\pi M f_0 + M\phi_m(t) + M\phi)$ at Mf_0 . But $M\phi_m(t)$ is always a multiple of 2π for MPSK modulation, and this M th power spectral line is now free of the data-dependent phase changes, given by $\cos(2\pi M f_0 + M\phi)$. A standard PLL can now used to track the phase, and the local oscillator signal is generated by dividing the frequency of the tracking signal by M .

However, the squaring device causes a significant squaring loss due to cross-products between noise and signal, which manifests itself as an SNR loss in the tracking loop, and is particularly severe for higher-order constellations [28, 14]. For 8-PSK, for example, this squaring loss w.r.t. an unmodulated carrier is on the order of 10 dB. Due to this, decision-directed PLL techniques are typically used in practice. Nonetheless, phase synchronization is a challenging task, and, more recently, integrated iterative phase estimation and data detection algorithms have shown much promise [16] (see also Chapter 12), a concept which has been extended to complete channel estimation [26].

Even with successful phase synchronization, phase offsets by multiples of constellation symmetry angle, $\pi/2$ for QPSK, cannot be identified by a typical PLL-based phase tracking circuit, leaving the carrier phase recovery system with a phase ambiguity, which have the undesirable effect that the received signal constellations may be rotated versions of the transmitted constellations, and the trellis decoder can usually not decode properly if the constellations are rotated. This necessitates that all possible phase rotations of the constellation have to be tried until the correct one is found, which can cause unacceptable delays. It is therefore desirable to design TCM systems such that they have as many phase

invariances as possible, that is, rotation angles which do not affect decoder operation. This will also assure more rapid resynchronization after temporary signal loss.

In decision-directed carrier phase acquisition and tracking circuits, this rotational invariance is even more important, since without proper output data, the tracking loop is in an undriven random-walk situation, from which it may take a long time to recover. This situation also can be avoided by making the trellis code invariant to constellation rotations.

A trellis code is called *rotationally invariant* with respect to the rotation of constituent constellation by an angle ϕ , if the decoder can correctly decode the transmitted information sequence when the local oscillator phase differs from the carrier phase by ϕ . Naturally, ϕ is restricted to be one of the phase ambiguity angles of the recovery circuit, which is a rotational symmetry angle of the signal constellation, i.e., an angle which rotates the signal constellation into itself. If \mathbf{x} is a sequence of coded symbols of a certain trellis code, denote by \mathbf{x}^ϕ the symbol sequence which is obtained by rotating each symbol x_r by the angle ϕ . We now have the following:

Definition 3.2 A TCM code is rotationally invariant with respect to a rotation by an angle ϕ , if $\mathbf{x} \rightarrow \mathbf{x}^\phi$ is a code sequence for all valid code sequences \mathbf{x} .

This definition is rather awkward to test or work with. But we can translate it into conditions on the transitions of the code trellis. Assume that there is a total of S states in the code's state space \mathcal{SP} and that there are P subsets which result from set partitioning the signal constellation. Assume further that the partitioning is done such that for each phase rotation, each subset rotates into either itself or another subset; i.e., the set of subsets is invariant under these rotations. This latter point is automatically true for 1- and 2-dimensional constellations [36] (see, e.g., Figure 3.10) if we use the type of lattice partitioning discussed in the previous sections. With these assumptions we now may state the following [36, 39]

Theorem 3.1 A TCM code is rotationally invariant with respect to a rotation by an angle ϕ , if there exists a (bijective) function $f_\phi : \mathcal{SP} \mapsto \mathcal{SP}$ with the following properties: For each transition from a state i to a state j , denote the associated signal subset by A . Denote by A^ϕ the subset obtained when A is rotated by the angle ϕ , $A \xrightarrow{\phi} A^\phi$. Then A^ϕ is the subset associated with the transition from state $f_\phi(i)$ to state $f_\phi(j)$.

Proof: The situation in Theorem 3.1 is illustrated in Figure 3.19. It is easy to see that if we concatenate successive trellis sections, for each path i, j, k, \dots , there exists a valid path $f_\phi(i), f_\phi(j), f_\phi(k), \dots$, of rotated symbols through the trellis. Q.E.D.

There are now two components to making a trellis code transparent to a phase rotation of the signal constellation. Firstly, the code must be rotationally invariant; that is,

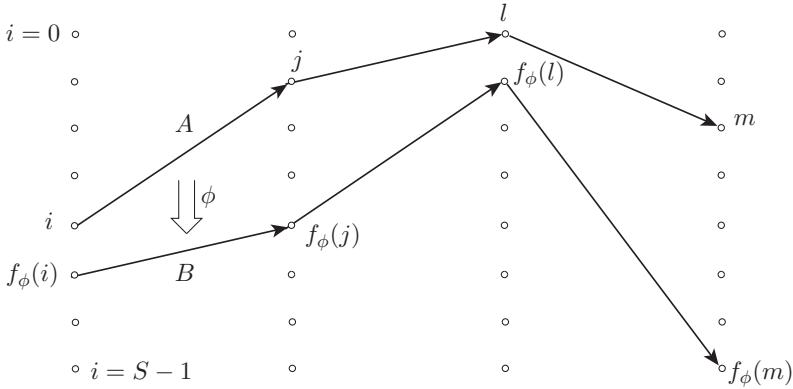


Figure 3.19: Illustration of Theorem 3.1 and its proof.

the decoder will still find a valid code sequence after rotation of \mathbf{x} by ϕ . Secondly, the rotated sequence \mathbf{x}^ϕ needs to map back into the same information sequence as the original sequence. This is achieved by differentially encoding the information bits, as illustrated below.

It turned out to be impossible to achieve rotational invariance of a code with a linear generating FSM in conjunction with 2-dimensional signal constellations [21, 22, 17], and hence “non-linear trellis codes” were investigated [37, 38]. Figure 3.20 shows such a non-linear eight-state trellis code for use with QAM constellations, illustrated for use with the 32-cross constellation to transmit 4 bits/symbol, as shown in Figure 3.21. This code was adopted in the CCITT V.32 Recommendation [18] and provides an asymptotic coding gain of 4 dB. The three leading uncoded bits which are in plain typeface in Figure 3.21 are not affected by 90 degree rotations of the constellation. Only the last two bits, marked in bold, are affected and need to be encoded differentially, as done in the encoder. The constellation is partitioned into the eight subsets D_0, \dots, D_7 . Note that successive 90 degree phase rotations take $D_0 \rightarrow D_1 \rightarrow D_2 \rightarrow D_3 \rightarrow D_0$, and $D_4 \rightarrow D_5 \rightarrow D_6 \rightarrow D_7 \rightarrow D_4$.

Equipped with this information we may now study a section of the trellis diagram of this non-linear, rotationally invariant trellis code. This trellis section is shown in Figure 3.22 below, together with the state correspondence function $f_\phi(i)$, for successive 90 degree rotations of the signal constellations (see Theorem 3.1). It is relatively easy to see that, if we take an example sequence D_4, D_0, D_3, D_6, D_4 and rotate it by 90 degrees into D_5, D_1, D_0, D_7, D_5 , we obtain another valid sequence of subsets. Careful checking reveals that the function required in Theorem 3.1 exists, and the corresponding state relations are illustrated in Figure 3.22.

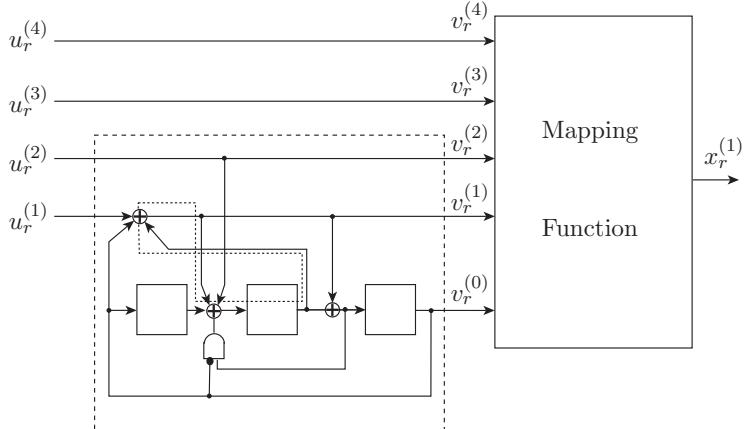


Figure 3.20: Non-linear rotationally invariant 8-state trellis code for QAM constellations.

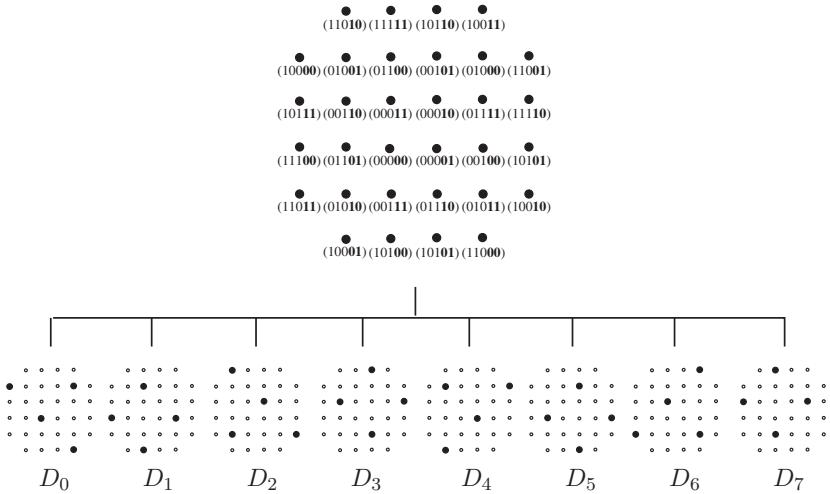


Figure 3.21: Example of a 32-cross constellation for the non-linear trellis code of Figure 3.20. The set partitioning of the 32-cross constellation into the eight subsets D_0, \dots, D_7 is also illustrated [34].

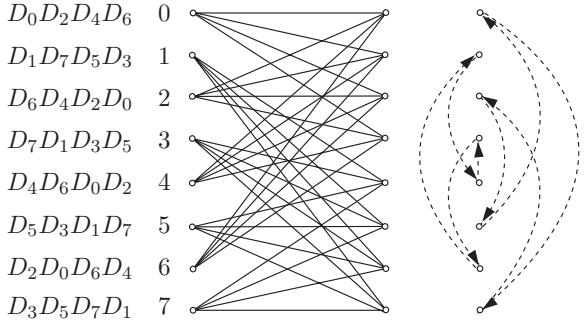


Figure 3.22: Trellis section of the non-linear 8-state trellis code. The subset labeling is such that the first signal set is the one on the top branch, and the last signal set is on the lowest branch, throughout all the states.

This then takes care of the rotational invariance of the code with respect to phase rotations of multiples of 90 degrees. However, the bit $v_r^{(1)} = u_r^{(1)}$ changes its value through such rotations. This is where the differential encoding comes into play. In Figure 3.20, this function is realized by the exor gate on the input line $u_r^{(1)}$ and the second delay cell; i.e., the differential encoder is integrated into the trellis encoder. This integrated differential encoder is indicated by the dotted loop in Figure 3.20. The coding gain of this code is 4 dB and the number of nearest neighbors is 16 [34].

The actual standards V.32 and V.33 [1] use the trellis code shown in Figure 3.23. The code is an 8-state trellis code with 90 degree phase invariance. The V.32 standard operates at duplex rates of up to 9600 bit/s, at a symbol rate of 2400 baud using the rotated cross constellation of Figure 3.24 below. We leave it as an exercise to the reader to show that the V.32 code is rotationally invariant to phase angles which are multiples of $\pi/2$.

The V.33 recommendation allows for data rates of up to 14,400 bit/s. It is designed to operate over point-to-point, four-wire leased telephone-type circuits. It uses the same encoder as V.32 but with an expanded 128-point cross signal constellation. It too is invariant to phase rotations of multiples of $\pi/2$.

While no linear trellis codes exist for 2-D constellations which are rotationally invariant, such codes can be found for higher dimensional signal sets. In fact, the encoder in Figure 3.18 generates such a code. The signal set partition chain for this encoder was given in Figure 3.17, and a 90 degree phase rotation will cause the following signal set

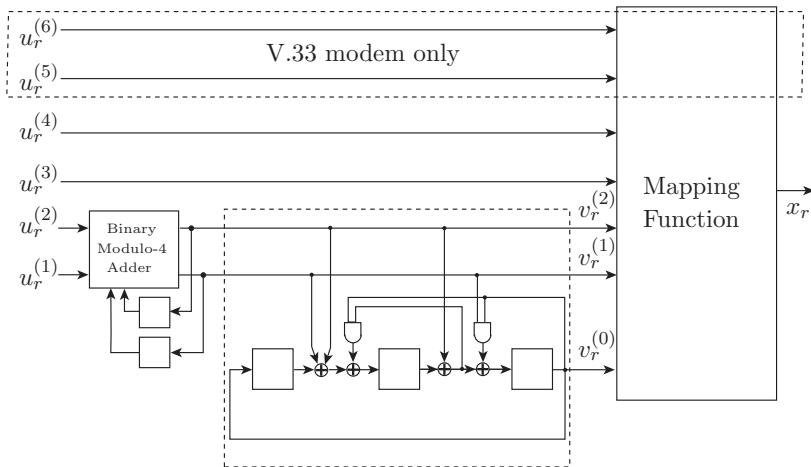


Figure 3.23: Non-linear 8-state trellis code used in the V.32 and V.33 recommendations.

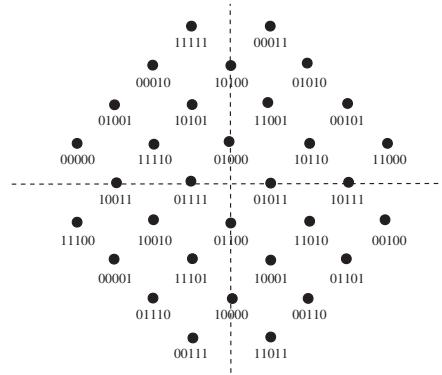


Figure 3.24: 32-point rotated cross signal constellation used in the V.32 recommendation for a 9600-bit/s voiceband modem.

changes⁵: $A \times A \cup B \times B \rightarrow C \times C \cup D \times D \rightarrow A \times A \cup B \times B$, $A \times B \cup B \times A \rightarrow C \times D \cup D \times C \rightarrow A \times B \cup B \times A$, $D \times A \cup C \times B \rightarrow B \times D \cup A \times C \rightarrow D \times A \cup C \times B$, and $A \times C \cup B \times D \rightarrow C \times B \cup D \times A$ (see Figures 3.14 and 3.17). A section of the trellis of this code is shown in Figure 3.25 below.

The state correspondences are worked out by checking through Theorem 3.1, from where we find that $f_{90^\circ}(0) = 4$, $f_{90^\circ}(4) = 0$, etc., as indicated in Figure 3.25 by the arrows.

Note that, due to the multi-dimensional nature of the signal set, 180 degree phase rotations map the subsets back onto themselves. The remaining ambiguities can be taken care of by differential encoding within the subsets.

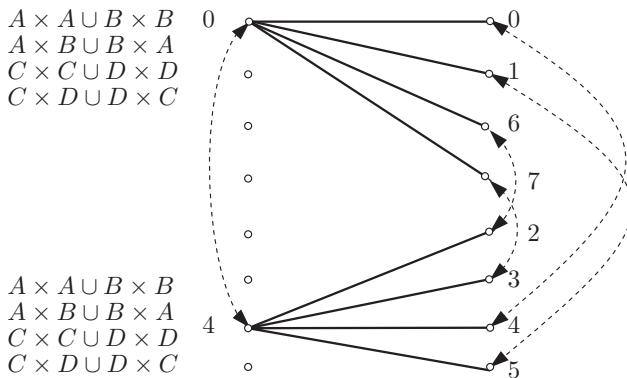


Figure 3.25: Portion of the trellis section of the linear 16-state trellis code, which achieves 90 degree rotational invariance.

Wei [39] and Pietrobon et al. [22] tackled the problem of designing rotationally invariant trellis codes for M-PSK signal constellations. This is somewhat more complicated, since the number of phase ambiguities of the constellations are larger than for QAM constellations. The basic philosophy, however, remains the same.

The fastest modems can achieve data transmission rates of up to 33,600 bit/s over the public switched telephone network (PSTN). This is roughly equal to the information theoretic capacity of an AWGN-channel with the same bandwidth. These modems adaptively select and choose the transmission band, the trellis code, constellation shaping, precoding and pre-emphasis for equalization, and signal set warping. These techniques are summarized in [7]. These modems use trellis coding and are described in the standard recommendation V.34, also nicknamed V.fast.

⁵Recall that the lattice is shifted by $(\frac{1}{2}, \frac{1}{2})$ in order to obtain the constellation.

3.6 V.fast

V.34 [19], or V.fast, is an analog modem standard for full-duplex transmission at rates up to 33.6 kbytes/s over standard telephone networks. Given the average signal-to-noise ratio, mainly due to quantization noise in the digital trunk lines, of typically between 28–38 dB, this rate is very close to the information theoretic capacity of this channel, and V.fast represents therefore the pinnacle of achievement and state of the art technology for digital data transmission over voiceband channels.

Due to the wide range of signal-to-noise ratio values and other channel conditions encountered on voiceband channels, the V.34 standard supports a large number of different rates, the highest supportable being negotiated between participating modems at the start of communications session [8, 19]. V.34 uses QAM modulation like all previous voiceband modems, but allows for an adaptive symbol rate and center frequency. The modem selects one of six symbol rates and one of two carrier frequencies according to channel conditions. These are 2400 (1600 Hz/1800 Hz), 2743 (1646 Hz/1829 Hz), 1680 (1680 Hz/1867 Hz), 3000 (1800 Hz/2000 Hz), 3200 (1829 Hz/1920 Hz), and 3429 (1959 Hz).

The trellis codes used in V.fast are 4-dimensional codes, whose small constellation expansion helps guard against non-Gaussian distortions. Three codes are specified in the standard, a 16-state 4-dimensional trellis code [36] with a coding gain of 4.2 dB, a 32-state code [40] with a coding gain of 4.5dB, and a 64-state code [35] with a coding gain of 4.7 dB. V.34 also specifies a shaping method called shell mapping in 16 dimensions to achieve a near spherical shaping of the signal constellation. This mapping achieves 0.8 dB of shaping gain, which is somewhat more than half of the theoretical maximum of 1.53 dB (see Section 3.3).

The large rates of these modems are mainly achieved by increasing the size of the constellations used, up to 1664 points for rates of 33.6 kbytes/s. The fundamental coding approach to counter the additive noise via trellis coding is unchanged and similar to the earlier modems discussed in this chapter. Adaptive equalization and precoding, however, have contributed as much to the success of V.34 as has trellis-coded modulation, by equalizing the channel and removing distortion. Every effort is made by the modem to use the available bandwidth, even if the attenuation at the edges of the band are as severe as 10 dB. Decision feedback equalization techniques have been chosen for this since linear equalizers generate too much noise enhancement in cases of deep attenuations. The equalization is implemented by precoding the data via a three-tap precoding filter. More details on precoding techniques can be found in [13].

Figure 3.26 illustrates the interplay of these different techniques at the transmitter. The differential encoder ensures rotational invariance to multiples of 90 degree rotations while the shell mapper shapes the signal constellation requiring a constellation expansion of 25%. The 4-dimensional trellis code requires a 50% constellation expansion making the total signal constellation expansion 75%.

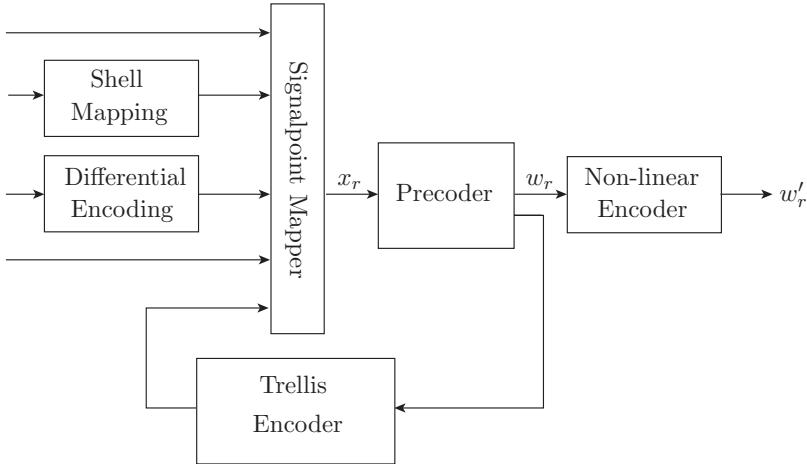


Figure 3.26: Encoder block diagram of a V.fast transmitter.

The precoder and trellis encoder are seemingly switched backwards in this setup, where the trellis encoder determines its next state from the output symbol chosen by the precoder, rather than directly from the input symbols. This helps minimize the predistortion that the precoder needs to add to the symbol x_r in order to generate the transmitted symbol w_r . The precoder compensates for the linear distortion of the channel and the effect is that the received sequence is a simple noise trellis sequence that can be decoded by a standard trellis decoder (Chapter 5). The non-linear encoder is optional and is designed to counter effects of non-linear distortion.

V.fast brings together a number of advanced techniques to exploit the information carrying capacity of the telephone voiceband channel, and can rightfully be considered a triumph of information technology. Since the information theoretic capacity does not allow any further significant increase in data speeds, V.fast has also been nicknamed V.last. It is important to note in this context that V.90, the 56 kbit/s downlink modem in common use in the 1990's, relies on a different technology and more importantly on a different channel. Since the digital trunk telephone lines carry 64 bits/s digital channels all the way to the local switching stations, and since the twisted pair cables from the switching stations to the end-user's telephones is analog and, not suffering from the quantization noise, has a much larger SNR, almost the entire digital data traffic can be transported to the end-user by a modem which treats the trunk lines as a digital channel rather than as a noise voiceband channel. This is precisely what V.90 does.

3.7 The IEEE 802.3an Standard

We conclude this chapter with a discussion of the physical layer of the IEEE 802.3an standard for 10 Gigabit Ethernet transmission over twisted pair copper cables. While not a “trellis-coded modulation” example, it is a “coded modulation” example of the latest generation and follows the same design principles as discussed here.

To run multi-gigabit data rates on four pairs of copper cabling, the BASE-T channel, sophisticated digital signal processing is used to eliminate the effects of cross-talk between pairs of cable and to remove the effects of signal reflections, but the main workhorse for the system is the coded modulation system discussed in this section.

The major differences between the IEEE 802.3an coding system and the trellis-coded modulation methods used for the voiceband channel lies in the adoption of a large regular LDPC code, to be discussed in detail in Chapter 6, as well as the choice of a novel 128-point, 2-dimensional QAM constellation for achieving high spectral efficiency.

Figure 3.27 shows the block diagram of a the IEEE 802.3an communication system, and we see the familiar blocks of the coded and uncoded transmit function and the signal mapping in the 2-dimensional QAM constellation. The receiver blocks are also completely analogous to those we discussed for trellis-based systems, consisting of the error control code decoder and the coset decoder. The only differences are the additional 2-D rotation of the incoming signals, which will be explained below, and the fact that the error control decoder is an LDPC code and not a convolutional code.

The twisted-pair copper cable channel is a baseband channel and therefore the basic signaling method is to use Pulse-Amplitude Modulation (PAM). But rather than standard 1-dimensional PAM, two successive PAM symbols are combined into a 2-dimensional signal proposed by BroadCom in [2]. The two dimensions are made up of two consecutive one-dimensional dithered 8-PAM signal constellations. A 16-level set partitioning of this constellation achieves enough intra-subset distance to permit uncoded transmission of 3 of the seven information bits.

The information symbol vector is therefore separated into 3 uncoded bits and 4 coded bits, the latter using a (2048,1723) LDPC code for noise protection. The constellation and mapping of bits is shown in Figure 3.28. The three uncoded bits select one of 8 shaded subsets. Within each such subset, there are 16 signal points, which are labeled by the 4 coded bits. Note that the differences in coded bits lead to signal points within a subset which can be close and thus are prone to errors. These bits are protected by the error control code. A change in an uncoded bit leads from one subset to another, at large distance which provides noise immunity, the basic principle of set partitioning.

The partition chain applied to this application is $D_2/2Z_2/2D_4/4Z^2$ with intraset distances $\sqrt{2}/2/2\sqrt{2}/4$, providing a 12 dB gain on the uncoded bit on an ideal AWGN channel. If we assume the points to lie on an integer grid with odd coordinates as in Figure 3.28, the average power per dimension equals $P_{\text{norm}} = 85$.

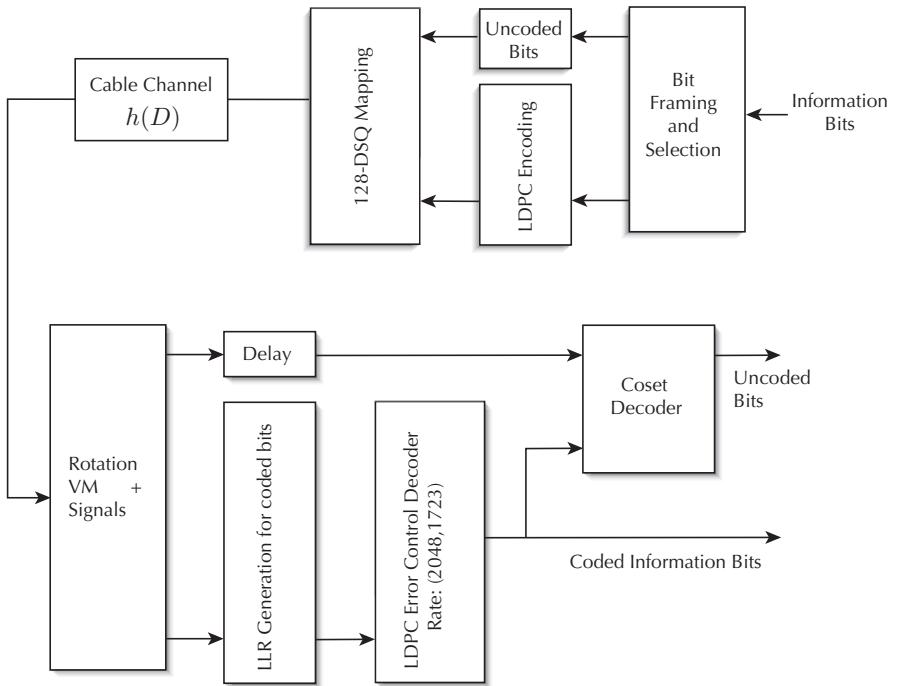


Figure 3.27: Block diagram of the IEEE 802.3an 10 Gbit/s transmission system.

Decoding of the coded bits, denoted as c_1, c_2, c_3, c_4 , needs to be accomplished first, and we now assume that $(\hat{c}_1, \hat{c}_2, \hat{c}_3, \hat{c}_4) = (c_1, c_2, c_3, c_4)$, i.e., no decoding errors occurred. (Otherwise the frame will be in error independently of the uncoded bits.) The coded bits $(\hat{c}_1, \hat{c}_2, \hat{c}_3, \hat{c}_4)$ now identify one of 16 subsets from the signal constellation in Figure 3.28.

For discussion purposes we assume that $(\hat{c}_1, \hat{c}_2, \hat{c}_3, \hat{c}_4) = (1, 1, 0, 1)$, in which case the subconstellation shown by the bold circles on the left-hand side of Figure 3.29 has been identified by the LDPC decoder. If $(\hat{c}_1, \hat{c}_2, \hat{c}_3, \hat{c}_4) = (0, 0, 0, 0)$, the subset on the right-hand side has been identified. The task of the coset decoder is now to extract the three uncoded bits which are encoded in the eight signal points of each subset. In order to simplify this process, we can translate each subset into the $(0,0,0,0)$ subset shown on the right in Figure 3.29. To accomplish this, each signal point is shifted to the left by two units, and down one unit in this case. Other subsets have different shifts according to the location of the

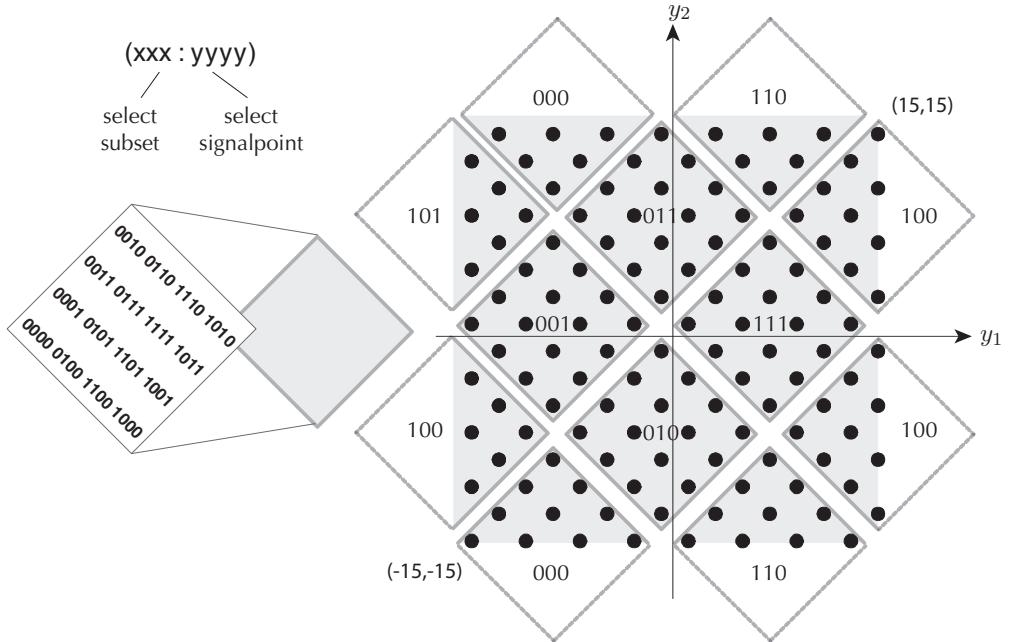


Figure 3.28: 128-DSQ constellation used in 802.3an.

binary 4-tuple $(\hat{c}_1, \hat{c}_2, \hat{c}_3, \hat{c}_4)$ in Figure 3.28. Points that fall outside the red square that defines the base constellation are wrapped around to the opposite side of the square, as illustrated for the points (000) and (110) . This modulo wrapping can be performed at any point in the signal processing path, but has to happen before the application of the decision equations (3.20).

This shift is given by the following set of shift equations, where the starting point for $(c_1, c_2, c_3, c_4) = (0, 0, 0, 0)$ is denoted by (y_1, y_2) .

(c_1, c_2, c_3, c_4)	Shift	(c_1, c_2, c_3, c_4)	Shift
$(0, 0, 0, 0)$	(y_1, y_2)	$(1, 0, 0, 0)$	$(y_1, y_2) + (-2, 2)$
$(0, 0, 0, 1)$	$(y_1, y_2) + (-2, -2)$	$(1, 0, 0, 1)$	$(y_1, y_2) + (-8, 4)$
$(0, 0, 1, 0)$	$(y_1, y_2) + (-6, -6)$	$(1, 0, 1, 0)$	$(y_1, y_2) + (-12, 0)$
$(0, 0, 1, 1)$	$(y_1, y_2) + (-4, -4)$	$(1, 0, 1, 1)$	$(y_1, y_2) + (-10, 2)$
$(0, 1, 0, 0)$	$(y_1, y_2) + (-2, 2)$	$(1, 1, 0, 0)$	$(y_1, y_2) + (-4, 4)$
$(0, 1, 0, 1)$	$(y_1, y_2) + (-4, 0)$	$(1, 1, 0, 1)$	$(y_1, y_2) + (-6, 2)$
$(0, 1, 1, 0)$	$(y_1, y_2) + (-8, -4)$	$(1, 1, 1, 0)$	$(y_1, y_2) + (-10, -2)$
$(0, 1, 1, 1)$	$(y_1, y_2) + (-6, -2)$	$(1, 1, 1, 1)$	$(y_1, y_2) + (-8, 0)$

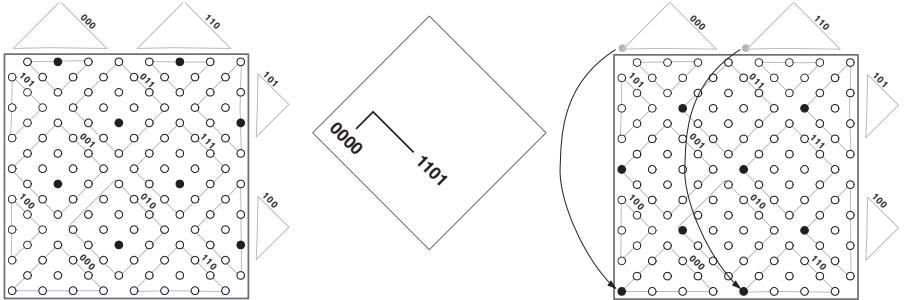


Figure 3.29: Subconstellation shift required for demodulating bits for $(c_1, c_2, c_3, c_4) = (1, 1, 0, 1)$. The subconstellation for $(0,0,0,0)$ is shown on the right.

Decoding of the uncoded bits can now be accomplished efficiently by realizing the relationships between the decision regions for the individual bits. Figure 3.30 shows these decision regions for the three uncoded bits u_1, u_2 , and u_3 . These decision regions are all congruent to each other, that is, a counterclockwise rotation of the set for u_1 by 90 degrees followed by a reflection on the vertical axis generates the decision region and boundaries for u_3 , and a simple reflection at the vertical axis generates the boundaries for u_2 .

Focussing on bit u_1 for subset $(c_1, c_2, c_3, c_4) = (0000)$, the decision region can be broken into four areas, depending on the location of the received coordinate s_2 , which can fall into four intervals: $s_2 \in [-15, -7], [-7, 1], [1, 9], [9, 17]$. Depending on where s_2 lies, the following decision are made for bit u_1 ; i.e., $u_1 = 1$

$$\begin{aligned} \text{if } s_2 \in [-15, -7] & \text{ AND } s_1 > s_2 + 8 & \text{ AND } s_1 \leq s_2 + 24 \\ \text{if } s_2 \in [-7, 1] & \text{ AND } s_1 > -s_2 - 6 & \text{ AND } s_1 \leq -s_2 + 10 \\ \text{if } s_2 \in [1, 9] & \text{ AND } s_1 > s_2 - 8 & \text{ AND } s_1 \leq s_2 + 8 \\ \text{if } s_2 \in [9, 17] & \text{ AND } s_1 > -s_2 + 10 & \text{ AND } s_1 \leq -s_2 + 26 \end{aligned} \quad (3.20)$$

The decisions for the other bits are made via the same process after the received signal sample operations discussed above, and the error probability of each uncoded bit is closely approximated by

$$P_{(b,u)} = 2Q\left(\sqrt{\frac{16 * 8}{2N_0}}\right) = Q\left(\sqrt{\frac{32}{\sigma^2}}\right) = 2Q\left(\sqrt{\frac{32 E_s}{85 \sigma^2}}\right), \quad (3.21)$$

giving a 12 dB gain over the raw bit error rate of the 128-DSQ constellation alone.

This concludes our discussion of the operation of coset-based coded modulation. Naturally, the performance of these systems is strongly dependent on the performance of the underlying error control codes. This will be discussed in subsequent chapters.

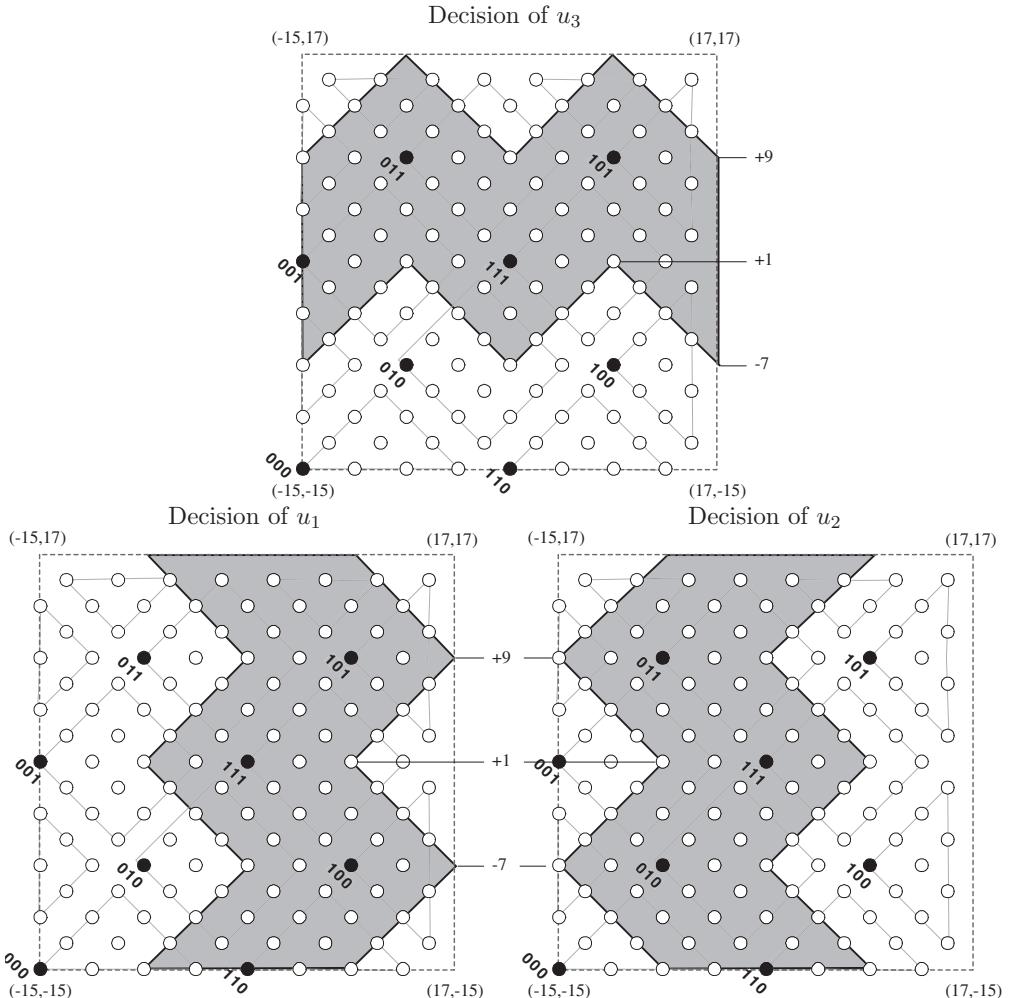


Figure 3.30: Decision region for uncoded bits u_1, u_2 and u_3 , derived from the prototype decision region for u_1 . Simply mirroring the s_1 variable allows us to use the same decision thresholds for both.

3.8 Historical Notes

While error control coding was long regarded as a discipline with applications mostly for channel with no bandwidth limitations such as the deep-space radio channel, the trellis-coded modulation schemes of Ungeröök [30, 31, 33, 34] provided the first strong evidence that coding could be used very effectively on bandlimited channels. Starting in the 1980's to this day, numerous researchers have discovered new results and new codes. Multi-dimensional schemes were first used by Wei [36] and Calderbank and Sloane [3], who also introduced the lattice viewpoint in [4]. Much of the material presented in this chapter is adapted from Forney's comprehensive treatments of the subject in [10, 11, 12].

The treatment of the IEEE 802.3an standard has been added here to illustrate the expansion of coded modulation into applications other than voice-band modems and is only one of the recent modern high-spectral efficiency coded modulation systems. It illustrates again, however, that redundant error control coding and signal set expansion can successfully be combined to achieve spectrally efficient transmission.

Bibliography

- [1] U. Black, *The V Series Recommendations, Protocols for Data Communications Over the Telephone Network*, McGraw-Hill, New York, 1991.
- [2] G. Ungerboeck et al., “LDPC (Low-Density Parity-Check) Coded 128 DSQ (Double-Square QAM) Constellation Modulation and Associated Labeling,” United States Patent Application, US 2006/0045197, March 2, 2006.
- [3] A.R. Calderbank and N.J.A. Sloane, “An eight-dimensional trellis code,” *Proc. IEEE*, vol. 74, pp. 757–759, 1986.
- [4] A.R. Calderbank and N.J.A. Sloane, “New trellis codes based on lattices and cosets,” *IEEE Trans. Inform. Theory*, vol. IT-33, pp. 177–195, 1987.
- [5] J.H. Conway and N.J.A. Sloane, *Sphere Packings, Lattices and Groups*, Springer-Verlag, New York, 1988.
- [6] J. Du and M. Kasahara, “Improvements of the information-bit error rate of trellis code modulation systems,” *IEICE, Japan*, vol. E 72, pp. 609–614, May 1989.
- [7] M.V. Eyuboglu, G.D. Forney, P. Dong, and G. Long, “Advanced modem techniques for V.Fast,” *Eur. Trans. Telecommun. ETT*, vol. 4, no. 3, pp. 234–256, May–June 1993.
- [8] G.D. Forney, Jr., L. Brown, M.V. Eyuboglu, and J.L. Moran III, “The V.34 high-speed modem standard,” *IEEE Commun. Mag.*, Dec. 1996.
- [9] 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. Select. Areas Commun.*, vol. SAC-2, no. 5, pp. 632–647, 1984.
- [10] G.D. Forney, “Coset codes—Part I: Introduction and geometrical classification,” *IEEE Trans. Inform. Theory*, vol. IT-34, pp. 1123–1151, 1988.
- [11] G.D. Forney, “Coset codes—Part II: Binary lattices and related codes,” *IEEE Trans. Inform. Theory*, vol. IT-34, pp. 1152–1187, 1988.
- [12] G.D. Forney, “Geometrically uniform codes,” *IEEE Trans. Inform. Theory*, vol. IT-37, pp. 1241 – 1260, 1991.

- [13] G.D. Forney and M.V. Eyuboglu, "Combined equalization and coding using precoding," *IEEE Commun. Mag.*, vol. 29, no. 12, pp. 25–34, Dec. 1991.
- [14] F.M. Gardner, *Phaselock Techniques*, 2nd edition, John Wiley & Sons, New York, 1979.
- [15] A. Gersho and R.M. Gray, *Vector Quantization and Signal Compression*, Kluwer Publishing, Dordrecht, 1996.
- [16] S. Howard, C. Schlegel, L. Perez, and F. Jiang, "Differential turbo-coded-modulation over unsynchronized channels," *Proceedings of the Wireless and Optical Communications Conference (WOC) 2002*, Banff, AB, Canada, July 2002.
- [17] IBM Europe, "Trellis-coded modulation schemes for use in data modems transmitting 3–7 bits per modulation interval," CCITT SG XVII Contribution COM XVII, no. D114, April 1983.
- [18] IBM Europe, "Trellis-coded modulation schemes with 8-state systematic encoder and 90° symmetry for use in data modems transmitting 3–7 bits per modulation interval," CCITT SG XVII Contribution COM XVII, no. D180, October 1983.
- [19] ITU-T Rec. V.34, "A modem operating at data signalling rates of up to 28,800 bit/s for use on the general switched telephone network and on leased point-to-point 2-wire telephone-type circuits," 1994.
- [20] J.L. Massey, T. Mittelholzer, T. Riedel, and M. Vollenweider, "Ring convolutional codes for phase modulation," *IEEE Int. Symp. Inform. Theory*, San Diego, CA, Jan. 1990.
- [21] S.S. Pietrobon, G.U Ungerboeck, L.C. Perez, and D.J. Costello, Jr., "Rotationally invariant nonlinear trellis codes for two-dimensional modulation," *IEEE Trans. Inform. Theory*, vol. IT-40, no. 6, pp. 1773–1791, Nov. 1994.
- [22] S.S. Pietrobon, R.H. Deng, A. Lafanechère, G. Ungerboeck, and D.J. Costello, Jr., "Trellis-coded multidimensional phase modulation," *IEEE Trans. Inform. Theory*, vol. IT-36, pp. 63–89, Jan. 1990.
- [23] S.S. Pietrobon and D.J. Costello, Jr., "Trellis coding with multidimensional QAM signal sets," *IEEE Trans. Inform. Theory*, vol. IT-39, pp. 325–336, March 1993.
- [24] J.E. Porath and T. Aulin, "Fast algorithmic construction of mostly optimal trellis codes," Technical Report no. 5, Division of Information Theory, School of Electrical and Computer Engineering, Chalmers University of Technology, Göteborg, Sweden, 1987.
- [25] J.E. Porath and T. Aulin, "Algorithmic construction of trellis codes," *IEEE Trans. Commun.*, vol. COM-41, no. 5, pp. 649–654, May. 1993.
- [26] C. Schlegel and A. Grant, "Differential space-time turbo codes," *IEEE Trans. Inform. Theory*, vol. 49, no. 9, Sep. 2003, pp. 2298–2306.

- [27] C. Schlegel and A. Pérez, *Trellis and Turbo Coding*, IEEE/Wiley, Hoboken, NJ, 2004.
- [28] B. Skar, *Digital Communications: Fundamentals and Applications*, Prentice Hall, Englewood Cliffs, NJ, 1988.
- [29] D. Slepian, “On neighbor distances and symmetry in group codes,” *IEEE Trans. Inform. Theory*, vol. IT-17, pp. 630–632, Sep. 1971.
- [30] G. Ungerboeck and I. Csajka, “On improving data-link performance by increasing channel alphabet and introducing sequence coding,” *International Symposium on Information Theory*, Ronneby, Sweden, June 1976.
- [31] G. Ungerboeck, “Channel coding with multilevel/phase signals,” *IEEE Trans. Inform. Theory*, vol. IT-28, no. 1, pp. 55–67, Jan. 1982.
- [32] G. Ungerboeck, J. Hagenauer, and T. Abdel-Nabi, “Coded 8-PSK experimental modem for the INTELSAT SCPC system,” *Proc. ICDSC, 7th*, pp. 299–304, 1986.
- [33] G. Ungerboeck, “Trellis-coded modulation with redundant signal sets Part I: Introduction,” *IEEE Commun. Mag.*, vol. 25, no. 2, pp. 5–11, Feb. 1987.
- [34] G. Ungerboeck, “Trellis-coded modulation with redundant signal sets Part II: State of the art,” *IEEE Commun. Mag.*, vol. 25, no. 2, pp. 12–21, Feb. 1987.
- [35] L.F. Wei, “A new 4D 64-state rate-4/5 trellis code,” Cont. D19, ITU-TSG 14, Geneva, Switzerland, Sep. 1993.
- [36] L.F. Wei, “Trellis-coded modulation with multidimensional constellations,” *IEEE Trans. Inform. Theory*, vol. IT-33, pp. 483 – 501, 1987.
- [37] L.F. Wei, “Rotationally invariant convolutional channel coding with expanded signal space–Part I: 180 degrees,” *IEEE J. Select. Areas Commun.*, vol. SAC-2,, pp. 659–672, Sep. 1984.
- [38] L.F. Wei, “Rotationally invariant convolutional channel coding with expanded signal space–Part II: nonlinear codes,” *IEEE J. Select. Areas Commun.*, vol. SAC-2,, pp. 672–686, Sep. 1984.
- [39] L.F. Wei, “Rotationally invariant trellis-coded modulations with multidimensional M-PSK”, *IEEE J. Select. Areas Commun.*, vol. SAC-7, no. 9, Dec. 1989.
- [40] R.G.C. Williams, “A trellis code for V.fast,” CCITT V.fast Rapporteur metting, Bath, U.K., Sep. 1992.
- [41] W. Zhang, “Finite-state machines in communications,” Ph.D. thesis, University of South Australia, Australia, 1995.
- [42] W. Zhang, C. Schlegel, and P. Alexander, “The BER reductions for systematic 8PSK trellis codes by a Gray scrambler,” Proceedings of the International Conference on Universal Wireless Access, Melbourne, Australia, April 1994.

Chapter 4

Trellis Representations

4.1 Preliminaries

While error control coding has traditionally largely been approach using finite-field algebra, this book concentrates on graphical descriptions and methods for these codes. These graphical representations lend themselves not only as visual aids, but also as blueprints for the algorithmic decoding methods discussed in Chapters 5, 6, and 8.

There are, in general, many different graphical representations of any given error control code, but some representations have distinct advantages. Among the most basic, and also richest graphical representations, are the trellises of codes discussed in this chapter. They will lead to famous optimal trellis decoding algorithms of Chapter 5 and are fundamental building blocks for the iterative decoding algorithms applied so successfully to turbo codes.

While we will not discuss algebraic code structures and decoding algorithm methods in any length in this book, we wish to point out that such approaches have been very successful in a number notable applications, such as compact disc (CD) storage, digital audio recording (DAT), and high-definition television (HDTV) [22]—primarily application where error correction is more central than error control.

Arguably, the single most important result which accounts for the use and popularity algebraic decoding are the finite-field methods for the class of Goppa codes [41], in particular the Berlekamp-Massey [4, 37] and Euclid's algorithm [54]. Reed-Solomon codes, and, to a lesser extent, Bose-Chaudhuri-Hocquenghem (BCH) codes, are the most popular subclasses of Goppa codes. These codes can be decoded very fast with a complexity on the order of $O(d^2)$, where d is the minimum distance of the code, using finite-field arithmetic which can be implemented efficiently in VLSI circuits. However, these efficient algebraic decoding algorithms only perform error correction, that is, the received noisy or corrupted symbols have to be mapped into the transmitted symbol alphabet before decoding. The

direct use of the received symbols, i.e., *soft-decision decoding*, cannot be used directly. This may explain why algebraic coding finds application predominantly in systems requiring error correction. Soft-decision decoding, referred to as error *control* decoding, can be done however, and several algorithms have been proposed [14, 62, 7, 39]. But these techniques have not found much attention by practitioners of error control coding. More recently, soft-decision decoding has seen a revival of interest [52, 20, 34, 18, 59, 60, 55, 57, 17], in particular in connection with the trellis complexity problem of codes addressed in this chapter.

In Chapter 5 we will discuss a number of powerful decoding techniques which make use of the trellis structure of codes. Unlike convolutional codes, algebraic codes have traditionally been constructed and decoded in purely algebraic ways, which do not lend themselves easily to soft-decision decoding extensions. However, with the trellis representations developed in this chapter, these codes can also be described by a trellis. Soft-decision decoding algorithms developed for trellis codes can therefore be applied. The algebraic picture of error correction codes are extensively discussed in a number of excellent reference books and others, [42, 33, 41, 5, 8].

4.2 The Parity-Check Matrix

One of the easiest ways to define a linear¹ code of rate $R = k/n$ is via its *parity-check matrix* $\mathbf{H} = [\mathbf{h}_1, \mathbf{h}_2, \dots, \mathbf{h}_n]$, which is an $n - k \times n$ matrix whose j th column is \mathbf{h}_j and has entries $h_{ij} \in \text{GF}(p)$, where $\text{GF}(p)$ is the finite field of the integers modulo p , and p is a prime. In the simplest case the code extends over $\text{GF}(2)$, i.e., over $\{0, 1\}$, and all operations are modulo 2, or EXOR and AND operations.

A linear code may now be characterized by the following

Definition 4.1 A linear code \mathcal{C} with parity check-matrix \mathbf{H} is the set of all n -tuples (vectors), or codewords \mathbf{x} , such that

$$\mathbf{H}\mathbf{x} = 0. \quad (4.1)$$

Algebraically, the codewords of \mathcal{C} lie in the (right) nullspace of \mathbf{H} .

For example, the family of single-error correcting binary Hamming codes have parity-check matrices \mathbf{H} whose columns are all the $2^m - 1$ non-zero binary vectors of length $m = n - k$. These codes have a rate $R = k/n = (2^m - m - 1)/(2^m - 1)$ and a minimum distance $d_{\min} = 3$ [42]. The first such code found by Richard Hamming in 1948 has the

¹We restrict our attention to linear codes since they are the largest and most important class of block codes, and there are no significant performance restrictions that result from this restriction.

parity-check matrix

$$\mathbf{H}_{[7,4]} = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 \end{bmatrix} = [\mathbf{h}_1 \mathbf{h}_2 \mathbf{h}_3 \mathbf{h}_4 \mathbf{h}_5 \mathbf{h}_6 \mathbf{h}_7]. \quad (4.2)$$

Note that each row of \mathbf{H} corresponds to a single parity-check equation.

Any parity check matrix can always be arranged such that $\mathbf{H} = [\mathbf{A} | \mathbf{I}_{n-k}]$ through column and row permutations and linear combinations of rows, where \mathbf{I}_{n-k} is the $n-k$ identity matrix and \mathbf{A} has dimensions $n-k \times k$. This has already been done in (4.2). From this form we can obtain the systematic $k \times n$ code generator matrix

$$\mathbf{G} = \left[\mathbf{I}_k \mid (-\mathbf{A}^T) \right] \quad (4.3)$$

through simple matrix manipulations. The code generator matrix has dimensions $k \times n$ and is used to generate directly the codewords \mathbf{x} via $\mathbf{x}^T = \mathbf{u}^T \mathbf{G}$, where \mathbf{u} is the information k -tuple. Algebraically, the codeword x lies in the row space of \mathbf{G} , since they are all linear combinations of the rows of \mathbf{G} .

4.3 Parity-Check Trellis Representations

In this section, we will use a trellis as a visual method of keeping track of the p^k codewords of a block code \mathcal{C} , and each distinct codeword will correspond to a distinct path through the trellis. This idea was first explored by Bahl, Cocke, Jelinek, and Raviv [3] in 1974, and later also by Wolf [64] and Massey [36] in 1978. Following the approach in [3, 64] we define the states of the trellis as follows: Let s_r be the label of the state at time r , then

$$s_{r+1} = s_r + x_{r+1} \mathbf{h}_{r+1} = \sum_{l=1}^r x_l \mathbf{h}_l, \quad (4.4)$$

where x_{r+1} runs through all permissible code symbols at time $r+1$. The state at the end of time interval $r+1$, s_{r+1} , is calculated from the preceding state s_r according to (4.4). If the states s_r and s_{r+1} are connected, they are joined in the trellis diagram by a branch labeled with the output symbol x_{r+1} which caused the connection. We see that (4.4) simply implements the parity-check equation (4.1) in a recursive fashion, since the final zero-state $s_{n+1} = \sum_{l=1}^n x_l \mathbf{h}_l = \mathbf{Hx}$ is the complete parity check equation! Since each intermediate state $s_r = \sum_{l=1}^r x_l \mathbf{h}_l$ is a vector of length $n-k$ with elements in $\text{GF}(p)$, also called the *partial syndrome* at time r , there can be at most p^{n-k} distinct states at time r , since this is the maximum number of distinct p -ary vectors of length $n-k$. We will see that often the number of states is significantly less than that.

At this point an example seems appropriate. Let us construct the trellis of the $[7, 4]$ Hamming code, whose parity check matrix is given by (4.2). The corresponding trellis is shown in Figure 4.1. The states s_r are labeled as ternary vectors $(\sigma_1, \sigma_2, \sigma_3)^T$, in accordance with (4.4). Note that all the path extensions which lead to states $s_{n+1} \neq (000)^T$ are dashed, since $s_n^{(i)}$ is the final syndrome and must equal $(000)^T$ for \mathbf{x} to be a codeword. We therefore force the trellis to terminate in the zero-state at time n .

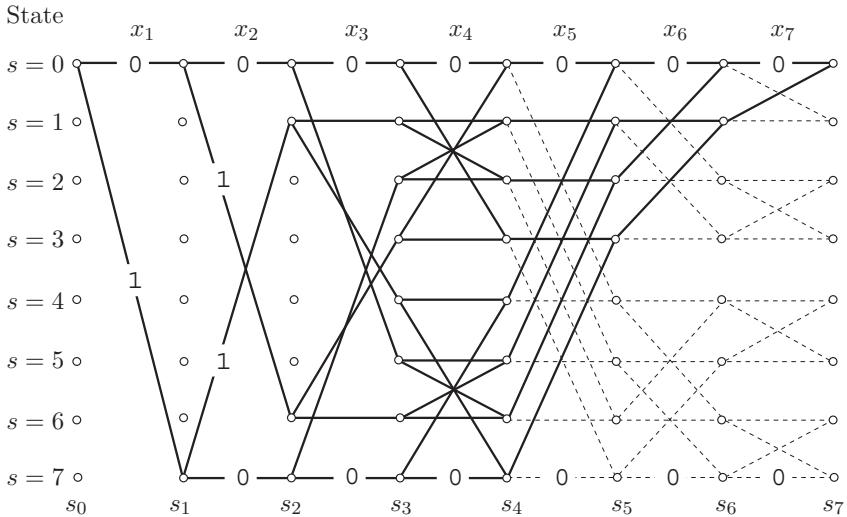


Figure 4.1: Trellis diagram of the $[7, 4]$ Hamming code. The labeling of the trellis is such that a “0” causes a horizontal transition and a “1” causes a sloped transition. This follows from (4.4) since $x_{r+1} = 0$ causes $s_{r+1} = s_r$. Hence, only a few transitions are labeled in the figure.

The trellis of a code represents the set of all codewords in the following sense: Each distinct path from the starting node to the final node is a different and distinct codeword. Codewords may share large portions of the trellis and only differ over short sections. This concept will be important when we discuss optimal decoding of codes on their trellis. The complexity of these decoding algorithms will be proportional to the number of branches and the number of states in the trellis. For codes whose maximum number of states is much less than the total number 2^k of codewords, this trellis-based decoding is very efficient.

4.4 Convolutional Codes and Their Trellis

In Chapter 4 we used finite-state machines as generators of our trellis, which then served as the matrix for the trellis code. This particular way of constructing the trellis is not only of historical importance, but it also allowed us to find all the structural theorems presented in Chapter 4. Later, in Chapter 5, we will prove information theoretic results which hold for trellis code generated in this way. If the mapping of the binary output bits is into the $\{-1, +1\}$ binary set of BPSK signals, we have the case of traditional convolutional codes. In fact, the trellis mapping function is basically omitted.

Convolutional codes have a long history. They were introduced by Elias [13] in 1955. Since then, much theory has evolved to understand them [12, 33, 8, 61].

Figure 4.2, which is the finite-state machine from Figure 3.1, is now a *convolutional encoder* which produces a *convolutional code*. This code maps a vector $\mathbf{u}_r = (u_r^{(2)}, u_r^{(1)})$ of input bits into a vector $\mathbf{v}_r = (v_r^{(2)}, v_r^{(1)}, v_r^{(0)})$ of output bits at time r . The rate $R = 2/3$ of this code is the ratio of the number of input bits to the number of output bits. If these output bits are mapped individually into BPSK signals, we quickly see that a convolutional code does not conserve the signal bandwidth, but requires $1/R$ times more bandwidth than uncoded transmission through rate expansion.

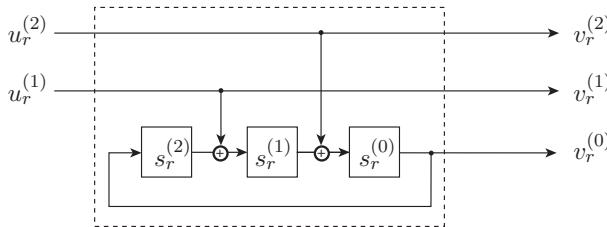


Figure 4.2: Rate $R = 2/3$ convolutional encoder which was used in Chapter 3, Figure 3.1, to generate an 8-PSK trellis code.

Convolutional codes have traditionally been used with BPSK or QPSK (Gray) mappings. In either case, due to the regularity of the mapper function, the Euclidean distance between two signal sequences depends only on the Hamming distance $H_d(\mathbf{v}^{(1)}, \mathbf{v}^{(2)})$ between the two output bit sequences (formerly branch label sequences), where the Hamming distance between two sequences is defined as the number of bit positions in which the two sequences differ. Furthermore, since the convolutional code is linear, we can choose any one of the sequences as the reference sequence, and we need to consider only the Hamming weights of $H_w(\mathbf{v}^{(1)} \oplus \mathbf{v}^{(2)}) = H_d(\mathbf{v}^{(1)} \oplus \mathbf{v}^{(2)}, 0) = H_d(\mathbf{v}^{(1)}, \mathbf{v}^{(2)})$.

It is interesting to note that the convolutional encoder in Figure 4.2 has an alternate “incarnation,” which is given in Figure 4.3 below. This form is called the controller canonical non-systematic form, the term stemming from the fact that inputs can be used to control the state of the encoder in a very direct way, in which the outputs have no influence. If no feedback is present, as in Figure 4.3, then the controller canonical form is commonly referred to as the non-systematic feedforward realization of the encoder. We will see later in this chapter that equivalent encoders, like the ones in Figures 4.2 and 4.3, generate the same code, although for a given input sequence they do not generate the same output sequence.

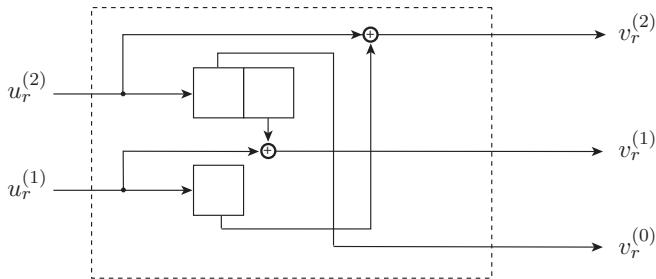


Figure 4.3: Rate $R = 2/3$ encoder from above in controller canonical non-systematic form.

Both encoders generate the same trellis, reproduced here from Chapter 3 and shown for a short length of $L = 5$ sections. One of the evident advantages of convolutional code trellises is that they can be of arbitrary length, and their state space is well-defined from a direct map from the generating finite-state machine of code.

We will show in Section 4.9 that every convolutional code has a minimal basic encoder of the form of Figure 4.3, an equivalent minimal systematic encoder (Figure 4.2). Minimal in this sense means that the number of states in the finite-state machine is equal to the number of states in the trellis—a fact which may seem obvious, but which is not in general the case.

Additionally, there are many other possible ways of representing the encoder, a particular one, the recursive systematic realization was made popular by the introduction of turbo codes. However, as shown later, in a certain sense only the two implementations discussed above are fundamental, and all others are easily reduced to one of these realizations. Note also that all encoder realizations generate the same code trellis, but that the labeling of the *information bits* to the different branches can change. This labeling, however, is important in the combinations of encoders, like the one used to generate the binary turbo codes in Chapter 8.

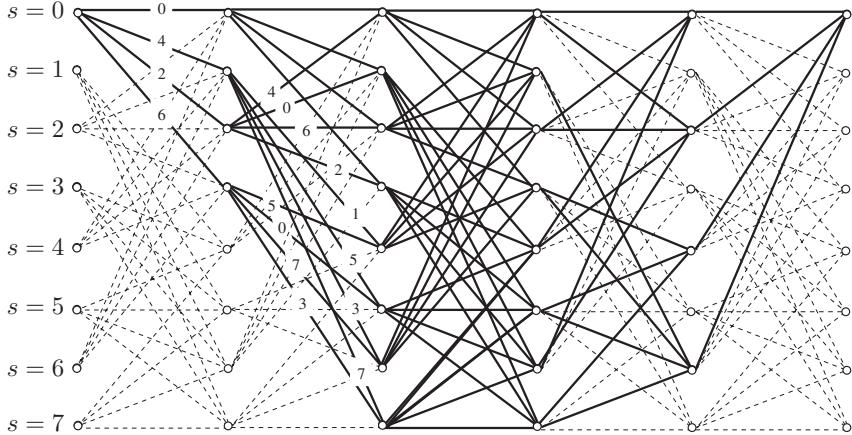


Figure 4.4: Trellis diagram for the encoders in Figure 4.2 and 4.3.

In order to illustrate these concepts, it is convenient to use the definition of the D -transform, given by

$$u(D) = \sum_{r=s}^{\infty} \mathbf{u}_r D^r, \quad (4.5)$$

where $s \in \mathbf{Z}$ guarantees that there are only finitely many negative terms in (4.5). The D -operator can be understood as a unit-delay operator, corresponding to passing a symbol through one delay element. Formally, (4.5) is a Laurent series with vector coefficients, which we may write as two (k in general) binary Laurent series

$$u(D) = (u^{(2)}(D), u^{(1)}(D)). \quad (4.6)$$

The binary Laurent series form a field² and division is understood to be evaluated

²That is, they possess all the field properties:

- (i) Closure under addition and multiplication.
- (ii) Every element $a(D)$ possesses an additive inverse $-a(D)$, and every element $a(D) \neq 0$ possesses a multiplicative inverse $1/a(D)$.
- (iii) The addition operation commutes: $a(D) + b(D) = b(D) + a(D)$, and the multiplication operation commutes also: $a(D)b(D) = b(D)a(D)$.
- (iv) There exists an additive unit element 0 such that $a(D) + 0 = a(D)$, and $a(D) + (-a(D)) = 0$, as well as a multiplicative unit element 1 such that $a(D) \cdot 1 = a(D)$, and $a(D)(1/a(D)) = 1$.
- (v) Multiplication distributes over addition: $a(D)(b(D) + c(D)) = a(D)b(D) + a(D)c(D)$.

by expanding the long division into positive exponents, e.g., $(1 + D)/(D^2 + D^3 + D^4) = D^{-2} + 1 + D + D^3 + \dots$. Arithmetic with Laurent series can now be carried out in the well-known conventional way, knowing that results from such computations are consistent with each other.

Applying this, let us then consider a rate $R = 1/2$ minimal basic encoder with the following encoding matrix

$$G_{FF}(D) = \begin{bmatrix} 1 + D^3 + D^4 & 1 + D + D^3 + D^4 \end{bmatrix} \quad (4.7)$$

which has the obvious non-systematic feedforward realization shown in Figure 4.5(a), also called the controller canonical form. The equivalent systematic, or observer canonical, encoder has the generator matrix

$$G_{FB}(D) = \begin{bmatrix} 1 & \frac{h_1(D)}{h_0(D)} \end{bmatrix} = \begin{bmatrix} 1 & \frac{1+D+D^3+D^4}{1+D^3+D^4} \end{bmatrix}. \quad (4.8)$$

with the systematic feedback realization shown in Figure 4.5(b).

As said above, a third realization, the recursive systematic realization based on the controller canonical form, has been used in the context of parallel and serial concatenated convolutional codes (turbo codes), and we now show how the recursive systematic realization can be derived from the systematic feedback encoder $G_{FB}(D)$. For a rate $R = 1/2$ systematic feedback encoder, we can write

$$\left(v^{(1)}(D), v^{(0)}(D) \right) = u^{(1)}(D) G_{FB}(D) = \left(u^{(1)}(D), u^{(1)}(D) \frac{h_1(D)}{h_0(D)} \right). \quad (4.9)$$

Introducing the auxiliary sequence

$$w(D) = \frac{u^{(1)}(D)}{h_0(D)} \quad (4.10)$$

and using $h_0(D) = 1 + D^3 + D^4$ for the selected example, we solve for

$$w(D) = (D^4 + D^3)w(D) + u^{(1)}(D) \quad (4.11)$$

The sequence $w(D)$ is now seen to be recursively generated by feeding it back to the input after $D = 3$ and $D = 4$ unit time delays and adding the input sequence $u^{(1)}(D)$ to it. This operation is implemented by the upper feedback part of the circuit in Figure 4.5(c). The lower part of the recursive systematic realization implements the multiplication by $h_1(D)$ to generate the sequence

$$v^{(0)}(D) = w(D)h_1(D) = \frac{u_0^{(1)}(D)}{h_0(D)}h_1(D), \quad (4.12)$$

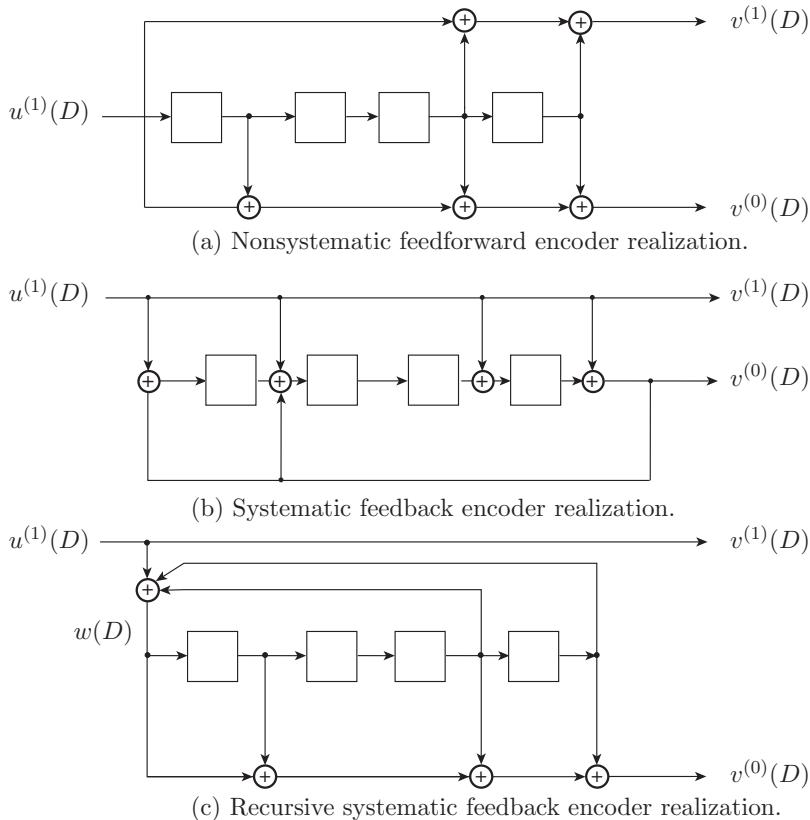


Figure 4.5: Three encoder realizations for a rate $R = 1/2$ convolutional code.

which, naturally, is identical to the one generated by the systematic feedback realization.

Since the systematic feedback and recursive systematic realizations are both based on the same encoder $G_{FB}(D)$, it is clear that they are not only equivalent, but identical. They implement the identical mapping from input sequences to output sequences. This then begs the question as to why one of these realizations would be preferred over the other. The answer lies in the fact that the recursive systematic realization is of the controller canonical form [28] and, as noted in Section 4.9, the state may be set by the input.

4.5 Minimal Trellises

In light of the importance the trellis plays for decoding the code, in particular the direct relationship between trellis complexity and the complexity of optimal decoding, we turn our attention now to the optimality of trellis representations. We can loosely define the *complexity of a trellis* as the number of branches in the trellis, which itself is strongly related to the maximum number of states, or the state-space dimension of the trellis. As we have seen, the complexity of the trellis of convolutional codes is directly related to the state-space of their finite-state machine implementations, and we will discuss how to minimize the size of that state space later in this chapter. For general codes, however, it is not immediately clear how complex their trellis representations are.

Following the results of McEliece [40] we will now discuss bounds on the state space of the trellis as well as its branch space and show that the parity-check trellis representation (or *PC-trellis*) from Section 4.3 is the best possible in the sense that it minimizes the number of states as well as the number of edges in the trellis for each time instant r . Since the number of edges is the relevant measure of complexity for the decoding algorithm (Theorem 7.2), the PC-trellis also minimizes decoding complexity.

From (4.4) we know that the set of states at time r , denoted by \mathcal{S}_r , is given by the mapping $\mathcal{C} \mapsto \mathcal{S}_r$

$$\{\mathbf{s}_r(\mathbf{x}) : \mathbf{s}_r(\mathbf{x}) = x_1\mathbf{h}_1 + \cdots + x_r\mathbf{h}_r\}. \quad (4.13)$$

This mapping is linear due to the linearity of the code and is in general not one-to-one. It can be viewed as a vector space of dimension $n - k$, since this is the dimension of the vectors $\mathbf{s}_r(\mathbf{x})$, but is, in general, much less (see, e.g., Figure 4.1). We denote its dimension by σ_r , i.e., $\sigma_r = \dim \mathcal{S}_r$. Note that in the case of binary codes, the dimension σ_r of the states space implies that the number of states $|\mathcal{S}_r|$ in the trellis at time r is 2^{σ_r} .

Likewise, an edge, or branch, in the trellis is specified by the state from where it originates, the state into which it leads, and the symbol attached to the branch. Formally we may describe the edge space $\mathcal{B}_{r,r+1}$ at time $r + 1$ by the triple

$$\{\mathbf{b}_{r,r+1}(\mathbf{x}) = (\mathbf{s}_r(\mathbf{x}), \mathbf{s}_{r+1}(\mathbf{x}), x_{r+1})\}. \quad (4.14)$$

Clearly, the edge mapping is also linear, and we denote its dimension by $\beta_{r,r+1}$.

Now, since codewords are formed via $\mathbf{x} = \mathbf{G}^T \mathbf{u}$, we have

$$\mathbf{s}_r(\mathbf{x}) = \mathbf{H}_r(x_1, \dots, x_r) = \mathbf{H}_r \mathbf{G}_r^T \mathbf{u}, \quad (4.15)$$

where \mathbf{H}_r and \mathbf{G}_r are the truncated matrices made up of the first r columns of \mathbf{H} and \mathbf{G} , respectively. So, $\mathbf{H}_r \mathbf{G}_r^T$ is a $n - k \times k$ matrix, and we have the following lemma:

Lemma 4.1 *The state space \mathcal{S}_r at the completion of time r is the column space of $\mathbf{H}_r \mathbf{G}_r^T$, and, consequently, its dimension σ_r is the rank of the matrix $\mathbf{H}_r \mathbf{G}_r^T$.*

Note that we can of course span the trellis up backwards from the end by changing the limits of the sum to $l = r + 1$ to n in (4.4), and then

$$\mathbf{s}_r(\mathbf{x}) = \overline{\mathbf{H}}_r(x_{r+1}, \dots, x_n)^T = \overline{\mathbf{H}}_r \overline{\mathbf{G}}_r^T \mathbf{u}, \quad (4.16)$$

where $\overline{\mathbf{H}}_r$ and $\overline{\mathbf{G}}_r$ are matrices made up of the last $n - r$ columns of \mathbf{H} and \mathbf{G} , respectively. The dimensions of $\overline{\mathbf{H}}_r \overline{\mathbf{G}}_r^T$ are also $n - k \times k$ matrix. Using the fact that the rank of a matrix is smaller or equal to its smallest dimension and that the rank of the product of two matrices is equal to or smaller than the rank of each factor matrix, we conclude from (4.15), (4.16), and the definitions of \mathbf{H}_r , \mathbf{G}_r , $\overline{\mathbf{H}}_r$, and $\overline{\mathbf{G}}_r$

Lemma 4.2 *The dimension σ_r of the state space \mathcal{S}_r at the end of time r is bounded by*

$$\sigma_r \leq \min(r, n - r, k, n - k). \quad (4.17)$$

Lemma 4.2 can be seen nicely in Figure 4.1.

Now, consider all the codewords or code sequences for which $x_j = 0; j > r$. These sequences are called the past subcode, denoted by P_r , and are illustrated in Figure 4.6 for $r = 5$ for the PC-trellis for the [7, 4] Hamming code. We denote the dimension of P_r by p_r . Likewise, we define the future subcode F_r as the set of code sequences for which $x_j = 0; j \leq r$, and denote its dimension by f_r . The future subcode F_2 for the [7, 4] Hamming code is also shown in Figure 4.6. Clearly, both of these subcodes are linear subcodes of the original code, i.e., they are subgroups of the original group (code) \mathcal{C} .

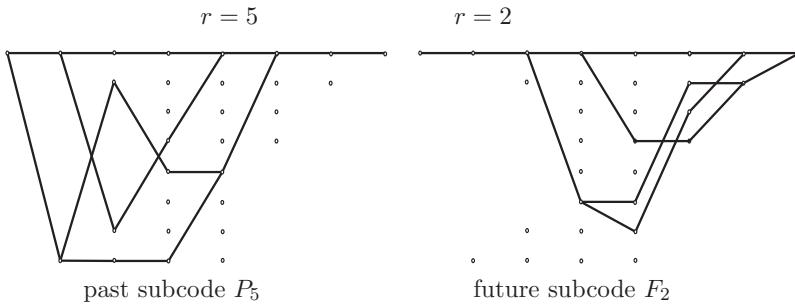


Figure 4.6: Trellis diagram of the past subcode P_5 and the future subcode F_2 of the [7, 4] Hamming code whose full trellis is shown in Figure 4.1.

Now, mathematically, $\mathbf{s}_r(\mathbf{x})$ is a linear map $\mathcal{C} \mapsto \mathcal{S}_r$ from the code space \mathcal{C} , which is k -dimensional, into the state space \mathcal{S}_r at time r , which has dimension σ_r . Such a linear map

has a kernel,³ which is the set of code words which are mapped into the state $\mathbf{s}_r(\mathbf{x}) = 0$ (i.e., all the code words whose trellis paths pass through the zero-state at time r). From Figure 4.6 we now guess the following

Lemma 4.3 *The kernel of the map $\mathbf{s}_r(\mathbf{x})$ is the sum of the past and future subcodes P_r and F_r , i.e.,*

$$\text{Ker}(\mathbf{s}_r) = P_r \oplus F_r. \quad (4.18)$$

The kernel $\text{Ker}(\mathbf{s}_r)$ is illustrated in Figure 4.7.

Proof: If $\mathbf{x} \in P_r \oplus F_r$, it can be expressed as $\mathbf{x} = \mathbf{x}^{(p)} + \mathbf{x}^{(f)}$, where $\mathbf{x}^{(p)} \in P_r$ and $\mathbf{x}^{(f)} \in F_r$. But since $\mathbf{x}^{(p)} \in C$, $\mathbf{H}\mathbf{x}^{(p)} = 0$, and hence $\mathbf{H}_r(x_1, \dots, x_r) = \mathbf{s}_r(\mathbf{x}^{(p)}) = 0$. Trivially, $\mathbf{s}_r(\mathbf{x}^{(f)}) = 0$, and since the $\mathcal{C} \mapsto \mathcal{S}_r$ is linear, we conclude $\mathbf{s}_r(\mathbf{x}) = \mathbf{s}_r(\mathbf{x}^{(p)} + \mathbf{x}^{(f)}) = 0$ and $P_r \oplus F_r \subseteq \text{Ker}(\mathbf{s}_r)$.

Conversely, assume $\mathbf{s}_r(\mathbf{x}) = 0$ and let $\mathbf{x} = \mathbf{x}^{(p)} + \mathbf{x}^{(f)}$ again, where we choose $\mathbf{x}^{(p)} = (x_1, \dots, x_r, 0, \dots, 0)$ and $\mathbf{x}^{(f)} = (0, \dots, 0, x_{r+1}, \dots, x_n)$. But due to $\mathbf{s}_r(\mathbf{x}) = 0$ and (4.15), we conclude that $\mathbf{H}\mathbf{x}^{(p)} = 0$ and therefore $\mathbf{x}^{(p)} \in \mathcal{C}$ and $\mathbf{x}^{(p)} \in P_r$. Likewise, applying (4.16) gives $\mathbf{H}\mathbf{x}^{(f)} = 0$ and $\mathbf{x}^{(f)} \in \mathcal{C}$ and $\mathbf{x}^{(f)} \in F_r$. Hence $\text{Ker}(\mathbf{s}_r) \subseteq P_r \oplus F_r$ also. Q.E.D.

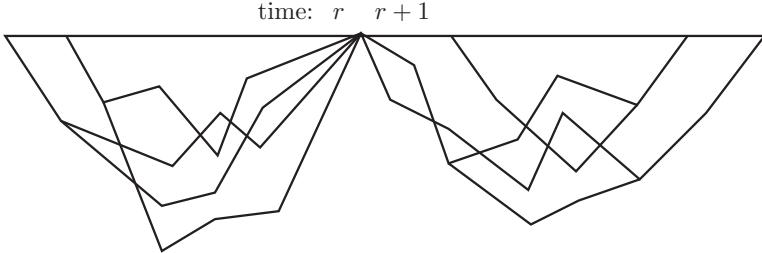


Figure 4.7: Illustration of the kernel of the map $\mathcal{C} \mapsto \mathcal{S}_r$ as the set of code words which pass through $\mathbf{s}_r(\mathbf{x}) = 0$.

As is the case with $\mathbf{s}_r(\mathbf{x})$, $\mathbf{b}_{r,r+1}(\mathbf{x})$ is a map from the code space \mathcal{C} into the $\beta_{r,r+1}$ -dimensional edge space $\mathcal{B}_{r,r+1}$. Its kernel is given by the following lemma:

Lemma 4.4 *The kernel of the map $\mathbf{b}_{r,r+1}(\mathbf{x})$ is the sum of the past and future subcodes P_r and F_{r+1} , i.e.,*

$$\text{Ker}(\mathbf{b}_{r,r+1}) = P_r \oplus F_{r+1}. \quad (4.19)$$

³The kernel of a linear map is the subset of point which map to zero under the map. In matrix algebraic formulation the kernel is the nullspace.

Proof: From the definition of the edge mapping (4.14) we see that $\text{Ker}(\mathbf{b}_{r,r+1})$ must be contained in $\text{Ker}(\mathbf{s}_r)$ and $\text{Ker}(\mathbf{s}_{r+1})$, as well as obey the condition $x_{r+1} = 0$. From this the lemma follows. Q.E.D.

The following theorem then determines the number of states and edges in the PC-trellis.

Theorem 4.5 *The number of states at depth r in the PC-trellis for a binary code is*

$$|\mathcal{S}_r| = 2^{k-p_r-f_r}, \quad (4.20)$$

and the number of edges at depth r is given by

$$|\mathcal{B}_{r,r+1}| = 2^{k-p_r-f_{r+1}}. \quad (4.21)$$

Proof: Since s_r is a linear map from $\mathcal{C} \rightarrow \mathcal{S}_r$, we may apply the rank theorem of linear algebra [53], i.e.,

$$\dim \text{Ker}(s_r) + \dim |\mathcal{S}_r| = \dim \mathcal{C}, \quad (4.22)$$

which leads to $p_r + f_r + \sigma_r = k$ and hence to (4.20). The proof of (4.21) is analogous. Q.E.D.

We now come to the heart of this section. The optimality of the PC-trellis is asserted by the following theorem:

Theorem 4.6 *The number of states $|\mathcal{S}_r|$ and the number of edges $|\mathcal{B}_{r,r+1}|$ at depth r in any trellis which represents the linear block code \mathcal{C} are bounded by*

$$|\mathcal{S}_r| \geq 2^{k-p_r-f_r}, \quad (4.23)$$

$$|\mathcal{B}_{r,r+1}| \geq 2^{k-p_r-f_{r+1}}. \quad (4.24)$$

Since the PC-trellis achieves the inequalities with equality, it simultaneously minimizes both the state and the edge count at every depth r .

Proof: The proof relies on the linearity of the code. Assume that T is a trellis which represents \mathcal{C} , and $\mathbf{s}_r(\mathbf{x})$ is a state at depth r in this trellis. Now let \mathcal{C}_{s_r} be the subset of codewords which pass through \mathbf{s}_r . Since every codeword must pass through at least one state at time r ,

$$\mathcal{C} = \bigcup_{\mathbf{s}_r} \mathcal{C}_{s_r}. \quad (4.25)$$

Now consider the codewords $\mathbf{x} = [\mathbf{x}_p \mathbf{x}_f]^T$ in \mathcal{C}_{s_r} , where $\mathbf{x}_p = (x_1, \dots, x_r)$ is the past portion of \mathbf{x} , and $\mathbf{x}_f = (x_{r+1}, \dots, x_n)$ is the future portion. Then let $\mathbf{x}^{(1)} = [\mathbf{x}_p^{(1)} \mathbf{x}_f^{(1)}]^T$ be a particular codeword in \mathcal{C}_{s_r} , e.g., the one with minimum Hamming weight.

Now, both codewords \mathbf{x} and $\mathbf{x}^{(1)}$ pass through s_r . Therefore $\mathbf{x}' = \left[\mathbf{x}_p \mathbf{x}_f^{(1)} \right]^T$ and $\mathbf{x}'' = \left[\mathbf{x}_p^{(1)} \mathbf{x}_f \right]^T$ are two codewords which also pass through the state s_r . Hence,

$$\mathbf{x}' - \mathbf{x} = \begin{bmatrix} 0 \\ \mathbf{x}_f^{(1)} - \mathbf{x}_f \end{bmatrix} \quad (4.26)$$

is in the future code F_r , and

$$\mathbf{x}'' - \mathbf{x} = \begin{bmatrix} \mathbf{x}_p^{(1)} - \mathbf{x}_p \\ 0 \end{bmatrix} \quad (4.27)$$

is in the past code P_r . We conclude that the difference of the original two codewords is

$$\mathbf{x}^{(1)} - \mathbf{x} = \begin{bmatrix} \mathbf{x}_p^{(1)} - \mathbf{x}_p \\ \mathbf{x}_f^{(1)} - \mathbf{x}_f \end{bmatrix} = \mathbf{x}' + \mathbf{x}'' - 2\mathbf{x} \quad (4.28)$$

and that therefore we have

$$\Rightarrow \mathbf{x} = \underbrace{\mathbf{x}' + \mathbf{x}''}_{\text{is in } P_r \oplus F_r} - \mathbf{x}^{(1)}. \quad (4.29)$$

This means that every codeword \mathbf{x} is generated from $\mathbf{x}^{(1)}$ through the addition of $\mathbf{x}' + \mathbf{x}'' \in P_r \oplus F_r$. Hence, \mathbf{x} is in the coset $P_r \oplus F_r - \mathbf{x}^{(1)}$. But the size of this coset is $|P_r \oplus F_r|$, and therefore $|\mathcal{C}_{s_r}| \leq 2^{p_r + f_r}$, which, together with $|\mathcal{C}| = 2^k$, immediately implies (4.23).

The proof of the second part of the theorem is analogous.

Q.E.D.

4.6 Minimum-Span Generator Matrices

The dimensions p_r and f_r can be found by inspecting the trellis, which, however, is a rather cumbersome undertaking. In addition to this, once the trellis is constructed, the number of states and branches at each time are known, and there is no need for the values p_r and f_r anymore. There is, however, a simpler way to obtain the dimensions p_r and f_r [40]. Let us consider the example of the [7, 4] Hamming code from the last section again, whose generator matrix is given by

$$\mathbf{G}_{[7,4]} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}. \quad (4.30)$$

By adding row #3 to row #1, then row #2 to row #1, then row #4 to row #3, and finally row #3 to row #2, we obtain the following sequence of equivalent generator matrices (leading and trailing zeros are not shown for better readability)

$$\begin{aligned} \mathbf{G}_{[7,4]} &\equiv \begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 1 \\ & 1 & 0 & 0 & 1 & 1 \\ & & 1 & 0 & 1 & 0 \\ & & & 1 & 0 & 1 \end{bmatrix} \equiv \begin{bmatrix} 1 & 1 & 1 & 0 & 1 \\ & 1 & 0 & 0 & 1 & 1 \\ & & 1 & 0 & 1 & 0 \\ & & & 1 & 0 & 1 \end{bmatrix} \\ &\equiv \begin{bmatrix} 1 & 1 & 1 & 0 & 1 \\ & 1 & 0 & 0 & 1 & 1 \\ & & 1 & 1 & 1 & 1 \\ & & & 1 & 0 & 1 \end{bmatrix} \equiv \begin{bmatrix} 1 & 1 & 1 & 0 & 1 \\ & 1 & 1 & 1 & 1 \\ & & 1 & 1 & 1 & 1 \\ & & & 1 & 0 & 1 \end{bmatrix}. \end{aligned} \quad (4.31)$$

What we have generated is an equivalent minimum span generator matrix (MSGM) [40] for the $[7, 4]$ Hamming code, where the span of a matrix is defined as the sum of the spans of the rows, and the span of a row is defined as its length without the trailing and leading zeros. More precisely, the span of a row vector \mathbf{x} is defined as $\text{Span}(\mathbf{x}) = R(\mathbf{x}) - L(\mathbf{x})$, where $R(\mathbf{x})$ is the index of the rightmost non-zero entry of \mathbf{x} , and $L(\mathbf{x})$ is the index of the leftmost non-zero entry of \mathbf{x} . An MSGM is a generator matrix for \mathcal{C} with minimum total span.

An MSGM can be obtained from an arbitrary generator matrix via the following simple algorithm:

Step 1: Find a pair of rows $\mathbf{x}^{(i)}$ and $\mathbf{x}^{(j)}$ in the generator matrix G such that $L(\mathbf{x}^{(i)}) = L(\mathbf{x}^{(j)})$ and $R(\mathbf{x}^{(i)}) \leq R(\mathbf{x}^{(j)})$, or $R(\mathbf{x}^{(i)}) = R(\mathbf{x}^{(j)})$ and $L(\mathbf{x}^{(i)}) \geq L(\mathbf{x}^{(j)})$.

Step 2: If **Step 1** fails and no such pair can be found go to **Step 4**.

Step 3: Let $\mathbf{x}^{(i)} = \mathbf{x}^{(i)} + \mathbf{x}^{(j)}$, i.e., replace the row $\mathbf{x}^{(i)}$ by the sum of the two rows. Go to **Step 1**.

Step 4: Output G , which is now a MSGM.

We will now prove the following:

Theorem 4.7 *The above algorithm always generates an MSGM.*

Proof: It is obvious from Step 3 in the algorithm that at each iteration the total span is reduced by one. The algorithm must therefore terminate in a finite number of steps, and it stops exactly when

$$\begin{aligned} L(\mathbf{x}^{(i)}) &\neq L(\mathbf{x}^{(j)}), \\ R(\mathbf{x}^{(i)}) &\neq R(\mathbf{x}^{(j)}) \end{aligned} \quad (4.32)$$

for all rows $\mathbf{x}^{(i)}, \mathbf{x}^{(j)}; \mathbf{x}^{(i)} \neq \mathbf{x}^{(j)}$ of G .

We now need to show that no other generator matrix G' can have smaller span than the one constructed above, which we denote by G . To this end, let $\mathbf{x}^{(1)'}, \dots, \mathbf{x}^{(k)'}$ be the rows of G' . Then, since G and G' are equivalent,

$$\mathbf{x}^{(j)'} = \sum_{\mathbf{x}^{(i)} \in \mathcal{I}_j} \mathbf{x}^{(i)}, \quad (4.33)$$

for every j , where \mathcal{I}_j is some subset of the set of rows of G . But due to (4.32) the sum in (4.33) can produce no cancellations at the endpoints of any member of \mathcal{I}_j , and therefore

$$\text{Span}(\mathbf{x}^{(j)'}) = \max_{\mathbf{x}^{(i)} \in \mathcal{I}_j} R(\mathbf{x}^{(i)}) - \min_{\mathbf{x}^{(i)} \in \mathcal{I}_j} L(\mathbf{x}^{(i)}) \geq \max_{\mathbf{x}^{(i)} \in \mathcal{I}_j} \text{Span}(\mathbf{x}^{(i)}). \quad (4.34)$$

But the k vectors $\mathbf{x}^{(j)'}$ must contain all $\mathbf{x}^{(i)}$'s, since otherwise G' has dimension less than k and cannot generate \mathcal{C} . Since every $\mathbf{x}^{(i)}$ is represented in at least one set \mathcal{I}_j , there exists an ordering of the indices i such that

$$\text{Span}(\mathbf{x}^{(j)'}) \geq \text{Span}(\mathbf{x}^{(i)}) \quad (4.35)$$

for every j , and hence also

$$\sum_{j=1}^k \text{Span}(\mathbf{x}^{(j)'}) \geq \sum_{i=1}^k \text{Span}(\mathbf{x}^{(i)}), \quad (4.36)$$

which proves the theorem

Q.E.D.

We have actually proven that the matrix G has the smallest possible span for every row (up to row permutations), i.e., it has the smallest span set. From this we immediately deduce the following

Corollary 4.8 *All MSGM's of a given code \mathcal{C} have the same span set.*

The significance of a MSGM lies in the fact that the dimension p_r of the past subcode P_r and the dimension f_r of the future subcode F_r , respectively, can be read off the generator matrix as explained by the following theorem.

Theorem 4.9 *Given an MSGM G , we obtain*

$$p_r = |i : R(\mathbf{x}^{(i)}) \leq r|, \quad (4.37)$$

i.e., p_r equals the number of rows in G for which the rightmost non-zero entry is at position r or before, and

$$f_r = |i : L(\mathbf{x}^{(i)}) \geq r + 1|, \quad (4.38)$$

i.e., f_r equals the number of rows in G for which the leftmost non-zero entry is at position $r + 1$ or later.

Proof: p_r is the dimension of the past subcode P_r , i.e., the set of all code words $\in \mathcal{C}$ which merge with the zero-state at position r or earlier. The rows of G are a basis for the code \mathcal{C} ; hence the rows of G which merge at time r or earlier, which are all independent, can be used as a basis for P_r . Due to (4.32), no other row, or linear combination of rows, can be $\in P_r$, since then a codeword would have non-zero entries x_j , for $j > r$, and therefore every codeword in P_r is a linear combination of said rows. There are exactly $|i : R(\mathbf{x}^{(i)}) \leq r|$ qualifying rows. This, together with Corollary 4.8, proves (4.37).

Analogously, the rows of G which start at position $r+1$ or later generate the subcode F_r . Since there are $|i : L(\mathbf{x}^{(i)}) \geq r+1|$ independent rows with that property is proven in (4.38). Q.E.D.

We summarize that the PC-trellis is optimal in that it simultaneously reduces the number of states and edges in its trellis representation of a linear block code \mathcal{C} , and, hence, would logically be the trellis of choice. It remains to point out that, while the PC-trellis is optimal for a given code, bit position permutations in a code, which generate equivalent codes, may bring further benefits. The problem of minimizing trellis complexity allowing such bit permutations is addressed in [25, 23, 24].

We now use Theorem 4.37 to read the values of p_r and f_r off the MSGM $\mathbf{G}_{[7,4]}$, given in (4.31). This produces the following table:

r	0	1	2	3	4	5	6	7
p_r	0	0	0	0	1	2	3	4
f_r	4	3	2	1	0	0	0	0

From this table we calculate $|\mathcal{S}_r| = 2^{k-f_r-p_r}$ and $|\mathcal{B}_{r,r+1}| = 2^{k-f_{r+1}-p_r}$, given by

r	0	1	2	3	4	5	6	7
$ \mathcal{S}_r $	1	2	4	8	8	4	2	1
$ \mathcal{B}_{r,r+1} $	2	4	8	16	8	4	2	-

These values correspond exactly with those in Figure 4.1, where we have constructed the trellis explicitly.

4.7 Systematic Construction of the PC-Trellis

The construction of the PC-trellis in Section 4.3 was relatively simple because the code was systematic and we did not have to deal with a large code. In this section we discuss a general method to construct the PC-trellis from a MSGM of a block code. This method was presented by McEliece in [40]. We will use the fact that the rows of G form bases for

P_r and F_r according to Theorems 4.9 and 4.5, i.e., $|\mathcal{S}_r| = 2^{k-p_r-f_r}$. Let us start with an example and consider the MSGM G for the extended [8, 4] Hamming code, given by

$$\mathbf{G}_{[8,4]} = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}. \quad (4.39)$$

At $r = 0$ we have a single starting state, i.e., $|\mathcal{S}_0| = 1$. By inspection we see that from $r = 1$ to $r = 3$, the first row $\mathbf{x}^{(1)} = (11110000)$ is active. By that we mean that any codeword which contains $\mathbf{x}^{(1)}$ will be affected by it in positions $r = 1$ to $r = 3 + 1$. Likewise, the second row $\mathbf{x}^{(2)}$ is active for $r \in [2, 6]$, $\mathbf{x}^{(3)}$ is active for $r \in [3, 5]$, and $\mathbf{x}^{(4)}$ is active for $r \in [5, 7]$. The basic idea of the trellis construction is that each row in the MSGM is needed as a basis only where it is active, and each active row doubles the number of states. We will now formalize this more precisely. Let w_r be a row vector with dimension σ_r whose i th entry is the i th active row. These vectors are given for $\mathbf{G}_{[8,4]}$ above as

$$\begin{array}{cccccccccc} \mathbf{w}_0 & \mathbf{w}_1 & \mathbf{w}_2 & \mathbf{w}_3 & \mathbf{w}_4 & \mathbf{w}_5 & \mathbf{w}_6 & \mathbf{w}_7 & \mathbf{w}_8 \\ \hline - & [w^{(1)}] & \left[\begin{matrix} w^{(1)} \\ w^{(2)} \end{matrix} \right]^T & \left[\begin{matrix} w^{(1)} \\ w^{(2)} \\ w^{(3)} \end{matrix} \right]^T & \left[\begin{matrix} w^{(2)} \\ w^{(3)} \end{matrix} \right]^T & \left[\begin{matrix} w^{(2)} \\ w^{(3)} \\ w^{(4)} \end{matrix} \right]^T & \left[\begin{matrix} w^{(2)} \\ w^{(4)} \end{matrix} \right]^T & [w^{(4)}] & - \end{array}$$

We now let the states \mathcal{S}_r be the vectors $\{\mathbf{w}_r\}$ with $w_r^{(i)} \in \{0, 1\}$. This gives the states of the trellis in Figure 4.8. Two states in this trellis are connected if either $\mathbf{w}_r \in \mathbf{w}_{r+1}$ or $\mathbf{w}_{r+1} \in \mathbf{w}_r$, e.g., $\mathbf{w}_2 = (01)$ and $\mathbf{w}_3 = (011)$ are connected since $\mathbf{w}_2 = (w^{(1)} = 0, w^{(2)} = 1)^T$ is contained in $\mathbf{w}_3 = (w^{(1)} = 0, w^{(2)} = 1, w^{(3)} = 1)$.

The trellis branch labels into state w_{r+1} are determined by the active rows in the two states at time r and $r + 1$. More precisely, let us pad the state vectors to equal length with zeros where needed; for example, $\tilde{\mathbf{w}}_6 = [0, w^{(2)}, 0, w^{(4)}]$. Now, the branch label is computed as

$$x_r = [\tilde{\mathbf{w}}_r \text{ OR } \tilde{\mathbf{w}}_{r+1}] \cdot \mathbf{g}_r^T, \quad (4.40)$$

where OR is the bit-wise logic OR operation, and \mathbf{g}_r is the r th column in the generator matrix. For instance, x_4 on the transition $(111) \rightarrow (11)$ is given by $(w^{(1)}, w^{(2)}, w^{(3)}) \cdot \mathbf{g}_4^T = (1110) \cdot (1110)^T = 1$. Since the state vector \mathbf{w}_r indicates which active rows are present in the codewords which pass through \mathbf{w}_r , it should be easy to see that Equation (4.40) simply equals the symbol which these active rows generate at time r .

Now note that the states at time $r = 3, r = 5$, and $r = 7$ are redundant, since there is only one branch leaving these states. We can therefore skip all odd-numbered states and combine pairs of branches into single branches with two attached symbols. This results

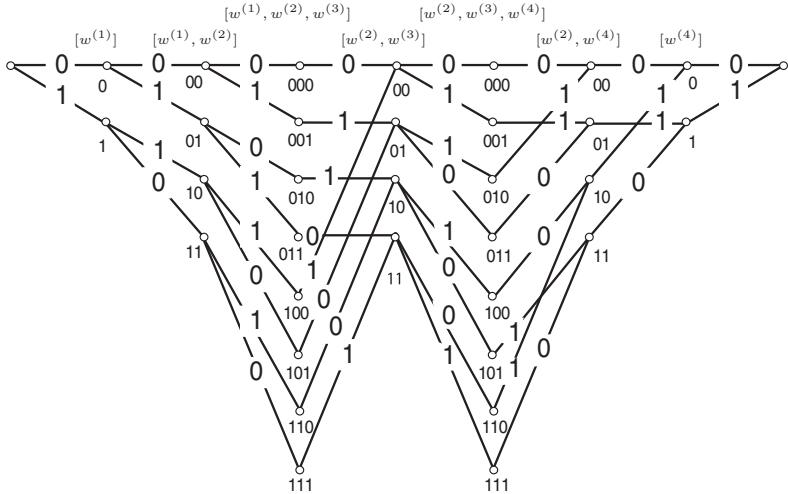


Figure 4.8: Trellis of $[8, 4]$ extended Hamming code, constructed in a systematic way.

in a new trellis that has a maximum number of states which is only four, i.e., half that of the original PC-trellis. This new trellis is shown in Figure 4.9, and we will construct this trellis again in Section 4.14 in our discussion on Reed–Muller codes. It becomes evident that there are many ways of graphically representing a code, and it is not always straightforward to determine which is the best method.

4.8 Tail-Biting Trellises

The trellises we have considered so far have a well-defined time axis starting at some time $r = 0$ and terminating at some time $r = n; n > 0$. In this section we consider the case of a *tail-biting* trellis, in which the index axis is circular, and the trellis “wraps” around from the end to the beginning. This new view introduces additional degrees of freedom in the design of the trellis for a given code, and often results in trellises which have significantly smaller complexities than their conventional counterparts. In fact, the *square root bound* [63], which we will derive later, asserts that the number of states in a tail-biting trellis could be as low as the square root of the number of states in the conventional trellis at its midpoint.

The procedure to construct a tail-biting trellis is exactly analogous to the construction

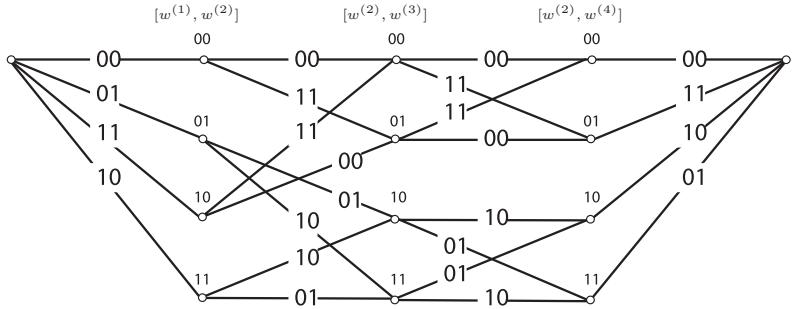


Figure 4.9: Contracted PC-trellis for the $[8, 4]$ extended Hamming code, obtained by taking pairs of symbols as branches.

of the PC-trellis discussed in the previous section; we only need to extend a few definitions and generalize the construction procedure. The procedure again starts with the generator matrix of a code with the only extension that the active span may wrap around from end to beginning. Consider the following slightly altered generator matrix for the extended $[8, 4]$ Hamming code, given by

$$\mathbf{G}'_{[8,4]} = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \end{bmatrix}, \quad (4.41)$$

which can be constructed by moving the fourth row from (4.39) into third place and constructing the fourth row in (4.41) by adding rows two and three. The reason for this transformation lies in the fact the this will allow us to construct, in fact, a minimal tail-biting trellis.

The beginning of activity of a row is now arbitrary, and may start at any non-zero element and extend, possibly wrapping around, to cover the last non-zero element. For the following we define the spans in (4.41) as indicated below by the stared entries:

$$\mathbf{G}'_{[8,4]} = \begin{bmatrix} \star & \star & \star & \star \\ & \star & \star & \star & \star \\ & & \star & \star & \star & \star \\ \star & \star & \star & & \star & \star & \star \end{bmatrix}; \quad (4.42)$$

that is, the activity spans for the four rows of $\mathbf{G}'_{[8,4]}$ are $[1, 3]$, $[3, 5]$, $[5, 7]$, and $[6, 2]$, which wraps around. The state-space vectors w_r for $\mathbf{G}'_{[8,4]}$ are then given by

$$\begin{array}{cccccccccc}
 w_0 = w_8 & w_1 & w_2 & w_3 & w_4 & w_5 & w_6 & w_7 & w_8 \\
 \hline
 [w^{(4)}]^T & \left[\begin{matrix} w^{(1)} \\ w^{(4)} \end{matrix} \right]^T & \left[\begin{matrix} w^{(1)} \\ w^{(4)} \end{matrix} \right]^T & \left[\begin{matrix} w^{(1)} \\ w^{(2)} \end{matrix} \right]^T & \left[\begin{matrix} w^{(2)} \end{matrix} \right]^T & \left[\begin{matrix} w^{(2)} \\ w^{(3)} \end{matrix} \right]^T & \left[\begin{matrix} w^{(3)} \\ w^{(4)} \end{matrix} \right]^T & \left[\begin{matrix} w^{(3)} \\ w^{(4)} \end{matrix} \right]^T & \left[\begin{matrix} w^{(4)} \end{matrix} \right]^T
 \end{array}$$

This leads to the following tail-biting trellis with maximum width four, which is smaller than the minimal conventional trellis for this code constructed in the last section:

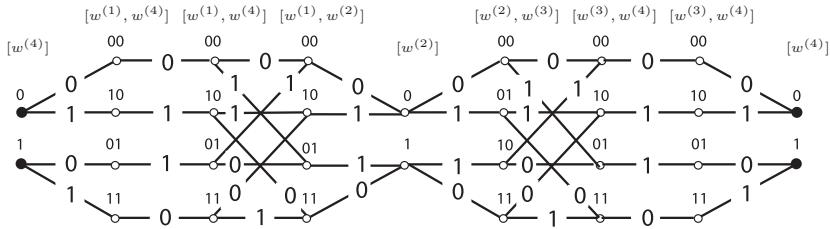


Figure 4.10: Tail-biting trellis for the [8, 4] extended Hamming code.

There exist exactly two non-isomorphic such minimal trellises for the [8,4] extended Hamming code [6]. Note that the states marked in black are those that "wrap around"; that is, a codeword ending in one of them must also start in the same state at the beginning of the trellis.

Similar to the case of the conventional trellis, we may contract states to generate a shorter trellis. For example, we may absorb the states at time $t = 1, 3, 6, 7$, since they are mere transition states and extend the branch labels to produce the trellis in Figure 4.11. This trellis has variable numbers of symbols on the branch labels, but it can be seen easily that if we reverse the second 4-symbol section in the trellis in Figure 4.10, a contracted trellis with two branch symbols on every branch results. Of course this new trellis describes a code which is equivalent to the original code only up those symbol permutations.

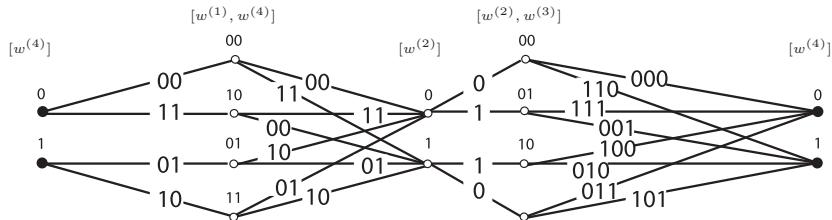


Figure 4.11: Contracted tail-biting trellis for the [8, 4] extended Hamming code.

We will now proceed to bring forth a strong argument that ascertains that the complexity of a tail-biting trellis for a given code can be substantially smaller than the complexity of the minimal PC-trellis for that code. But first we note that we may interpret all trellises as tail-biting. In the case of a single start and ending state it is simply not necessary to remember the state across the boundary from end to beginning.

The following theorem is called the *cut-set lower bound* and is due to Wiberg et al. [63, 6].

Theorem 4.10 *The product of the state space sizes at time r and r' in a tailbiting trellis is at least as large as the maximal state space size $|S_c|$ in a conventional trellis for the same code, i.e.,*

$$|S_r||S_{r'}| \geq |S_c|. \quad (4.43)$$

Proof. We cut the tail-biting trellis into two disjoint sections by splitting the states at r and r' into two and cutting the connection between them as shown in Figure 4.12.

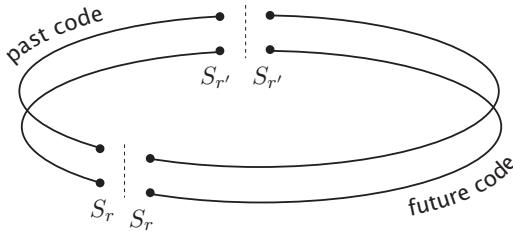


Figure 4.12: Illustration of the cut-set lower bound.

Call the section $[r', r]$ the past code, and call the section $[r, r']$ the future code. We will now construct a conventional trellis from these two pieces which has a state space at time r which is the product $|S_r||S_{r'}|$.

Let $\mathcal{S}_c = \mathcal{S}_r \times \mathcal{S}_{r'}$ be the Cartesian product of the state spaces at r and r' . The states S_c will form the state space at time r of a conventional two-section trellis for the code. This conventional trellis cannot have a smaller total state space than the minimal conventional trellis, hence the inequality in (4.43).

It remains to show how to construct this two-section trellis. Let the set of branches from the initial state to state $S_c = (S_r, S_{r'})$ of the conventional trellis be the set of paths from $S_{r'}$ to S_r in the past code, and, analogously, let the set of branches from this state $(S_{r'}, S_r)$ to the final state be the set of paths from S_r to $S_{r'}$ in the future code, i.e.,

the conventional trellis remembers the different states $S_{r'}$ not at the beginning, but by extending the state S_r . Since we can combine any codeword from the past code with any codeword from the future code as long as they share the same states S_r and $S_{r'}$, both trellises describe the same code, but the conventional trellis has a state space of size $|S_r||S_{r'}|$.

Q.E.D.

The following corollary is known as the *square-root bound*, and is a straightforward consequence of Theorem 4.10:

Corollary 4.11 *The maximum state size $|S_r|$ in a tail-biting trellis is at least as large as the square root of the maximum state size $|S_c|$ in the minimal conventional trellis for the same code, i.e.,*

$$|S_r| \geq \sqrt{|S_c|}, \quad (4.44)$$

An application of Theorem 4.10 to the trellis of the [8,4] extended Hamming code in Figure 4.8 shows that a tail-biting trellis with a state space profile $(2, 4, 2, 4, 2, 4, 2, 4, 2)$ would be possible. However, Calderbank et. al. [6] show through refined arguments that the minimal state profile for this code is $(2, 4, 4, 4, 2, 4, 4, 4, 2)$, as we constructed in Figure 4.10. The same paper also presents a minimal 16-state tail-biting trellis for the extended [24,12] Golay code, whose minimal conventional trellis implementation has 256 states.

However, apart from these examples of provably minimal tail-biting codes which achieve the square root lower bound, little is known on how to systematically construct minimal tail-biting trellises.

As a last point we wish to note that the trellis of a convolutional code can be made to tail-bite for any desired length n by preloading the initial states of the shift registers which generate the trellis code with the state bits it will have at time n . This is particularly straightforward in the feed-forward realization of an encoder, since there the state bits equal the information bits. The trellis is now no longer constrained to be driven back to the all-zero state at time n and has thus a marginally higher rate than a terminated trellis code.

4.9 The Minimal Trellis of Convolutional Codes

The question of state minimality in the case of convolutional codes has a slightly different flavor than that for general codes. Of course, we can also approach the subject via the construction of minimum span generators, and it is easily appreciated that such a trellis has a constant state space, owing to the fact that the parity-check and generator matrices of the convolutional code are band-diagonal.

However, given that the code is generated from a finite-state machine, as in Section 4.4, the state space of the trellis can be no larger than the number of distinct states of the generating finite-state machine. The question of minimizing the trellis representation is

therefore the same as that of finding a *minimal encoder* for a given code. Such an encoder has no redundant states, and the state space of the trellis and that of the encoder FSM are identical.

First, we formally define a convolutional code in the following definition.

Definition 4.2 A rate $R = k/n$ convolutional code over the field of binary Laurent series $F_2(D)$ is an injective (one-to-one) linear mapping of the k -dimensional vector Laurent series $u(D) \in F_2^k(D)$ into the n -dimensional vector Laurent series $v(D) \in F_2^n(D)$, i.e.,

$$u(D) \rightarrow v(D) : \quad F_2^k(D) \rightarrow F_2^n(D). \quad (4.45)$$

From basic linear algebra (see, e.g., [26]) we know that any such linear map can be represented by a matrix multiplication, in our case we write

$$v(D) = u(D)G(D), \quad (4.46)$$

where $G(D)$ is known as a *generator matrix* of the convolutional code, or simply a *generator*, and consists of $k \times n$ entries $g_{ij}(D)$ which are Laurent series. Algebraically, a convolutional code is the image set of the linear operator $G(D)$.

We will concentrate on *delay-free* generator matrices, i.e., those which have no common multiple of D in the numerator of $g_{ij}(D)$. In other words, a general generator matrix $G_n(D)$ can always be written as

$$G_n(D) = D^i G(D), \quad i \geq 1, \quad (4.47)$$

by pulling out the common term D^i of all $g_{ij}(D)$, where i is the delay. This restriction does not affect the generality of the results we present in this chapter.

From Definition 4.2 we see that a convolutional code is the set of output sequences, irrespective of the particular mapping of input to output sequences, and there exist an infinite number of generator matrices for the same code. We define the equivalence of two generator matrices in the following:

Definition 4.3 Two generator matrices $G(D)$ and $G'(D)$ are equivalent, written as $G(D) \equiv G'(D)$, if they generate the same convolutional code $\{v(D)\}$, where $\{v(D)\}$ is the set of all possible output sequences $v(D)$.

Examining Figure 4.3, we can read off quite easily that

$$G_2(D) = \begin{bmatrix} 1 & D^2 & D \\ D & 1 & 0 \end{bmatrix} \quad (4.48)$$

and, upon further examination,⁴ that

$$\begin{aligned} v(D) &= u(D)G_2(D) = u(D) \begin{bmatrix} 1 & D^2 \\ D & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & \frac{D}{1+D^3} \\ 0 & 1 & \frac{D^2}{1+D^3} \end{bmatrix} \\ &= u'(D) \begin{bmatrix} 1 & 0 & \frac{D}{1+D^3} \\ 0 & 1 & \frac{D^2}{1+D^3} \end{bmatrix} = u'(D)G_1(D), \end{aligned} \quad (4.49)$$

where $G_1(D)$ is the generator matrix for the encoder shown in Figure 4.2. The set of sequences $\{u(D)\}$ is identical to the set of sequences $\{u'(D)\}$, since

$$T(D) = \begin{bmatrix} 1 & D^2 \\ D & 1 \end{bmatrix} \implies T^{-1}(D) = \frac{1}{1+D^3} \begin{bmatrix} 1 & D^2 \\ D & 1 \end{bmatrix} \quad (4.50)$$

is invertible, and $u'(D) = u(D)T(D)$.

Since we have cast our coding world into the language of linear systems, the matrix $T(D)$ is invertible if and only if its determinant $\det(T(D)) = 1 + D^3 \neq 0$. In this case the set of possible input sequences $\{u(D)\}$ is mapped onto itself; $\{u(D)\} = \{u'(D)\}$, as long as $T(D)$ is a $k \times k$ matrix of full rank.

The generator matrix $G_1(D)$ above is called *systematic*, corresponding to the systematic encoder⁵ of Figure 4.2, since the input bits $(u_r^{(2)}, u_r^{(1)})$ appear unaltered as $(v_r^{(2)}, v_r^{(1)})$. Both $G_1(D)$ and $G_2(D)$ have the same number of states, as can be seen from Figures 4.2 and 4.3, and they both generate the same code $\{v(D)\}$.

Appealing to linear systems theory again, we know that there are an infinite number of matrix representations for a given linear map, each such representation amounting to a different choice of bases. Another such choice, or generator, for our code is

$$G_3(D) = \begin{bmatrix} 1+D & 1+D^2 & D \\ D+D^2 & 1+D & 0 \end{bmatrix}. \quad (4.51)$$

This new encoder has four delay elements in its realization, and therefore has more states than $G_1(D)$ or $G_2(D)$. But there is a more serious problem with $G_3(D)$. Since we can always transform one basis representation into another basis representation via a matrix multiplication, we find that

$$G_3(D) = \begin{bmatrix} 1 & 1 \\ 0 & 1+D \end{bmatrix} G_2(D) = T_3(D)G_2(D), \quad (4.52)$$

⁴Note that all coefficient operations are in the field GF(2), i.e., additions are EXOR operations, and multiplications are AND operations.

⁵We will use the terms generator matrix and encoder interchangeably, realizing that while $G(D)$ may have different physical implementations, these differences are irrelevant from our viewpoint.

and the input sequence

$$u(D) = \left[0, \frac{1}{1+D} = 1 + D + D^2 + \dots \right] \quad (4.53)$$

generates the output sequence

$$v(D) = \left[0, \frac{1}{1+D} \right] G_3(D) = [D, 1, 0], \quad (4.54)$$

whose Hamming weight $H_w(v(D)) = 2$. We have the unsettling case that an infinite weight ($H_w(u(D)) = \infty$) input sequence generates a finite weight output sequence. Such an encoder is called *catastrophic*, since a finite number of channel errors in the reception of $v(D)$ can cause an infinite number of errors in the data $u(D)$. Such encoders are suspect. We define formally:

Definition 4.4 An encoder $G(D)$ for a convolutional code is catastrophic if there exists an input sequence $u(D)$ such that $H_w(u(D)) = \infty$ and $H_w(u(D)G(D)) < \infty$.

Note that the property of being catastrophic is one of the encoder, since, while $G_3(D)$ is catastrophic, neither $G_2(D)$ nor $G_1(D)$ are, and all generate the same code! We now note that the problem stemmed from the fact that

$$T_3^{-1}(D) = \begin{bmatrix} 1 & 1 \\ 0 & 1+D \end{bmatrix}^{-1} = \begin{bmatrix} 1 & \frac{1}{1+D} \\ 0 & \frac{1}{1+D} \end{bmatrix} \quad (4.55)$$

was not well-behaved, which turned $G_2(D)$ into a catastrophic $G_3(D)$. The problem was that some of the entries of $T_3^{-1}(D)$ have an infinite number of coefficients, i.e., they are fractional.

Since, in the case of a catastrophic encoder, a finite-weight sequence $v(D)$ maps into an infinite-weight sequence $u(D)$, the encoder right inverse⁶ $G^{-1}(D)$ must have fractional entries. We therefore require that for a “useful” encoder, $G^{-1}(D)$ must have no fractional entries; in fact, all its entries are required to be polynomials in D , i.e., they have no negative powers and only a finite number of non-zero coefficients. This then is a sufficient condition for $G(D)$ to be not catastrophic. We will see later that it is also a necessary condition. Note that both $G_1(D)$ and $G_2(D)$ have polynomial right inverses, i.e.,

$$G_1^{-1}(D) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix} \quad (4.56)$$

⁶Such an inverse must always exist, since we defined a convolutional code to be a *injective* map, i.e., one which can be inverted.

and

$$G_2^{-1}(D) = \begin{bmatrix} 1 & D \\ D & 1 + D^2 \\ D^2 & 1 + D + D^3 \end{bmatrix} \quad (4.57)$$

and are therefore not catastrophic.

Let us further define the class of basic encoders as follows:

Definition 4.5 An encoder $G(D)$ is basic if it is polynomial and has a polynomial right inverse.

$G_2(D)$, for example, is a basic encoder. We will use basic encoders as a main tool to develop the algebraic theory of convolutional codes. For a basic encoder we define the constraint length

$$\nu = \sum_{i=1}^k \max_j (\deg(g_{ij}(D))) = \sum_{i=1}^k \nu_i. \quad (4.58)$$

Note that $\nu_i = \max_j (\deg(g_{ij}(D)))$, the maximum degree among the polynomials in row i of $G(D)$, corresponds to the maximum number of delay units needed to store the i th input bits $u^{(i)}(D)$ in the controller canonical realization.

Again, we find for $G_2(D)$ that $\nu = 3$, i.e., the number of delay elements needed in the controller canonical encoder realization shown in Figure 4.3 is three. Since the inputs are binary, the number of states of $G_2(D)$ is $S = 2^\nu = 8$, and we see that for basic encoders the states and the number of states are easily defined and determined.

One wonders whether 2^ν is the minimum number of states which are necessary to generate a certain convolutional code. To pursue this question further, we need the following

Definition 4.6 A minimal basic encoder is a basic encoder which has the smallest constraint length among all equivalent basic encoders.

We will see later that $G_2(D)$ is indeed minimal.

As another example of an equivalent encoder, consider

$$G_4(D) = T_4(D)G_2(D) = \begin{bmatrix} 1 & 1 + D + D^2 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & D^2 & D \\ D & 1 & 0 \end{bmatrix} \quad (4.59)$$

$$= \begin{bmatrix} 1 + D + D^2 + D^3 & 1 + D & D \\ D & 1 & 0 \end{bmatrix}, \quad (4.60)$$

whose constraint length $\nu = 4$. Note that, since

$$T_4(D)^{-1} = \begin{bmatrix} 1 & 1 + D + D^2 \\ 0 & 1 \end{bmatrix}^{-1} = \begin{bmatrix} 1 & 1 + D + D^2 \\ 0 & 1 \end{bmatrix}, \quad (4.61)$$

$G_4(D)$ has a polynomial right inverse,

$$G_4^{-1}(D) = G_2^{-1}(D) \begin{bmatrix} 1 & 1+D+D^2 \\ 0 & 1 \end{bmatrix} \quad (4.62)$$

$$= \begin{bmatrix} 1 & D \\ D & 1+D^2 \\ D^2 & 1+D+D^3 \end{bmatrix} \begin{bmatrix} 1 & 1+D+D^2 \\ 0 & 1 \end{bmatrix} \quad (4.63)$$

$$= \begin{bmatrix} 1 & 1+D^2 \\ D & 1+D+D^3 \\ D^2 & 1+D+D^2+D^4 \end{bmatrix}. \quad (4.64)$$

We have seen above that two encoders are equivalent if and only if $T(D)$ has a non-zero determinant and is therefore invertible. We may now strengthen this result for basic encoders in the following

Theorem 4.12 *Two basic encoders $G(D)$ and $G'(D)$ are equivalent if and only if $G(D) = T(D)G'(D)$, where $T(D)$ is a $k \times k$ polynomial matrix with unit determinant, $\det(T(D)) = 1$.*

Proof: Since both $T(D)$ and $G'(D)$ are polynomial, and since $T(D)$ has full rank, we conclude $\{u(D)G(D)\} = \{u(D)T(D)G'(D)\} = \{u'(D)G'(D)\}$, and $G(D) \equiv G'(D)$.

Conversely, if $G(D) \equiv G'(D)$, $T^{-1}(D)$ must exist. Since $G(D)$ is basic, it has a polynomial right inverse, $G^{-1}(D)$, and consequently $T^{-1}(D) = G'(D)G^{-1}(D)$ is polynomial also. Therefore $G'(D) = T^{-1}(D)T(D)G'(D)$ and $T^{-1}(D)T(D) = I_k$, the $k \times k$ identity matrix. But since both $T(D)$ and $T^{-1}(D)$ are polynomial, $\det(T(D))\det(T^{-1}(D)) = \det(I_k) = 1$, and $\det(T(D)) = 1$. Q.E.D.

We want to remark at this point that the binary polynomials in D , denoted by $F[D]$, form a commutative ring, i.e., they possess all the field properties except division. Other “famous” examples of rings are the integer numbers, \mathbf{Z} , as well as the integer numbers modulo m , \mathbf{Z}_m . Certain elements in a ring do have inverses: They are called *units*. In \mathbf{Z} , the units are $\{-1, 1\}$, in $F[D]$ it is only the unit element 1. In the proof of Theorem 4.12, we could also have used the following basic algebraic result [26, Page 96]:

Theorem 4.13 *A square matrix G with elements from a commutative ring R is invertible if and only if $\det(G) = r_u$, where $r_u \in R$ is a unit, i.e., if and only if $\det(G)$ is invertible in R .*

A square polynomial matrix $T(D)$ with a polynomial inverse is also called a *scrambler*, since such a $T(D)$ will simply scramble the input sequences $\{u(D)\}$, that is, relabel them.

4.10 Fundamental Theorems from Basic Algebra

In this section we explore what formal algebra has to say about encoders. With the preliminaries from the last section, we are now ready for our first major theorem. Let us decompose the basic encoder $G(D)$ into two parts, by splitting off the largest power terms in each row, i.e.,

$$G(D) = \tilde{G}(D) + \hat{G}(D) = \tilde{G}(D) + \begin{bmatrix} D^{\nu_1} & & & \\ & D^{\nu_2} & & \\ & & \ddots & \\ & & & D^{\nu_k} \end{bmatrix} G_h, \quad (4.65)$$

where G_h is a matrix with $(0, 1)$ entries, a 1 indicating the position where the highest degree term D_i^ν occurs in row i , for example:

$$G_2(D) = \begin{bmatrix} 1 & 0 & D \\ 0 & 1 & 0 \end{bmatrix} + \begin{bmatrix} D^2 & & \\ & D \end{bmatrix} \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} \quad (4.66)$$

and

$$G_4(D) = \begin{bmatrix} 1 + D + D^2 & 1 + D & D \\ 0 & 1 & 0 \end{bmatrix} + \begin{bmatrix} D^3 & & \\ & D \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}. \quad (4.67)$$

A minimal basic encoder is then characterized by the following theorem [27].

Theorem 4.14 *A polynomial $G(D)$ is a minimal basic encoder if and only if*

- (i) *G_h has full rank ($\det(G_h) \neq 0$), or, equivalently, if and only if*
- (ii) *the maximum degree of all $k \times k$ subdeterminants of $G(D)$ equals the constraint length ν .*

Proof: Assume that G_h does not have full rank. Then there exists a sum of $d \leq k$ rows \mathbf{h}_{i_j} of G_h such that

$$\mathbf{h}_{i_1} + \mathbf{h}_{i_2} + \cdots + \mathbf{h}_{i_d} = 0, \quad (4.68)$$

as in (4.67). Assume now that we have ordered the indices in increasing maximum row degrees, such that $\nu_{i_d} \geq \nu_{i_j}, d \geq j$. Adding

$$D^{\nu_{i_d}} (\mathbf{h}_{i_1} + \mathbf{h}_{i_2} + \cdots + \mathbf{h}_{i_{d-1}}) \quad (4.69)$$

to row i_d of $\hat{G}(D)$ reduces it to an all zero row, and, similarly, adding

$$D^{\nu_{i_d}-\nu_{i_1}} g_{i_1}(D) + D^{\nu_{i_d}-\nu_{i_2}} g_{i_2}(D) + \cdots + D^{\nu_{i_d}-\nu_{i_{d-1}}} g_{i_{d-1}}(D) \quad (4.70)$$

to row i_d of $G(D)$ will reduce the highest degree of row i_d and produce an equivalent generator. The new generator matrix now has a constraint length which is less than that of the original generator matrix and we have a contradiction to the original assumption that our $G(D)$ was minimal basic.

After some thought (working with $\tilde{G}(D)$), it is quite easy to see that conditions (i) and (ii) in the theorem are equivalent. Q.E.D.

Since Theorem 4.14 has a constructive proof, there follows a simple algorithm to obtain a minimal basic encoder from any basic encoder, given by the following:

Step 1: If G_h has full rank, $G(D)$ is a minimal basic encoder and we stop, else

Step 2: let $\mathbf{h}_{i_j}, j \leq k$ be a set of rows of G_h such that

$$\mathbf{h}_{i_1} + \mathbf{h}_{i_2} + \cdots + \mathbf{h}_{i_d} = 0. \quad (4.71)$$

Further let $g_{i_j}(D)$ be the corresponding rows of $G(D)$, and add

$$D^{\nu_{i_d} - \nu_{i_1}} g_{i_1}(D) + D^{\nu_{i_d} - \nu_{i_2}} g_{i_2}(D) + \cdots + D^{\nu_{i_d} - \nu_{i_{d-1}}} g_{i_{d-1}}(D) \quad (4.72)$$

to row i_d of $G(D)$. Go to Step 1.

Note that a minimal basic encoder for a given convolutional code is not necessarily unique. For example,

$$G_5(D) = \begin{bmatrix} 1 & D \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & D^2 & D \\ D & 1 & 0 \end{bmatrix} \quad (4.73)$$

$$= \begin{bmatrix} 1 + D^2 & D^2 & D \\ D & 1 & 0 \end{bmatrix} \quad (4.74)$$

is a basic encoder with constraint length $\nu = 3$ equivalent to $G_2(D)$, and both are minimal.

Our treatment of convolutional encoders has focused on basic encoders so far. We will now justify why this class of encoders is so important. But before we can continue, we need some more basic algebra.

Figure 4.13 relates the algebraic concepts needed in the rest of this chapter. We start with the (commutative) ring R . If R is commutative and has no non-zero zero divisors, i.e., there exist no non-zero numbers $r_1, r_2 \in R$, such that $r_1 r_2 = 0$, the ring R is called an *integral domain*. In an integral domain we may use the cancelation law: $r_1 r_2 = r_1 r_3 \Rightarrow r_2 = r_3$, which is something like division, but not quite as powerful. Examples of integral domains are \mathbf{Z} , $F[D]$ (the ring of binary polynomials), and \mathbf{Z}_p , the integers modulo p , when p is a prime.

On the other hand, a subring N of a ring R is an *ideal* if, for every $r \in R$, $rN \in N$ and $Nr \in N$. That is, the ideal brings every element of the original ring R into itself through multiplication. If the ring R (and hence also N) is commutative, we define a principal ideal N as one which is generated by all the multiples of a single element n with R , i.e., $\langle n \rangle = \{rn | r \in R\}$. An example of a principal ideal in \mathbf{Z} is $\langle 2 \rangle$, the ideal of all even numbers. Now, if every ideal in a commutative ring R is principal, we have a *principal ideal domain*. This is an algebraic structure with powerful properties, one of the most well known of which is the following:

Theorem 4.15 (Unique Factorization Theorem) *In a principal ideal domain, every element can be factored uniquely, up to unit elements, into primes, or irreducibles. These are elements which cannot be factored.*

In \mathbf{Z} , this is the popular and famous integer prime factorization. Remember that the units in \mathbf{Z} are 1 and -1 , and hence the primes can be taken positive or negative by multiplying with the unit element -1 . One usually agrees on the convention to take only positive primes. Note that, technically, this factorization applies to fields also, (such as \mathbf{Z}_p); but, alas, in a field every element is a unit, and hence the factorization is not unique. In $F[D]$, however, the only unit is 1 and we have an unambiguous factorization.

For principal ideal domains we have the important theorem (see, e.g., [26, Page 181 ff], and [15]):

Theorem 4.16 (Invariant Factor Theorem) *Given a $k \times n$ matrix \mathbf{P} with elements from the principal ideal domain R , \mathbf{P} can be written as*

$$\mathbf{P} = \mathbf{A}\Gamma\mathbf{B}, \quad (4.75)$$

where

$$\Gamma = \begin{bmatrix} \gamma_1 & & & & \\ & \gamma_2 & & & 0 \\ & & \ddots & & \\ & & & \gamma_k & 0 \\ 0 & & & & \ddots & 0 \end{bmatrix}, \quad (4.76)$$

and $\gamma_i|\gamma_j$, if $i \leq j$. Furthermore, the $k \times k$ matrix \mathbf{A} and the $n \times n$ matrix \mathbf{B} both have unit determinants and are therefore invertible in R . The invariant factors are given by $\gamma_i = \Delta_i/\Delta_{i-1}$, where Δ_i is the greatest common divisor (g.c.d.) of the $i \times i$ subdeterminants of \mathbf{P} .

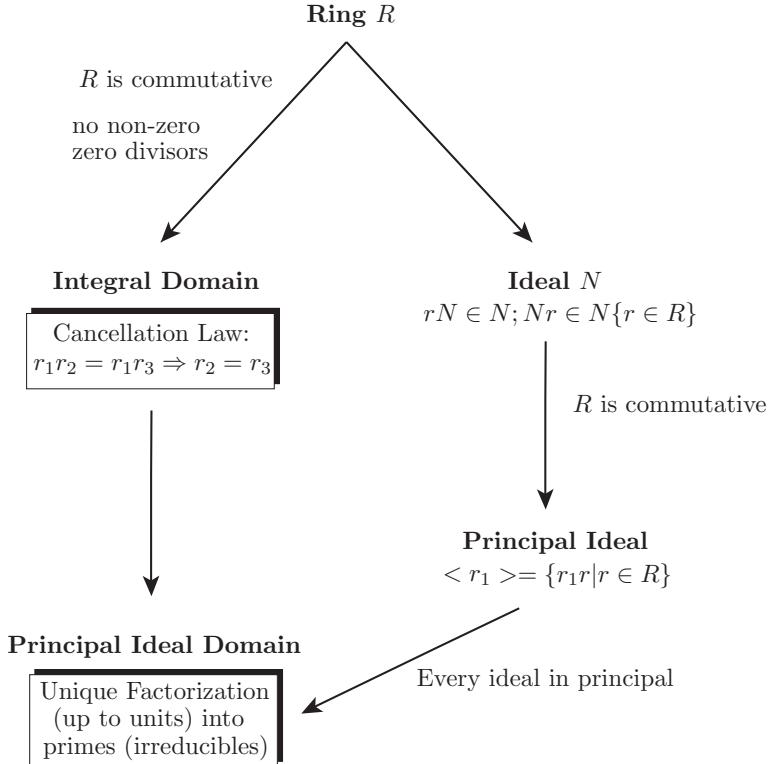


Figure 4.13: Diagram of ring-algebraic concepts. Some examples of rings to which these concepts apply are the integer numbers \mathbf{Z} , the integers modulo m , \mathbf{Z}_m and the polynomials in D over the field F , $F[D]$.

The invariant factor theorem can easily be extended to rational matrices. Let \mathbf{R} be a $k \times n$ matrix whose entries are fractions of elements of R . Furthermore, let ϕ be the least common multiple of all denominators of the entries in \mathbf{R} . We may now apply the invariant factor theorem to the matrix $\mathbf{P} = \phi\mathbf{R}$, which has all its elements in R , and obtain

$$\phi\mathbf{R} = \mathbf{A}\Gamma'\mathbf{B} \Rightarrow \mathbf{R} = \mathbf{A}\Gamma\mathbf{B}, \quad (4.77)$$

where the new entries of $\mathbf{\Gamma}$ are $\gamma_i = \gamma'_i/\phi$, where, again, $\gamma_i|\gamma_j$ for $i \leq j$, since $\gamma'_i|\gamma'_j$.

Applying these concepts to our description of encoders now, we find the following decomposition for a generating matrix $G(D)$ with rational entries:

$$G(D) = A(D)\Gamma(D)B(D), \quad (4.78)$$

where $A(D)$ is a $k \times k$ scrambler and $B(D)$ is an $n \times n$ scrambler. Now, since the last $n - k$ diagonal elements of $\Gamma(D) = 0$, we may strike out the last $n - k$ rows in $B(D)$ and obtain a $k \times n$ polynomial matrix, which we call $G_b(D)$. Since $A(D)$ is invertible and $\Gamma(D)$ is invertible, $G(D) \equiv G_b(D)$, i.e., $G(D) = A(D)\Gamma(D)G_b(D)$ and $G_b(D)$ generate the same code. But $G_b(D)$ has entries in $F[D]$, and, furthermore, since $B(D)$ has unit determinant and has a polynomial inverse, so does $G_b(D)$. We conclude that $G_b(D)$ is a basic encoding matrix which is equivalent to the original rational encoding matrix $G(D)$. Hence the following

Theorem 4.17 *Every rational encoding matrix has an equivalent basic encoding matrix.*

In the process of the above discussion we have also developed the following algorithm to construct an equivalent basic encoding matrix for any rational encoding matrix:

Step 1: Compute the invariant factor decomposition of the rational matrix $G(D)$, given by

$$G(D) = A(D)\Gamma(D)B(D). \quad (4.79)$$

Step 2: Delete the last $n - k$ rows of $B(D)$ to obtain the equivalent basic $k \times n$ encoding matrix $G_b(D)$.

An algorithm to calculate the invariant factor decomposition is shown in Appendix 4.16.

While we now have a pretty good idea about basic encoders and how to obtain a minimal version thereof, the question whether some non-polynomial encoders might have a less complex internal structure is still daunting. We know how to turn any encoder into a basic encoder and then into a minimal basic encoder, but do we lose anything in doing so? We are now ready to address this question and we will see that a minimal basic encoder is minimal in a more general sense than we have been able to show up to now. In order to do so, we need the concept of abstract states.

An abstract state is, loosely speaking, the internal state of an encoder which produces a particular output ($v(D)$ in our case), if the input sequence is all zero. This is known as the zero-input response. Different abstract states must generate different output sequences, otherwise they are not distinguishable and are the same state. Obviously, different abstract states correspond to different physical states (the states of the shift registers in an encoder implementation), but different physical states might correspond to the same abstract state.

This will appear again in Chapter 6 as state equivalence. Hence, the number of abstract states will always be equal to or smaller than the number of physical states of a given encoder. The number of abstract states is therefore a more basic measure of the inherent complexity of an encoder.

In order to generate all possible abstract states, we simply use all possible input sequences from $-\infty$ to time unit 0, at which time we turn off the input sequences and observe the set of possible outputs $\{v(D)\}$. This set we define as our abstract states. To formalize, let P be the projection operator which sets the input sequence to 0 for all non-negative time units, i.e., $u(D)P = (\dots, \mathbf{u}_{-2}, \mathbf{u}_{-1}, 0, 0, 0, \dots)$ and let Q be the projection operator which sets the output sequence to zero for all negative time units, i.e., $v(D)Q = (\dots, 0, 0, \mathbf{v}_0, \mathbf{v}_1, \dots)$. The abstract state corresponding to an input sequence $u(D)$ is then formally given by

$$v_S(D) = (u(D)PG(D))Q. \quad (4.80)$$

Our first result is

Theorem 4.18 *The number of abstract states of a minimal basic encoder is 2^ν , equal to the number of physical states.*

Proof: Let

$$s(D) = \left(u_{-\nu_k}^{(k)} D^{-\nu_k} + \dots + u_{-1}^{(k)} D^{-1}, \dots, u_{-\nu_1}^{(1)} D^{-\nu_1} + \dots + u_{-1}^{(1)} D^{-1} \right) \quad (4.81)$$

be a physical state of the basic encoder in its controller canonical form. For example, for the code from Figure 4.3, $v_S(D) = (u_{-2}^{(2)} D^{-2} + u_{-1}^{(2)} D^{-1}, u_{-1}^{(1)} D^{-1})$. There are 2^ν different physical states in our basic encoder. Let us assume now that two different physical states $s_1(D)$ and $s_2(D)$ correspond to the same abstract state, i.e.,

$$v(D)Q = s_1(D)G(D)Q = s_2(D)G(D)Q. \quad (4.82)$$

This is equivalent to

$$(s_1(D) - s_2(D))G(D)Q = s(D)G(D)Q = 0, \quad (4.83)$$

i.e., there exists a non-zero state $s(D)$ whose corresponding abstract state is 0, according to (4.83). We will now show that the only state which fulfills (4.83) is $s(D) = 0$, and hence $s_1(D) = s_2(D)$. This will prove the theorem by contradiction.

We need to have $v(D) = 0$, i.e., $v_i = 0$ for all $i \geq 0$, in order for (4.83) to hold. Now, without loss of generality, assume that

$$\nu_k = \nu_{k-1} = \dots = \nu_{k-l} > \nu_{k-l-1} \geq \dots \geq \nu_1. \quad (4.84)$$

The coefficient v_{ν_k} of D^{ν_k} in $v(D)$ is then given by

$$v_{-\nu_k} = \left(0, \dots, 0, u_{-\nu_k}^{(k-l)}, \dots, u_{-\nu_k}^{(k)}, \right) G_h(D) = 0, \quad (4.85)$$

but, since $G_h(D)$ has full rank for a minimal basic encoding matrix, we must have $u_{-\nu_k}^{(k)} = u_{-\nu_k}^{(k-1)} = \dots = u_{-\nu_k}^{(1)} = 0$. Continuing by induction, we then prove $v_{-\nu_{k-1}} = 0$ etc., i.e., that $s(D) = 0$, which proves the theorem. Q.E.D.

We see that the abstract states capture the memory of an encoder $G(D)$ in a very general way, and we are therefore interested in finding the $G(D)$ which minimizes the number of abstract states, realizing that this is all we need to keep track of in the decoder. Hence the following

Definition 4.7 *A minimal encoder is an encoder $G(D)$ which has the smallest number of abstract states over all equivalent encoders (basic or not).*

Now, if $G(D)$ and $G'(D)$ are two equivalent encoders with abstract states $v_S(D)$ and $v'_S(D)$, respectively, we can relate these abstract states as follows:

$$\begin{aligned} v_S(D) &= u(D)PG(D)Q = u(D)PT(D)G'(D)Q \\ &= u(D)PT(D)(P+Q)G'(D)Q \\ &= u(D)PT(D)PG'(D)Q + u(D)PT(D)QG'(D)Q, \end{aligned} \quad (4.86)$$

but the first term is an abstract state of $G'(D)$, denoted $v'_S(D)$, and the second term is a codeword, i.e.,

$$v'(D) = u(D)PT(D)QG'(D)Q \quad (4.87)$$

$$= u(D)PT(D)QG'(D), \quad (4.88)$$

where we were allowed to drop the operator Q in the above equation, since, for any *realizable* encoder, $G'(D)$ has to be a causal encoding matrix. Therefore, since $u'(D) = u(D)PT(D)Q$ is an input sequence which starts at time 0, i.e., $u'_j = 0; j < 0$, and since $G'(D)$ must be causal, $v'(D)$ cannot have any non-zero negative components either. Hence the trailing Q -operator in (4.88) is superfluous, and we have the representation given in the following theorem.

Theorem 4.19

$$v_S(D) = v'_S(D) + v'(D), \quad (4.89)$$

where, if $G'(D)$ is a minimal basic encoder, the representation (4.89) is unique.

Proof: To see this, assume the contrary, i.e.,

$$v_{S_{(\text{mb})}}(D) + v(D) = v'_{S_{(\text{mb})}}(D) + v'(D), \quad (4.90)$$

that is,

$$v_{S_{(\text{mb})}}(D) + v'_{S_{(\text{mb})}}(D) = v''_{S_{(\text{mb})}}(D) = v''(D) = v(D) + v'(D), \quad (4.91)$$

and $v''(D)$ is both a codeword and an abstract state $v''_{S_{(\text{mb})}}(D)$ of the minimal basic encoder. But then

$$v''(D) = u''(D)G'(D) \Rightarrow u''(D) = v''(D)[G'(D)]^{-1}(D), \quad (4.92)$$

and $u''(D)$ is polynomial since $[G'(D)]^{-1}$ is polynomial. Furthermore, since $v''_{S_{(\text{mb})}}(D) = u(D)G'(D)Q$, for some input sequence $u(D)$ with no non-zero terms $u_j, j \geq 0$, and

$$v''_{S_{(\text{mb})}}(D) = u(D)G'(D)Q = u''(D)G'(D), \quad (4.93)$$

we obtain

$$(u''(D) + u(D)) G'(D)Q = 0. \quad (4.94)$$

Using the same method as in the proof of Theorem 4.18, we can show that $(u''(D) + u(D)) = 0$, which implies $u''(D) = u(D) = 0$, and $v''_{S_{(\text{mb})}} = 0$, leading to a contradiction. This proves the theorem. Q.E.D.

Due to $v_{S_{(\text{mb})}}(D) = v_S(D) + v(D)$ and the uniqueness of (4.89), the map $v_S(D) \rightarrow v_{S_{(\text{mb})}}(D) : v_S(D) = v_{S_{(\text{mb})}}(D) + v'(D)$ is surjective (i.e., onto), and we have our next theorem.

Theorem 4.20 *The number of abstract states of an encoder $G(D)$ is always larger than or equal to the number of abstract states of an equivalent minimum basic encoder $G_{\text{mb}}(D)$.*

We also immediately notice the following

Corollary 4.21 *A minimal basic encoder is a minimal encoder.*

While this proves that the minimal basic encoders are desirable since they represent an implementation with the minimum number of abstract states, more can be said about minimal encoders in general, expressed by the following central theorem.

Theorem 4.22 *$G(D)$ is a minimal encoder if and only if*

- (i) *its number of abstract states equals the number of abstract states of an equivalent minimal basic encoder, or, if and only if*

(ii) $G(D)$ has a polynomial right inverse in D and also a polynomial right inverse in D^{-1} .

Proof:

(i) Part (i) is obvious from Theorem 4.20.

(ii) Let $u(D)$ be given and assume that the output sequence $v(D) = u(D)G(D)$ is polynomial in the inverse power D^{-1} , i.e., $v_r = 0$ for $r > 0$ and $r < s$, where $s \leq 0$ is arbitrary. Then

$$D^{-1}v(D)Q = 0. \quad (4.95)$$

We now show that $u(D)$ must also be polynomial in D^{-1} . Breaking up (4.95) into two components, a past and a future term, we obtain

$$\begin{aligned} D^{-1}v(D)Q &= 0 = D^{-1}u(D)(P + Q)G(D)Q \\ &= D^{-1}u(D)PG(D)Q + D^{-1}u(D)QG(D)Q, \end{aligned} \quad (4.96)$$

and, since $G(D)$ is causal, $D^{-1}u(D)QG(D)Q = D^{-1}u(D)QG(D)$ is also a codeword. Therefore the abstract state $D^{-1}u(D)PG(D)Q$ is a codeword, and, following (4.91) ff., we conclude that it must be the zero codeword. Since $G(D)$ has full rank, $D^{-1}u(D)QG(D) = 0$ implies

$$D^{-1}u(D)Q = 0, \quad (4.97)$$

i.e., $u(D)$ contains no positive powers of D . Since $v(D) = u(D)G(D)$ and $G(D)$ is delay-free, $u(D)$ must be polynomial in D^{-1} , i.e., $u_r = 0$ for $r < s$. Therefore the map $v(D) \rightarrow u(D) = G^{-1}(D)v(D)$ maps polynomials in D^{-1} into polynomials in D^{-1} , and $G^{-1}(D)$, the right inverse of $G(D)$, must be polynomial in D^{-1} also.

The fact that a minimal encoder $G(D)$ also has a polynomial right inverse in D is proven similarly, i.e., assume this time that $v(D) = u(D)G(D)$ is polynomial in D . Then

$$v(D) = u(D)PG(D) + u(D)QG(D), \quad (4.98)$$

where $u(D)Q$ is a power series (only positive terms). Then, due to causality, $u(D)QG(D)$ is also a power series, which implies that $u(D)QG(D)$ must also be a power series. But now

$$u(D)PG(D) = u(D)PG(D)Q \quad (4.99)$$

is again both a codeword and an abstract state, which implies that it must be the zero codeword, i.e., $u(D)PG(D) = 0$. But since $G(D)$ has full rank we conclude that $u(D)P = 0$, that is, $u(D)$ is a power series. Now from above take $G^{-1}(D^{-1})$, a

polynomial right inverse in D^{-1} . Then $G^{-1}(D) = D^s G^{-1}(D^{-1})$ is (pseudo)-inverse which is polynomial in D such that

$$u(D)D^s = v(D)G^{-1}(D^{-1}), \quad (4.100)$$

and hence a polynomial $v(D)$ can only generate a polynomial $u(D)$, i.e., $u(D)$ has only finitely many terms. Therefore the inverse map $G^{-1}(D) : v(D) \rightarrow u(D) = v(D)G^{-1}(D)$ must also be polynomial.

Now for the reverse part of the theorem assume that $G(D)$ has a polynomial inverse in D^{-1} and in D . Assume further that the abstract state $v_s(D) = u(D)G(D)Q = u'(D)G(D)$ is a codeword, and that $u(D)$ is polynomial in D^{-1} without a constant term. $v_s(D)$ is therefore a power series and since $G(D)$ has a polynomial inverse in D by assumption, it follows that $u'(D)$ is also a power series. Now, using the right inverse $G^{-1}(D^{-1})$ which is polynomial in D^{-1} , we write

$$\begin{aligned} u'(D)G(D)G^{-1}(D^{-1}) &= u(D)G(D)QG^{-1}(D^{-1}), \\ u'(D) &= u(D)G(D)QG^{-1}(D^{-1}). \end{aligned} \quad (4.101)$$

But the left-hand side in (4.101) has no negative powers in D , and the right-hand side has no positive powers since $u(D)G(D)Q$ has no positive powers and $G^{-1}(D^{-1})$ is polynomial in D^{-1} . We conclude that $v_s(D) = 0$. Using Theorems 4.19 and 4.20, we conclude that $G(D)$ is minimal. Q.E.D.

As pointed out by Johannesson and Wan, part (ii) of the theorem provides us with a practical minimality test for encoders. Furthermore, since a minimal encoder $G(D)$ has a polynomial right inverse $(G(D))^{-1}$, it follows immediately that $G(D)$ is not catastrophic, i.e., we have the following corollary.

Corollary 4.23 *A minimal encoder is not catastrophic.*

Johannesson and Wan pointed out the interesting fact [27] that there are minimal encoders which are not minimal basic. Quoting their example, the basic encoder

$$G(D) = \begin{bmatrix} 1+D & D & 1 \\ 1+D^2+D^3 & 1+D+D^2+D^3 & 0 \end{bmatrix} \quad (4.102)$$

has constraint length $\nu = 4$, but is not minimal basic. In fact

$$G'(D) = \begin{bmatrix} 1+D & D & 1 \\ D^2 & 1 & 1+D+D^2 \end{bmatrix} \quad (4.103)$$

is an equivalent minimal basic encoder with constraint length $\nu' = 3$.

Nevertheless, $G(D)$ has a polynomial right inverse in D^{-1} , given by

$$(G(D^{-1}))^{-1} = \begin{bmatrix} 1 + D^{-1} + D^{-2} + D^{-3} & D^{-1} \\ 1 + D^{-1} + D^{-3} & D^{-1} \\ D^{-2} + D^{-3} & D^{-1} \end{bmatrix} \quad (4.104)$$

and is therefore, according to Theorem 4.22, a minimal encoder with $2^3 = 8$ abstract states.

4.11 Systematic Encoders

The encoder in Figure 4.2, whose encoding matrix is

$$G_s(D) = \begin{bmatrix} 1 & 0 & \frac{D}{1+D^3} \\ 0 & 1 & \frac{D^2}{1+D^3} \end{bmatrix}, \quad (4.105)$$

is systematic, i.e., the k information bits appear unchanged in the output sequence $v(D)$.

We now show that every code has a systematic encoder. Let us assume without loss of generality that the code is generated by the basic encoder $G(D)$. Then $G^{-1}(D)$ exists and is a polynomial matrix. Using the invariant factor theorem we write

$$G^{-1}(D) = (A(D)\Gamma(D)B(D))^{-1} = B^{-1}(D)\Gamma^{-1}(D)A^{-1}(D), \quad (4.106)$$

and, since $B^{-1}(D)$ and $A^{-1}(D)$ are polynomial, $\Gamma^{-1}(D)$ must be a polynomial also. This means that all the invariant factors must be units, i.e., $\gamma_i = 1$, and thus the greatest common divisor (g.c.d.) of the $k \times k$ subdeterminants of $G(D)$ must equal 1. It follows that there exists a $k \times k$ subdeterminant $\Delta_k(D)$ of $G(D)$, which is a delay-free polynomial (no multiple of D), since otherwise the g.c.d. of all the $k \times k$ subdeterminant would be a multiple of D .

We now rearrange the columns of $G(D)$ such that the first k columns form that matrix $T(D)$ whose determinant is $\Delta_k(D)$. Since $T(D)$ is invertible, we form

$$G_s(D) = T^{-1}(D)G(D) = [\mathbf{I}_k | P(D)], \quad (4.107)$$

where $P(D)$ is a $n-k \times k$ parity-check matrix with (possibly) rational entries. For example, for the code from Figure 4.2 we have

$$P(D) = \begin{bmatrix} \frac{D}{1+D^3} \\ \frac{D^2}{1+D^3} \end{bmatrix}. \quad (4.108)$$

Note that the systematic encoder for a given code is unique, whereas there may exist several different equivalent minimal basic encoders for the same code.

Systematic encoders have another nice property, given by the following theorem.

Theorem 4.24 Every systematic encoder for a convolutional code is minimal.

Proof: The proof of this result comes easily from Theorem 4.22. Consider the inverse of a systematic encoder, which is simply the $k \times k$ identity matrix, which, trivially, is both polynomial in D and in D^{-1} . Q.E.D.

To obtain a systematic encoder for a given code, the procedure outlined above is practical, i.e., we find a $k \times k$ submatrix whose determinant is a polynomial which is not a multiple of D . Then simply calculate (4.107) to obtain $G_s(D)$.

The inverse operation is more difficult. Given a systematic encoder $G_s(D)$, we want to find an equivalent minimal basic encoder. We can proceed by calculating the invariant factor decomposition of $G_s(D)$. This will give us a basic encoder. Then we apply the algorithm on page 140 to obtain a minimal basic encoder.

Often it is easier to apply a more ad hoc technique, based on examining the trellis of a given code [48]. Let us take the example of Figure 4.2 again (Equation (4.105)). This systematic encoder has 8 states, and, since systematic encoders are minimal, we are looking for a minimal basic encoder with 8 states also. The code rate is $2/3$, and therefore the two values for ν_1, ν_2 must be 1 and 2 (or 0 and 3, which is quickly ruled out). The minimal basic encoder blueprint for our code is shown in Figure 4.14.

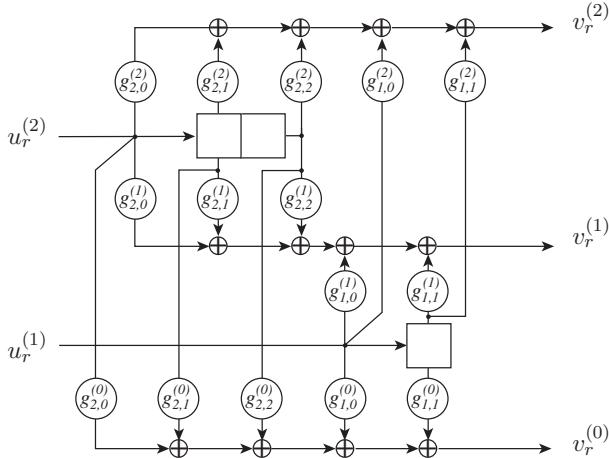


Figure 4.14: Encoder blueprint for the minimal basic encoder equivalent to $G_S(D)$.

We now look at the trellis generated by the systematic encoder, whose first few transitions originating from the zero-state are shown in Figure 4.15. We start out by trac-

ing paths from the zero-state back to the zero-state and map them into the connectors $g_{1,m}^{(j)}$ and $g_{2,m}^{(j)}$. The first merger occurs after two branches with the output sequence $(v_0, v_1) = ((010), (100))$. The first triple v_0 is generated by setting $g_{1,0}^{(2)} = 0, g_{1,0}^{(1)} = 1$ and $g_{1,0}^{(0)} = 0$. The second triple v_1 is generated by setting $g_{1,1}^{(2)} = 1, g_{1,1}^{(1)} = 0$ and $g_{1,1}^{(0)} = 0$. Now we move onto the next path. It is advantageous to choose paths for which $u_r^{(1)} = 0$, since then the connectors $g_{1,m}^{(j)}$ will not interfere. One such path is $(100), (001), (010)$. From this we can determine the connectors $g_{2,0}^{(2)} = 1, g_{2,0}^{(1)} = 0$, and $g_{2,0}^{(0)} = 0$. From the second triple we obtain $g_{2,1}^{(2)} = 0, g_{2,1}^{(1)} = 0$, and $g_{2,1}^{(0)} = 1$ and, from the third triple $g_{2,2}^{(2)} = 0, g_{2,2}^{(1)} = 1$, and $g_{2,2}^{(0)} = 0$. Checking back with Figure 4.3, we see that we have obtained the same encoder. (Note that choosing other paths could have generated another, equivalent encoder.) This procedure can easily be generalized for larger encoders.

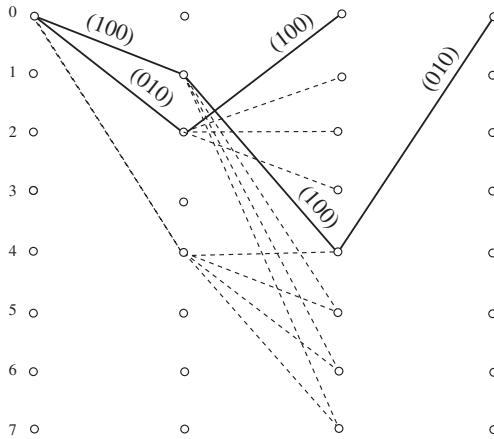


Figure 4.15: Initial trellis section generated by the systematic encoder from Figure 4.2.

4.12 Maximum Free-Distance Convolutional Codes

This chapter has developed fundamental results of convolutional codes and their encoders. These results do not give explicit constructions for good convolutional codes and, as in the case for trellis codes, computer searches are usually used to find good codes [31, 46, 47].

Unlike trellis codes, convolutional code searches can be based on Hamming distance.

If the output bits $v(D)$ of a convolutional code are mapped into the signals $\{-1, +1\}$ of a binary phase-shift keyed signal constellation, the minimum squared Euclidean distance d_{free}^2 depends only on the number of binary differences between the closest code sequences. This number is the minimum Hamming distance of a convolutional code, denoted by d_{free} . Since convolutional codes are linear, finding the minimum Hamming distance between two sequences $v^{(1)}(D)$ and $v^{(2)}(D)$, $H_d(v^{(1)}(D), v^{(2)}(D))$, amounts to finding the minimum Hamming weight of any code sequence $v(D)$. Finding convolutional codes with large minimum Hamming distance is as difficult as finding good trellis codes with large free distance. Most often the controller canonical form (Figure 4.3) of an encoder is preferred in these searches, which is targeted at finding a minimal basic encoder. The procedure is then to search for a code with the largest minimum Hamming weight by varying the connector taps $g_{i,m}^{(j)}$, either exhaustively or according to heuristic rules. Once an encoder is found, it is tested for minimality, which will then ensure that it is not catastrophic.

In this fashion the codes in Tables 4.1–4.7 were found [11, 31, 33, 49]. They are the rate $R = 1/n$, with $n = 1, 2, \dots, 8$ and $R = 2/3$ codes with the largest minimum Hamming distance d_{free} for a given constraint length. The free distance of a convolutional code is the most common parameter optimized in code searches. Extensive tables of a variety of good convolutional codes may be found in Chapter 8 of [28] and the references therein.

ν	$g^{(1)}$	$g^{(0)}$	d_{free}	$g^{(2)}$	$g^{(1)}$	$g^{(0)}$	d_{free}
2	5	7	5	2	5	7	8
3	15	17	6	3	13	15	10
4	23	35	7	4	25	33	12
5	65	57	8	5	47	53	13
6	133	171	10	6	133	145	15
7	345	237	10	7	225	331	16
8	561	753	12	8	557	663	18
9	1161	1545	12	9	1117	1365	20
10	2335	3661	14	10	2353	2671	22
11	4335	5723	15	11	4767	5723	24
12	10533	17661	16	12	10533	10675	24
13	21675	27123	16	13	21645	35661	26
14	56721	61713	18				
15	111653	145665	19				
16	347241	246277	20				

Table 4.1: Connectors and free Hamming distance of the best $R = 1/2$ convolutional codes [33] on the left. The connectors are given in octal notation, e.g., $g = 17 = 1111$, and connectors and d_{free} of the best $R = 1/3$ convolutional codes on the right [33].

	$g_2^{(2)}, g_1^{(2)}$	$g_2^{(1)}, g_1^{(1)}$	$g_2^{(0)}, g_1^{(0)}$	d_{free}
2	3, 1	1, 2	3, 2	3
3	2, 1	1, 4	3, 7	4
4	7, 2	1, 5	4, 7	5
5	14, 3	6, 10	16, 17	6
6	15, 6	6, 15	15, 17	7
7	14, 3	7, 11	13, 17	8
8	32, 13	5, 33	25, 22	8
9	25, 5	3, 70	36, 53	9
10	63, 32	15, 65	46, 61	10

Table 4.2: Connectors and d_{free} of the best $R = 2/3$ convolutional codes [33].

	$g^{(3)}$	$g^{(2)}$	$g^{(1)}$	$g^{(0)}$	d_{free}
2	5	7	7	7	10
3	13	15	15	17	13
4	25	27	33	37	16
5	53	67	71	75	18
6	135	135	147	163	20
7	235	275	313	357	22
8	463	535	733	745	24
9	1117	1365	1633	1653	27
10	2387	2353	2671	3175	29
11	4767	5723	6265	7455	32
12	11145	12477	15537	16727	33
13	21113	23175	35527	35537	36

Table 4.3: Connectors and d_{free} of the best $R = 1/4$ convolutional codes [31].

	$g^{(4)}$	$g^{(3)}$	$g^{(2)}$	$g^{(1)}$	$g^{(0)}$	d_{free}
2	7	7	7	5	5	13
3	17	17	13	15	15	16
4	37	27	33	25	35	20
5	75	71	73	65	57	22
6	175	131	135	135	147	25
7	257	233	323	271	357	28

Table 4.4: Connectors and d_{free} of the best $R = 1/5$ convolutional codes [11].

	$g^{(5)}$	$g^{(4)}$	$g^{(3)}$	$g^{(2)}$	$g^{(1)}$	$g^{(0)}$	d_{free}
2	7	7	7	7	5	5	16
3	17	17	13	13	15	15	20
4	37	35	27	33	25	35	24
5	73	75	55	65	47	57	27
6	173	151	135	135	163	137	30
7	253	375	331	235	313	357	34

Table 4.5: Connectors and d_{free} of the best $R = 1/6$ convolutional codes [11].

	$g^{(6)}$	$g^{(5)}$	$g^{(4)}$	$g^{(3)}$	$g^{(2)}$	$g^{(1)}$	$g^{(0)}$	d_{free}
2	7	7	7	7	5	5	5	18
3	17	17	13	13	13	15	15	23
4	35	27	25	27	33	35	37	28
5	53	75	65	75	47	67	57	32
6	165	145	173	135	135	147	137	36
7	275	253	375	331	235	313	357	40

Table 4.6: Connectors and d_{free} of the best $R = 1/7$ convolutional codes [11].

4.13 The Squaring Construction and the Trellis of Lattices

In this section we take a reverse approach to that of the previous sections, that is, we construct lattices and codes by *constructing* their trellises first. Lattices were used in Chapter 3 as signal sets for trellis codes, and we now learn that they, too, can be described by trellises. To this end we introduce a general building block, the squaring construction, which has found application mainly in the construction of lattices [9].

Let us assume that some set S of signals is the union of M disjoint subsets T_i . This is a partition of S into M subsets, and we denote this partition by S/T . Furthermore,

	$g^{(7)}$	$g^{(6)}$	$g^{(5)}$	$g^{(4)}$	$g^{(3)}$	$g^{(2)}$	$g^{(1)}$	$g^{(0)}$	d_{free}
2	7	7	5	5	5	7	7	7	21
3	17	17	13	13	13	15	15	17	26
4	37	33	25	25	35	33	27	37	32
5	57	73	51	65	75	47	67	57	36
6	153	111	165	173	135	135	147	137	40
7	275	275	253	371	331	235	313	357	45

Table 4.7: Connectors and d_{free} of the best $R = 1/8$ convolutional codes [11].

let $d(S)$ be the minimum Euclidean distance between any two elements of S , i.e., $d(S) = \min_{\substack{s_1, s_2 \in S \\ (s_1 \neq s_2)}} |s_1 - s_2|^2$, and let $d(T_j)$ be the minimum distance between any two elements of the subset T_j . Define $d(T) = \min_j T_j$ as the minimum distance between elements in any one of the subsets. As an example consider the lattice partition $\Lambda = \Lambda' + [\Lambda/\Lambda']$ from Chapter 3, where the subsets are the cosets of the sublattice Λ' .

We now define the *squaring construction* [16] as the set of signals U , given by the following definition.

Definition 4.8 *The union set U , also denoted by $|S/T|^2$, resulting from the squaring construction of the partition S/T is the set of all pairs (s_1, s_2) , such that $s_1, s_2 \in T_j$, i.e., s_1 and s_2 are in the same subset T_j .*

The squaring construction can conveniently be depicted by a trellis diagram, shown in Figure 4.16. This trellis has M states, corresponding to the M subsets T_j of S . This is logical, since we need to remember the subset of s_1 in order to restrict the choice of s_2 to the same subset.

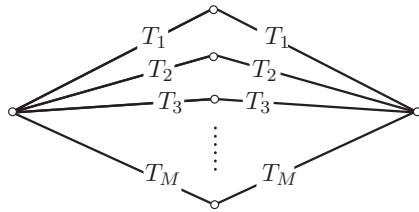


Figure 4.16: Trellis diagram of the squaring construction.

Continuing our lattice example, let us apply the squaring construction to the lattice partition $Z^2/RZ^2 = Z^2/D_2$ (compare Figure 3.13, Chapter 3), where T_1 is the lattice RZ^2 and T_2 is its coset $RZ^2 + (0, 1)$. This is illustrated in Figure 4.17, and we see that this construction yields D_4 , the Schläfli lattice, whose points have an even coordinate sum. (The points in RZ^2 have even coordinate sums and the points in its coset $RZ^2 + (0, 1)$ have odd coordinate sums.)

Note that in applying the squaring construction to Z^2/D_2 we have obtained another lattice, D_4 , i.e., any two points $d_1, d_2 \in D_4$ can be added as 4-dimensional vectors to produce $d_3 = d_1 + d_2 \in D_4$, another point in the lattice. That this is so can be seen by inspection of Figure 4.17. This property results of course from the fact that RZ^2 is a subgroup of Z^2 under vector addition (i.e., a sublattice), inducing the group partition Z^2/RZ^2 . Our lattice examples have therefore more algebraic structure than strictly needed for the squaring construction.

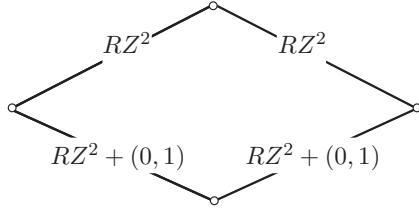


Figure 4.17: Using the squaring construction to generate D_4 from the partition Z^2/D_2 .

The squaring construction is an important tool for the following reason, stated in the following lemma.

Lemma 4.25 *Given the distances $d(T)$ and $d(S)$ in the partition S/T , the minimum distance $d(U)$ of U obeys*

$$d(U) = \min[d(T), 2d(S)]. \quad (4.109)$$

Proof: There are two cases we need to examine:

- (a) If the two elements u_1, u_2 achieving $d(U)$ have their first components, $t_1, t_2 \in T_j$, $t_1 \neq t_2$, in the same subset, i.e., $u_1 = (t_1, s_1)$, $u_2 = (t_2, s_2)$, where $s_1 = s_2$, then their distance $|u_1 - u_2|^2 = d(T)$, and, consequently, $d(U) = d(T)$.
- (b) If their first components t_1, t_2 lie in different subsets T_j and T_i , their second components s_1, s_2 must lie in the same two respective subsets by virtue of the squaring construction, i.e., $s_1 \in T_j$ and $s_2 \in T_i$. Since $T_j \neq T_i \Rightarrow s_1 \neq s_2$. Likewise, $t_1 \neq t_2$, and, since t_1, t_2, s_1 , and s_2 can be chosen such that $d(t_1, t_2) = d(S)$ and $d(s_1, s_2) = d(S)$, we conclude that $d(U) = 2d(S)$ in this case. This proves the lemma. Q.E.D.

We now extend the squaring construction to two levels. Let $S/T/V$ be a two-level partition chain, that is, each subset T_j is made up of P disjoint subsets V_{ji} . Let $T_j \times T_j = |T_j|^2$ be the set of all elements $(t_1, t_2); t_1, t_2 \in T_j$. Then each $|T_j|^2$ can be represented by the trellis in Figure 4.18, i.e., each element $t_1 \in V_{ji}$ can be combined with a second element t_2 from any $V_{ji} \in T_j$, and we have a 2-stage trellis with P states representing $|T_j|^2$ in terms of its subsets V_{ji} . Since each $|T_j|^2$ can be broken down into a trellis like the one in Figure 4.18, we are led to the two-level representation of the set U shown in Figure 4.19. From the above, it should be straightforward to see that Figures 4.16 and 4.19 show the same object with different degrees of detail.

There is another way of looking at Figure 4.18. The paths to the first node at the second stage (solid paths in the figure) are in fact the set obtained by the squaring construction

of the partition T_j/V_j , denoted by W_j . Obviously, W_j is a subset of $|T_j|^2$. In fact, $|T_j|^2$ is the union of P sets W_{ji} , each of which, except for W_j , is obtained from a squaring-type construction similar to the squaring construction, but with the indices of the second component sets permuted cyclically. This is called a *twisted squaring construction*.

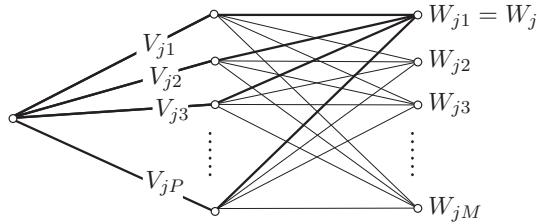


Figure 4.18: Representation of $|T_j|^2 = T_j \times T_j$, using the second level partition T/V .

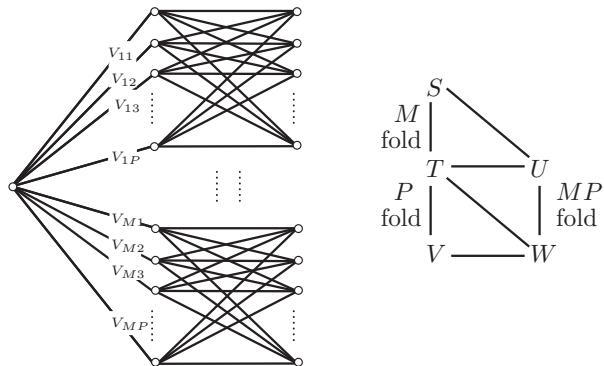


Figure 4.19: Representation of the original squaring construction refined to the two-level partition $S/T/V$ and schematic representation of the induced partition U/W .

This then induces an MP -partition of the set U , denoted by U/W , as shown in Figure 4.19; i.e., the set U is partitioned into MP disjoint sets W_{ji} . The right-hand side of Figure 4.19 shows a schematic diagram of this construction. The partition chain $S/T/V$ induces a partition U/W via the squaring construction, whereby U is obtained from the squaring construction $|S/T|^2$ and W from $|T/V|^2$. Since the partition S/T is M -fold and the partition T/V is P -fold, the resulting partition U/W is MP -fold.

We now apply the squaring construction to the new partition U/W and obtain a *two-level squaring construction* (Figure 4.20). This is done by concatenating back to back two trellis sections of the type in Figure 4.19, just as in the one-level squaring construction. We denote the two-level squaring construction by $|S/T/V|^4$, which is, in fact, identical to the one-level squaring construction $|U/W|^2$. Using Lemma 4.25 twice, we get

$$\begin{aligned} d(|S/T/V|^4) &= \min[d(W), 2d(U)] \\ &= \min[d(V), 2d(T), 4d(S)]. \end{aligned} \quad (4.110)$$

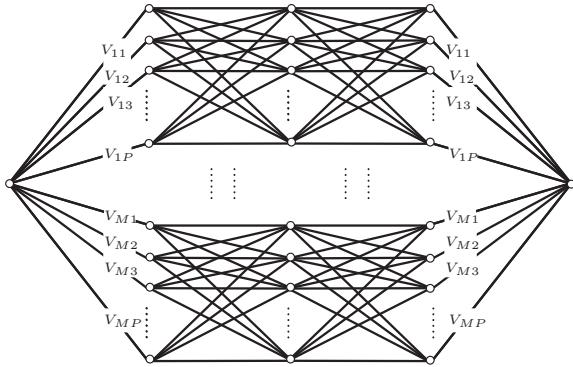


Figure 4.20: Two-level squaring construction for the partition $S/T/V$.

As an example of a two-level squaring construction, we start with the binary lattice partition chain, $Z^2/RZ^2/2Z^2$. The resulting lattice⁷ has eight dimensions and its 4-state trellis diagram is shown in Figure 4.21. Its minimum squared Euclidean distance between lattice points is $d(|Z^2/RZ^2/2Z^2|^4) = 4$. This is the famous Gosset lattice E_8 [9].

This game of two-level squaring constructions may now be continued with the newly constructed lattice, i.e., we can recursively construct the lattices $\Lambda(n) = |\Lambda(n-1)/R\Lambda(n-1)|^2 = |\Lambda(n-2)/R\Lambda(n-2)/2\Lambda(n-2)|^4$ starting with $D_4 = \Lambda(1) = |Z^2/RZ^2|^2$ and $E_8 = \Lambda(2) = |Z^2/RZ^2/2Z^2|^4$. The sequence of these lattices is known as the sequence of *Barnes-Wall* lattices $\Lambda_N = \Lambda(n)$ of dimension $N = 2^{n+1}$ and minimum squared Euclidean distance $2^n = N/2$ [16, 9].

The minimum squared distance results from the squaring construction, i.e., $d(\Lambda_N) = \min(d(R\Lambda_{N/2}), 2d(\Lambda_{N/2})) = \min(d(2\Lambda_{N/4}), 2d(R\Lambda_{N/4}), 4d(\Lambda_{N/4}))$. The distance sequence now follows by induction, starting with $d(\Lambda_2 = Z^2) = 1$, and $d(\Lambda_4 = D_4) = 2$ and using the fact that $d(R\Lambda) = 2d(\Lambda)$, i.e., $d(\Lambda_N) = 2d(\Lambda_{N/2}) = 4d(\Lambda_{N/4})$, and hence $d(\Lambda_N) = N/2$.

⁷The fact that the resulting union set is also a lattice is argued analogously to the case of D_4 .

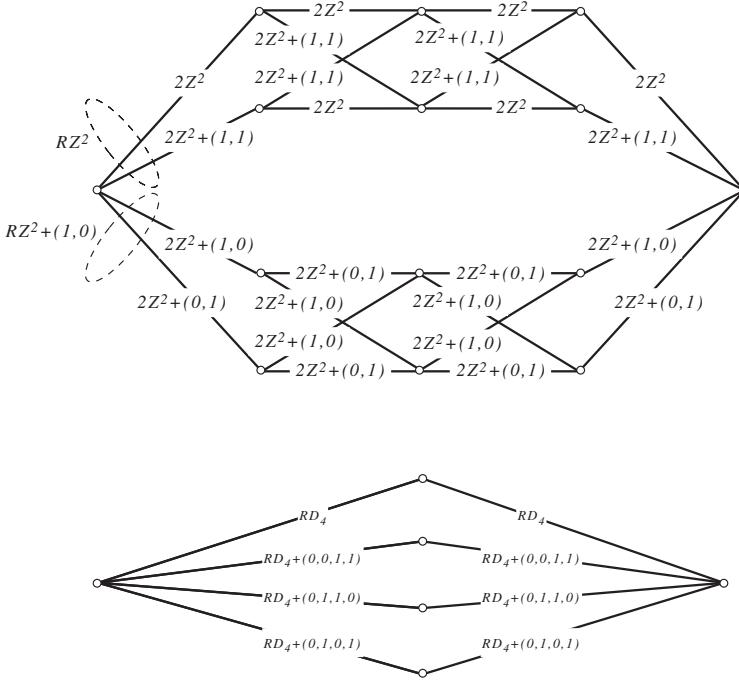


Figure 4.21: Gosset lattice E_8 obtained via the two-level squaring construction. The subsets are the cosets $Z_2 + \mathbf{c}$, where \mathbf{c} is the coset representative (Section 3.5). The lower figure shows E_8 as a single-level squaring construction from the partition $U/V = D_4/RD_4$.

Note that, as can be seen from the squaring construction, Λ_N has the same minimum distance as $R\Lambda_{N/2} \times R\Lambda_{N/2}$, namely $N/2$, but has $2^{N/4}$ times as many lattice points ($2^{N/4}$ is also the number of cosets in the $\Lambda_{N/2}/R\Lambda_{N/2}$ partition). The asymptotic coding gain of Λ_N , given by $\gamma(\Lambda_N) = d(\Lambda_N)/V(\Lambda_N)^{2/N}$ (see Equation (3.9)), is therefore $2^{1/2}$ times that of $R\Lambda_{N/2} \times R\Lambda_{N/2}$, which is also the coding gain of $\Lambda_{N/2}$. Starting with $\gamma(Z^2) = 1$ this leads to an asymptotic coding gain of the Barnes-Walls lattices given by $\gamma = 2^{n/2} = N/2$, which increases without bound. The relationships and construction of the infinite sequence of Barnes-Walls lattices is schematically represented in Figure 4.22, which also includes the relevant parent lattices.

Since the partitions $\Lambda_N/R\Lambda_N$ and $R\Lambda_N/2\Lambda_N$ are both of the order $2^{N/2}$, the number of states in the trellis diagram for Λ_{4N} resulting from the two-level squaring construction

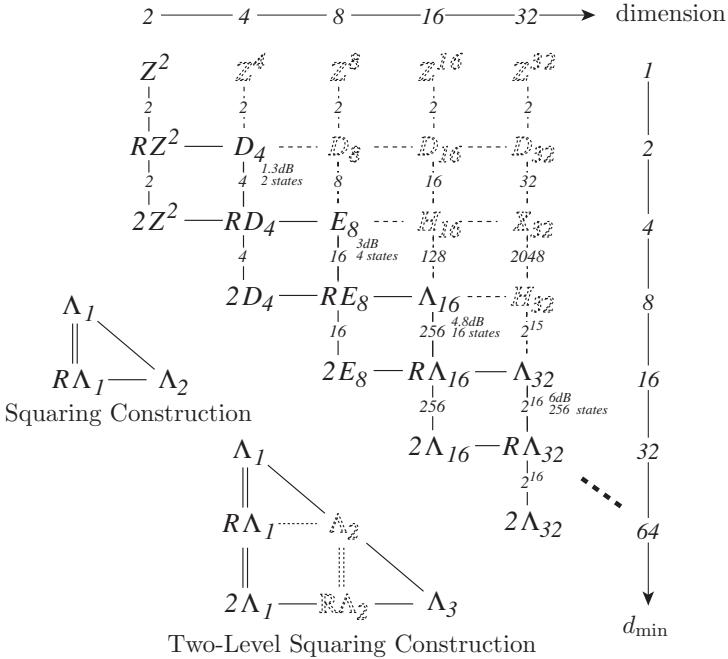


Figure 4.22: Schematic representation of the relationships of the infinite sequence of Barnes-Walls and related lattices. The small numbers indicate the order of the lattice partition $|\Lambda/\Lambda'|$. The asymptotic coding gain and the number of trellis states is also given for the main sequence of lattices.

is 2^N , and therefore the number of states in the trellis of the Barnes-Walls lattices Λ_N is given by $2^{N/4} = 2^{2^{n-1}}$, which grows exponentially with the dimension N . Thus we have 2 states for D_4 , 4 states for E_8 , 16 states for Λ_{16} , and then 256, 65,536, and 2^{64} states for $\Lambda_{32}, \Lambda_{64}$, and Λ_{128} , respectively; and the number of states, or the complexity of the lattice, also increases without bound.

If we want to use E_8 as a code for a telephone line modem, we would choose quadrature modulation and transmit four 2-dimensional signals to make up one 8-dimensional code word or lattice point. A typical baud rate over telephone channels is 2400 symbols/second (baud). To build a modem which transmits 9600 bits/s, we require 16 signal points every two dimensions, and the total number of signal points from E_8 is $2^{16} = 65,536$. In order

to transmit 14,400 bits/s, we already need 2^{24} signal points, approximately 17 million. It becomes evident that efficient decoding algorithms are needed since exhaustive look-up tables clearly become infeasible. The trellis structure of the lattices provides an excellent way of breaking down the complexity of decoding, and we will see in Section 4.15 that a decoder for the E_8 lattice becomes rather simple indeed. Further aspects of implementation of lattice modems are discussed by Lang and Longstaff [30] and by Conway and Sloane [10].

4.14 The Construction of Reed–Muller Codes

If instead of the integer lattices, we use the binary field $\text{GF}(2)$ with $\{0, 1\}$, we can apply the same squaring construction to build the entire class of Reed–Muller codes.

We start with the two-level partition chain $(2, 2)/(2, 1)/(2, 0)$ of length-two binary codes. $(2, 2)$ is the binary code over $\text{GF}(2)$ consisting of all four length-2 binary code words. $(2, 1)$ is its subcode consisting of the code words $\{(0, 0), (1, 1)\}$, and $(2, 0)$ is the trivial single code word $(0, 0)$. Since $(2, 1)$ is a subgroup of $(2, 2)$ under vector addition over $\text{GF}(2)$, $(2, 2)/(2, 1)$ is a true partition, and the same holds for $(2, 1)/(2, 0)$. We now define [16] the Reed–Muller codes $\text{RM}(r, n)$ with parameters r, n recursively as

$$\text{RM}(r, n) = |\text{RM}(r, n - 1)/\text{RM}(r - 1, n - 1)|^2 \quad (4.111)$$

and start the recursion with $\text{RM}(-1, 1) = (2, 0)$, $\text{RM}(0, 1) = (2, 1)$, and $\text{RM}(1, 1) = (2, 2)$. The first two codes constructed are illustrated in Figure 4.23 below. The code $\text{RM}(1, 2) = |(2, 2)/(2, 1)|^2$ is a $(4, 3)$ single parity-check code, and the code $\text{RM}(0, 2) = |(2, 1)/(2, 0)|^2$ is the $(4, 1)$ repetition code.

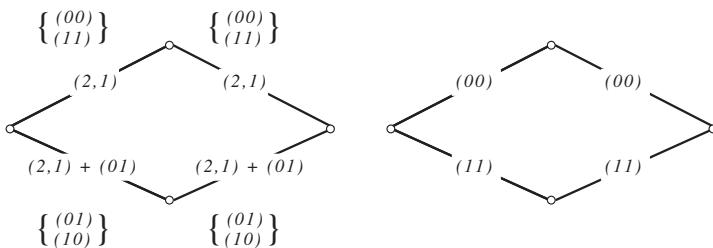


Figure 4.23: The construction of $\text{RM}(1, 2)$ and $\text{RM}(0, 2)$ via the squaring construction from the binary codes $(2, 2)$, $(2, 1)$, and $(2, 0)$.

Figure 4.24 shows the construction table for the Reed–Muller codes. Every column is completed at the top with $\text{RM}(n, n)$, the binary $(2^n, 2^n)$ code consisting of all length- 2^n

binary vectors, and at the bottom with $\text{RM}(-1, n)$, the binary code consisting of the single codeword $(0, \dots, 0)$ of length 2^n . The partition orders are also indicated in the figure.

We now need to determine the code parameters of the $\text{RM}(r, n)$ codes. Their length N equals 2^n , and, from Lemma 4.25, we establish recursively the minimum distance $d_{\min} = 2^{n-r}$. The code rate is found as follows. The partition order of $|\text{RM}(r, n)/\text{RM}(r-1, n)|$, denoted by $m(r, n)$, follows from the squaring construction and obeys the recursion

$$m(r, n) = m(r, n-1)m(r-1, n-1). \quad (4.112)$$

Starting this recursion with $m(1, 1) = 2$ and $m(0, 1) = 2$ leads to the partition numbers in Figure 4.24. Converting to logarithms, i.e., $M(r, n) = \log_2(m(r, n))$, (4.112) becomes

$$M(r, n) = M(r, n-1) + M(r-1, n-1), \quad (4.113)$$

which, after initialization with $M(1, 1) = 1$ and $M(0, 1) = 1$, generates Pascal's triangle, whose numbers can also be found via the combinatorial generating function $(1+x)^n$. The information rates of the $\text{RM}(r, n)$ codes are now found easily. The rate of $\text{RM}(r, n)$ is given by the rate of $\text{RM}(r-1, n)$ plus $M(r, n)$. This generates the code rates indicated in Figure 4.24.

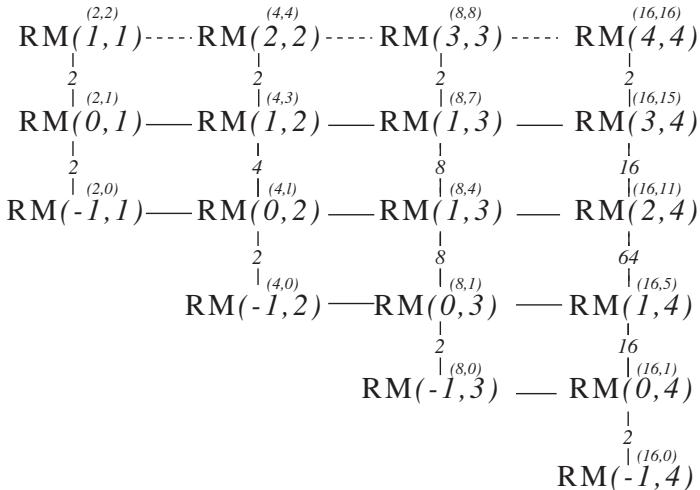


Figure 4.24: Diagram for the construction of the Reed–Muller codes via the squaring and two-level squaring constructions.

Among the general class of Reed–Muller codes we find the following special classes:

- $\text{RM}(n-1, n)$ are the single parity-check codes of length 2^n .
- $\text{RM}(n-2, n)$ is the family of extended Hamming codes of length 2^n with minimum distance $d_{\min} = 4$ [16, 9].
- $\text{RM}(1, n)$ are the first-order Reed–Muller codes of length $N = 2^n$, rate $R = (n+1)/N$ and minimum distance $d_{\min} = N/2$ [9].
- $\text{RM}(0, n)$ are the repetition codes of length 2^n .

Figure 4.25 below shows the four-section trellis diagrams of $\text{RM}(2, 4)$, a $[16, 11]$ code, and of $\text{RM}(1, 4)$, a $[16, 5]$ code.

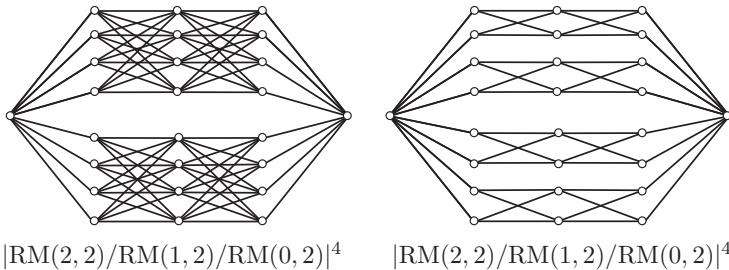


Figure 4.25: The trellis diagrams of $\text{RM}(2, 4)$ and $\text{RM}(1, 4)$, constructed via the two-level squaring construction.

4.15 A Decoding Example

We will discuss general trellis decoding procedures at length in Chapter 7 and all of those methods are applicable to trellis decoding of block codes. However, the trellises of the codes and lattices constructed in the preceding sections lend themselves to more efficient decoder implementations due to their regular structure. The construction also resulted in efficient trellises in terms of their numbers of states. The extended Hamming codes constructed in Section 4.14 above have $N/2$ states, while the PC-trellises of the original and the extended Hamming codes, constructed according to Section 4.3, have $2^{n-k} = N$ states, a savings of a factor of 2, obtained by taking pairs of coded bits as branch symbols.

Often the trellis decoding operation can be mapped into an efficient decoder. Let us consider the extended Hamming code $\text{RM}(1, 3)$, whose trellis is shown in Figure 4.26. Note that the trellis is the same as that of the Gosset lattice E_8 in Figure 4.21, and, after

rearranging the states at time $r = 1$, identical to the trellis in Figure 4.9. Let us assume that, as is the usual practice in telecommunications, zeros are mapped into -1 's and 1's are retained, i.e., we map the output signals $\{0, 1\}$ of the code into a BPSK signal set and we obtain the modified code words \mathbf{x}' from the original \mathbf{x} . With this the decoder now operates as follows. For each of the signals y_i from the received signal vector $\mathbf{y} = (y_1, \dots, y_8)$, two metrics need to be computed: one is $(y_i - 1)^2$ and the other is $(y_i + 1)^2$, or equivalently $m_0 = y_i$ and $m_1 = -y_i$. We see that m_0 and m_1 are negatives of each other, and only one must be calculated. Since the all-one vector $\mathbf{x} = (1, \dots, 1)$ is a codeword, the negative of every modified codeword \mathbf{x}' is itself a codeword, as can be seen by applying the linearity of the code and adding \mathbf{x} to that modified codeword. The sign of the metric sum can now be used to identify the sign of the codeword.

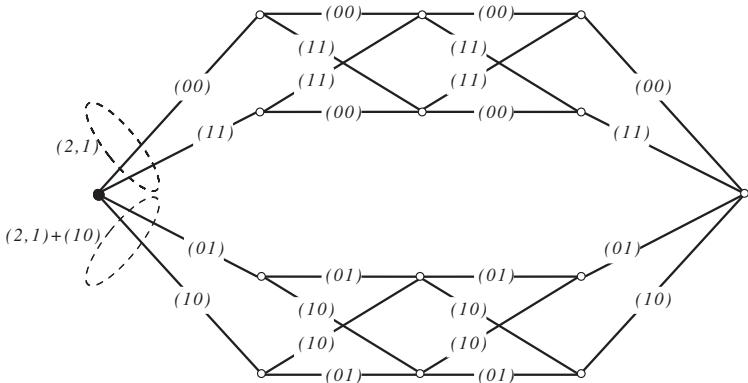


Figure 4.26: Trellis of the $[8, 4]$ Extended Hamming code. This is also the trellis of the Gosset lattice from Figure 4.21 as can be seen by comparing Figures 4.22 and 4.24.

The operation of the decoder is illustrated in Figure 4.27, where the operator \mathcal{O} produces the sum and difference of the inputs, i.e., the outputs of the first operator box are $y_1 + y_2$ and $y_1 - y_2$. The first two stages in the decoder decode the first and the second trellis sections, and the sign indicates whether the codeword or its complement are to be selected. After the absolute magnitude operation, the two sections are combined. This is done by the last section of the decoder. Finally, the selection decides through which of the four central states the best codeword leads. Since the components have already been decoded, this determines the codeword with the best total metric.

In order to use the above decoder to decode E_8 , one additional initial step needs to be

added. According to Figure 4.21 and Figure 3.14, the transition of a “0” corresponds to transmitting an odd integer, and the transition of a “1” corresponds to transmitting an even integer. For every received y_i , we therefore need to find the closest even integer I_{ei} and the closest odd integer I_{oi} . These integers are stored and used later to reconstruct the exact lattice point. The metrics $m_{0i} = (I_{oi} - y_i)^2$ and $m_{1i} = (I_{ei} - y_i)^2$ are then used in the algorithm above to find the most likely sequence of sublattices via a decoding operation which is identical to that of the extended Hamming code. The decoded sublattice sequence determines now if the odd or even integers are retained, i.e., the decoded sublattices $Z^2 \equiv (I_{oi}, I_{o(i+1)})$, $Z^2 + (1, 1) \equiv (I_{ei}, I_{e(i+1)})$, $Z^2 + (1, 0) \equiv (I_{ei}, I_{o(i+1)})$, and $Z^2 + (0, 1) \equiv (I_{oi}, I_{e(i+1)})$. This sequence of decoded integers identifies the most likely lattice point.

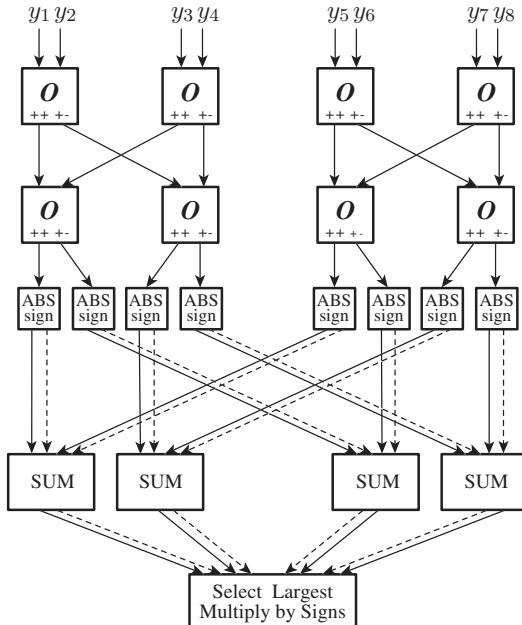


Figure 4.27: Schematic diagram of an efficient decoder for the [8, 4] Extended Hamming code. The operator \mathcal{O} produces the sum $y_i + y_j$ and the difference $y_i - y_j$ of the inputs, ABS forms the absolute value and SUM adds $y_i + y_j$.

Finally we wish to refer the interested reader to more in-depth material on this subject available as recent books by Lin et al. [32] and by Honary and Markarian [21].

4.16 Polar Codes and Their Relationship to RM Codes

RM codes are binary linear codes of length N with dimension K , and their generator matrices are derived from a common set of basis vectors. Let us therefore construct this set as the n th Kronecker power of the basic *kernel matrix*

$$F_2 = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}, \quad (4.114)$$

that is, we generate $F_2^{\otimes n}$. For example,

$$F_2^{\otimes 3} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \quad (4.115)$$

and clearly, the columns form an independent set of basis vectors. We now define $R_N = F_2^{\otimes n}$ as our base matrix, and the Reed–Muller codes as well as the Polar codes are based on R_N . RM codes were discovered by Muller [44], but a decoding algorithm was first invented by Reed [50]. Polar codes, on the other hand, were discovered by Arikan [1] more than 50 years later. These two ways of deriving codes from the same foundation elegantly illustrate the shift in perspective that has taken place over the last six decades and which is a major thread of this book.

Define the N -tuple $\mathbf{v}_i = \mathbf{r}_{2^i}$, $i \in [0, n-1]$, where \mathbf{r}_k is the k th row of R_N . So, for $n=3$ we obtain

$$\begin{aligned} \mathbf{v}_1 &= [10101010], \\ \mathbf{v}_2 &= [11001100], \\ \mathbf{v}_3 &= [11110000]. \end{aligned}$$

Furthermore, the code generators also use the boolean products $\mathbf{v}_k \cdot \mathbf{v}_l$, where the products are taken componentwise as logic AND operation. In this fashion the RM codes are generated by the basis vectors consisting of the products of \mathbf{v}_k up to degree r , i.e.,

$$\text{RM}(r, n) = \{\mathbf{v}_0, \mathbf{v}_1, \dots, \mathbf{v}_n, \mathbf{v}_1 \cdot \mathbf{v}_2, \dots, \mathbf{v}_{n-1} \mathbf{v}_n, \dots, \text{higher products}\}. \quad (4.116)$$

By inspection we can see that this construction generates the same basis vectors as the squaring construction from the previous section.

For example, the generator matrix for RM(3, 2) is then given by

$$\mathbf{G}_{\text{RM}(3,2)} = \begin{bmatrix} \mathbf{v}_0 \\ \mathbf{v}_1 \\ \mathbf{v}_2 \\ \mathbf{v}_3 \\ \mathbf{v}_1\mathbf{v}_2 \\ \mathbf{v}_1\mathbf{v}_3 \\ \mathbf{v}_2\mathbf{v}_3 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}, \quad (4.117)$$

which is, of course, the [8,7] parity check code as discussed above.

The row selection in the Reed–Muller codes are based on the weight of the rows in the matrix R_N , and this leads to asymptotically unreliable codes. The polar code construction below proceeds with a different row selection principle.

Polar coding operates with bit channels, so let us introduce the basic channel model

$$y = W(x), \quad (4.118)$$

where W stands for any binary input channel, such as a simple erasure channel for example. This channel has a mutual information denoted by $I_W(x; y)$, which in the case of an erasure channel is simple $I_W(x; y) = 1 - \epsilon$, the erasure probability. In the case of symmetric output channels, $I_W(x; y)$ coincides with the capacity of the channel, which then requires a symmetric input distribution on x .

The basis function generation matrix R_N is built recursively according to (4.115), so we recreate this process with the binary input channels attached. This is shown in Figure 4.28 for the first two stages of the process. On the left hand, the input bits u_1 and u_2 are passed through F_2 , then each output bit is send through an identical channel W with mutual information I_W . We now consider the two virtual channels $u_1 \rightarrow \mathbf{y}$ and $u_2 \rightarrow \mathbf{y}$ and compute their mutual information as

$$I([u_1, u_2]; \mathbf{y}) = I(u_2; \mathbf{y}|u_1) + I(u_1; \mathbf{y}). \quad (4.119)$$

Technically, this decomposition creates two synthetic channels, whose combined capacity is equal to $2I_W$. If our example channel W is a BEC with erasure probability 0.5, then $I(u_2; \mathbf{y}|u_1) = 0.75$, and $I(u_1; \mathbf{y}) = 0.25$. This can be seen by drawing out the two channels and realizing that for $I(u_1; \mathbf{y})$ the output signal is statistically independent of u_1 with probability 0.75 and therefore all the corresponding outputs are represented by an erasure of u_1 . On the other hand, the channel with the higher capacity requires knowledge of u_1 , the bit that passes through the lower-capacity channel. If we choose to “freeze” that bit u_1 to a fixed level – either 0 or 1, the channel for the bit u_2 becomes useable.

This basic observation is at the core of the phenomenon of channel polarization which is exploited in polar codes. Arikan [1] showed that, in general, this channel splitting

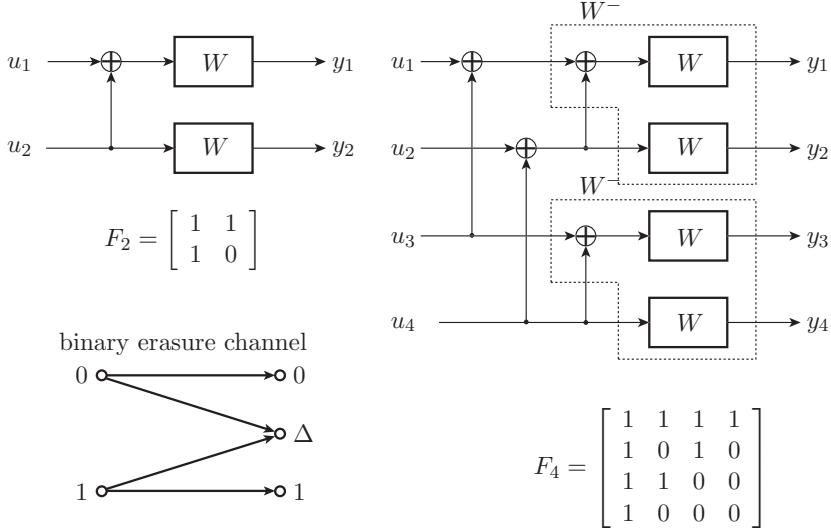


Figure 4.28: Encoding principle of polar codes with the binary input channels W . The binary erasure channel (BEC) model shown is used in the text to illustrate the concepts.

generates a lower-capacity channel W^- and a higher-capacity channel W^+ , whose sum capacity equals the original channel capacity, such that

$$I(W^-) \leq I(W) \leq I(W^+). \quad (4.120)$$

Furthermore, in this combination we have $I(W^-) = I^2(W)$.

If we study Figure 4.28 carefully, we note that the splitting argument is recursive. In F_4 , two channels W^- , and two channels W^+ are combined into a total of four channels, denoted by $W^{--}, W^{-+}, W^{+-}, W^{++}$. We note that the combining of the two W^- channels is separate from the combining of the two W^+ channels, and that the channel combination can therefore be applied recursively; from this we obtain $I(W^{++}) = 9375, I(W^{+-}) = 0.5625, I(W^{-+}) = 0.4375, I(W^{--}) = 0.0625$, for a total sum of $4I(W) = 2$ bits/channel use.

Figure 4.29, presented in [45], develops this recombination further recursively for blocks of length $N = 2^n$, following the pattern established by F_{2^n} . The figure nicely illustrates the effect called “polarization,” whereby the binary channels synthesized through F_{2^n} separate into “good channels” and “bad channels,” that is, into channels with a mutual information approaching unity and also channels whose mutual information approaches

zero. The mutual information of the respective channels can be computed recursively from the relationships $I(W^+) = 2I(W) - I^2(W)$, and $I(W^-) = I^2(W)$. As $N \rightarrow \infty$ this leads to the spread of mutual information values shown in Figure 4.29, and Arikan [1] showed that the proportion of channels whose mutual information goes to unity is exactly equal to $I(W) = \delta$, that is, the erasure probability of the channel.

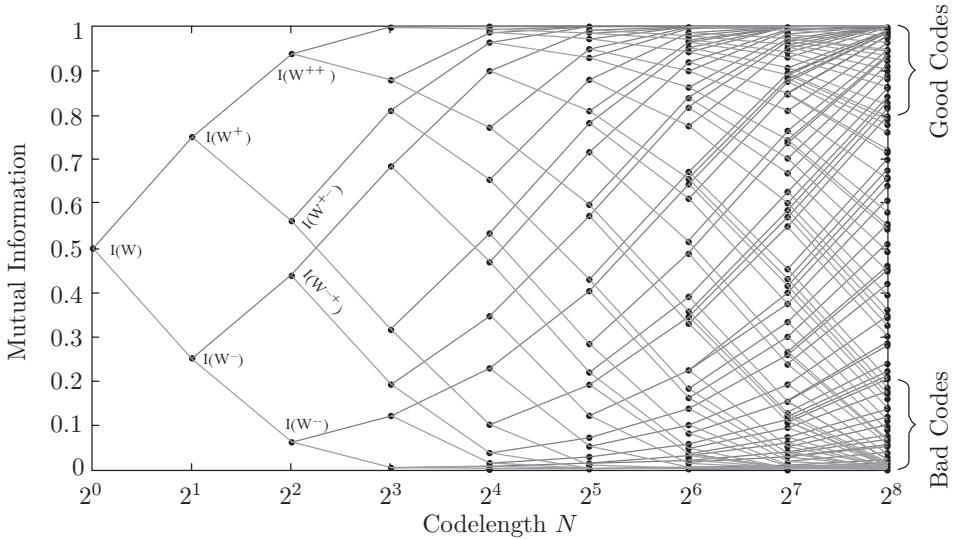


Figure 4.29: Illustration of the channel polarization effect for a BEC with $\delta = 0.5$.

As N increases, the channels become increasingly polarized, meaning that they converge to either channels with mutual information $\rightarrow 1$ or $\rightarrow 0$, and the proportion of channels with an intermediate mutual information goes to zero. This is illustrated in Figure 4.30, which plots the cumulative probability function of the Bhattacharyya bound $Z(W_n)$ for the polarized channels for several n . $Z(W_n)$ is related to $I_{W_n} \geq \sqrt{1 - Z^2(W_n)}$ and was used in [1] to derive the polarization effect. If $I(W_n) = 1$, $Z(W_n) = 0$, and $I(W_n) = 0 \rightarrow Z(W_n) = 1$. Consequently polarization of the Bhattacharyya bound $Z(W_n)$ implies polarization of the mutual information $I(W_n)$.

In general, Arikan [1] shows that the proportion of good channels with a mutual information that goes to unity is given by $I(x; y)$ for any binary symmetric memoryless channels. A polar code now consists of sending binary bits only through those good channels, and not using the bad channels, or more precisely, the bad channels are frozen to a specific predetermined value.

This selection of the good channels is the encoding process for a *polar code*, or equiv-

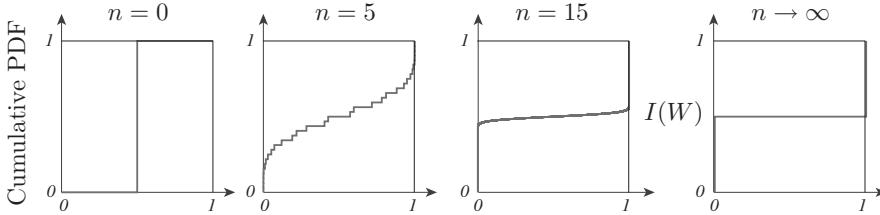


Figure 4.30: Statistical illustration of polarization via the cumulative PDF of the Bhattacharyya parameter $Z(W_n)$. For $n = 0$, the channel has only a value $Z(W_n)$, while for $n \rightarrow \infty$, $Z(W_n)$ has a bi-modal distribution with probability mass at 0 and 1, source [19].

alently is what defines the code. Formally, encoding consists of selecting an information vector \mathbf{u} , by populating the unfrozen positions with the K information bits, and setting the other postions to zero, for example. The codeword \mathbf{x} is now found via $\mathbf{x} = \mathbf{u}F_N$. We note that this operation is in essence the Hadamard transform, which can be carried out via the fast Hadamard transform (in structure identical to the fast fourier transform). Encoding can therefore be done with a complexity of order $\mathcal{O}(N \log(N))$, or $\mathcal{O}(\log(N))$ operations per bit.

Constructing a polar code can be challenging, however. In the case of BEC channels, the synthetic channel capacities of the individual bit channels can be computed recursively as illustrated in Figure 4.28 with an effort of order $\mathcal{O}(N \log(N))$, however, for other channels the complexity of identifying the good channels grows exponentially with codelength. Density evolution—discussed in detail in Chapter 6—has been used for polar code construction in [43, 56]. A lower complexity version of DE, using the Gaussian approximation of densities, has also been applied, see [58].

A last point we wish to mention is that the construction of polar codes is dependent on the actual channel and the channel parameter, such as the erasure rate. These codes are therefore not “universal” codes, and conceptually would have to be redesigned for each value of the signal-to-noise ratio or erasure rate. This is, however, not a serious impediment in practice.

As we have seen in the derivation of these codes, the channel decomposition leads to channels which are conditioned on the knowledge of previous symbols, such as the example of $I(u_2; \mathbf{y}|u_1)$ in (4.119), which in general will have the form $I(u_k; \mathbf{y}|[u_1, \dots, u_{k-1}])$. As long as all the conditioning binary symbols $[u_1, \dots, u_{k-1}]$ are from frozen positions, the resulting channel mutual information directly represent that maximal rate that can be achieved on this channel. However, in general, parts of $[u_1, \dots, u_{k-1}]$ will be information bearing binary digits themselves. In light of this, it is not too surprising that the decoding method proposed by Arikan [1] is of a recursive cancellation nature; that is, previously

computed symbols $[u_1, \dots, u_{k-1}]$ are used to define the channel for u_k . The resulting successive cancelation algorithm has much in common with the graph-based iterative decoding algorithm discussed in subsequent chapters of this book, and we therefore relegate a discussion of polar code decoding to Chapter 9.

Suffice to say that successive or partially parallel cancelation algorithms are the state-of-the art decoding methods for polar codes, and in recent years have made great progress to the point where fast algorithms have been implemented in VLSI.

Appendix 4.A

We will generate a sequence of matrix operations which turn \mathbf{P} into the diagonal matrix $\mathbf{\Gamma}$ according to Theorem 4.16 (see also [26, Section 3.7], or [15]).

First, let us define some elementary matrix operations of size either $k \times k$ or $n \times n$, depending on whether we premultiply or postmultiply \mathbf{P} . The first such operation is $\mathbf{T}_{ij}(b) = \mathbf{I} + b\mathbf{E}_{ij}$, where \mathbf{I} is the identity matrix, \mathbf{E}_{ij} is a matrix with a single 1 in position (i, j) ; $i \neq j$ and 0's elsewhere, and $b \in R$, the principal ideal domain.⁸ $\mathbf{T}_{ij}(b)$ is invertible in R , since

$$\mathbf{T}_{ij}(b)\mathbf{T}_{ij}(-b) = (\mathbf{I} + b\mathbf{E}_{ij})(\mathbf{I} + (-b)\mathbf{E}_{ij}) = \mathbf{I}. \quad (4.121)$$

Left multiplication of \mathbf{P} by a $k \times k$ matrix $\mathbf{T}_{ij}(b)$ adds b times the j th row to the i th row of \mathbf{P} , leaving the remaining rows unchanged. Right multiplication of \mathbf{P} by an $n \times n$ matrix $\mathbf{T}_{ij}(b)$ adds b times the i th column to the j th column of \mathbf{P} .

Next, let u be a unit in R , and define $\mathbf{D}_i(u) = \mathbf{I} + (u - 1)\mathbf{E}_{ii}$, with element u on the i th position on the diagonal. Again, $\mathbf{D}_i(u)$ is invertible with inverse $\mathbf{D}_i(1/u)$. Left multiplication with $\mathbf{D}_i(u)$ multiplies the i th row of \mathbf{P} by u , while right multiplication multiplies the i th column by u .

Finally, define $\mathbf{Q}_{ij} = \mathbf{I} - \mathbf{E}_{ii} - \mathbf{E}_{jj} + \mathbf{E}_{ij} + \mathbf{E}_{ji}$, and \mathbf{Q}_{ij} is its own inverse. Left multiplication by \mathbf{Q}_{ij} interchanges the i th and j th rows of \mathbf{P} , while right multiplication interchanges the i -th and j -th rows, leaving the remainder of the matrix unchanged.

Last, define

$$\mathbf{U} = \begin{bmatrix} x & s \\ y & t \\ & 1 & & 0 \\ & & \ddots & \\ 0 & & & 1 \\ & & & & 1 \end{bmatrix}, \quad (4.122)$$

where x, y, s , and t will be chosen such that the submatrix $\begin{bmatrix} x & s \\ y & t \end{bmatrix}$ is invertible.

⁸It might be helpful to think in terms of integers, i.e., $R = \mathbf{Z}$.

Let us start now, and assume $p_{11} \neq 0$ (otherwise bring a non-zero element into its position via elementary Q_{ij} operations), and assume further that p_{11} does not divide p_{12} (otherwise, again, move such an element into its position via elementary operations). Now, according to Euclid's division theorem [26], there exist elements $x, y \in R$, such that

$$p_{11}x + p_{12}y = \gcd(p_{11}, p_{12}) = d, \quad (4.123)$$

where d , the greatest common divisor of p_{11} and p_{12} , can be found via Euclid's algorithm. Let $s = p_{12}/d$ and $t = -p_{11}/d$ in (4.122), which makes it invertible, i.e.,

$$\begin{bmatrix} x & p_{12}/d \\ y & -p_{11}/d \end{bmatrix}^{-1} = \begin{bmatrix} p_{11}/d & p_{12}/d \\ y & -x \end{bmatrix}. \quad (4.124)$$

Multiplying \mathbf{P} on the right by this matrix gives a matrix whose first row is $(d, 0, p_{13}, \dots, p_{1k})$. If d does not divide all elements of the new matrix, move such an element into position $(2, 1)$ via elementary matrix operation and repeat the process until p'_{11} divides all p'_{ij} , for all i, j .

Via elementary matrix operations the entire first row and column can be cleared to zero, leaving the equivalent matrix

$$\mathbf{P}'' = \begin{bmatrix} \gamma_1 & 0 \\ 0 & \mathbf{P}_1'' \end{bmatrix}, \quad (4.125)$$

where $\gamma_1 = p''_{11}$, up to units, and γ_1 divides every element in \mathbf{P}_1'' , since the elementary matrix operations do not affect divisibility.

Iterating this procedure now, continuing with \mathbf{P}_1'' yields

$$\mathbf{P}''' = \begin{bmatrix} \gamma_1 & 0 & 0 \\ 0 & \gamma_2 & 0 \\ 0 & 0 & \mathbf{P}_2''' \end{bmatrix}, \quad (4.126)$$

where the divisibility $\gamma_1|\gamma_2$ is assured since γ_1 divides all elements in \mathbf{P}_1'' , which implies that any elementary matrix operation preserves this divisibility, and γ_2 divides every element in \mathbf{P}_2''' .

We now simply iterate this procedure to obtain the decomposition in Theorem 4.16, in the process producing the desired matrices \mathbf{A} and \mathbf{B} . It can further be shown that this decomposition is unique in the sense that all equivalent matrices yield the same diagonal matrix $\mathbf{\Gamma}$ (see [26], Section 3.7).

Note that this procedure also provides a proof for Theorem 4.16.

Bibliography

- [1] E. Arikan, “Channel polarization: A method for constructing capacity-achieving codes for symmetric binary-input memoryless channels,” *IEEE Trans. Inform. Theory*, vol. 55, no. 7, July 2009, pp. 3051–3073.
- [2] J.B. Anderson and S.M. Hladik, “Tailbiting MAP decoders,” *IEEE J. Select. Areas Commun.*, vol. SAC-16, pp. 297–302, Feb. 1998.
- [3] L.R. Bahl, J. Cocke, F. Jelinek, and J. Raviv, “Optimal decoding of linear codes for minimizing symbol error rate,” *IEEE Trans. Inform. Theory*, vol. IT-20, pp. 284–287, 1974.
- [4] E.R. Berlekamp, *Algebraic Coding Theory*, McGraw-Hill, New York, 1968.
- [5] R.E. Blahut, *Principles and Practice of Information Theory*, Addison-Wesley, Reading, MA, 1987.
- [6] R. Calderbank, G.D. Forney, and A. Vardy, ”Minimal tail-biting trellises: The Golay code and more,” *IEEE Trans. Inform. Theory*, vol. IT-45, no. 5, pp. 1435–1455, July 1999.
- [7] D. Chase, “A class of algorithms for decoding block codes with channel measurement information,” *IEEE Trans. Inform. Theory*, vol. IT-18, pp. 170–182, Jan. 1972.
- [8] G.C. Clark and J.B. Cain, *Error-Correction Coding for Digital Communications*, Plenum Press, New York, 1983.
- [9] J.H. Conway and N.J.A. Sloane, *Sphere Packings, Lattices and Groups*, Springer-Verlag, New York, 1988.
- [10] J.H. Conway and N.J.A. Sloane, “Decoding techniques for codes and lattices, including the Golay code and the Leech lattice,” *IEEE Trans. Inform. Theory*, vol. IT-32, pp. 41–50, 1986.
- [11] D.G. Daut, J.W. Modestino, and L.D. Wismer, “New short constraint length convolutional code construction for selected rational rates,” *IEEE Trans. Inform. Theory*, vol. IT-28, pp. 793–799, Sept. 1982.

- [12] A. Dholakia, *Introduction to Convolutional Codes*, Kluwer Academic Publishers, Boston, MA, 1994.
- [13] P. Elias, “Coding for noisy channels,” *IRE Conv. Rec.*, pt. 4, pp. 37–47, 1955.
- [14] G.D. Forney, Jr., “Generalized minimum distance decoding,” *IEEE Trans. Inform. Theory*, vol. IT-12, pp. 125–131, April 1966.
- [15] G.D. Forney, “Convolutional codes I: Algebraic structure,” *IEEE Trans. Inform. Theory*, vol. IT-16, no. 6, pp. 720–738, Nov. 1970.
- [16] G.D. Forney, “Coset Codes—Part II: Binary lattices and related codes,” *IEEE Trans. Inform. Theory*, vol. IT-34, no. 5, pp. 1152–1187, Sept. 1988.
- [17] M.P.C. Fossorier and S. Lin, “Soft-decision decoding of linear block codes based on ordered statistics,” *IEEE Trans. Inform. Theory*, vol. IT-41, pp. 1379–11396, Sept. 1995.
- [18] Y.S. Han, C.R.P. Hartman, and C.C. Chen, “Efficient priority-first search maximum-likelihood soft-decision decoding of linear block codes,” *IEEE Trans. Inform. Theory*, vol. IT-39, pp. 1514–1523, Sept. 1993.
- [19] S.H. Hassani, *Polarization and Spatial Coupling: Two Techniques to Boost Performance*, PhD Thesis No. 5706, École Polytechnique Fédérale de Lausanne, March 2013.
- [20] F. Hemmati, “Closest coset decoding of $|u|u + v|$ codes,” *IEEE J. Select. Areas Commun.*, vol. SAC-7, pp. 982–988, Aug. 1989.
- [21] B. Honary and G. Markarivan, *Trellis Decoding of Block Codes: A Practical Approach*, Kluwer Academic Publisher, Boston, MA, 1997.
- [22] H. Imai et al. *Essentials of Error-Control Coding Techniques*, Academic Press, New York, 1990.
- [23] T. Kasami et al., “On the optimum bit orders with respect to the state complexity of trellis diagrams for binary linear codes,” *IEEE Trans. Inform. Theory*, vol. IT-39, pp. 242–245, 1993.
- [24] T. Kasami et. al., “On the trellis structure of block codes”, *IEEE Trans. Inform. Theory*, vol. IT-39, pp. 1057–1064, 1993.
- [25] A.B. Kiely, S.J. Dolinar, Jr., R.J. McEliece, L.L. Ekroot, and W. LinL, “Trellis complexity of linear block codes,” *IEEE Trans. Inform. Theory*, vol. IT-42, no. 6, Nov. 1996.
- [26] N. Jacobson, *Basic Algebra I*, Freeman and Company, New York, 1985.
- [27] R. Johannesson and Z.-X. Wan, “A linear algebra approach to minimal convolutional encoders,” *IEEE Trans. Inform. Theory*, vol. IT-39, no. 4, pp. 1219–1233, July 1993.

- [28] R. Johannesson and K.S. Zigangirov *Fundamentals of Convolutional Coding*, IEEE Press, 1999.
- [29] F. Kshischang and V. Sorokine, "On the trellis structure of block codes," *IEEE Trans. Inform. Theory*, vol. IT-41, pp. 1924–1937, Nov. 1995.
- [30] G.R. Lang and F.M. Longstaff, "A Leech lattice modem," *IEEE J. Select. Areas Commun.*, vol. SAC-7, No. 6, Aug. 1989.
- [31] K.J. Larsen, "Short convolutional codes with maximum free distance for rates 1/2, 1/3, and 1/4," *IEEE Trans. Inform. Theory*, vol. IT-19,, pp. 371-372, May 1973.
- [32] S. Lin (Editor) *Trellises and Trellis-Based Decoding Algorithms for Linear Block Codes*, Kluwer Academic Publisher, Dordrecht, 1998.
- [33] S. Lin and Daniel J. Costello, Jr., *Error Control Coding: Fundamentals and Applications*, Prentice Hall, Englewood Cliffs, NJ, 1983.
- [34] N.J.C. Lous, P.A.H. Bours, and H.C.A. van Tilborg, "On maximum likelihood soft-decision decoding of binary linear codes," *IEEE Trans. Inform. Theory*, vol. IT-39, pp. 197–203, Jan. 1993.
- [35] J.H. Ma and J.K. Wolf, "On tailbiting convolutional codes," *IEEE Trans. Commun.*, vol. COM-34, pp. 104–111, Feb. 1986.
- [36] J.L. Massey, "Foundations and methods of channel coding," *Proc. Int. Conf. Inform. Theory and Systems, NTG-Fachberichte* vol. 65, pp. 148–157, 1978.
- [37] J.L. Massey, "Shift-register synthesis and BCH decoding," *IEEE Trans. Inform. Theory*, vol. IT-15, pp. 122-127, Jan. 1969.
- [38] J.L. Massey and M.K. Sain, "Inverses of linear sequential circuits," *IEEE Trans. Computers*, vol. C-17, pp. 310–337, April 1968.
- [39] K.R. Matis and J.W. Modestino, "Reduced-state soft-decision trellis decoding of linear block codes," *IEEE Trans. Inform. Theory*, vol. IT-28, pp. 61–68, Jan. 1982.
- [40] R.J. McEliece, "On the BCJR trellis for linear block codes," *IEEE Trans. Inform. Theory*, vol. IT-42, no. 4, July 1996.
- [41] R. McEliece, *The Theory of Information and Coding* Encyclopedia of Mathematics and its Applications, Addison-Wesley, Reading, MA, 1977.
- [42] F.J. MacWilliams and N.J.A. Sloane, *The Theory of Error Correcting Codes*, North Holland, New York, 1988.
- [43] R. Mori and T. Tanaka, "Performance of polar codes with the construction using density evolution," *IEEE Commun. Lett.*, vol. 13, no. 7, pp. 519–521, July 2009.
- [44] D.E. Muller, "Applications of boolean algebra to switching circuits design and to error detection," *IRE Trans.*, EC-3, pp. 6–12, Sept. 1954.

- [45] K. Niu, K. Chen, J. Lin, and Q.T. Zhang, “Polar codes: Primary concepts and practical decoding algorithms,” *IEEE Comm. Mag.*, pp. 192–203, July 2014.
- [46] J.P. Odenwalder, “Optimal decoding of convolutional codes,” Ph.D. thesis, University of California, Los Angeles, 1970.
- [47] E. Paaske, “Short binary convolutional codes with maximum free distance for rates $2/3$ and $3/4$,” *IEEE Trans. Inform. Theory*, vol. IT-20, pp. 683–689, Sept. 1974.
- [48] J.E. Porath, “Algorithms for converting convolutional codes from feedback to feed-forward form and vice versa,” *Electronic Lett.*, vol. 25, no. 15, pp. 1008–1009, July 1989.
- [49] J.G. Proakis, *Digital Communications*, third edition, McGraw-Hill, New York, 1995.
- [50] I.S. Reed, “A class of multiple-error-correcting codes and their decoding scheme,” *IRE Trans.*, IT-4, pp. 38–49, Sept. 1954.
- [51] M.K. Sain and J.L. Massey, “Invertibility of linear time-invariant dynamical systems,” *IEEE Trans. Automatic Control*, vol. AC-14, pp. 141–149, April 1969.
- [52] J. Snyders and Y. Be’ery, “Maximum likelihood soft decoding of binary block codes and decoders for the Golay codes,” *IEEE Trans. Inform. Theory*, vol. IT-35,, pp. 963–975, Sept. 1989.
- [53] G. Strang, *Linear Algebra and Its Applications*, Harcourt Brace Jovanovich, San Diego, 1988.
- [54] Y. Sugiyama, M. Kasahara, S. Hirasawa, and T. Namekawa, “A method for solving key equation for decoding Goppa codes,” *Inform. Contr.*, vol. 27, pp. 81–99, 1975.
- [55] D.J. Taipale and M.J. Seo, “An efficient soft-decision Reed–Solomon decoding algorithm,” *IEEE Trans. Inform. Theory*, vol. IT-40, pp. 1130–1139, July 1994.
- [56] I. Tal and A. Vardy, “How to construct polar codes,” *IEEE Trans. Inform. Theory*, vol. 59, no. 10, pp. 6562–6582, Oct. 2013.
- [57] T. Taneko, T. Nishijima, H. Inazumi, and S. Hirasawa, “Efficient maximum likelihood decoding of linear block codes with algebraic decoder,” *IEEE Trans. Inform. Theory*, vol. IT-40, pp. 320–327, March 1994.
- [58] P. Trifonov, “Efficient design and decoding of polar codes,” *IEEE Trans. Commun.*, vol. 60, no. 11, pp. 3221–3227, Nov. 2012.
- [59] A. Vardy and Y. Be’ery, “More efficient soft decoding of the Golay codes,” *IEEE Trans. Inform. Theory*, vol. IT-37, pp. 667–672, May 1991.
- [60] A. Vardy and Y. Be’ery, “Maximum likelihood soft decoding of BCH codes,” *IEEE Trans. Inform. Theory*, vol. IT-40, pp. 546–554, March 1994.

- [61] A.J. Viterbi and J.K. Omura, *Principles of Digital Communication and Coding*, McGraw-Hill, New York, 1979.
- [62] E.J. Weldon, Jr., "Decoding binary block codes on q-ary output channels," *IEEE Trans. Inform. Theory*, vol. IT-17, pp. 713–718, Nov. 1971.
- [63] N. Wiberg, H.-A. Loeliger, and R. Kötter, "Codes and iterative decoding on general graphs," *Eur. Trans. Telecommun.*, vol. 6, pp. 513–526, Sept. 1995.
- [64] J.K. Wolf, "Efficient maximum-likelihood decoding of linear block codes using a trellis," *IEEE Trans. Inform. Theory*, vol. IT-24, no. 1, pp. 76–80, Jan. 1978.

Chapter 5

Trellis and Tree Decoding

5.1 Background and Introduction

In this chapter we will study decoding algorithms that operate on the trellis representation of the code. Our objective is to extract the information symbols that were encoded and are carried by the received signals with computationally efficient algorithms that lead to decoders.

There a great variety of decoding algorithms for trellis, some heuristic and some derived from well-defined optimality criteria. Until very recently, the main objective of a decoding algorithm was the successful identification of the transmitted symbol sequence, accomplished a sequence decoder. These sequence decoders fall into two main groups: Tree decoders and trellis decoders. Tree decoders explore the code tree, to be defined below, and their most well-known representatives are the sequential algorithms and limited-size breadth first algorithms, such as the *M-algorithm*. Trellis decoders make use of the more structured trellis of a code, and its main algorithm is the maximum-likelihood sequence detector, also known widely simply as the *Viterbi algorithm*.

Recently, and in conjunction with the emergence of turbo coding, symbol probability algorithms have become important. They calculate the reliability of individual transmitted or information symbols, rather than decoding sequences. Symbol probability algorithms are essential for the iterative algorithms used to decode large concatenated codes, such as turbo codes. Their most popular and widely used representative is the A Posteriori Probability (APP) algorithm, also known as the BCJR algorithm, or the forward–backward algorithm. The APP algorithm works on the trellis of the code, and it is discussed in detail in Section 5.7.

Let us now set the stage for these decoding algorithms. In Chapters 2 and 3 we have discussed how a trellis encoder generates a sequence $\boldsymbol{x}^{(i)} = (x_{-l}^{(i)}, \dots, x_l^{(i)})$ of correlated complex symbols $x_r^{(i)}$ for message i , and how this sequence is modulated, using the pulse

waveform $p(t)$, into the (baseband) output signal

$$s^{(i)}(t) = \sum_{r=-l}^l x_r^{(i)} p(t - rT). \quad (5.1)$$

From Chapter 2 we also know the structure of the optimal decoder for such a system. We have to build a matched filter for each possible signal $s^{(i)}(t)$ and select the message which corresponds to the signal which produces the largest sampled output value.

The matched filter for $s^{(i)}(t)$ is given by

$$s^{(i)}(-t) = \sum_{r=-l}^l x_r^{(i)} p(-t - rT), \quad (5.2)$$

and, if $r(t)$ is the received signal, the sampled response of the matched filter (5.2) to $r(t)$ is given by

$$\begin{aligned} \mathbf{r} \cdot \mathbf{s}^{(i)} &= \int_{-\infty}^{\infty} r(t) s^{(i)}(t) dt \\ &= \sum_{r=-l}^l x_r^{(i)} y_r = \mathbf{x}^{(i)} \cdot \mathbf{y}, \end{aligned} \quad (2.23)$$

where $y_r = \int_{-\infty}^{\infty} r(\alpha) p(\alpha - rT) d\alpha$ is the output of the filter matched to the pulse $p(t)$ sampled at time $t = rT$ as discussed in Section 2.5 (Equation (2.26)), and $\mathbf{y} = (y_{-l}, \dots, y_l)$ is the vector of sampled signals y_r .

If time-orthogonal pulses (e.g., Nyquist pulses) $p(t)$ with unit energy ($\int_{-\infty}^{\infty} p^2(t) dt = 1$) are used, the energy of the signal $s^{(i)}(t)$ is given by

$$\begin{aligned} |\mathbf{s}^{(i)}|^2 &= \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} s^{(i)}(\alpha) s^{(i)}(\beta) d\alpha d\beta \\ &= \sum_{r=-l}^l |x_r^{(i)}|^2, \end{aligned} \quad (5.3)$$

and, from (2.23), the maximum likelihood receiver will select the sequence $\mathbf{x}^{(i)}$ which maximizes

$$J^{(i)} = 2 \sum_{r=-l}^l \operatorname{Re}\left\{ x_r^{(i)} v_r^* \right\} - \sum_{r=-l}^l |x_r^{(i)}|^2. \quad (5.4)$$

$J^{(i)}$ in Equation (5.4) is called the *metric* of the sequence $\mathbf{x}^{(i)}$, and this metric is to be maximized over all allowable choices of $\mathbf{x}^{(i)}$. Clearly, exhaustive cataloguing of (5.4) for all $\mathbf{x}^{(i)}$ is inconceivable, and strategic methods for reducing this complexity to manageable levels is precisely what the decoders in the next sections accomplish.

5.2 Tree Decoders

From (5.4) we define the partial metric at time n as

$$J_n^{(i)} = 2 \sum_{r=-l}^n \operatorname{Re} \left\{ x_r^{(i)} v_r^* \right\} - \sum_{r=-l}^n |x_r^{(i)}|^2, \quad (5.5)$$

which allows us to rewrite (5.4) in the recursive form

$$J_n^{(i)} = J_{n-1}^{(i)} + 2\operatorname{Re} \left\{ x_n^{(i)} v_n^* \right\} - |x_n^{(i)}|^2. \quad (5.6)$$

Equation (5.6) implies a tree structure which can be used to evaluate the metrics for all the allowable signal sequences as illustrated in Figure 5.1 for the code from Figure 3.1, Chapter 3. This tree has, in general, 2^k branches leaving each node, since there are 2^k possible different choices of the signal x_n at time n . Each node is labeled with the partial sequence¹ $\tilde{\mathbf{x}}^{(i)} = (x_{-l}^{(i)}, \dots, x_n^{(i)})$ which leads to it. The intermediate metric $J_n^{(i)}$ is also associated with each node. A tree decoder starts at time $n = -l$ at the single root node with $J_{-l} = 0$, and extends through the tree evaluating and storing (5.6) until time unit $n = l$, at which time the largest accumulated metric identifies the most likely sequence $\mathbf{x}^{(i)}$.

It becomes obvious that the size of this tree grows very quickly. In fact, its final width is k^{2l+1} , which is an outlandish number even for small values of l , i.e., short encoded sequences. We therefore need to reduce the complexity of decoding, and one way of accomplishing this is can by performing only a partial search of the tree.

There are a number of different approaches to tree decoding and we will discuss the more fundamental types in the subsequent sections. Before we tackle these decoding algorithms, however, we wish to modify the metric such that it can take into account the different lengths of paths, since we will come up against the problem of comparing paths of different lengths.

Consider then the set \mathcal{X}_M of M partial sequences $\tilde{\mathbf{x}}^{(i)}$ with lengths $\{n_i\}$, and let $n_{\max} = \max\{n_1, \dots, n_M\}$ be the maximum length among the M partial sequences. The decoder must make its likelihood ranking of the paths based on the partial received sequence $\tilde{\mathbf{y}}$ of length n_{\max} .

From (2.12) we know that an optimum receiver would choose the $\tilde{\mathbf{x}}^{(i)}$ which maximizes

$$P[\tilde{\mathbf{x}}^{(i)}|\tilde{\mathbf{y}}] = P[\tilde{\mathbf{x}}^{(i)}] \frac{\prod_{r=-l}^{-l+n_i+1} p_n(y_r - x_r) \prod_{r=-l+n_i}^{-l+n_{\max}} p(y_r)}{p(\tilde{\mathbf{y}})}, \quad (5.7)$$

where the second product reflects the fact that we have no hypotheses $x_r^{(i)}$ for $r > n_i$, since $\tilde{\mathbf{x}}^{(i)}$ extends only up to $-l + n_i$. We therefore have to use the a priori probabilities

¹We denote partial sequences by tildes to distinguish them from complete sequences or codewords.

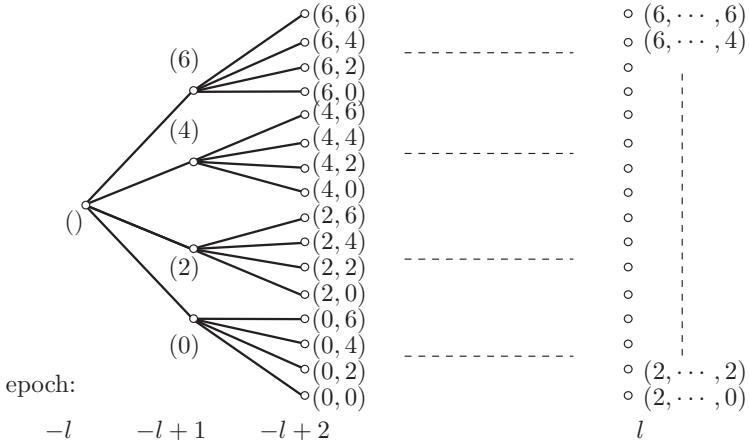


Figure 5.1: Code tree extending from time $-l$ to time l for the code from Figure 3.1, Chapter 3.

$p(y_r|x_r^{(i)}) = p(y_r)$ for $r > n_i$. Using $p(\tilde{\mathbf{y}}) = \prod_{r=-l}^{-l+n_{\max}} p(y_r)$ Equation (5.7) can be rewritten as

$$P[\tilde{\mathbf{x}}^{(i)}|\tilde{\mathbf{y}}] = P[\tilde{\mathbf{x}}^{(i)}] \prod_{r=-l}^{-l+n_i} \frac{p_n(y_r - x_r^{(i)})}{p(y_r)}, \quad (5.8)$$

and we see that we need not be concerned with the tail samples not affected by $\tilde{\mathbf{x}}^{(i)}$. Taking logarithms gives the “additive” metric

$$L(\tilde{\mathbf{x}}^{(i)}, \tilde{\mathbf{y}}) = \sum_{r=-l}^{-l+n_i} \log \frac{p_n(y_r - x_r^{(i)})}{p(y_r)} - \log \frac{1}{P[\tilde{\mathbf{x}}^{(i)}]}. \quad (5.9)$$

Since $P[\tilde{\mathbf{x}}^{(i)}] = (2^{-k})^{n_i}$ is the a priori probability of the partial sequence $\tilde{\mathbf{x}}^{(i)}$, assuming that all the inputs to the trellis encoder have equal probability, (5.9) becomes

$$L(\tilde{\mathbf{x}}^{(i)}, \tilde{\mathbf{y}}) = L(\tilde{\mathbf{x}}^{(i)}, \mathbf{y}) = \sum_{r=-l}^{-l+n_i} \left[\log \frac{p_n(y_r - x_r^{(i)})}{p(y_r)} - k \right], \quad (5.10)$$

where we have extended $\tilde{\mathbf{y}} \rightarrow \mathbf{y}$ since (5.10) ignores the tail samples $y_r; r > n_i$ anyhow. The metric (5.10) was introduced for decoding tree codes by Fano [21] in 1963 and was analytically derived by Massey [36] in 1972.

Since Equation (2.13) explicitly gives the conditional probability distribution $p_n(y_r - x_r)$, the metric in (5.10) can be specialized for additive white Gaussian noise channels to

$$\begin{aligned} L(\tilde{\mathbf{x}}^{(i)}, \mathbf{y}) &= \sum_{r=-l}^{-l+n_i} \left[\log \frac{\exp(-|x_r^{(i)} - y_r|^2/N_0)}{\sum_{x \in \mathcal{A}} p(x) \exp(-|x - y_r|^2/N_0)} - k \right] \\ &= - \sum_{r=-l}^{-l+n_i} \frac{|x_r^{(i)} - y_r|^2}{N_0} - c_r(y_r), \end{aligned} \quad (5.11)$$

where $c_r(y_r) = \log \left(\sum_{x \in \mathcal{A}} p(x) \exp(-|x - y_r|^2/N_0) \right) + k$ is a term independent of $x_r^{(i)}$ which is subtracted from all the metrics at time r . Note that c_r can be positive or negative, which causes some of the problems with *sequential decoding*, as we will see later.

It is worth noting here that if the paths examined are of the same length, say n , they all contain the same cumulative constant $-\sum_{r=-l}^{-l+n} c_r$ in their metrics, which therefore may be discarded from all the metrics. This allows us to simplify (5.11) to

$$L(\tilde{\mathbf{x}}^{(i)}, \mathbf{y}) \equiv \sum_{r=-l}^{-l+n} 2\text{Re} \left\{ x_r^{(i)} v_r^* \right\} - |x_r^{(i)}|^2 = J_n^{(i)}, \quad (5.12)$$

by neglecting terms common to all the metrics. The metric (5.12) is equivalent to the accumulated Euclidean distance between the received partial sequence $\tilde{\mathbf{y}}$ and the hypothesized symbols on the i -th path to up to length n . The restriction to path of equal length makes this metric much simpler than the general metric (5.10) (and (5.11)) and finds application in the so called *breadth-first* decoding algorithms which we will discuss in subsequent sections.

5.3 The Stack Algorithm

The stack algorithm is one of the many variants of what has become known as *sequential decoding*, and was introduced by Wozencraft and Reiffen [58] for convolutional codes. It has subsequently experienced many changes and additions. Sequential decoding describes any algorithm for decoding trellis codes which successively explores the code tree by moving to new nodes from an already explored node.

From the introductory discussion in the preceding section, one way of sequential decoding becomes apparent. We start exploring the tree and store the metric (5.10) (or (5.11)) for every node explored. At each stage now we simply extend the node with the

largest such metric. This, in essence, is the *stack algorithm* first proposed by Zigangirov [64] and Jelinek [32]. This basic algorithm is:

Step 1: Initialize an empty stack S of visited nodes and their metrics. Deposit the empty partial sequence $()$ at the top of the stack with its metric $L((), \mathbf{y}) = 0$.

Step 2: Extend the node corresponding to the top entry $\{\tilde{\mathbf{x}}_{\text{top}}, L(\tilde{\mathbf{x}}_{\text{top}}, \mathbf{y})\}$ by forming $L(\tilde{\mathbf{x}}_{\text{top}}, \mathbf{y}) - |x_r - y_r|^2/N_0 - c_r$ for all 2^k extensions of $\tilde{\mathbf{x}}_{\text{top}} \rightarrow (\tilde{\mathbf{x}}_{\text{top}}, x_r) = \tilde{\mathbf{x}}^{(i)}$. Delete $\{\tilde{\mathbf{x}}_{\text{top}}, L(\tilde{\mathbf{x}}_{\text{top}}, \mathbf{y})\}$ from the stack.

Step 3: Place the new entries $\{\tilde{\mathbf{x}}^{(i)}, L(\tilde{\mathbf{x}}^{(i)}, \mathbf{y})\}$ from Step 2 into the stack such that the stack remains ordered with the entry with the largest metric at the top of the stack.

Step 4: If the top entry of the stack is a path to one of the terminal nodes at depth l , stop and select \mathbf{x}_{top} as the transmitted symbol sequence. Otherwise, go to Step 2.

There are some practical problems associated with the stack algorithm. Firstly, the number of computations which the algorithms performs is very dependent on the quality of the channel. If we have a very noisy channel, the received sample value y_r will be very unreliable and a large number of possible paths will have similar metrics. These paths all have to be stored in the stack and explored further. This causes a computational speed problem, since the incoming symbols have to be stored in a buffer while the algorithm performs the decoding operation. This buffer is now likely to overflow if the channel is very noisy and the decoder will have to declare a decoding failure. This phenomenon is explored further in Section 5.11. In practice, the transmitted data will be framed and the decoder will declare a frame erasure if it experiences input buffer overflow.

A second problem with the stack algorithm is the increasing complexity of Step 2, i.e., of reordering the stack. This sorting operation depends on the size of the stack, which, again, for very noisy channels becomes large. This problem is addressed in all practical applications by ignoring small differences in the metric and collecting all stack entries with metrics within a specified “quantization interval” in the same bucket. Bucket j contains all stack entries with metrics

$$j\Delta \leq L(\tilde{\mathbf{x}}^{(i)}, \mathbf{r}) \leq (j+1)\Delta, \quad (5.13)$$

where Δ is a variable quantization parameter. Incoming paths are now sorted only into the correct bucket, avoiding the sorting complexity of the large stack. The depositing and removal of the paths from the buckets can occur on a “last in, first out” basis.

There are a number of variations of this basic theme. If Δ is a fixed value, the number of buckets can also grow to be large, and the sorting problem, originally avoided, reappears. An alternative is to let the buckets vary in size, rather than in metric range. In that way, the critical dependence on the stack size can be avoided.

An associated problem with the stack is that of stack overflow. This is less severe and the remedy is simply to drop the last path in the stack from future consideration. The probability of actually loosing the correct path is very small, a much smaller problem than that of a frame erasure. A large number of variants of this algorithm are feasible and have been explored in the literature. Further discussion of the details of implementation of these algorithms are found in [3, 4, 30].

5.4 The Fano Algorithm

Unlike the stack algorithm, the Fano algorithm is a depth-first tree search procedure in its purest form. Introduced by Fano [21] in 1963, this algorithm stores only one path and thus, essentially, requires no storage. Its drawback is a certain loss in speed compared to the stack algorithm for higher rates [27], but for moderate rates the Fano algorithm decodes faster than the stack algorithm [28]. It seems that the Fano algorithm is the preferred choice for practical implementations of sequential decoding algorithms.

Since the Fano algorithm only stores one path, it must allow for backtracking. Also, there can be no jumping between non-connected nodes, i.e., the algorithm only moves between adjacent nodes which are connected in the code tree. The algorithm starts at the initial node and moves in the tree by proceeding from one node to a successor node with a suitably large metric. If no such node can be found, the algorithm backtracks and looks for other branches leading off from previously visited nodes. The metrics of all these adjacent nodes can be computed by adding or subtracting the metric of the connecting branch and no costly storing of metrics is required. If a node is visited more than once, its metric is recomputed. This is part of the computation/storage trade-off of sequential decoding.

The algorithm proceeds along a chosen path as long as the metric continues to increase. It does that by continually tightening a metric threshold to the current node metric as it visits nodes for the first time. If new nodes along the path have a metric smaller than the threshold, the algorithm backs up and looks at other node extensions. If no other extensions with a metric above the threshold can be found, the value of the threshold is decreased and the forward search is resumed. In this fashion, each node visited in the forward direction more than once is reached with a progressively lower threshold each time. This prevents the algorithm from getting caught in an infinite loop. Eventually this procedure reaches a terminal node at the end of the tree and a decoded symbol sequence can be output.

Figure 5.2 depicts an example of the search behavior of the Fano algorithm. Assume that there are two competing paths, where the solid path is the most likely sequence and the dashed path is a competitor. The vertical height of the nodes in Figure 5.2 is used to illustrate the values of the metrics for each node. Also assume that the paths shown

are those with the best metrics, i.e., all other branches leading off from the nodes lead to nodes with smaller metrics. Initially, the algorithm will proceed to node A, at which time it will start to backtrack since the metric of node D is smaller than that of node A. After exploring alternatives and successively lowering the threshold to t_1 and then to t_2 , it will reach node O again and proceed along the dashed path to node B and node C. Now it will start to backtrack again, lowering its threshold to t_3 and then to t_4 . It will now again explore the solid path beyond node D to node E, since the lower threshold will allow that. From there on, the path metrics pick up again and the algorithm proceeds along the solid path. If the threshold decrement Δ had been twice as large, the algorithm would have moved back to node O faster, but would also have been able to move beyond the metric dip at node F, and would have chosen the erroneous path.

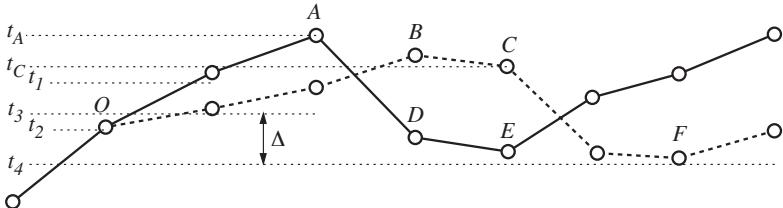


Figure 5.2: Illustration of the operation of the Fano algorithm when choosing between two competing paths.

It becomes obvious that at some point the metric threshold t will have to be lowered to the lowest metric value which the maximum likelihood path assumes, and, consequently, a large decrement Δ allows the decoder to achieve this low threshold faster. Conversely, if the decrement Δ is too large, t may drop to a value which allows several erroneous path to be potentially decoded before the maximum metric path. The optimal value of the metric threshold is best determined by experience and simulations. Figure 5.3 shows the flowchart of the Fano algorithm.

5.5 The M -Algorithm

The M -algorithm is a purely breadth-first synchronous algorithm which moves from time unit to time unit. It keeps M candidate paths at each iteration and deletes all others from further consideration. At each time unit the algorithm extends all M currently held nodes to form $2^k M$ new nodes, from among which those M with the best metrics are retained. Due to the breadth-first nature of the algorithm, the metric in (5.12) can be used. The algorithm is very simple:

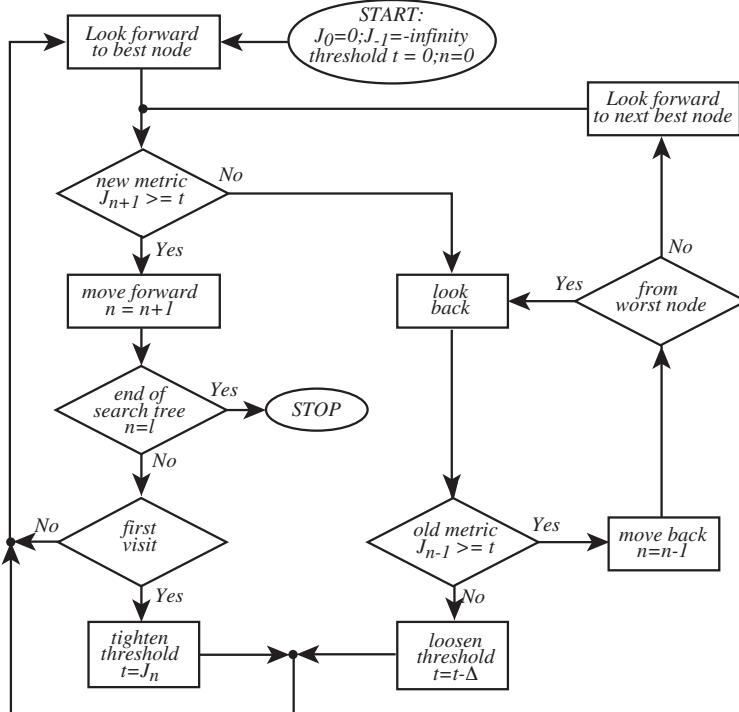


Figure 5.3: Flowchart of the Fano algorithm. The initialization of $J_{-1} = -\infty$ has the effect that the algorithm can lower the threshold for the first step, if necessary.

Step 1: Initialize an empty list L of candidate paths and their metrics. Deposit the zero-length path $()$ with its metric $L((), \mathbf{y}) = 0$ in the list. Set $n = -l$.

Step 2: Extend 2^k partial paths $\tilde{\mathbf{x}}^{(i)} \rightarrow (\tilde{\mathbf{x}}^{(i)}, x_r^{(i)})$ from each of the at most M paths $\tilde{\mathbf{x}}^{(i)}$ in the list. Delete the entries in the original list.

Step 3: Find the at most M partial paths with the best metrics among the extensions² and save them in the list L . Delete the rest of the extensions. Set $n = n + 1$.

Step 4: If at the end of the tree, i.e., $n = l$, release the output symbols corresponding to the path with the best metric in the list L , otherwise go to Step 2.

²Note that from two or more extensions leading to the same state (see Section 5.6), all but the one with the best metric may be discarded. This will improve performance slightly by eliminating some paths which cannot be the ML path.

The M -algorithm appeared in the literature for the first time in a paper by Jelinek and Anderson [33], where it was applied to source coding. At the time, there were no real algorithms for sequential source coding other than this, so it was viewed as miraculous. In the early 1980s, applications of the algorithm to channel coding began to appear. The research book by Anderson and Mohan on algorithmic source and channel coding [4], along with [2], collects much of this material, and these are the most comprehensive sources on the subject.

This algorithm is straightforward to implement, and its popularity is partly due to the simple metric as compared to sequential decoding. The decoding problem with the M -Algorithm is the loss of the correct path from the list of candidates, after which the algorithm might spend a long time resynchronizing. This problem is usually addressed by framing the data. With each new frame resynchronization is achieved. The computational load of the M -algorithm is independent of the size of the code; it is proportional to M . Unlike depth-first algorithms, it is also independent of the quality of the channel, since M paths are retained irrespective of the channel quality.

A variant of the M -algorithm is the so-called T -algorithm. It differs from the M -algorithm only in Step 3, where instead of a fixed number M , all path with metrics $L(\tilde{\mathbf{x}}^{(i)}, \mathbf{y}) \geq \lambda_t - T$ are retained, where λ_t is the metric of the best path and T is some arbitrary threshold. The T -algorithm is therefore in a sense a hybrid between the M -algorithm and a stack-type algorithm. Its performance depends on T , but is very similar to that of the M -algorithm, and we will not discuss it further.

The major problem with the M -algorithm is the possibility that the correct path is not among the M retained candidates at time n . If this happens, we lose the correct path and it usually takes a long time to resynchronize.

Since the complexity of the M -algorithm is largely independent of the code size and constraint length, one usually chooses very long constraint-length codes to assure that free distance of the code, d_{free} , is very large, and therefore errors due to neighboring codewords are not the dominant error mechanism, correct path loss primarily determines the performance of the decoder.

Let us then take a closer look at the probability of losing the correct path at time n . To that end we assume that at time $n - 1$ the correct path was among the M retained candidates as illustrated in Figure 5.4. Each of these M nodes is extended into 2^k nodes at time n , of which M are to be retained. There are then a total of $\binom{M2^k}{M}$ ways of choosing the new M retained paths at time n .

Let us denote the correct partial path by $\tilde{\mathbf{x}}^{(c)}$. The optimal strategy of the decoder will then be to retain that particular set of M candidates which maximizes the probability of containing $\tilde{\mathbf{x}}^{(c)}$. Let \mathcal{C}_p be one of the $\binom{M2^k}{M}$ possible sets of M candidates at time n . We wish to maximize

$$\max_p \Pr \left\{ \tilde{\mathbf{x}}^{(c)} \in \mathcal{C}_p | \tilde{\mathbf{y}} \right\}. \quad (5.14)$$

Since all the partial paths $\tilde{\mathbf{x}}^{(p_j)} \in \mathcal{C}_p, j = 1, \dots, M$ are distinct, the events $\{\tilde{\mathbf{x}}^{(c)} = \tilde{\mathbf{x}}^{(p_j)}\}$ are all mutually exclusive for different j , i.e., the correct path can be at most only one of the M different candidates $\tilde{\mathbf{x}}^{(p_j)}$. Equation (5.14) can therefore be evaluated as

$$\max_p \Pr\left\{\tilde{\mathbf{x}}^{(c)} \in \mathcal{C}_p | \tilde{\mathbf{y}}\right\} = \max_p \sum_{j=1}^M \Pr\left\{\tilde{\mathbf{x}}^{(c)} = \tilde{\mathbf{x}}_j^{(p_j)} | \tilde{\mathbf{y}}\right\}. \quad (5.15)$$

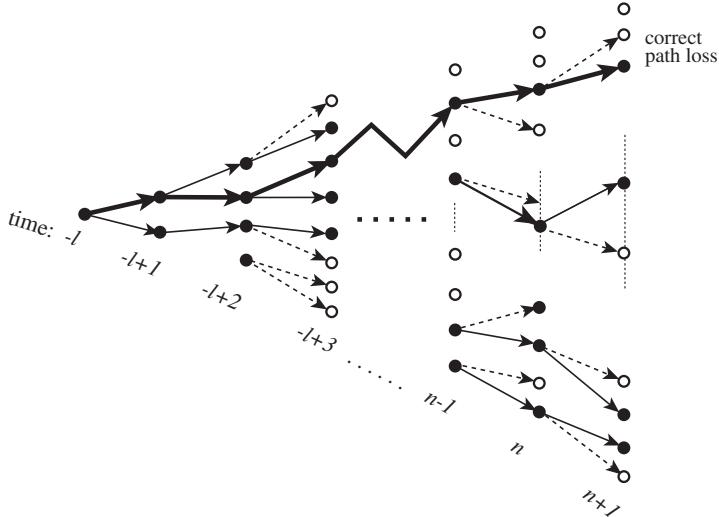


Figure 5.4: Extension of $2^k M = 2 \cdot 4$ paths from the list of the best $M = 4$ paths. The solid paths are those retained by the algorithm, the path indicated by the heavy line corresponds to the correct transmitted sequence.

From (5.7), (5.9), and (5.12) we know that

$$\Pr\left\{\tilde{\mathbf{x}}^{(c)} = \tilde{\mathbf{x}}^{(p_j)} | \tilde{\mathbf{y}}\right\} \propto \exp\left(-\sum_{r=-l}^{-l+n} \left(2\operatorname{Re}\left\{x_r^{(p_j)} v_r^*\right\} - |x_r^{(p_j)}|^2\right)\right), \quad (5.16)$$

where the proportionality constant is independent of $\tilde{\mathbf{x}}^{(p_j)}$. The maximization in (5.14) now becomes equivalent to (considering only the exponent from above)

$$\begin{aligned} \max_p \Pr\left\{\tilde{\mathbf{x}}^{(c)} \in \mathcal{C}_p | \tilde{\mathbf{y}}\right\} &\equiv \max_p \sum_{j=1}^M \sum_{r=-l}^{-l+n} \left(2\operatorname{Re}\left\{x_r^{(p_j)} v_r^*\right\} - |x_r^{(p_j)}|^2\right) \\ &= \max_p J_n^{(p_j)}, \end{aligned} \quad (5.17)$$

i.e., we simply collect the M paths with the best partial metrics $J_n^{(p_j)}$ at time n . This argument was derived by Aulin [5]. Earlier we showed that the total metric can be broken up into the recursive form of (5.6), but now we have shown that if the detector is constrained to considering only a maximum of M path at each stage, retaining those M paths $\tilde{\mathbf{x}}^{(p_j)}$ with maximum partial metrics is the optimal strategy.

The probability of correct path loss, denoted by $\Pr(\text{CPL})$, can now be addressed. Follow the methodology of Aulin [5], we need to evaluate the probability that the correct path $\tilde{\mathbf{x}}^{(c)}$ is not among the M candidate paths. This will happen if M paths $\tilde{\mathbf{x}}^{(p_j)} \neq \tilde{\mathbf{x}}^{(c)}$ have a partial metric $J_n^{(p_j)} \geq J_n^{(c)}$, or equivalently if all the M metric differences

$$\delta_n^{(j,c)} = J_n^{(c)} - J_n^{(p_j)} = \sum_{r=-l}^{-l+n} \left(|x_r^{(p_j)}|^2 - |x_r^{(c)}|^2 - 2\text{Re}\left\{ x_r^{(p_j)} - x_r^{(c)} \right\} v_r^* \right) \quad (5.18)$$

are smaller than or equal to zero. That is,

$$\Pr(\text{CPL}|\mathcal{C}_p) = \Pr\{\boldsymbol{\delta}_n \leq \mathbf{0}\}, \quad \tilde{\mathbf{x}}^{(p_j)} \in \mathcal{C}_p, \quad (5.19)$$

where $\boldsymbol{\delta}_n = (\delta_n^{(1,c)}, \dots, \delta_n^{(M,c)})$ is the vector of metric differences at time n between the correct path and the set of paths in a given set \mathcal{C}_p , which does not contain $\tilde{\mathbf{x}}^{(c)}$. $\Pr(\text{CPL}|\mathcal{C}_p)$ depends on the correct path $\mathbf{x}^{(c)}$ and, strictly speaking, has to be averaged over all correct paths. We shall be satisfied with the correct path which produces the largest $\Pr(\text{CPL}|\mathcal{C}_p)$.

In Appendix 5.C we show that the probability of losing the correct path decreases exponentially with the signal-to-noise ratio and is overbounded by

$$\Pr(\text{CPL}|\mathcal{C}_p) \leq Q\left(\sqrt{\frac{d_l^2}{2N_0}}\right). \quad (5.20)$$

The parameter d_l^2 depends on \mathcal{C}_p and is known as the *Vector Euclidean distance* [5] of the path $\tilde{\mathbf{x}}^{(c)}$ with respect to the M error paths $\tilde{\mathbf{x}}^{(p_i)} \in \mathcal{C}_p$. It is important to note here that (5.20) is an upper bound of the probability that M specific error paths have a metric larger than $\tilde{\mathbf{x}}^{(c)}$. The problem is finding the minimal d_l^2 among all sets \mathcal{C}_p , denoted by $\min(d_l^2)$. This is a rather complicated combinatorial problem, since essentially all combinations of M candidates for each correct path at each time n have to be analyzed from the growing set of possibilities. Aulin [5] has studied this problem and gives several rejection rules which alleviate the complexity of finding d_l^2 , but the problem remains complex.

Note that d_l^2 is a non-decreasing function of M , the decoder complexity, and one way of selecting M is to choose it such that

$$\min(d_l^2) \geq d_{\text{free}}^2. \quad (5.21)$$

This choice should guarantee that the performance of the M -algorithm is approximately equal to the performance of maximum-likelihood decoding. To see this, let $\overline{P_e(M)}$ be the probability of an error event (compare equation (5.66)). Then

$$\begin{aligned}\overline{P_e(M)} &\leq \frac{\overline{P_e}}{\overline{P_e} + \Pr(\text{CPL})} (1 - \Pr(\text{CPL})) + \Pr(\text{CPL}) \\ &\leq \frac{\overline{P_e}}{\overline{P_e} + \Pr(\text{CPL})},\end{aligned}\quad (5.22)$$

where $\overline{P_e}$ is the probability that a maximum-likelihood decoder starts an error event (see Section 5.9). For high values of the signal-to-noise ratio, equation (5.22) can be approximated by

$$\overline{P_e(M)} \approx N_{d_{\text{free}}} Q\left(\frac{d_{\text{free}}}{\sqrt{2N_0}}\right) + \kappa Q\left(\frac{\min(d_l)}{\sqrt{2N_0}}\right), \quad (5.23)$$

where κ is some constant, which, however, is difficult to determine in the general case. Now, if (5.21) holds, the suboptimality of the M -algorithm does not dominate the error performance for high signal-to-noise ratios.

Aulin [5] has analyzed this situation for 8-PSK trellis codes and found that, in general, $M \approx \sqrt{S}$ will fulfill condition (5.21), where S is the number of states in the code trellis.

Figure 5.5 shows the simulated performance of the M -algorithm versus M for the 64-state trellis code from Table 3.1 with $d_{\text{free}}^2 = 6.34$. $M = 8$ meets (5.21) according to [5], but from Figure 5.5 it is apparent that the performance is still about 1.5 dB poorer than ML-decoding. This is attributable to the resynchronization problems and the fact that we are operating at rather low values of the signal-to-noise ratio, where neither d_{free}^2 nor $\min(d_l^2)$ are necessarily dominating the error performance.

Figures 5.6 and 5.7 show the empirical probability of correct path loss $P(\text{CPL})$ and the BER for two convolutional codes and various values of M . Figure 5.6 shows simulation results for the 2048-state convolutional code, $\nu = 11$, from Table 4.1. The bit error rate and the probability of losing the correct path converge to the same asymptotic behavior, indicating that the probability of correct path loss and not recovery errors is the dominant error mechanism for very large values of the signal-to-noise ratio.

Figure 5.7 shows simulation results for the $\nu = 15$ large constraint-length code for the same values of M . For this length code, path loss will be the dominant error scenario. We note that both codes have a very similar error performance, demonstrating that the code complexity has little influence.

Once the correct path is lost, the algorithm may spend a long time before it finds it again, i.e., before the correct path is again one of the M retained paths. We use this terminology in the following way: Technically, each path through the tree is distinct, and, once lost, the correct path is lost forever, or at least until the end of the frame. However, it can easily be appreciated that due to the finite memory of codes described by “small” trellises, even after loss of the correct path, the decoder may find a sub-tree later on, which

Symbol Error Probability (BER)

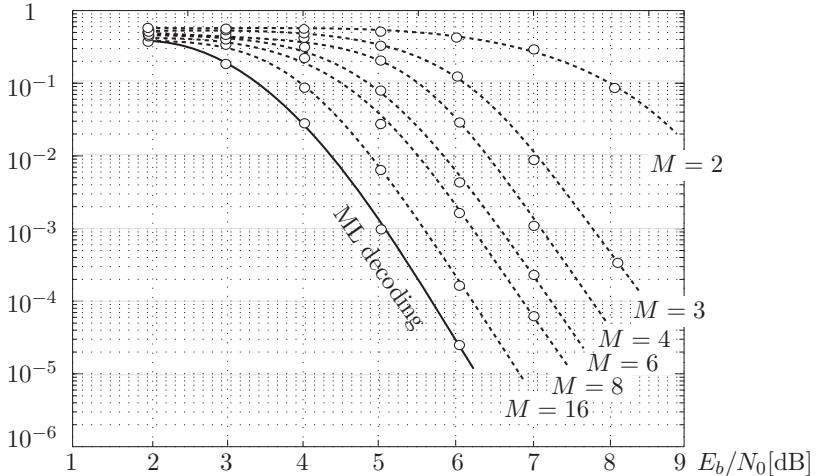


Figure 5.5: Simulation results for the 64-state optimal distance 8-PSK trellis codes decoded with the M -algorithm, using $M = 2, 3, 4, 5, 6, 8$, and 16 . The performance of maximum likelihood decoding is also included in the figure (Source: [5]).

is identical with the correct sub-tree. Once the algorithm joins this section of the tree, we say it has recovered the correct path. This terminology will make more sense once we use the trellis as decoder map.

Correct path recovery, however, is a very complex problem and no complete analytical approach exists to date. There are a few theoretical studies of the recovery problem, such as [6], but most of the understanding into the dynamics of the decoder during a recovery has been gained through simulation studies.

Figure 5.8 presents such a simulation showing the average number of steps taken for the algorithm to recover the correct path for the 2048-state, $\nu = 11$ convolutional code, whose error performance is shown in Figure 5.6. Each instance of the simulation was performed such that the algorithm was initiated and run until the correct path was lost, and then the number of steps until recovery were counted [34].

In contrast, Figure 5.9 shows the average number of steps until recovery for the rate $1/2$, $\nu = 11$ systematic convolutional code with generator polynomials $g^{(0)} = 4000, g^{(1)} = 7153$. This code has a free Hamming distance of only $d_{\text{free}} = 9$, but its recovery performance is much superior to that of the non-systematic code. In fact, the average number of steps

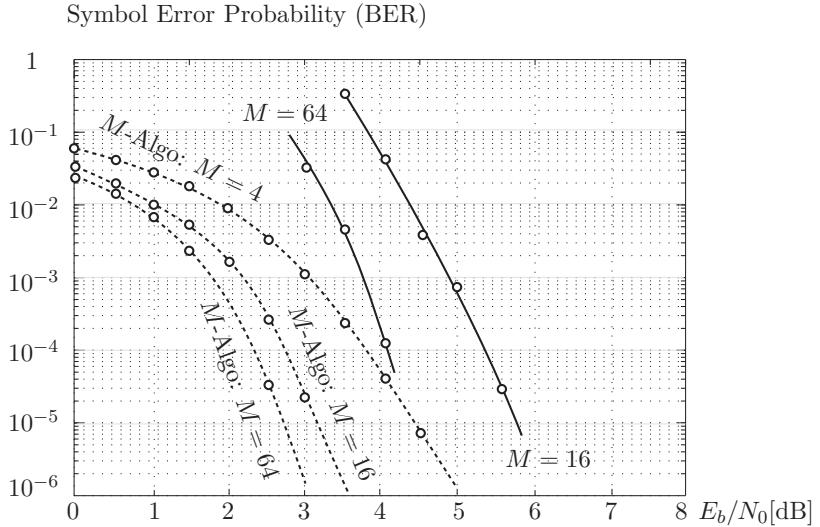
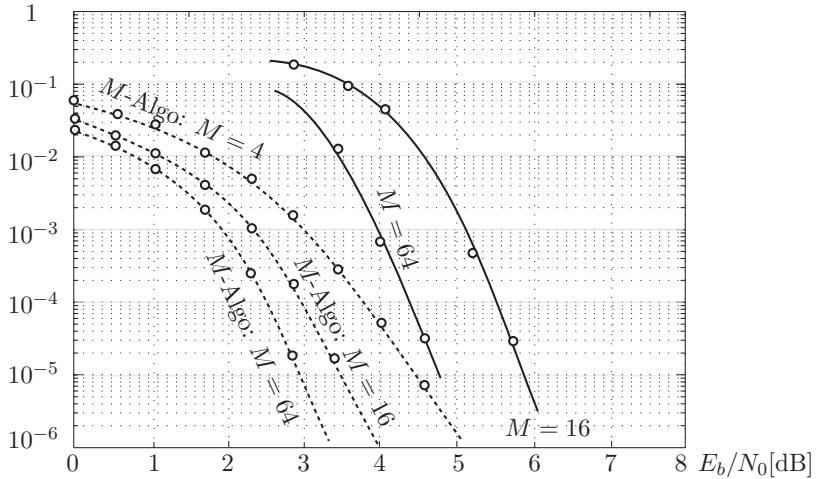


Figure 5.6: Simulation results for the 2048-state, rate $R = 1/2$ convolutional code using the M -algorithm, for $M = 4, 16$, and 64 . The dashed curves are the path loss probability and the solid curves are BER's.

until recovery is independent of the signal-to-noise ratio, while it increases approximately linearly with E_b/N_0 for the non-systematic code. This rapid recovery results in superior error performance of the systematic code compared to the non-systematic code, shown in Figure 5.10. However, what can also clearly be seen in Figure 5.10 is that, for very large values of E_b/N_0 , the “stronger” code will win out due to its larger free distance.

The observation that systematic convolutional codes outperform non-systematic codes for error rates $P_b \gtrsim 10^{-6}$ has also been made by Osthoff et al. [38]. The reason for this lies in the *return barrier* phenomenon, which can be explained with the aid of Figure 5.11. In order for the algorithm to recapture the correct path after a correct path loss, one of the M retained paths must correspond to a trellis state that connects to the correct state. In Figure 5.11 we assume that the all-zero sequence is the correct sequence, and hence the all-zero state is the correct state for all time. This assumption is made without loss of generality for convolutional codes due to their linearity. For a feed-forward realization of a rate $1/2$ code, the only state which connects to the all-zero state is the state $(0, \dots, 0, 1)$, denoted by s_m in the figure. In the case of a systematic code with $g_0^{(1)} = g_\nu^{(1)} = 1$ (see Chapter 4) the two possible branch signals are (01) and (10) as indicated in Figure 5.11.

Symbol Error Probability (BER)

Figure 5.7: Same simulation results for the $\nu = 15$, $R = 1/2$ convolutional code.

Average number of steps until recovery

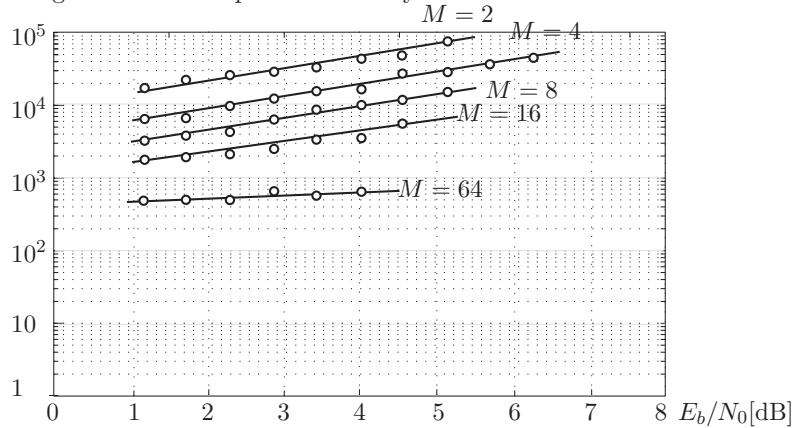


Figure 5.8: Average number of steps until recovery for the code from Figure 5.6 [34].

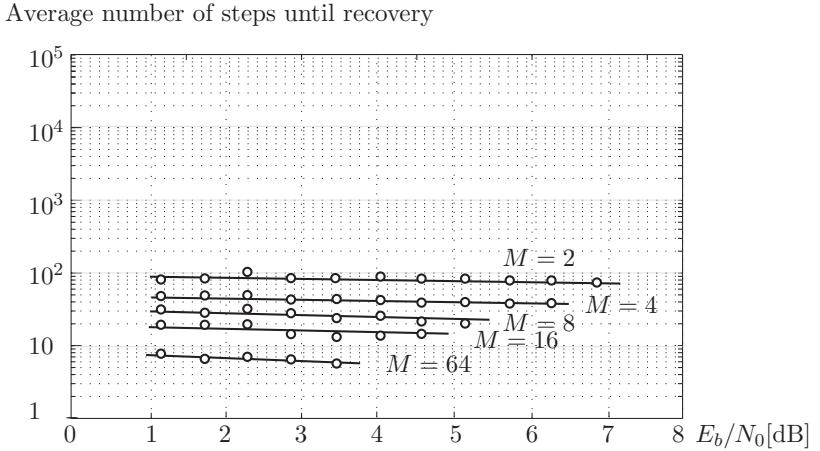


Figure 5.9: Average number of steps until recovery of the correct path for the systematic convolutional code with $\nu = 11$ ([34]).

For a non-systematic, maximum free distance code on the other hand, the two branch signals are (11) and (00), respectively. Since the correct branch signal is (00), the probability that the metric of s_f (for failed) exceeds the metric of s_c equals 1/2, since both branch signals are equidistant from the correct signal. For the non-systematic code on the other hand, this probability equals $Q(\sqrt{E_s/N_0})$. This explains the dependence of the path recovery on E_b/N_0 for non-systematic codes, as well as why systematic codes recapture the correct path faster with a recovery behavior which is independent of E_b/N_0 .

The M -algorithm impresses with its simplicity. Unfortunately, a theoretical understanding of the algorithm is not related to this simplicity at all, and it seems that much more work in this area is needed before a coherent theory is available. This lack of a theoretical basis for the algorithm is, however, no barrier to its implementation. Early work on the application of the M -algorithm to convolutional codes, apart from Anderson [2, 3, 4, 38], was presented by Zigangirov and Kolesnik [65], while Simmons and Wittke [47], Aulin [7], and Balachandran [10], among others, have applied the M -algorithm to continuous-phase modulation. General trellis codes have not yet seen much action from the M -algorithm. An notable exception is [40].

It is generally felt that the M -algorithm is not a viable candidate algorithm for decoding binary convolutional codes, in particular with the emergence of turbo codes and iterative decoding; however, it seems to work very well with non-binary modulations such as

Symbol Error Probability (BER)

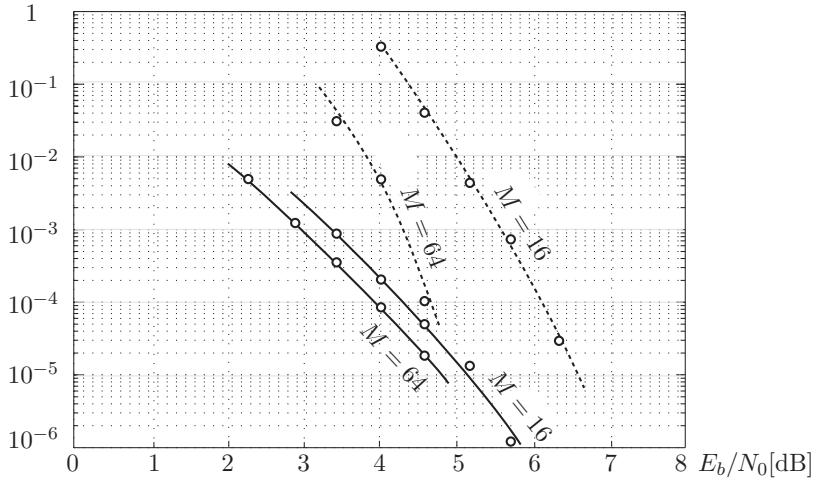


Figure 5.10: Simulation results for the superior 2048-state systematic code using the M -algorithm. The dashed curves are the error performance of the same constraint length non-systematic code from Figure 5.6 (Source [34]).

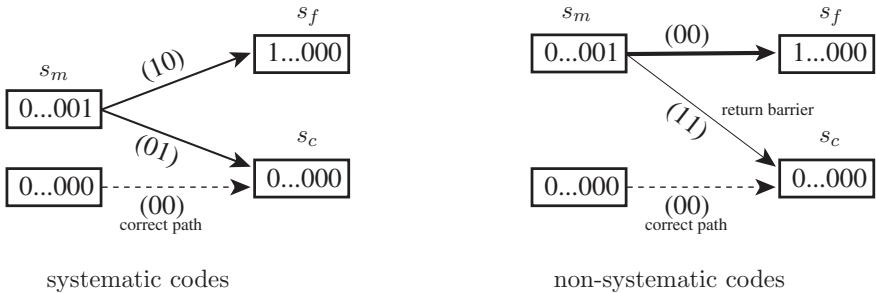


Figure 5.11: Heuristic explanation of the return barrier phenomenon in the M -algorithm.

CPM, trellis-coded modulation, and certain multiple access systems such as code-division multiple (CDMA) access, where it may have a place in practical implementations.

5.6 Maximum Likelihood Decoding

The difficulty in decoding arises from the exponential size of the growing decoding tree. In this section we will show that this tree can be reduced by merging nodes, such that the tree only grows to a maximum size of 2^S nodes, where S is the number of encoder states. This merging leads diverging paths together again and we obtain a structure resembling a trellis, as discussed for encoders in Chapter 4. In fact, because the paths through the trellis of the code represent all possible codewords, it should be intuitively evident that a decoder will also need to explore no more than all those paths, which can be done by operating such a decoder on the trellis of the code.

In order to see how this happens, let $J_{n-1}^{(i)}$ and $J_{n-1}^{(j)}$ be the metrics of two nodes corresponding to the partial sequences $\tilde{\mathbf{x}}^{(i)}$ and $\tilde{\mathbf{x}}^{(j)}$ of length $n-1$, respectively. Let the encoder states which correspond to $\tilde{\mathbf{x}}^{(i)}$ and $\tilde{\mathbf{x}}^{(j)}$ at time $n-1$ be $s_{n-1}^{(i)}$ and $s_{n-1}^{(j)}$; $s_{n-1}^{(i)} \neq s_{n-1}^{(j)}$, and assume that the next extension of $\tilde{\mathbf{x}}^{(i)} \rightarrow (\tilde{\mathbf{x}}^{(i)}, x_n^{(i)})$ and $\tilde{\mathbf{x}}^{(j)} \rightarrow (\tilde{\mathbf{x}}^{(j)}, x_n^{(j)})$ is such that $s_n^{(i)} = s_n^{(j)}$, i.e., the encoder states at time n are identical. See also Figure 5.12 below.

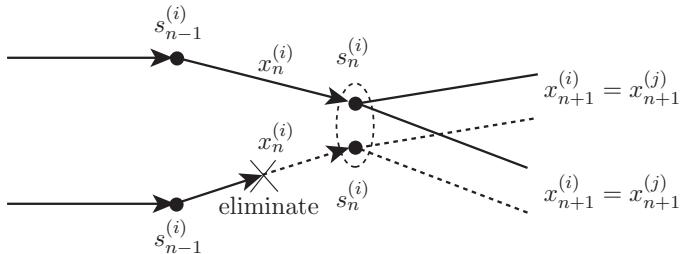


Figure 5.12: Merging nodes.

Now we propose to merge the two nodes $(\tilde{\mathbf{x}}^{(i)}, x_n^{(i)})$ and $(\tilde{\mathbf{x}}^{(j)}, x_n^{(j)})$ into one node, which we now call a (decoder) state. We retain the partial sequence which has the larger metric J_n at time n and discard the partial sequence with the smaller metric. Ties are broken arbitrarily. We are now ready to prove the following

Theorem 5.1 (*Theorem of Non-Optimality*) *The procedure of merging nodes which correspond to identical encoder states and then discarding the path with the smaller metric never eliminates the maximum-likelihood path.*

Theorem 5.1 is sometimes referred to as the theorem of non-optimality and allows us to construct a maximum-likelihood decoder whose complexity is significantly smaller than that of an all-out exhaustive tree search.

Proof: The metric at time $n + k$ for path i can be written as

$$J_{n+k}^{(i)} = J_n^{(i)} + \sum_{h=1}^k \beta_{n+h}^{(i)} \quad (5.24)$$

for every future time index $n + k; 0 < k \leq l - n$, where $\beta_n^{(i)} = 2\text{Re}\left\{x_n^{(i)}v_n^*\right\} - |x_n^{(i)}|^2$ is the metric increment, now also called the *branch metric*, at time n . Now, if the nodes of path i and j correspond to the same encoder state at time n , there exists for every possible extension of the i th path $(x_{n+1}^{(i)}, \dots, x_{n+k}^{(i)})$ a corresponding identical extension $(x_{n+1}^{(j)}, \dots, x_{n+k}^{(j)})$ of the j th path. Let us then assume without loss of generality that the i th path accumulates the largest metric at time l , i.e., $J_l^{(i)} \geq J_l^{(j)}$. Therefore

$$J_n^{(i)} + \sum_{h=1}^{n-l} \beta_{n+h}^{(i)} \geq J_n^{(j)} + \sum_{h=1}^{n-l} \beta_{n+h}^{(j)}, \quad (5.25)$$

and $\sum_{h=1}^{n-l} \beta_{n+h}^{(i)}$ is the maximum metric sum for the extensions from node $(\tilde{x}^{(i)}, x_n^{(i)})$. (Otherwise another path would have a higher final metric.) But since the extensions for both paths are identical, $\sum_{h=1}^{n-l} \beta_{n+h}^{(i)} = \sum_{h=1}^{n-l} \beta_{n+h}^{(j)}$ and $J_n^{(i)} \geq J_n^{(j)}$. Path j can therefore never accumulate a larger metric than path i , and we may discard it with impunity at time n . Q.E.D.

The tree now folds back on itself and forms a trellis with exactly S , identical to the encoder trellis (e.g., Figure 3.3), and there are 2^k paths merging in a single state at each step. Note then that there are now at most S retained partial sequences $\tilde{x}^{(i)}$, called the *survivors*. The most convenient labeling convention is that each state is labeled by the corresponding encoder state, plus the survivor which leads to it. This trellis is a replica of the encoder trellis discussed in Chapter 4, and the task of the decoder is to retrace the path the encoder took through this trellis. Theorem 5.1 guarantees that this procedure is optimal. This method was introduced by Viterbi in 1967 [51, 39] in the context of analyzing convolutional codes, and has since become widely known as the *Viterbi Algorithm* [23]:

Step 1: Initialize the S states of the decoder with a metric $J_{-l}^{(i)} = -\infty$ and survivors $\tilde{x}^{(i)} = \{\}$. Initialize the starting state of the encoder, usually state $i = 0$, with the metric $J_{-l}^{(0)} = 0$. Let $n = -l$.

Step 2: Calculate the branch metric

$$\beta_n = 2\text{Re}\{x_n v_n^*\} - |x_n|^2 \quad (5.26)$$

for each state $s_n^{(i)}$ and each extension $x_n^{(i)}$.

Step 3: Follow all trellis transitions $s_n^{(i)} \rightarrow s_{n+1}^{(i)}$ determined by the encoder FSM and, from the 2^k merging paths, retain the survivor $\tilde{\mathbf{x}}^{(i)}$ for which $J_{n+1}^{(i)}$ is maximized.

Step 4: If $n < l$, let $n = n + 1$ and go to Step 2.

Step 5: Output the survivor $\mathbf{x}^{(i)}$ which maximizes $J_l^{(i)}$ as the maximum-likelihood estimate of the transmitted sequence.

Steps 2 and 3 are the central operations of the Viterbi algorithm and are referred to as the Add–Compare–Select (ACS) step. That is, branch metrics are added to state metrics, comparisons are made among all incoming branches, and the largest-metric path is selected.

The Viterbi algorithm and the M -algorithm are both breadth-first searches and share some similarities. In fact, one often introduces the concept of mergers also in the M -algorithm in order to avoid carrying along suboptimal paths. In fact, the M -algorithm can be operated in the trellis rather than in the tree. The Viterbi algorithm has enjoyed tremendous popularity, not only in decoding trellis codes, but also in symbol sequence estimation over channels affected by intersymbol interference [41, 24], multi-user optimal detectors [50], and speech recognition. Whenever the underlying generating process can be modeled as a finite-state machine, the Viterbi algorithm finds application.

A rather large body of literature deals with the Viterbi decoder, and there are a number of good books dealing with the subject (e.g., [30, 41, 13, 52]). One of the more important results is that it can be shown that one does not have to wait until the entire sequence is decoded before starting to output the estimated symbols $x_n^{(i)}$, or the corresponding data. The probability that the symbols in all survivors $\tilde{\mathbf{x}}^{(i)}$ are identical for $m < n - n_t$, where n is the current active decoding time and n_t , called the *truncation length* or *decision depth* (Section 3.2 and Equation (4.16)), is very close to unity for $n_t \approx 5\nu$. This has been shown to be true for rate 1/2 convolutional codes [15, Page 182], but the argument can easily be extended to general codes. We may therefore modify the algorithm to obtain a fixed-delay decoder by modifying Step 4 and 5 of the above Viterbi algorithm as follows:

Step 4: If $n \geq n_t$, output $x_{n-n_t}^{(i)}$ from the survivor $\tilde{\mathbf{x}}^{(i)}$ with the largest metric $J_n^{(i)}$ as the estimated symbol at time $n - n_t$. If $n < l - 1$, let $n = n + 1$ and go to Step 2.

Step 5: Output the remaining estimated symbols $x_n^{(i)}; l - n_t < n \leq l$ from the survivor $\mathbf{x}^{(i)}$ which maximizes $J_l^{(i)}$.

We recognize that we may now let $l \rightarrow \infty$, i.e., the complexity of our decoder is no longer determined by the length of the sequence, and it may be operated in a continuous fashion. The simulation results in Section 5.9 were obtained with a Viterbi decoder according to the modified algorithm.

Let us spend some thoughts on the complexity of the Viterbi algorithm. Denote by E the total number of branches in the trellis; i.e., there are $S2^k$ branches per time epoch for a trellis generated by a rate k/n convolutional code. The complexity requirements of the Viterbi algorithm can then be captured by the following theorem [35]

Theorem 5.2 *The Viterbi algorithm requires a complexity which is linear in the number of edges E , i.e., it performs $O(E)$ arithmetic operations (multiplications, additions and comparisons).*

Proof: Step 2 in the Viterbi algorithm requires the calculation of β_n , which needs two multiplies and an addition, as well as the addition $J_n^{(i)} + \beta_n$ for each branch. Some of the values β_n may be identical, the number of arithmetic operations is therefore larger than E additions and less than $2E$ multiplications and additions.

If we denote the number of branches entering state s by $\rho(s)$, then Step 3 in the above algorithm requires $\sum_{\text{states } s} (\rho(s) - 1) \leq E/2l$ comparisons per time epoch. $\rho(s) = 2^k$ in our case, and the total number of comparisons is therefore less than E , and larger than $E - 2lS$.

There are then together $O(E)$ arithmetic operations required.

Q.E.D.

5.7 A Posteriori Probability Symbol Decoding

The purpose of the a posteriori probability (APP) algorithm is to compute a posteriori probabilities on either the information bits or the encoded symbols. These probabilities are mainly important in the iterative decoding algorithms for turbo codes discussed later in this book. Maximizing the a posteriori probabilities by themselves leads to only minor improvements in terms of bit error rates compared to the Viterbi algorithm. The algorithm was originally presented by Bahl, Cocke, Jelinek, and Raviv [9] in 1972 and was used to maximize the probability of each symbol being correct, referred to as the maximum a posteriori probability (MAP) algorithm. However, it provided no significant improvement over maximum-likelihood sequence decoding and is significantly more complex. Prior to the age of turbo coding, it was consequently not much used.

With the invention of turbo codes in 1993, however, the situation turned, and the APP became the major representative of the so-called soft-in soft-out (SISO) algorithms for providing probability information of the individual symbols of a trellis code. These probabilities are required for iterative decoding schemes and concatenated coding schemes with soft decision decoding of the inner code, such as iterative decoding of turbo codes, which is discussed in Chapter 8.

The APP algorithm is a representative of a general class of belief propagation algorithms, which we will examine in Chapter 6, applied to the special case of trellis graphs.

Due to its importance, we will first give a functional description of the algorithm before deriving the formulas in detail. Figure 5.13 shows the example trellis of a short terminated trellis code with seven sections. The transmitted signal is $\mathbf{x} = [x_0, \dots, x_6]$, and the information symbols are $\mathbf{u} = [u_0, \dots, u_4, u_5 = 0, u_6 = 0]$, i.e., there are two tail bits that drive the encoder back into the zero-state.

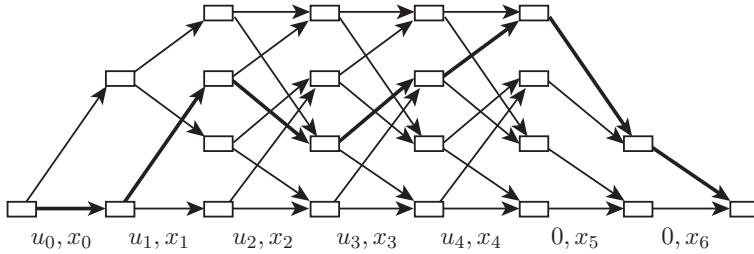


Figure 5.13: Example trellis of a short terminated trellis code.

The ultimate purpose of the algorithm is the calculation of a posteriori probabilities, such as $\Pr[u_r|\mathbf{y}]$ or $\Pr[x_r|\mathbf{y}]$, where \mathbf{y} is the received sequence observed at the output of a channel, whose input is the transmitted sequence \mathbf{x} . However, conceptually, it is more immediate to calculate the probability that the encoder traversed a specific transition in the trellis, i.e., $\Pr[s_r = i, s_{r+1} = j|\mathbf{y}]$, where s_r is the state at epoch r , and s_{r+1} is the state at epoch $r + 1$. The algorithm computes this probability as the product of three terms:

$$\begin{aligned}\Pr[s_r = i, s_{r+1} = j|\mathbf{y}] &= \frac{1}{\Pr(\mathbf{y})} \Pr[s_r = i, s_{r+1} = j, \mathbf{y}] \\ &= \frac{1}{\Pr(\mathbf{y})} \alpha_{r-1}(i) \gamma_r(j, i) \beta_r(j).\end{aligned}\quad (5.27)$$

The α -values are internal variables of the algorithm and are computed by the *forward recursion*

$$\alpha_{r-1}(i) = \sum_{\text{states } l} \alpha_{r-2}(l) \gamma_{r-1}(i, l). \quad (5.28)$$

This forward recursion evaluates α -values at time $r - 1$ from previously calculated α -values at time $r - 2$, and the sum is over all states l at time $r - 2$ that connect with state i at time $r - 1$. The forward recursion is illustrated in Figure 5.14. The α -values are initiated as $\alpha(0) = 1, \alpha(1) = \alpha(2) = \alpha(3) = 0$. This automatically enforces the boundary condition that the encoder starts in state 0.

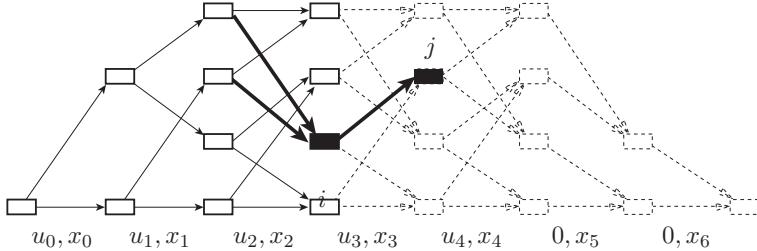


Figure 5.14: Illustration of the forward recursion of the APP algorithm.

The β -values are calculated by an analogous procedure, called the *backward recursion*

$$\beta_r(j) = \sum_{\text{states } k} \beta_{r+1}(k) \gamma_{r+1}(k, j), \quad (5.29)$$

and initialized as $\beta(0) = 1, \beta(1) = \beta(2) = \beta(3) = 0$ to enforce the terminating condition of the trellis code. The sum is over all states k at time $r + 1$ to which state j at time r connects. The backward recursion is illustrated in Figure 5.15.

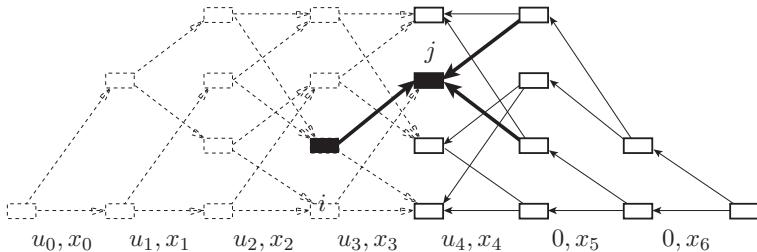


Figure 5.15: Illustration of the backward recursion of the APP algorithm.

The γ -values are conditional transition probabilities and are the inputs to the algorithm. $\gamma_r(j, i)$ is the joint probability that the state at time $r + 1$ is $s_{r+1} = j$ and that y_r is received, and it is calculated as

$$\gamma_r(j, i) = \Pr(s_{r+1} = j, y_r | s_r = i) = \Pr[s_{r+1} = j | s_r = i] \Pr(y_r | x_r). \quad (5.30)$$

The first term, $\Pr[s_{r+1} = j | s_r = i]$, is the a priori transition probability and is related to the probability of u_r . In fact, in our example, the top transition is associated with $u_r = 1$

and the bottom transition with $u_r = 0$. This factor can and will be used to account for a priori probability information on the bits u_r . In the sequel we will abbreviate this transition probability by

$$p_{ij} = \Pr(s_{r+1} = j | s_r = i) = \Pr(u_r). \quad (5.31)$$

The second term, $\Pr(y_r|x_r)$, is simply the conditional channel transition probability, given that symbol x_r is transmitted. Note that x_r is the symbol associated with the transition from state $i \rightarrow j$.

The a posteriori symbol probabilities $\Pr[u_r|y]$ can now be calculated from the a posteriori transition probabilities (5.27) by summing over all transitions corresponding to $u_r = 1$ and, separately, by summing over all transitions corresponding to $u_r = 0$, to obtain

$$p[u_r = 1|y] = \frac{1}{\Pr(y)} \sum_{\text{solid}} \Pr[s_r = i, s_{r+1} = j, y] \quad (5.32)$$

$$p[u_r = 0|y] = \frac{1}{\Pr(y)} \sum_{\text{dashed}} \Pr[s_r = i, s_{r+1} = j, y], \quad (5.33)$$

as shown in Figure 5.16, where the solid transitions correspond to $u_r = 1$, and the dashed transitions correspond to $u_r = 0$ as illustrated on the left.

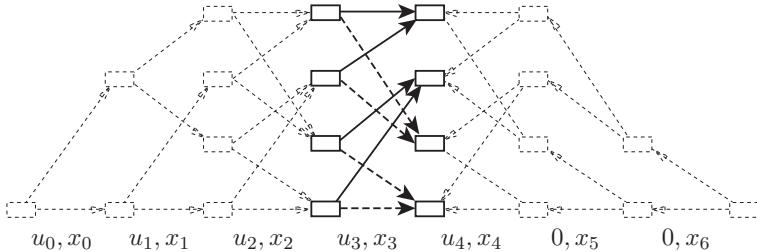


Figure 5.16: Computational dynamics of the final APP calculation.

A formal algorithm description is given at the end of this section, but first we present a rigorous derivation of the APP algorithm. This derivation was first given by Bahl et al. [9].

In the general case we will have need for the probability

$$q_{ij}(x) = \Pr(\tau(u_r, s_r) = x | s_r = i, s_{r+1} = j), \quad (5.34)$$

that is, the a priori probability that the output x_r at time r assumes the value x on the transition from state i to state j . This probability is typically a deterministic function of i

and j , unless there are parallel transitions, in which case x_r is determined by the uncoded information bits.

Before we proceed with the derivation, let us define the internal variables α and β by their probabilistic meaning. These are

$$\alpha_r(j) = \Pr(s_{r+1} = j, \tilde{\mathbf{y}}), \quad (5.35)$$

the joint probability of the partial sequence $\tilde{\mathbf{y}} = (y_{-l}, \dots, y_r)$ up to and including time epoch r and state $s_{r+1} = j$; and

$$\beta_r(j) = \Pr((y_{r+1}, \dots, y_l) | s_{r+1} = j), \quad (5.36)$$

the conditional probability of the remainder of the received sequence \mathbf{y} given that the state at time $r + 1$ is j .

With the above we now calculate

$$\begin{aligned} \Pr(s_{r+1} = j, \mathbf{y}) &= \Pr(s_{r+1} = j, \tilde{\mathbf{y}}, (y_{r+1}, \dots, y_l)) \\ &= \Pr(s_{r+1} = j, \tilde{\mathbf{y}}) \Pr((y_{r+1}, \dots, y_l) | s_{r+1} = j, \tilde{\mathbf{y}}) \\ &= \alpha_r(j) \beta_r(j), \end{aligned} \quad (5.37)$$

where we have used the fact that $\Pr((y_{r+1}, \dots, y_l) | s_{r+1} = j, \tilde{\mathbf{y}}) = \Pr((y_{r+1}, \dots, y_l) | s_{r+1} = j)$, i.e., if $s_{r+1} = j$ is known, events after time r are independent of the history $\tilde{\mathbf{y}}$ up to s_{r+1} .

In the same way we calculate via Bayes' expansion

$$\begin{aligned} \Pr(s_r = i, s_{r+1} = j, \mathbf{y}) &= \Pr(s_r = i, s_{r+1} = j, (y_{-l}, \dots, y_{r-1}), y_r, (y_{r+1}, \dots, y_l)) \\ &= \Pr(s_r = i, (y_{-l}, \dots, y_{r-1})) \Pr(s_{r+1} = j, y_r | s_r = i) \\ &\quad \times \Pr((y_{r+1}, \dots, y_l) | s_{r+1} = j) \\ &= \alpha_{r-1}(i) \gamma_r(j, i) \beta_r(j). \end{aligned} \quad (5.38)$$

Now, again applying Bayes' rule and $\sum_b p(a, b) = p(a)$, we obtain

$$\begin{aligned} \alpha_r(j) &= \sum_{\text{states } i} \Pr(s_r = i, s_{r+1} = j, \tilde{\mathbf{y}}) \\ &= \sum_{\text{states } i} \Pr(s_r = i, (y_{-l}, \dots, y_{r-1})) \Pr(s_{r+1} = j, y_r | s_r = i) \\ &= \sum_{\text{states } i} \alpha_{r-1}(i) \gamma_r(j, i). \end{aligned} \quad (5.39)$$

For a trellis code started in the zero state at time $r = -l$ we have the starting conditions

$$\alpha_{-l-1}(0) = 1, \alpha_{-l-1}(j) = 0, \quad j \neq 0. \quad (5.40)$$

As above, we similarly develop an expression for $\beta_r(j)$, i.e.,

$$\begin{aligned}\beta_r(j) &= \sum_{\text{states } i} \Pr(s_{r+2} = i, (y_{r+1}, \dots, y_l) | s_{r+1} = j) \\ &= \sum_{\text{states } i} \Pr(s_{r+2} = i, y_{r+1} | s_{r+1} = j) \Pr((y_{r+2}, \dots, y_l) | s_{r+2} = i) \\ &= \sum_{\text{states } i} \beta_{r+1}(i) \gamma_{r+1}(i, j).\end{aligned}\quad (5.41)$$

The boundary condition for $\beta_r(j)$ is

$$\beta_l(0) = 1, \beta_l(j) = 0, \quad j \neq 0, \quad (5.42)$$

for a trellis code which is terminated in the zero state.

Furthermore, the general form of the γ -values is given by

$$\begin{aligned}\gamma_r(j, i) &= \sum_{x_r} \Pr(s_{r+1} = j | s_r = i) \Pr(x_r | s_r = i, s_{r+1} = j) \Pr(y_r | x_r) \\ &= \sum_{x_r} p_{ij} q_{ij}(x_r) p_n(y_r - x_r),\end{aligned}\quad (5.43)$$

where we have used the conditional density function of the AWGN channel from (2.11), i.e., $\Pr(y_r | x_r) = p_n(y_r - x_r)$. The calculation of $\gamma_r(j, i)$ is not very complex and can most easily be implemented by a table lookup procedure.

Equations (5.39) and (5.41) are iterative and we can now compute the a posteriori state and transition probabilities via the following algorithm:

Step 1: Initialize $\alpha_{-l-1}(0) = 1, \alpha_{-l-1}(j) = 0$ for all non-zero states ($j \neq 0$) of the encoder FSM, and $\beta_l(0) = 1, \beta_l(j) = 0, j \neq 0$. Let $r = -l$.

Step 2: For all states j calculate $\gamma_r(j, i)$ and $\alpha_r(j)$ via (5.43) and (5.39).

Step 3: If $r < l$, let $r = r + 1$ and go to Step 2, else $r = l - 1$ and go to Step 4.

Step 4: Calculate $\beta_r(j)$ using (5.41). Calculate $\Pr(s_{r+1} = j, \mathbf{y})$ from (5.37), and $\Pr(s_r = i, s_{r+1} = j; \mathbf{y})$ from (5.27).

Step 5: If $r > -l$, let $r = r - 1$ and go to Step 4.

Step 6: Terminate the algorithm and output all the values $\Pr(s_{r+1} = j, \mathbf{y})$ and $\Pr(s_r = i, s_{r+1} = j, \mathbf{y})$.

Contrary to the maximum likelihood algorithm, the APP algorithms needs to go through the trellis twice, once in the forward direction and once in the reverse direction. What is worse, all the values $\alpha_r(j)$ must be stored from the first pass through the trellis. For a rate k/n convolutional code, for example, this requires $2^{k\nu}2l$ storage locations since there are $2^{k\nu}$ states for each of which we need to store a different value $\alpha_r(j)$ at each time epoch r . The storage requirement grows exponentially in the constraint length ν and linearly in the block length $2l$.

The a posteriori state and transition probabilities produced by this algorithm can now be used to calculate a posteriori information bit probabilities, i.e., the probability that the information k -tuple $u_r = u$, where u can vary over all possible binary k -tuples. Starting from the transition probabilities $\Pr(s_r = i, s_{r+1} = j | \mathbf{y})$, we simply sum over all transitions $i \rightarrow j$ which are caused by $u_r = u$. Denoting these transitions by $A(u)$, we obtain

$$\Pr(u_r = u | \mathbf{y}) = \sum_{(i,j) \in A(u)} \Pr(s_r = i, s_{r+1} = j | \mathbf{y}). \quad (5.44)$$

As mentioned above, another most interesting product of the APP decoder is the a posteriori probability of the transmitted output symbol x_r . Arguing analogously as above, and letting $B(x)$ be the set of transitions on which the output signal x can occur, we obtain

$$\begin{aligned} \Pr(x_r = x | \mathbf{y}) &= \sum_{(i,j) \in B(x)} \Pr(x | y_r) \Pr(s_r = i, s_{r+1} = j | \mathbf{y}) \\ &= \sum_{(i,j) \in B(x)} \frac{p_n(y_r - x_r)}{p(y_r)} q_{ij}(x) \Pr(s_r = i, s_{r+1} = j | \mathbf{y}), \end{aligned} \quad (5.45)$$

where the a priori probability of y_r can be calculated via

$$p(y_r) = \sum_{\substack{x' \\ ((i,j) \in B(x))}} p(y_r | x') q_{ij}(x'), \quad (5.46)$$

and the sum extends over all transitions $i \rightarrow j$.

Equation (5.45) can be much simplified if there is only one output symbol on the transition $i \rightarrow j$ as in the introductory discussion. In that case the transition automatically determines the output symbol, and

$$\Pr(x_r = x) = \sum_{(i,j) \in B(x)} \Pr(s_r = i, s_{r+1} = j | \mathbf{y}). \quad (5.47)$$

One problem we have to address is that of numerical stability. The α - and β -values in (5.39) and (5.41) decay rapidly and will underflow in any fixed precision implementation. We therefore normalize both values at each epoch, i.e.,

$$\alpha_r(i) \rightarrow \frac{\alpha_r(i)}{\sum_s \alpha_r(s)}, \quad \beta_r(i) \rightarrow \frac{\beta_r(i)}{\sum_s \beta_r(s)}. \quad (5.48)$$

This normalization has no effect on our final results such as (5.45), since these are similarly normalized. In fact, this normalization allows us to ignore the division by $p(y_r)$ in (5.45), and division by $\Pr(\mathbf{y})$ in (5.27), (5.32), and (5.33).

5.8 Log-APP and Approximations

While the APP algorithm is concise and consists only of multiplications and additions, current direct digital hardware implementations of the algorithm lead to complex circuits due to many real number multiplications involved in the algorithm. In order to avoid these multiplications, we transform the algorithm into the logarithm-domain. This results in the so-called *log-APP* algorithm.

First we transform the forward recursion (5.28), (5.39) into the logarithm-domain using the definitions

$$A_r(i) = \log(\alpha_r(i)), \quad \Gamma_r(i, l) = \log(\gamma_r(i, l)) \quad (5.49)$$

to obtain the *log-domain* forward recursion

$$A_{r-1}(i) = \log \left(\sum_{\text{states } l} \exp \left(A_{r-2}(l) + \Gamma_{r-1}(i, l) \right) \right). \quad (5.50)$$

Likewise the backward recursion can be transformed into the logarithm domain using the analogous definition $B_r(j) = \log(\beta_r(j))$, and we obtain

$$B_r(j) = \log \left(\sum_{\text{states } k} \exp \left(B_{r+1}(k) + \Gamma_{r+1}(k, j) \right) \right). \quad (5.51)$$

The product in (5.27) and (5.38) now turns into the simple sum

$$\alpha_{r-1}(i)\gamma_r(j, i)\beta_r(j) \rightarrow A_{r-1}(i) + \Gamma_r(j, i) + B_r(j). \quad (5.52)$$

Unfortunately, Equations (5.50) and (5.51) contain $\log()$ and $\exp()$ functions, which seem even more complex than the original multiplications. This is true; however, in most cases of current practical interest, the APP algorithm is used to decode binary codes, i.e., there are only two branches involved at each state and therefore only sums of two terms in (5.50) and (5.51). The logarithm of such a binary sum can be expanded as

$$\begin{aligned} \log(\exp(a) + \exp(b)) &= \log \left(\exp \left(\max(a, b) \right) \left(1 + \exp(-|a - b|) \right) \right) \\ &= \max(a, b) + \log \left(\left(1 + \exp(-|a - b|) \right) \right). \end{aligned}$$

It seems that little is gained from these manipulations, but the second term is now the only complex operation left, and there are a number of ways to approach this. The first, and most complex but precise method is to store the function

$$\log\left(\left(1 + \exp(-x)\right)\right), \quad x = |a - b|, \quad (5.53)$$

in a ROM lookup table. Given an example quantization of 4 bits, this is a 16×16 value lookup table, which is very manageable. Figure 5.17 shows the signal flow of this binary log-domain operation on the example of a node operation in the forward recursion.

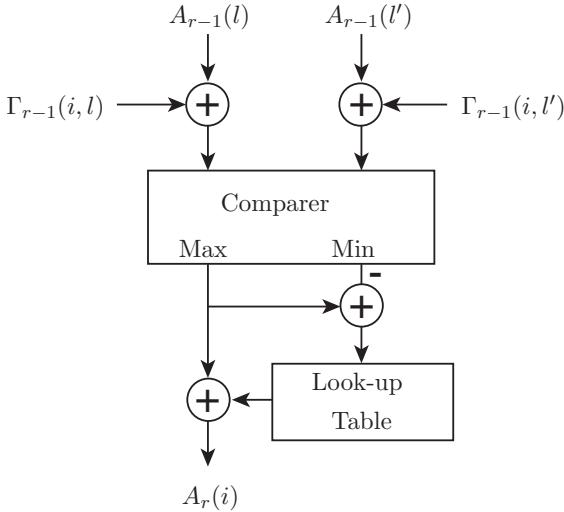


Figure 5.17: Signal flow diagram of the node calculation of the Log-APP algorithm.

Finally, to binary codes the algorithm computes the log-likelihood ratio (LLR) $\lambda(u_r)$ of the information bits u_r using the a posteriori probabilities (5.44) as

$$\begin{aligned} \lambda(u_r) &= \log\left(\frac{\Pr(u_r = 1)}{\Pr(u_r = 0)}\right) = \log\left(\frac{\sum_{(i,j) \in A(u=1)} \alpha_{r-1}(i)\gamma_r(j,i)\beta_r(j)}{\sum_{(i,j) \in A(u=0)} \alpha_{r-1}(i)\gamma_r(j,i)\beta_r(j)}\right), \\ \lambda(u_r) &= \log\left(\frac{\sum_{(i,j) \in A(u=1)} \exp(A_{r-1}(i) + \Gamma_r(j,i) + B_r(j))}{\sum_{(i,j) \in A(u=0)} \exp(A_{r-1}(i) + \Gamma_r(j,i) + B_r(j))}\right). \end{aligned} \quad (5.54)$$

The LLR is the quantity which is used in the iterative decoding algorithms of binary turbo codes as discussed later in this book. The range of the LLR is $[-\infty, \infty]$, where a large value signifies a high probability that $u_r = 1$.

A straightforward way of reducing the complexity of the Log-APP is to eliminate the ROM lookup table in Figure 5.17, [42]. This has the effect of approximating the forward and backward recursions by

$$\begin{aligned} A_{r-1}(i) &= \log \left(\sum_{\text{states } l} \exp \left(A_{r-2}(l) + \Gamma_{r-1}(i, l) \right) \right) \\ &\approx \max_{\text{states } l} \left(A_{r-2}(l) + \Gamma_{r-1}(i, l) \right) \end{aligned} \quad (5.55)$$

and

$$\begin{aligned} B_r(j) &= \log \left(\sum_{\text{states } k} \exp \left(B_{r+1}(k) + \Gamma_{r+1}(k, j) \right) \right) \\ &\approx \max_{\text{states } k} \left(B_{r+1}(k) + \Gamma_{r+1}(k, j) \right) \end{aligned} \quad (5.56)$$

It is very interesting to note that (5.55) is nothing else than our familiar Viterbi algorithm for maximum-likelihood sequence decoding. Furthermore, equation (5.56) is also a Viterbi algorithm, but operated in the reverse direction.

Analogously, the final LLR calculation in (5.54) is approximated by

$$\begin{aligned} \lambda(u_r) &\approx \max_{(i,j) \in A(u=1)} \left(A_{r-1}(i) + \Gamma_r(j, i) + B_r(j) \right) \\ &\quad - \max_{(i,j) \in A(u=1)} \left(A_{r-1}(i) + \Gamma_r(j, i) + B_r(j) \right). \end{aligned} \quad (5.57)$$

This algorithm is known the *Log-Max-APP algorithm*, and its big advantage is that it only uses additions and maximization operations to approximate the LLR of u_r . This computational savings is paid for by an approximate 0.5 dB loss when these decoders are used to decode turbo codes.

Further insight into the relationship between the Log-APP and its approximation can be gained by considering the expressing the LLR of u_r in its basic form, i.e.,

$$\lambda(u_r) = \log \left(\frac{\sum_{\mathbf{x};(u_r=1)} \exp \left(-\frac{|\mathbf{y} - \mathbf{x}|^2}{N_0} \right)}{\sum_{\mathbf{x};(u_r=1)} \exp \left(-\frac{|\mathbf{y} - \mathbf{x}|^2}{N_0} \right)} \right), \quad (5.58)$$

where the sum in the numerator extends over all coded sequences \mathbf{x} which correspond to information bit $u_r = 1$, and the denominator sum extends over all \mathbf{x} corresponding to $u_r = 0$.

It is quite straightforward to see that the MAX-Log-APP retains only the path in each sum which has the best metrics, and therefore the MAX-Log-APP calculates an approximation to the true LLR, given by

$$\lambda(u_r) \approx \min_{\mathbf{x};(u_r=0)} \frac{|\mathbf{y} - \mathbf{x}|^2}{N_0} - \min_{\mathbf{x};(u_r=1)} \frac{|\mathbf{y} - \mathbf{x}|^2}{N_0}, \quad (5.59)$$

i.e., the metric difference between the nearest path to \mathbf{y} with $u_r = 0$ and the nearest path with $u_r = 1$. For constant energy signals this simplifies to

$$\lambda(u_r) \approx \frac{(\mathbf{x}_r^{(1)} - \mathbf{x}_r^{(0)}) \cdot \mathbf{y}}{N_0/2}, \quad (5.60)$$

where $x_r^{(1)} = \arg \min_{\mathbf{x};(u_r=1)} |\mathbf{y} - \mathbf{x}|^2$, and $x_r^{(0)} = \arg \min_{\mathbf{x};(u_r=0)} |\mathbf{y} - \mathbf{x}|^2$.

For high-speed turbo decoding applications requiring up to ten iterations, evaluation of Equation (5.53) may be too complex, yet one is not readily willing to accept the half a dB loss entailed by using the Max-Log-APP. A very effective way of approximating (5.53) is [29]

$$\begin{aligned} & \max(a, b) + \log \left(\left(1 + \exp(-|a - b|) \right) \right) \\ & \approx \max(a, b) + \begin{cases} 0 & \text{if } |a - b| > T \\ C & \text{if } |a - b| \leq T. \end{cases} \end{aligned} \quad (5.61)$$

This simple threshold approximation is called *constant-Log-APP* algorithm. It is used in the UMTS turbo code [20] and leads to a degradation with respect to the full Log-APP of only 0.03 dB on this code [16], where the optimal parameters for this code are determined to be $C = 0.5$ and $T = 1.5$. This reduces the ROM lookup table of the Log-APP to a simple comparator circuit.

APP decoders are mainly used in decoders for turbo codes of various sizes. It is therefore desirable to make the APP algorithm itself independent of the block size of the overall code. While the forward recursion can be performed in synchrony with the incoming data, the backward recursion poses a problem, since the end of the block would need to be received before it can be started. A solution lies performing a *partial backward recursion*, starting some D symbol epochs in the future and using these values to calculate the LLRs at epoch r . The basic notion of this *sliding window* implementation is illustrated in Figure 5.18.

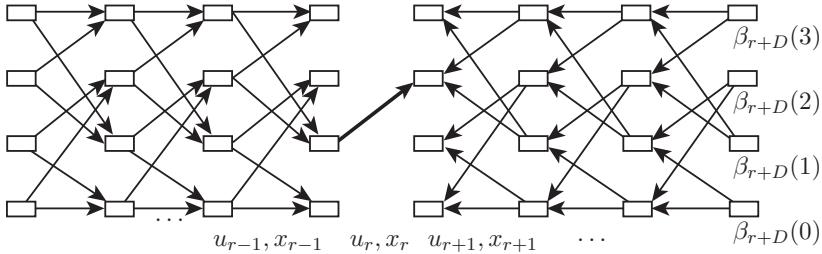


Figure 5.18: Sliding window approximation to the APP algorithm.

The question now is how to initialize the values $\beta_{r+D}(j)$, and the most typical method is to give them all the same value; a uniform initialization. Note that the exact values is irrelevant since the LLR eliminates constants.

Note that at first sight it seems that we have traded in a largely increased computational load, since for each forward recursion step, D backward recursion steps are needed to find the values of β at epoch r . However, it is computationally much more efficient to operate this algorithm in a block fashion. That is, for every D backward recursion steps, not only a single forward step at r is executed, but a number R of forward steps. Typical values are $D = 2R$, which leads to efficient shared memory implementations.

5.9 Error Analysis and Distance Spectrum

The error performance analysis of trellis codes under optimal decoding (Section 5.6) is almost exclusively based on a code's distance spectrum used in union bounding techniques. This approach leads to very tight bounds over a large region of signal-to-noise ratios of practical interest.

In contrast, the same bounding techniques have had only limited success in accurately bounding the performance of turbo codes and other iteratively decoded concatenated codes, and a second method, based on statistical analyses, is used to complement the distance spectrum based techniques.

As discussed in Chapter 3, the encoder for a convolutional code is a finite-state machine (FSM), whose transitions are determined by the input data sequence. The optimum trellis decoder uses a duplicate of this finite-state machine, which attempts to retrace the path (i.e., the state sequence) taken by the encoder FSM. The same principles apply to all codes, given their trellis representation is used for decoding.

Possibly the single most important measure of interest is the information bit error probability (BER) associated with our decoder. Unfortunately, it turns out that the

calculation of the BER is extremely complex and no efficient exact methods exist. We therefore look for more accessible ways of obtaining a measure of performance. We will proceed to consider the probability that a codeword error occurs, or, more appropriately, for trellis codes a code sequence error occurs. Such an error happens when the decoder follows a path in the trellis which diverges from the correct path somewhere in the trellis.

The decoder will make an error if the path it follows through its trellis does not coincide with the path taken by the encoder. Such a scenario is illustrated in Figure 5.19, where the decoder starts an error at time j by following an incorrect path which remerges with the correct sequence at time $j + L$.

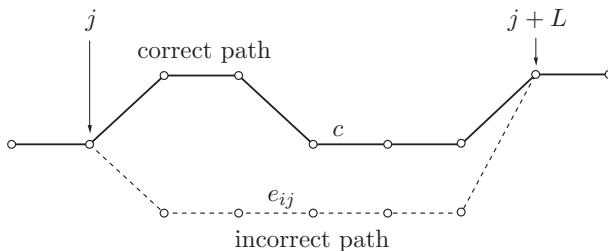


Figure 5.19: The correct path and an error path of length L shown in the code trellis.

In general there are many overlapping error paths possible, and Figure 5.20 shows an example trellis where the correct path is indicated by a solid line, and all possible paths, i.e., all error paths, by dotted lines. The total length of the trellis and the code is l , and we assume that the code is started in the zero-state and also terminated in the zero-state.

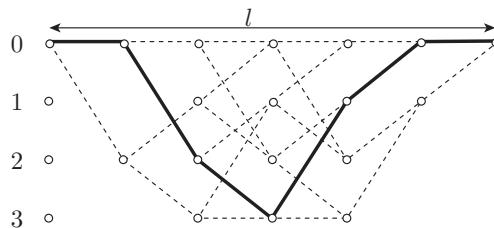


Figure 5.20: Trellis diagram showing a correct path and the set of possible error paths.

The probability of error, P , is the probability that any of the dotted paths in Figure 5.20 is chosen; that is, P is the probability of the union of the individual errors $e_{i,j}$, given

by

$$P = \Pr \left(\bigcup_j \bigcup_i e_{i,j} \mid c \right), \quad (5.62)$$

where $e_{i,j}$ is the i th error path departing from the correct path c at time j .

To obtain an average error probability, we also need to average over all correct paths, i.e.,

$$\bar{P} = \sum_c p(c) \Pr \left(\bigcup_j \bigcup_i e_{i,j} \mid c \right), \quad (5.63)$$

where $p(c)$ is the probability that the *encoder* chooses path c .

The probability in (5.63) is still difficult to calculate and we use the union bound³ to simplify the expression further to obtain

$$\bar{P} \leq \sum_c p(c) \sum_j \Pr \left(\bigcup_i e_{i,j} \mid c \right). \quad (5.64)$$

If the total length l of the encoded sequence is very large, as is typically the case for convolutional codes which don't have a predetermined codelength, \bar{P} will become large; in fact, it will approach unity as $l \rightarrow \infty$, and its probability may therefore not be a good measure for the performance of a trellis code. We therefore normalize \bar{P} per unit time and consider

$$\bar{P}_e = \lim_{l \rightarrow \infty} \frac{1}{l} \bar{P}. \quad (5.65)$$

Since, in many cases, the infinite trellis looks identical at every time unit, we can eliminate the sum over j in (5.64) to obtain

$$\bar{P}_e \leq \sum_c p(c) \Pr \left(\bigcup_i e_i \mid c \right), \quad (5.66)$$

where e_i is the event that an error starts at an arbitrary but fixed time unit, say j . Also, the correct sequence c up to node j and after node $j + L$ is irrelevant for the error path e_i of length L . Equation (5.66) can also be interpreted as the *first event error probability*, i.e., the probability that the decoder starts its first error event at node j . \bar{P} is upperbounded by the first event error probability.

In order to make the bound manageable, we apply the union bound again to obtain

$$\bar{P}_e \leq \sum_c p(c) \sum_{e_i} \Pr(e_i \mid c). \quad (5.67)$$

³The union bound states that the probability of the union of distinct events is smaller or equal to the sum of the probabilities of the individual events, i.e., $\Pr(\bigcup E_i) \leq \sum_i \Pr(E_i)$.

Let us denote $\Pr(e_i|c)$ by $P_{c \rightarrow e_i}$, which, since there are only two hypotheses involved now, is easily evaluated as (see Equation (2.14))

$$P_{c \rightarrow e_i} = Q\left(\sqrt{d_{ci}^2 \frac{RE_b}{2N_0}}\right), \quad (5.68)$$

where $R = k/n$ is the code rate in bits/symbol, N_0 is the one-sided noise power spectral density, E_b is the energy per information bit, and d_{ci}^2 is the squared Euclidean distance between the energy normalized signals on the error path e_i and the signals on the correct path c . Equation (5.68) is commonly called the *pair-wise error probability*.

The upper bound on \overline{P}_e now becomes

$$\overline{P}_e \leq \sum_c p(c) \sum_{e_i|c} Q\left(\sqrt{d_{ci}^2 \frac{RE_b}{2N_0}}\right), \quad (5.69)$$

which can be rearranged into

$$\overline{P}_e \leq \sum_{\substack{i \\ (d_i^2 \in \mathcal{D})}} A_{d_i^2} Q\left(\sqrt{d_i^2 \frac{RE_b}{2N_0}}\right), \quad (5.70)$$

by counting how often each of the squared Euclidean distances d_i^2 occurs in (5.69) between the signals on c and e_i in a particular trellis code. \mathcal{D} is the set of all possible such distances d_i^2 , and $A_{d_i^2}$ is the number of times d_i^2 occurs in this list, termed the *multiplicity* of d_i^2 . The multiplicity $A_{d_i^2}$ can be fractional since not all paths c may have the same set of distances d_i^2 with respect to their error paths. The smallest d_i^2 which can be found in the trellis is called d_{free}^2 , the free squared Euclidean distance or *free distance* for short.

The infinite set of pairs $\{d^2, A_{d^2}\}$ is called the *distance spectrum* of the code, and we immediately see its connection to the error probability of the code. Figure 5.21 shows the distance spectrum of the 16-state 8-PSK code from Table 3.1, whose free squared Euclidean distance $d_{\text{free}}^2 = 5.17$.

From the first error event probability we can obtain a bound on the average bit error probability (BER) by the following reasoning. Each error event $c \rightarrow e_i$ will cause a certain number of bit errors. If we replace A_{d^2} with B_{d^2} , which is the average number of bit errors on error paths with distance d^2 , we obtain a bound on the bit errors per time unit. Since our trellis code processes k bits per time unit, the average bit error probability is bounded by

$$\overline{P}_b \leq \sum_{\substack{i \\ (d_i^2 \in \mathcal{D})}} \frac{1}{k} B_{d_i^2} Q\left(\sqrt{d_i^2 \frac{RE_b}{2N_0}}\right). \quad (5.71)$$

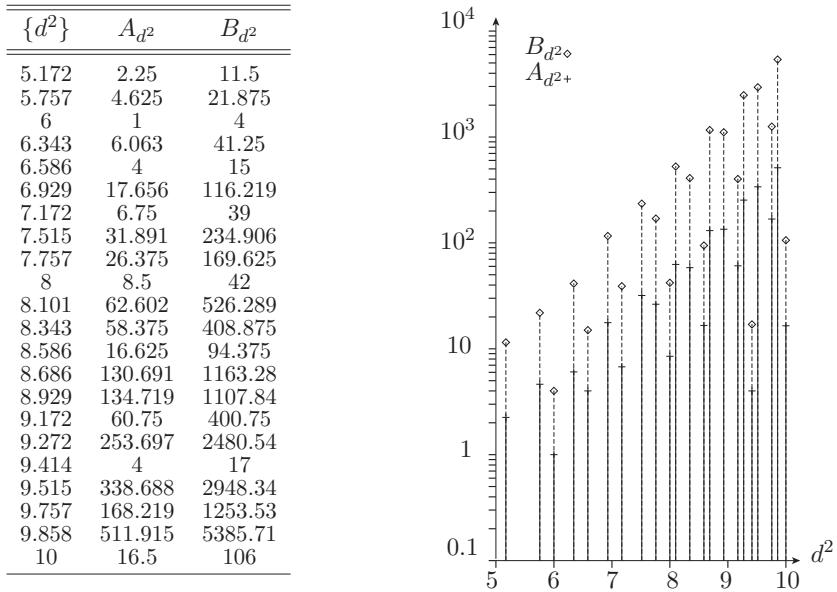


Figure 5.21: Distance spectrum of the code from Figure 3.1.

Figure 5.21 also shows the values B_{d^2} for this code.

Figure 5.22 is a plot of (5.71) for some popular 8-PSK Ungeröck trellis codes (solid curves) and compares it to results obtained by simulation (dashed curves). This illustrates the tightness of (5.71), which is a good measure of the codes' performance for $\overline{P}_b \lesssim 10^{-2}$.

Algorithms to compute a code's distance spectrum can be generated relatively easily from any one of the decoding algorithms described earlier in this chapter, by changing the decoding metric to d_{ci}^2 and keeping all the merged distances and their multiplicities.

If a trellis code is *regular* (or *geometrically uniform*), the averaging over c in (5.69) is not necessary and any code sequence may be taken as reference sequence. In the case of geometrically uniform codes, the Voronoi regions of all the code sequences are congruent, and hence the error probability is the same for all code sequences. For the more general case of regular codes, the distances between sequences only depend on their label differences, and the distance spectrum is identical for all sequences. However, many trellis codes are not regular, and we wish to have available a more general method.

We know that both the encoder and the optimal decoder are essentially identical FSMs denoted by M . Since we are interested in the set of squared Euclidean distances which can occur if those two FSM's follow different paths, we find it helpful to consider a new FSM $\mathcal{M} = M \otimes M$ with states (p, q) ; $p, q \in M$ and outputs $\delta((p, q) \rightarrow (p_1, q_1)) = d_{(p,q),(p_1,q_1)}^2$,

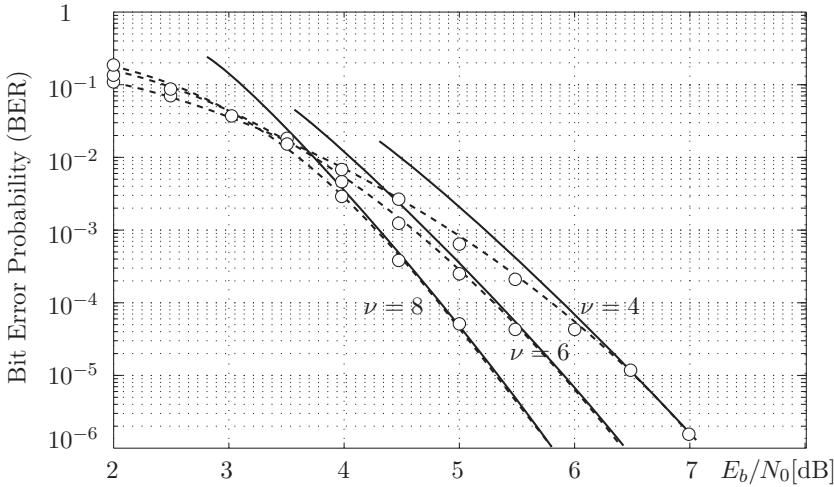


Figure 5.22: Bounds and bit error probabilities for three 8-PSLK codes.

where $d_{(p,q),(p_1,q_1)}^2$ is the *distance increment* accrued when the correct path c advances from state p to p_1 and the incorrect path e_i advances from q to q_1 . If the transition $(p, q) \rightarrow (p_1, q_1)$ is not possible, $\delta((p, q) \rightarrow (p_1, q_1)) = \infty$, by definition.

Let us pause here for a moment and discuss what motivates the construction of \mathcal{M} . It allows us to keep track of the evolution of both the correct path c and the error path e_i . Consider, for example, the case where the correct path progresses through the states $p \rightarrow p_1 \rightarrow p_2 \cdots \rightarrow p_{L-1} \rightarrow p_L$ and the error path leads through the states $q \rightarrow q_1 \rightarrow q_2 \cdots \rightarrow q_{L-1} \rightarrow q_L$, where $q = p$ and $q_L = p_L$. This is illustrated in Figure 5.23 for $L = 5$.

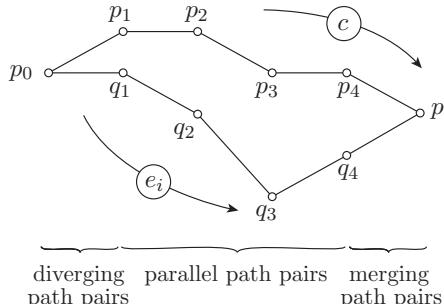


Figure 5.23: The correct path and an error path of length $L = 5$.

Multiplying the distance increments $\delta((p_{i-1}, q_{i-1}) \rightarrow (p_i, q_i))$ of \mathcal{M} , we obtain

$$\prod_{i=1}^L X^{\delta((p_{i-1}, q_{i-1}) \rightarrow (p_i, q_i))} = X^{\sum_{i=1}^L d_{(p_{i-1}, q_{i-1}), (p_i, q_i)}^2} = X^{d_{ci}^2}, \quad (5.72)$$

that is, the total distance between c and e_i appears in the exponent of our dummy base X .

Note that \mathcal{M} is a random FSM, that is, we are not assigning any inputs to the transitions. Each transition is taken with equal probability $1/2^k$, or if different with the probability of the transition $p \rightarrow p_1$ of the correct path c .

Now define the output transition matrix of our FSM \mathcal{M} having the entries

$$\mathbf{B} = \{b_{(pq)(p_1q_1)}\} = \left\{ \frac{1}{2^k} X^{\delta((p,q) \rightarrow (p_1,q_1))} \right\} = \left\{ \frac{1}{2^k} X^{d_{(p,q), (p_1,q_1)}^2} \right\}, \quad (5.73)$$

and it is quite straightforward to see that the $((p,p), (q,q))$ -entry of the L th power of \mathbf{B} is a polynomial in X whose exponents are all the distances between path pairs originating in (p,p) and terminating in (q,q) , and whose coefficients are the average multiplicities of these distances. The weighting factor $1/2^k$ weighs each transition with the probability that c moves from $p \rightarrow p_1$. This achieves the weighting of the distances with the probability of c . We now can use matrix multiplication to keep track of the distances between paths.

The size of \mathbf{B} quickly becomes unmanageable, since it grows with the square of the number of states in the code FSM. As an example, the matrix \mathbf{B} for the 4-state 8-PSK trellis code with $h^{(0)} = 5, h^{(1)} = 4, h^{(2)} = 2$ has 16×16 entries⁴ and is given by $\mathbf{B} =$

$$\begin{array}{ccccccccccccccccccccc} & 00 & 01 & 02 & 03 & 10 & 11 & 12 & 13 & 20 & 21 & 22 & 23 & 30 & 31 & 32 & 33 \\ \begin{matrix} 00 \\ 01 \\ 02 \\ 03 \\ 10 \\ 11 \\ 12 \\ 1 \\ 13 \\ 4 \\ 20 \\ 21 \\ 22 \\ 23 \\ 30 \\ 31 \\ 32 \\ 33 \end{matrix} & \left(\begin{array}{ccccccccccccccccccccc} 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 & 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^{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^{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^{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^{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^{0.6} X^{0.6} 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^{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^2 \\ 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^{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^{0.6} 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^{3.4} X^{0.6} 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^{0.6} X^{3.4} X^{3.4} X^{3.4} X^{0.6} X^{0.6} X^{0.6} X^{3.4} \\ 1 & 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^{0.6} X^{3.4} 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^{3.4} X^{0.6} X^{0.6} X^{0.6} X^{3.4} \\ 21 & 1 & X^2 & X^4 & X^2 & 1 & X^2 & X^4 & X^4 & X^2 & 1 & X^2 & X^2 & X^4 & X^2 & 1 & X^2 \\ 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^{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^{0.6} X^{3.4} \\ 22 & 1 & X^2 & X^4 & X^2 & 1 & X^2 & X^4 & X^4 & X^2 & 1 & X^2 & X^2 & X^4 & X^2 & 1 & X^2 \\ 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^{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^{0.6} X^{3.4} \\ 23 & 1 & X^2 & X^4 & X^2 & 1 & X^2 & X^4 & X^4 & X^2 & 1 & X^2 & X^2 & X^4 & X^2 & 1 & X^2 \\ 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^{0.6} X^{0.6} 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} \\ 30 & 1 & X^2 & X^4 & X^2 & 1 & X^2 & X^4 & X^4 & X^2 & 1 & X^2 & X^2 & X^4 & X^2 & 1 & X^2 \\ 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^{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^{0.6} X^{3.4} \\ 31 & 1 & X^2 & X^4 & X^2 & 1 & X^2 & X^4 & X^4 & X^2 & 1 & X^2 & X^2 & X^4 & X^2 & 1 & X^2 \\ 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^{3.4} X^{3.4} X^{0.6} X^{0.6} X^{0.6} X^{3.4} X^{0.6} X^{0.6} X^{3.4} X^{3.4} \\ 32 & 1 & X^2 & X^4 & X^2 & 1 & X^2 & X^4 & X^4 & X^2 & 1 & X^2 & X^2 & X^4 & X^2 & 1 & X^2 \\ X^2 & 1 & X^2 & X^4 & X^2 & 1 & X^2 & X^4 & X^4 & X^2 & 1 & X^2 & X^2 & X^4 & X^2 & 1 & X^2 \end{array} \right). \end{array}$$

⁴The exponents have been rounded to 1 decimal place for space reasons.

We wish to partition our matrix \mathbf{B} into a diverging, a parallel, and a merging section, corresponding to these different stages of an error event. For the 4-state code from above, this partition is given by

$$\mathbf{D} = \frac{1}{4} \begin{matrix} 01 & 02 & 03 & 10 & 12 & 13 & 20 & 21 & 23 & 30 & 31 & 32 \\ 00 & X^2 & X^4 & X^2 & X^2 & X^2 & X^4 & X^4 & X^2 & X^2 & X^4 & X^2 \\ 11 & X^2 & X^4 & X^2 & X^2 & X^2 & X^4 & X^4 & X^2 & X^2 & X^4 & X^2 \\ 22 & X^2 & X^4 & X^2 & X^2 & X^2 & X^4 & X^4 & X^2 & X^2 & X^4 & X^2 \\ 33 & X^2 & X^4 & X^2 & X^2 & X^2 & X^4 & X^4 & X^2 & X^2 & X^4 & X^2 \end{matrix}, \quad (5.74)$$

$$\mathbf{P} = \frac{1}{4} \begin{matrix} 01 & 02 & 03 & 10 & 12 & 13 & 20 & 21 & 23 & 30 & 31 & 32 \\ 01 & X^{0.6} & X^{0.6} & X^{3.4} & X^{3.4} & X^{0.6} & X^{0.6} & X^{3.4} & X^{0.6} & X^{0.6} & X^{0.6} & X^{3.4} \\ 02 & 1 & X^2 & X^4 & 1 & X^4 & X^2 & X^2 & X^4 & 1 & X^4 & X^2 & 1 \\ 03 & X^{3.4} & X^{3.4} & X^{0.6} & X^{3.4} & X^{0.6} & X^{0.6} & X^{3.4} & X^{0.6} & X^{3.4} & X^{0.6} & X^{3.4} \\ 10 & X^{3.4} & X^{0.6} & X^{0.6} & X^{3.4} & X^{0.6} & X^{0.6} & X^{3.4} & X^{3.4} & X^{0.6} & X^{0.6} & X^{0.6} \\ 12 & X^{3.4} & X^{0.6} & X^{0.6} & X^{3.4} & X^{0.6} & X^{3.4} & X^{0.6} & X^{0.6} & X^{3.4} & X^{0.6} & X^{3.4} \\ 13 & 1 & X^2 & X^4 & 1 & X^4 & X^2 & X^2 & X^2 & 1 & X^4 & X^2 & 1 \\ 20 & 1 & X^2 & X^4 & 1 & X^4 & X^2 & X^2 & X^4 & 1 & X^4 & X^2 & 1 \\ 21 & X^{3.4} & X^{0.6} & X^{3.4} & X^{0.6} & X^{3.4} & X^{0.6} & X^{3.4} & X^{0.6} & X^{3.4} & X^{3.4} & X^{3.4} \\ 23 & X^{3.4} & X^{0.6} & X^{0.6} & X^{3.4} & X^{0.6} & X^{0.6} & X^{3.4} & X^{3.4} & X^{0.6} & X^{0.6} & X^{0.6} \\ 30 & X^{3.4} & X^{3.4} & X^{0.6} & X^{3.4} & X^{0.6} & X^{0.6} & X^{3.4} & X^{0.6} & X^{3.4} & X^{0.6} & X^{3.4} \\ 31 & 1 & X^2 & X^4 & 1 & X^4 & X^2 & X^2 & X^4 & 1 & X^4 & X^4 & X^2 \\ 32 & X^{0.6} & X^{0.6} & X^{3.4} & X^{3.4} & X^{0.6} & X^{0.6} & X^{0.6} & X^{3.4} & X^{0.6} & X^{0.6} & X^{3.4} \end{matrix}, \quad (5.75)$$

$$\mathbf{M} = \frac{1}{4} \begin{matrix} 00 & 11 & 22 & 33 \\ 01 & X^{3.4} & X^{3.4} & X^{3.4} & X^{3.4} \\ 02 & X^2 & X^2 & X^2 & X^2 \\ 03 & X^{0.6} & X^{3.4} & X^{0.6} & X^{3.4} \\ 10 & X^{3.4} & X^{3.4} & X^{3.4} & X^{3.4} \\ 12 & X^{3.4} & X^{0.6} & X^{3.4} & X^{0.6} \\ 13 & X^2 & X^2 & X^2 & X^2 \\ 20 & X^2 & X^2 & X^2 & X^2 \\ 21 & X^{3.4} & X^{0.6} & X^{3.4} & X^{0.6} \\ 23 & X^{3.4} & X^{3.4} & X^{3.4} & X^{3.4} \\ 30 & X^{0.6} & X^{3.4} & X^{0.6} & X^{3.4} \\ 31 & X^2 & X^2 & X^2 & X^2 \\ 32 & X^{3.4} & X^{3.4} & X^{3.4} & X^{3.4} \end{matrix}, \quad (5.76)$$

where the diverging matrix \mathbf{D} is $N \times (N^2 - N)$, the parallel matrix \mathbf{P} is $(N^2 - N) \times (N^2 - N)$, and the merging matrix \mathbf{M} is $(N^2 - N) \times N$ and N is the number of states in M .

From Figure 5.23 we see that each error event starts when the correct path and the error path diverge and terminates when the two paths merge again. In between, e_i and c are never in the same state at the same time unit, and we refer to this middle part of the error event as the parallel section.

We may now describe all distances of error events of length exactly L by

$$\mathbf{G}_L = \mathbf{D}\mathbf{P}^{L-2}\mathbf{M}, \quad L \geq 2. \quad (5.77)$$

The $((p, p), (q, q))$ entry of \mathbf{G}_L is essentially a table of all weighted distances between length- L path pairs which originate from a given state p and merge in a given state q , L time units later. (5.77) can now be used to find the distance spectrum of a code up to any desired length. While (5.77) is a convenient way of describing the distance spectrum, its calculation is best performed using an adaptation of one of the decoding algorithms described earlier in this chapter.

The union bounds (5.70) and (5.71) are very tight for error probabilities smaller than about 10^{-2} as evidenced in Figure 5.22. Its disadvantage is that we need to evaluate the distance spectrum and also need some way of terminating the sum since (5.70) and (5.71) have no closed form expression.

Not so for our next bound, which is traditionally referred to as the *transfer-function bound* since it can also be derived via state transfer functions from a systems theory point of view. The transfer function bound using the pair-state trellis was first applied to TCM in [11]. The first step we take is loosening (5.70) by applying the inequality (see, e.g., [60, Page 83])

$$Q\left(\sqrt{d_i^2 \frac{RE_b}{2N_0}}\right) \leq \exp\left(-d_i^2 \frac{RE_b}{4N_0}\right). \quad (5.78)$$

Now, using (5.77), which is essentially a table of the distances between path pairs of length L , it is easy to see that

$$\overline{P_e} \leq \sum_{\substack{i \\ (d_i^2 \in \mathcal{D})}} A_{d_i^2} \exp\left(-\sqrt{d_i^2 \frac{RE_b}{2N_0}}\right) \quad (5.79)$$

$$= \frac{1}{N} \sum_{L=2}^{\infty} \mathbf{1}^T \mathbf{D} \mathbf{P}^{L-2} \mathbf{M} \mathbf{1} \Big|_{X=\exp\left(-\frac{RE_b}{4N_0}\right)}, \quad (5.80)$$

where $\mathbf{1} = (1, 1, \dots, 1)^T$ is an $N \times 1$ vector of all 1's and acts by summing all merger states while $\frac{1}{N} \mathbf{1}^T$ averages uniformly over all diverging states.

Pulling the sum into the matrix multiplication we obtain

$$\overline{P}_e \leq \frac{1}{N} \mathbf{1}^T \mathbf{D} \sum_{L=2}^{\infty} \mathbf{P}^{L-2} \mathbf{M} \mathbf{1} \Big|_{X=\exp\left(-\frac{RE_b}{4N_0}\right)} \quad (5.81)$$

$$= \frac{1}{N} \mathbf{1}^T \mathbf{D} (\mathbf{I} - \mathbf{P})^{-1} \mathbf{M} \mathbf{1} \Big|_{X=\exp\left(-\frac{RE_b}{4N_0}\right)}. \quad (5.82)$$

The above equation involves the inversion⁵ of an $(N^2 - N) \times (N^2 - N)$ matrix, which might be a sizeable task. In going from (5.81) to (5.82) we assumed that the infinite series in (5.81) converges. Using matrix theory, it can be shown that this series converges, if the largest eigenvalue of \mathbf{P} , $\lambda_{\max} < 1$ [48], and hence $(\mathbf{I} - \mathbf{P})^{-1}$ exists. This is always the case for non-catastrophic codes and for E_b/N_0 sufficiently large.

Note also that it is necessary to invert this matrix symbolically in order to obtain a closed form expression for the transfer function bound as a function of the signal-to-noise ratio E_b/N_0 . In practice, one might want to apply an iterative matrix inversion technique [8] to achieve a computationally fast procedure.

Since using (5.77) to search for all the distances up to a certain length L_{\max} is often easier, it is more efficient to use the tighter union bound for the distances up to a certain path pair length L_{\max} and use the transfer function only to bound the tail, i.e., we break (5.77) into two components, and obtain

$$\begin{aligned} \overline{P}_e \leq & \sum_{L=1}^{L_{\max}} \sum_{c_k^{(L)}} p(c_k^{(L)}) \sum_{e_i^{(L)} | c_k^{(L)}} Q\left(\sqrt{d_{ci}^2 \frac{RE_b}{2N_0}}\right) \\ & + \frac{1}{N} \mathbf{1}^T \mathbf{D} \sum_{L=L_{\max}+1}^{\infty} \mathbf{P}^{L-2} \mathbf{M} \mathbf{1} \Big|_{X=\exp\left(-\frac{RE_b}{4N_0}\right)}, \end{aligned} \quad (5.83)$$

where $c^{(L)}$ and $e_i^{(L)}$ are a correct path and incorrect path of length exactly L . The second term, the tail of the transfer function bound, can be overbounded by

$$\epsilon = \frac{1}{N} \mathbf{1}^T \mathbf{D} \sum_{L=L_{\max}+1}^{\infty} \mathbf{P}^{L-2} \mathbf{M} \mathbf{1} \Big|_{X=\exp\left(-\frac{RE_b}{4N_0}\right)} \leq \frac{\lambda_{\max}^{L_{\max}-1}}{1 - \lambda_{\max}}, \quad (5.84)$$

where λ_{\max} is the largest eigenvalue⁶ of \mathbf{P} .

⁵The matrix $\mathbf{I} - \mathbf{P}$ is sparse in most cases of interest and can therefore be inverted efficiently numerically [1].

⁶The Perron–Frobenius theorem from linear algebra tells us that a non-negative matrix has a non-negative largest eigenvalue [48]. A matrix is called non-negative, if every entry is a non-negative real number, which is the case for \mathbf{P} .

The computation of the eigenvalues of a matrix and its inversion are closely related, and not much seems to have been gained. The complexity of inverting $\mathbf{I} - \mathbf{P}$ and finding λ_{\max} are comparable. However, there are many efficient ways of bounding λ_{\max} . And we do not even need a very tight bound since we are only interested in the order of magnitude of ϵ .

One straightforward way of bounding λ_{\max} we will use now is obtained by applying Gershgorin's circle theorem (see Appendix 5.A). Doing this, we find that

$$\lambda_{\max} \leq \max_i \sum_j p_{ij} = g, \quad (5.85)$$

i.e., λ_{\max} is smaller than the maximum row sum of \mathbf{P} . A quick look at (5.75) shows that g , which in general is a polynomial in X , contains the constant term 1 for our 4-state code. This means that $g \geq 1$ for all real X and (5.85) is not useful as a bound in (5.84).

Since working with \mathbf{P} did not lead to a satisfactory bound, we go to \mathbf{P}^2 . From basic linear algebra we know that the maximum eigenvalue of \mathbf{P}^2 is λ_{\max}^2 . Let us then apply the Gershgorin bound to \mathbf{P}^2 , and we obtain

$$\lambda_{\max} \leq \sqrt{\max_i \sum_j p_{ij}^{(2)}} = \sqrt{g^{(2)}}. \quad (5.86)$$

Figure 5.24 shows plots of

$$\frac{\left(\sqrt{g^{(2)}}\right)^{L_{\max}-1}}{\left(1 - \sqrt{g^{(2)}}\right)}$$

versus the signal-to-noise ratio E_b/N_0 for various maximum lengths L_{\max} . From such plots one can quickly determine the maximum length L_{\max} in the trellis which needs to be searched for the distances d_{ci}^2 in order to achieve a prescribed accuracy.

There are many approaches to minimize the state space of \mathcal{M} (see for example [63, 62], and [27, Section 5.5]), most of them based on finite-state machine reduction techniques.

The approach chosen in this section does not reflect the actual historical development of these results. Methods to overcome the non-linearity of Ungeröök trellis codes and reducing the size of the computation FSM \mathcal{M} from N^2 states down to N were first presented by Zehavi and Wolf [61], and independently by Rouanne and Costello [43] with generalizations in [44] and appeared under the terminology of “quasi-regularity.” Later a slightly different, but equivalent approach was given by Biglieri and McLane [12], who called the property necessary for reduction “uniformity.”

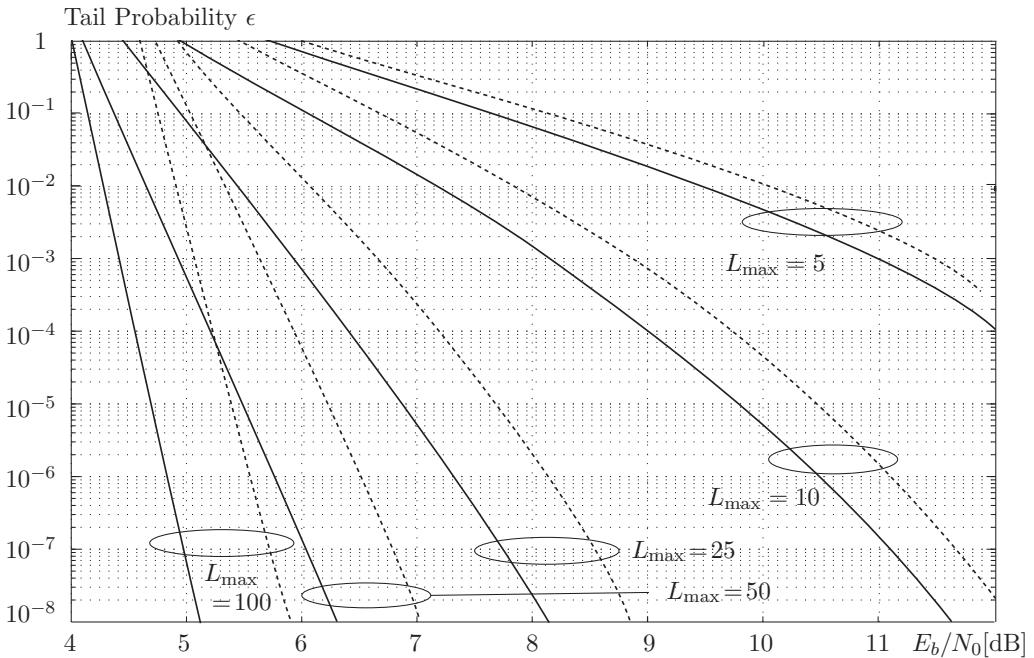


Figure 5.24: Plots of the bound on ϵ (5.84) using the exact value of λ_{\max} (solid curves) as well as the Gershgorin bound (dotted curves) for the 4-state 8-PSK trellis code for $L_{\max} = 5, 10, 25, 50, 100$, and 500.

5.10 Random Coding Analysis of Optimal Decoding

The calculation of the error probability is very difficult for any code of decent size, and, there is, in general, little hope of calculating an exact error probability for these codes. Consequently, finding bounds on their performance becomes the next best thing one can hope for. But, as we have seen, even finding bounds for specific codes is rather cumbersome. It requires the distance spectrum of the code as discussed earlier.

If we average the performance, i.e., the block, bit, or event error probabilities over all possible codes, assuming some distribution on the selection of codes, we find to our great astonishment that this average can be calculated relatively easily. This is the essence of Shannon's random coding bound idea [45], which we apply to trellis codes in this section.

Knowing the average of the performance of all codes, we can rest assured that there must exist at least one code whose individual performance is better than that calculated

average. This leads to the famous existence argument of good codes—and, immediately to the dilemma that while knowing of the existence of good codes, the argument tells us little on how to find these good codes. Nonetheless, random coding bounds are of great interest; they tell us about the achievable performance of good codes in general, and, together with performance lower bounds, accurately describe the limits of performance of large codes. They also give us a way to evaluate the inherent performance potential of different signal constellations.

In this section we commence with random coding bounds on the first event error probability of trellis codes, which we later use to derive bounds on the bit error probability. Let us then assume that we use an ML-decoder⁷ which selects the hypothesized transmitted sequence $\hat{\mathbf{x}}$ according to the highest probability metric $p(\mathbf{y}|\mathbf{x})$, where \mathbf{y} is the received sequence $\mathbf{x} + \mathbf{n}$, $\mathbf{x}^{(c)}$ is the transmitted sequence and \mathbf{n} is a noise sequence (see Section 2.5). This decoder makes an error if it decodes a sequence \mathbf{x}' , given that the transmitted sequence was \mathbf{x} . For an ML decoder this happens if and only if $p(\mathbf{y}|\mathbf{x}) \leq p(\mathbf{y}|\mathbf{x}')$.

Let c and e be paths through the trellis. Remember that c and e describe only the paths through the trellis (state sequences), and not the signals attached to these paths. The signal assignment is the actual encoding function, and we will average over all these functions. As in the case of specific trellis codes, let c be the correct path, that is, the one taken by the encoder, and let e denote an incorrect path which diverges from c at node j . Further, let \mathcal{E} denote the set of all incorrect paths that diverge from c at node j . A necessary condition for an error event to start at node j is that the incorrect path e accumulates a higher conditional probability than the correct path c over their unmerged segments of the trellis.

If we are using a convolutional code to generate the code trellis, the sets \mathcal{E} for different correct paths are congruent. They contain the same number of paths of a certain length. In a particular trellis code then, \mathbf{x} is the sequence of signals assigned along the correct path c and \mathbf{x}' is the sequence of signals assigned to e .

Using this notation, we rewrite (5.66) as

$$\overline{P}_e \leq \sum_c p(c) \int_{\mathbf{y}} p(\mathbf{y}|\mathbf{x}) \mathcal{I}\left(\bigcup_{e \in \mathcal{E}} e(p(\mathbf{y}|\mathbf{x}') \geq p(\mathbf{y}|\mathbf{x}))\right) d\mathbf{y}, \quad (5.87)$$

where $\mathcal{I}(B)$ is an indicator function such that $\mathcal{I}(B) = 0$ if $B = \emptyset$, the empty set, $\mathcal{I}(B) = 1$ if $B \neq \emptyset$, and $e(p(\mathbf{y}|\mathbf{x}') \geq p(\mathbf{y}|\mathbf{x}))$ is a path e for which $p(\mathbf{y}|\mathbf{x}') \geq p(\mathbf{y}|\mathbf{x})$. $\mathcal{I}\left(\bigcup_{e \in \mathcal{E}} e(p(\mathbf{y}|\mathbf{x}') \geq p(\mathbf{y}|\mathbf{x}))\right)$ simply says whether or not there exists an error path with a posteriori probability larger than the correct path.

We now specify the structure of our trellis code to be the one shown in Figure 5.25. The binary input vector at time r , u_r , has k components and enters a feed-forward shift

⁷Note that the assumption of an ML decoder is nowhere necessary to prove the capacity theorem in general; only a “good” decoder is needed, as shown by Shannon [45].

register at time unit r . The shift register stores the $\nu - 1$ most recent binary input vectors. The mapper function assigns a symbol x_r as a function of the input and the state, i.e., $x_r = \tau(u_r, s_r) = \tau(u_r, \dots, u_{r-\nu+1})$.

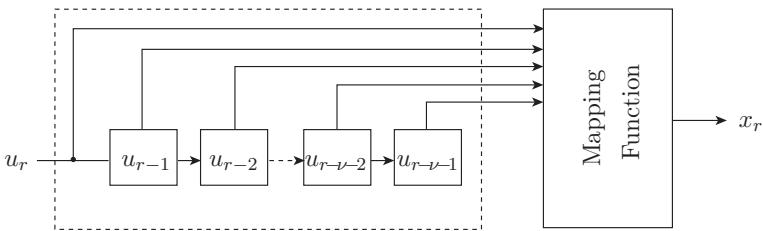


Figure 5.25: Trellis encoder structure used for the random coding bound arguments.

The particular realization of the trellis code as in Figure 5.25 has been chosen merely for convenience. The bounds which follow can be calculated for other structures as well, but the formalism can become quite unpleasant.

The symbol x_r is chosen from a signal set \mathcal{A} of size $|\mathcal{A}| = A$. The channel considered here is a memoryless channel used without feedback, i.e., the output symbol y is described by the conditional probability distribution $p(y|x)$, and the conditional probability of a symbol vector $p(\mathbf{y}|\mathbf{x})$ is then the product of the individual conditional symbol probabilities. We will need this property later in the proof to obtain an exponential bound on the error probability.

In order for an incorrect path e , which diverges from c at node j , to merge with the correct path at node $j + L$, the last $\nu - 1$ entries in the information sequence u'_j, \dots, u'_{j+L} associated with e must equal the last $\nu - 1$ entries in the information sequences u_j, \dots, u_{j+L} associated with c . This has to be the case since in order for the two paths e and c to merge at node $j + L$, their associated encoder states must be identical. Because an information vector u_j entering the encoder can affect the output for ν time units, this is also the maximum time it takes to force the encoder into any given state from any arbitrary starting state. Because the remaining information bits are arbitrary, we have $N_p \leq (2^k - 1)2^{k(L-\nu)}$ incorrect paths e of length L . Note that the choice of the information bits at node j is restricted because we stipulated that the incorrect path diverges at node j , which rules out the one path that continues to the correct state at node $j + 1$. This accounts for the factor $2^k - 1$ in the expression for N_p above.

We now proceed to express (5.87) as the sum over sequences of length L , and we rewrite

it as

$$\overline{P_e} \leq \sum_{L=\nu}^{\infty} \sum_{c^{(L)} \in \mathcal{C}^{(L)}} p(c^{(L)}) \int_{\mathbf{y}} p(\mathbf{y}|\mathbf{x}) \mathcal{I} \left(\bigcup_{e^{(L)} \in \mathcal{E}^{(L)}} e^{(L)} (p(\mathbf{y}|\mathbf{x}) \leq p(\mathbf{y}|\mathbf{x}')) \right) d\mathbf{y}, \quad (5.88)$$

where $\mathcal{C}^{(L)}$ is the set of all correct paths $c^{(L)}$ of length L starting at node j and $\mathcal{E}^{(L)}$ is the set of all incorrect paths $e^{(L)}$ of length L unmerged with $c^{(L)}$ from node j to node $j+L$. Note that $\bigcup_L \mathcal{E}^{(L)} = \mathcal{E}$.

Now we observe that if an error occurs at node j , then for at least one path e

$$p(\mathbf{y}|\mathbf{x}') \geq p(\mathbf{y}|\mathbf{x}). \quad (5.89)$$

Therefore, for any real parameter $\gamma \geq 0$

$$\sum_{e^{(L)} \in \mathcal{E}^{(L)}} \left[\frac{p(\mathbf{y}|\mathbf{x}')}{p(\mathbf{y}|\mathbf{x})} \right]^\gamma \geq 1. \quad (5.90)$$

We can raise both sides of (5.90) to the power of some non-negative parameter $\rho \geq 0$ and preserve the inequality, i.e.,

$$\left(\sum_{e^{(L)} \in \mathcal{E}^{(L)}} \left[\frac{p(\mathbf{y}|\mathbf{x}')}{p(\mathbf{y}|\mathbf{x})} \right]^\gamma \right)^\rho \geq 1. \quad (5.91)$$

This Chernoff bounding technique was introduced by Gallager [26] to derive random coding theorems for block codes, and then applied to convolutional codes by Viterbi [53].

We now use (5.91) to overbound the indicator function $\mathcal{I}(\cdot)$ by an exponential, i.e.,

$$\mathcal{I} \left(\bigcup_{e^{(L)} \in \mathcal{E}^{(L)}} e^{(L)} (p(\mathbf{y}|\mathbf{x}) \leq p(\mathbf{y}|\mathbf{x}')) \right) \leq \left(\sum_{e^{(L)} \in \mathcal{E}^{(L)}} \left[\frac{p(\mathbf{y}|\mathbf{x}')}{p(\mathbf{y}|\mathbf{x})} \right]^\gamma \right)^\rho. \quad (5.92)$$

Using (5.92) in (5.88), we obtain

$$\overline{P_e} \leq \sum_{L=\nu}^{\infty} \sum_{c^{(L)}} p(c^{(L)}) \int_{\mathbf{y}} p(\mathbf{y}|\mathbf{x}) \left(\sum_{e^{(L)} \in \mathcal{E}^{(L)}} \left[\frac{p(\mathbf{y}|\mathbf{x}')}{p(\mathbf{y}|\mathbf{x})} \right]^\gamma \right)^\rho d\mathbf{y}. \quad (5.93)$$

It is worth pausing here for a moment to reflect on what we are doing. If we set the parameter $\rho = 1$, the sum over $e^{(L)}$ will pull out all the way and we have in effect the union bound as in (5.67). We know, however, that the union bound is rather loose for low signal-to-noise ratios, or, alternatively, for high rates. Hence the parameter ρ , which allows us to tighten the bound in these areas.

Since we are free to choose γ , we select $\gamma = 1/(1 + \rho)$, which simplifies (5.93) to

$$\overline{P}_e \leq \sum_{L=\nu}^{\infty} \sum_{c^{(L)}} p(c^{(L)}) \int_{\mathbf{y}} p(\mathbf{y}|\mathbf{x})^{1/(1+\rho)} \left(\sum_{e^{(L)} \in \mathcal{E}^{(L)}} p(\mathbf{y}|\mathbf{x}')^{1/(1+\rho)} \right)^{\rho} d\mathbf{y}. \quad (5.94)$$

\overline{P}_e is the event error probability of a particular code since it depends on the signal sequences \mathbf{x} and \mathbf{x}' of the code. The aim of this section is to obtain a bound on an ensemble average of trellis codes, and we therefore average \overline{P}_e over all the codes in the ensemble, i.e.,

$$\text{Avg}\{\overline{P}_e\} \leq \text{Avg} \left\{ \sum_{L=\nu}^{\infty} \sum_{c^{(L)}} p(c^{(L)}) \int_{\mathbf{y}} p(\mathbf{y}|\mathbf{x})^{1/(1+\rho)} \left(\sum_{e^{(L)} \in \mathcal{E}^{(L)}} p(\mathbf{y}|\mathbf{x}')^{1/(1+\rho)} \right)^{\rho} d\mathbf{y} \right\}, \quad (5.95)$$

where $\text{Avg}\{\cdot\}$ denotes this ensemble average.

Using the linearity of the averaging operator and noting that there are exactly $N = 2^{kL}$ equiprobable paths in $\mathcal{C}^{(L)}$ of length L , because at each time unit there are 2^k possible choices to continue the correct path, we obtain

$$\text{Avg}\{\overline{P}_e\} \leq \sum_{L=\nu}^{\infty} \frac{1}{2^{kL}} \text{Avg} \left\{ \sum_{c^{(L)}} \int_{\mathbf{y}} p(\mathbf{y}|\mathbf{x})^{1/(1+\rho)} \left(\sum_{e^{(L)} \in \mathcal{E}^{(L)}} p(\mathbf{y}|\mathbf{x}')^{1/(1+\rho)} \right)^{\rho} d\mathbf{y} \right\}. \quad (5.96)$$

To continue let $\mathcal{X} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N\}$ be a possible assignment of signal sequences of length L associated with the paths $c^{(L)}$, i.e., \mathcal{X} is a particular code and let $q_{LN}(\{\mathcal{X}\}) = q_{LN}(\mathbf{x}_1, \dots, \mathbf{x}_N)$ be the probability of choosing this code. Note that $\mathcal{E}^{(L)} \subset \mathcal{C}^{(L)}$ since each incorrect path is also a possible correct path. Averaging over all codes means averaging over all signal sequences in these codes, i.e., over all assignments \mathcal{X} . Doing this we obtain

$$\begin{aligned} \text{Avg}\{\overline{P}_e\} &\leq \sum_{L=\nu}^{\infty} \frac{1}{2^{kL}} \sum_{c^{(L)}} \int_{\mathbf{y}} \sum_{\mathbf{x}_1} \dots \sum_{\mathbf{x}_N} q_{LN}(\mathcal{X}) p(\mathbf{y}|\mathbf{x})^{1/(1+\rho)} \\ &\quad \times \left(\sum_{e^{(L)} \in \mathcal{E}^{(L)}} p(\mathbf{y}|\mathbf{x}')^{1/(1+\rho)} \right)^{\rho} d\mathbf{y}, \end{aligned} \quad (5.97)$$

where \mathbf{x} is the signal sequence on $c^{(L)}$ and \mathbf{x}' is the one on $e^{(L)}$.

We can rewrite $q_{LN}(\mathcal{X}) = q_{L(N-1)}(\mathcal{X}'|\mathbf{x})$, where $\mathcal{X}' = \mathcal{X} \setminus \{\mathbf{x}\}$ is the set of signal sequences without \mathbf{x} and $q_{L(N-1)}(\mathcal{X}'|\mathbf{x})$ is the probability of $\mathcal{X}' \setminus \{\mathbf{x}\}$, conditioned on \mathbf{x} .

Restricting ρ to the unit interval, i.e., $0 \leq \rho \leq 1$, allows us to apply Jensen's inequality, $\sum p_i \alpha_i^\rho \leq (\sum p_i \alpha_i)^\rho$, to move the sum inside the exponential, and we obtain

$$\begin{aligned} \text{Avg} \{ \overline{P}_e \} &\leq \sum_{L=\nu}^{\infty} \frac{1}{2^{kL}} \sum_{c^{(L)}} \int_{\mathbf{y}} \sum_{\mathbf{x}} q(\mathbf{x}) p(\mathbf{y}|\mathbf{x})^{1/(1+\rho)} \\ &\times \left(\sum_{e^{(L)} \in \mathcal{E}^{(L)}} \underbrace{\sum_{\mathbf{x}_1} \cdots \sum_{\mathbf{x}_N}}_{(\mathbf{x}_i \neq \mathbf{x})} q_{L(N-1)}(\mathcal{X}'|\mathbf{x}) p(\mathbf{y}|\mathbf{x}')^{1/(1+\rho)} \right)^\rho d\mathbf{y}. \end{aligned} \quad (5.98)$$

But the inner term in (5.98) depends only on \mathbf{x}' and we can reduce (5.98) by summing over all other signal assignments, i.e.,

$$\begin{aligned} \text{Avg} \{ \overline{P}_e \} &\leq \sum_{L=\nu}^{\infty} \frac{1}{2^{kL}} \sum_{c^{(L)}} \int_{\mathbf{y}} \sum_{\mathbf{x}} q(\mathbf{x}) p(\mathbf{y}|\mathbf{x})^{1/(1+\rho)} \\ &\times \left(\sum_{e^{(L)} \in \mathcal{E}^{(L)}} \sum_{\mathbf{x}'} q(\mathbf{x}'|\mathbf{x}) p(\mathbf{y}|\mathbf{x}')^{1/(1+\rho)} \right)^\rho d\mathbf{y}. \end{aligned} \quad (5.99)$$

We observe that the last equation depends only on one correct and one incorrect signal sequence. We now further assume that the signals on $c^{(L)}$ and $e^{(L)}$ are assigned independently, i.e., $q(\mathbf{x}'|\mathbf{x}) = q(\mathbf{x}')$, and that each signal is chosen independently, making $q(\mathbf{x}) = \prod_{i=1}^{L-1} q(x_j)$ a product. In order to make this possible we must assume that the trellis codes are *time-varying* in nature, for otherwise each symbol would also depend on the choices of the ν most recent symbols. Note also that the signal assignments \mathbf{x} and \mathbf{x}' can be made independently since $e^{(L)}$ is in a different state than $c^{(L)}$ over the entire length of the error event.

This generalization to time-varying codes is quite serious, since almost all practical codes are time-invariant. It allows us, however, to obtain a much simpler version of the above bound, starting with

$$\begin{aligned} \text{Avg} \{ \overline{P}_e \} &\leq \sum_{L=\nu}^{\infty} \frac{1}{2^{kL}} \sum_{c^{(L)}} \int_{\mathbf{y}} \sum_{\mathbf{x}} \prod_{i=1}^L q(x_i) p(y_i|x_i)^{1/(1+\rho)} \\ &\left(\sum_{e^{(L)} \in \mathcal{E}^{(L)}} \sum_{\mathbf{x}'} \prod_{i=1}^L q(x'_i) p(y_i|x'_i)^{1/(1+\rho)} \right)^\rho d\mathbf{y}, \end{aligned} \quad (5.100)$$

where we have assumed that $p(\mathbf{y}|\mathbf{x}) = \prod_{i=1}^L p(y_i|x_i)$, that is, the channel is memoryless. In a final step we realize that the products are now independent of the index j and the

correct and incorrect path $c^{(L)}$ and $e^{(L)}$. We obtain

$$\begin{aligned} \text{Avg}\{\overline{P}_e\} &\leq \sum_{L=\nu}^{\infty} \int_{\mathbf{y}} \left(\sum_x q(x) p(y|x)^{1/(1+\rho)} \right)^L \\ &\quad \times \left(N_p \left(\sum_{x'} q(x') p(y|x')^{1/(1+\rho)} \right)^L \right)^{\rho} d\mathbf{y} \\ &= \sum_{L=\nu}^{\infty} N_p^{\rho} \int_{\mathbf{y}} \left(\left(\sum_x q(x) p(y|x)^{1/(1+\rho)} \right)^{1+\rho} \right)^L d\mathbf{y}, \end{aligned} \quad (5.101)$$

$$\leq \sum_{L=\nu}^{\infty} (2^k - 1)^{\rho} 2^{\rho k(L-\nu)} \left(\int_y \left(\sum_x q(x) p(y|x)^{1/(1+\rho)} \right)^{1+\rho} dy \right)^L, \quad (5.102)$$

where we have broken up the integral over \mathbf{y} into individual integrals over y in the last step.

Let us now define the error exponent

$$E_0(\rho, \mathbf{q}) \equiv -\log_2 \int_y \left(\sum_x q(x) p(y|x)^{1/(1+\rho)} \right)^{1+\rho} dy, \quad (5.103)$$

where $\mathbf{q} = (q_1, \dots, q_A)$ is the probability with which x is chosen from a signal set \mathcal{A} of size A . This allows us to write the error bound in the standard form:

$$\text{Avg}\{\overline{P}_e\} \leq (2^k - 1)^{\rho} 2^{-\nu E_0(\rho, \mathbf{q})} \sum_{L=0}^{\infty} 2^{\rho k L} 2^{-L E_0(\rho, \mathbf{q})} \quad (5.104)$$

$$= \frac{(2^k - 1)^{\rho} 2^{-\nu E_0(\rho, \mathbf{q})}}{1 - 2^{-(E_0(\rho, \mathbf{q}) - \rho k)}}, \quad \rho k < E_0(\rho, \mathbf{q}), \quad (5.105)$$

where we have used the summation formula for a geometric series and the condition $\rho k < E_0(\rho, \mathbf{q})$ assures convergence.

Since k is the number of information bits transmitted in one channel symbol x , we may call it the information rate in bits per channel use and denote it by the symbol R . This gives us our final bound on the event error probability:

$$\text{Avg}\{\overline{P}_e\} = \frac{(2^R - 1)^{\rho} 2^{-\nu E_0(\rho, \mathbf{q})}}{1 - 2^{-(E_0(\rho, \mathbf{q}) - \rho R)}}, \quad \rho < E_0(\rho, \mathbf{q})/R. \quad (5.106)$$

Using (5.104), we may easily obtain a bound on the average number of bit errors \overline{P}_b .

Since on an error path of length L there can be at most $L - \nu + 1$ bit errors, we obtain

$$\text{Avg} \{ \overline{P_b} \} \leq (2^R - 1)^\rho 2^{-\nu E_0(\rho, \mathbf{q})} \sum_{L=0}^{\infty} (L+1) 2^{\rho RL} 2^{-LE_0(\rho, \mathbf{q})} \quad (5.107)$$

$$= \frac{(2^R - 1)^\rho 2^{-\nu E_0(\rho, \mathbf{q})}}{(1 - 2^{-(E_0(\rho, \mathbf{q}) - \rho k)})^2}, \quad \rho < E_0(\rho, \mathbf{q})/R, \quad (5.108)$$

For large constraint lengths ν , only the exponent $E_0(\rho, \mathbf{q})$ will matter. $E_0(\rho, \mathbf{q})$ is a function of ρ and we wish to explore some of its properties before interpreting our bound. Clearly $E_0(\rho, \mathbf{q}) \geq 0$ for $\rho \geq 0$, since $\sum_x q(x)p(y|x)^{1/(1+\rho)} \leq 1$ for $\rho \geq 0$, that is, our exponent is positive, making the bound non-trivial. Furthermore, we have

Lemma 5.3 $E_0(\rho, \mathbf{q})$ is a monotonically increasing function of ρ .

Proof: We will apply the inequality⁸

$$\left(\sum_i p_i \alpha_i^r \right)^{\frac{1}{r}} \leq \left(\sum_i p_i \alpha_i^s \right)^{\frac{1}{s}}, \quad 0 < r < s; \alpha_i \geq 0, \quad (5.109)$$

which holds with equality if for some constant c : $p_i \alpha_i = cp_i$ for all i . Using (5.109) in the expression for the error exponent (5.103), we obtain

$$\begin{aligned} E_0(\rho, \mathbf{q}) &= -\log_2 \int_y \left(\sum_x q(x)p(y|x)^{1/(1+\rho)} \right)^{1+\rho} dy \\ &\leq -\log_2 \int_y \left(\sum_x q(x)p(y|x)^{1/(1+\rho_1)} \right)^{1+\rho_1} dy \\ &= E_0(\rho_1, \mathbf{q}), \end{aligned} \quad (5.110)$$

for $\rho_1 > \rho > -1$. Equality holds in (5.110) if and only if $p(y|x) = c$ for all $x \in \mathcal{A}$, but that implies that our channel has capacity $C = 0$. Therefore $E_0(\rho, \mathbf{q})$ is monotonically increasing for all interesting cases. Q.E.D.

Lemma 5.3 tells us that in order to obtain the best bound, we want to choose ρ as large as possible, given the constraint in (5.106), i.e., we choose

⁸This inequality is easily derived from Hölder's inequality

$$\sum_i \beta_i \gamma_i \leq \left(\sum_i \beta_i^{\frac{1}{\mu}} \right)^\mu \left(\sum_i \gamma_i^{\frac{1}{1-\mu}} \right)^{1-\mu}, \quad 0 < \mu < 1; \beta_i, \gamma_i \geq 0,$$

by letting $\beta_i = p_i^\mu \alpha_i^r$, $\gamma_i = p_i^{1-\mu}$ and $\mu = r/s$.

$$\rho = \frac{E_0(\rho, \mathbf{q})}{R} (1 - \epsilon), \quad (5.111)$$

where ϵ is the usual infinitesimally small number.

Our next lemma says that, unfortunately, the larger ρ and hence the error exponent, the smaller the associated rate.

Lemma 5.4 *The function $R(\rho) \equiv E_0(\rho, \mathbf{q})/\rho$ is a monotonically decreasing function of ρ .*

Proof: We will use the fact that $E_0(\rho, \mathbf{q})$ is a \cap -convex function in ρ . This fact is well-known and proven in [53], Appendix 3A.3. We therefore know that $\frac{\partial^2}{\partial \rho^2} E_0(\rho, \mathbf{q}) \leq 0$, i.e., the function $\frac{\partial}{\partial \rho} E_0(\rho, \mathbf{q})$ is strictly monotonically decreasing. Let us then consider

$$\frac{\partial}{\partial \rho} \frac{E_0(\rho, \mathbf{q})}{\rho} = \frac{1}{\rho^2} \left(\rho \frac{\partial E_0(\rho, \mathbf{q})}{\partial \rho} - E_0(\rho, \mathbf{q}) \right). \quad (5.112)$$

The first term in parentheses, $\rho \frac{\partial E_0(\rho, \mathbf{q})}{\partial \rho}$, lies on a straight line through the origin with slope $\frac{\partial E_0(\rho, \mathbf{q})}{\partial \rho}$. Furthermore, $E_0(\rho, \mathbf{q})$, as a function of ρ also passes through the origin, and its slope at the origin is larger than $\frac{\partial E_0(\rho, \mathbf{q})}{\partial \rho}$ for all $\rho > 0$. By virtue of the \cup -convexity of $E_0(\rho, \mathbf{q})$, the derivative (5.112) is negative, proving the lemma. Q.E.D.

We are therefore faced with the situation that the larger the rate R , the smaller the maximizing ρ and, hence, the smaller the error exponent $E_0(\rho, \mathbf{q})$ and $E_0(\rho, \mathbf{q}) \rightarrow 0$ as $\rho \rightarrow 0$, which can easily be seen from (5.103). Let us find out at what limiting rate R the error exponent approaches its value of zero, i.e., let us calculate

$$\lim_{\rho \rightarrow 0} R(\rho) = \lim_{\rho \rightarrow 0} \frac{E_0(\rho, \mathbf{q})}{\rho}. \quad (5.113)$$

Since this limit is of the form $\frac{0}{0}$, we can use the rule of de l'Hôpital and obtain the limit

$$\lim_{\rho \rightarrow 0} R(\rho) = \left. \frac{\partial E_0(\rho, \mathbf{q})}{\partial \rho} \right|_{\rho=0}. \quad (5.114)$$

This derivative is easily evaluated and to our great surprise and satisfaction we obtain

$$\left. \frac{\partial E_0(\rho, \mathbf{q})}{\partial \rho} \right|_{\rho=0} = \int_y \sum_x q(x) p(y|x) \log_2 \left(\frac{p(y|x)}{\sum_{x'} q(x') p(y|x')} \right) dy = C(\mathbf{q}), \quad (5.115)$$

the Shannon channel capacity, using the signal set \mathcal{A} and the input probability distribution \mathbf{q} . (See also [18, 53]). Maximizing over the input probability distribution \mathbf{q} will then achieve the channel capacity C .

We have now proved the important fact that random trellis codes can achieve the Shannon capacity, and, as we will see, with a very favorable error exponent (5.103). The bounds thus derived are an important confirmation of our decision to use convolutional codes to generate the code trellis, since they tell us that these codes can, in principle at least, achieve channel capacity.

Let us now concern ourselves with additive white Gaussian noise (AWGN) channels, since they are arguably the most important class of memoryless channels. For an AWGN-channel with complex input and output symbols x and y (from (2.14)), we have

$$p(y|x) = \frac{1}{2\pi N_0} \exp\left(-\frac{|y-x|^2}{2N_0}\right) \quad (5.116)$$

Furthermore, for $\rho = 1$, $E_0(\rho, \mathbf{q})$ can be simplified considerably by evaluating the integral over y , i.e.,

$$\begin{aligned} E_0(1, \mathbf{q}) &= -\log_2 \int_y \left(\sum_x q(x) \frac{1}{\sqrt{2\pi N_0}} \exp\left(-\frac{|y-x|^2}{4N_0}\right) \right)^2 \\ &= -\log_2 \sum_x \sum_{x'} q(x) q(x') \int_y \frac{1}{2\pi N_0} \exp\left(-\frac{|y-x|^2}{4N_0} - \frac{|y-x'|^2}{4N_0}\right) \\ &= -\log_2 \sum_x \sum_{x'} q(x) q(x') \exp\left(-\frac{|x-x'|^2}{4N_0}\right) = R_0(\mathbf{q}), \end{aligned} \quad (5.117)$$

where $R_0(\mathbf{q}) = E_0(1, \mathbf{q})$ is known in the literature as the cutoff rate of the channel. It used to be believed [59, 37] that R_0 represents a “practical” limit for coded systems. However, with the appearance of turbo codes, the significance of R_0 as a measure of practical importance has vanished.

On AWGN-channels the uniform input symbol distribution $\mathbf{q} = 1/A, \dots, 1/A$ is particularly popular, and we define $E_0(\rho, \text{uniform distribution}) = E_0(\rho)$ and the cutoff rate $R_0(\text{uniform distribution}) = R_0$, i.e., we simply omit the probability distribution vector \mathbf{q} .

Figure 5.26 shows $E_0(\rho)$ for an 8-PSK constellation on an AWGN-channel with signal-to-noise ratio $E_s/N_0 = 10$ dB. The error exponent function has a constant value of R_0 up to the rate $R = R_0$ and then rapidly drops to zero as $R \rightarrow C$. The figure also shows the error exponent for general codes (dashed curve), that is, the exponent in

$$\overline{P_B} < 2^{-NE}, \quad E(R) = \max_{\mathbf{q}} \max_{0 \leq \rho \leq 1} [E_0(\rho, \mathbf{q}) - \rho R], \quad (5.118)$$

which bounds the error probability of the ensemble of all (linear) codes of length N [53, Section 3.1]. A very thorough discussion about error exponents can be found in either [53] or [26].

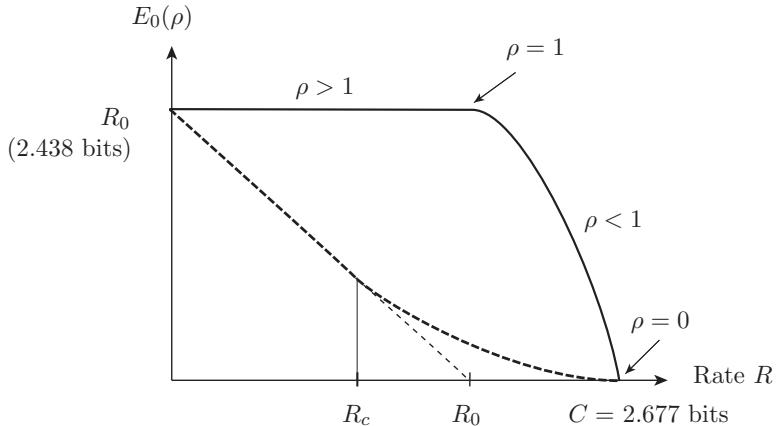


Figure 5.26: Error exponent as a function of the rate R for an 8-PSK constellation on an AWGN-channel at a signal-to-noise ratio of $E_s/N_0 = 10$ dB.

We discern the interesting fact that the error exponent for trellis codes is significantly larger than that for block codes, especially at high rates of transmission. Since the general code error exponent multiplies the block codelength, this is taken as an indication that the constraint length ν needed to obtain a prescribed error probability \bar{P}_b is much smaller than the blocklength of an equivalent block code. This may account for the popularity of trellis codes and their often superior behavior on high-noise channels, since, while the majority of academic publications are devoted to block codes, trellis (and in particular convolutional) codes are more widespread in applications.

While trellis codes can theoretically approach capacity, this has not turned out to be a practical avenue, since the decoding complexity of the corresponding decoders becomes prohibitively large. The achievement of approaching capacity with manageable decoder complexity had to await the invention of turbo codes, which, incidentally, do not rely on maximum-likelihood decoding.

5.11 Random Coding Analysis of Sequential Decoding

In the previous section we presented random coding performance bounds for trellis codes for maximum likelihood decoding. Since sequential decoding is not a maximum likelihood decoding method, the results in Section 5.10 do not apply.

The error analysis of sequential decoding is very difficult, and, again, we find it easier to generate results for an ensemble of codes via random coding arguments. The evaluation

of the error probability is not the main problem here, since, if properly dimensioned, both the stack and the Fano algorithm will almost always find the same error path as the maximum likelihood detector, and thus one might argue that we could simply let the code constraint length become large and thus achieve the bounds from Section 5.10. In this section then, we show that the problem of path loss will become dominant and determine the code's performance limits. In doing so, we will encounter R_0 , the *computational cutoff rate*, at the maximum communications rate above which the decoder fails due to buffer overflow (or ultimately, path loss).

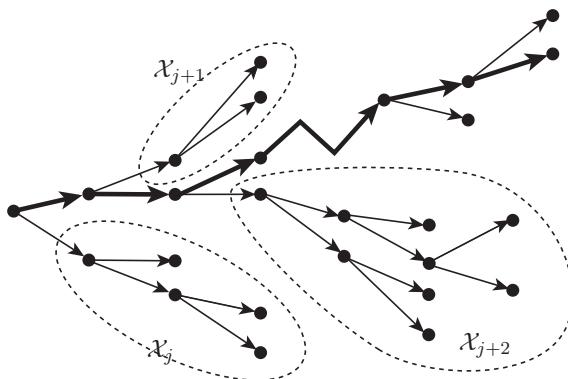


Figure 5.27: Incorrect subsets explored by a sequential decoder. The solid path is the correct one.

The difference with sequential decoding is, in contrast to ML-decoding, that its computational load is variable. And it is this computational load which can cause problems as we will see. Figure 5.27 shows an example of the search procedure of sequential decoding. The algorithm explores at each node an entire set of incorrect partial paths before finally continuing. This set at each node includes all the incorrect paths explored by the possibly multiple visits to that node as, for example, in the Fano algorithm. The sets \mathcal{X}'_j denote the sets of incorrect signal sequences \tilde{x}' diverging at node j , which are searched by the algorithm. Further, denote the number of signal sequences in \mathcal{X}'_j by C_j . Note that C_j is also the number of computations that need to be performed at node j , since each new path requires one additional metric calculation. This is the case because the algorithm explores two extensions at each step for a binary code, both resulting in distinct extension paths (Figure 5.27).

The problem becomes quite evident now, the number of computations at each node is variable, and it is this distribution of the computations which we want to examine. Again,

let $\tilde{\mathbf{x}}$ be the partial correct path through the trellis and $\tilde{\mathbf{x}}'_j$ be a partial incorrect path which diverges from $\tilde{\mathbf{x}}$ at node j . Furthermore, let $L_n(\tilde{\mathbf{x}}') = L(\tilde{\mathbf{x}}', \mathbf{y})$ be the metric of the incorrect path at node n , and let $L_m(\tilde{\mathbf{x}})$ be the metric of the correct path at node m . A path is searched further if and only if it is at the top of the stack, and hence, if $L_n(\tilde{\mathbf{x}}') < L_m(\tilde{\mathbf{x}})$, the incorrect path is not searched further until the metric of $\tilde{\mathbf{x}}$ falls below $L_n(\tilde{\mathbf{x}}')$. If

$$\min_{m \geq j} L_m(\tilde{\mathbf{x}}) = \lambda_j > L_n(\tilde{\mathbf{x}}'), \quad (5.119)$$

the incorrect path $\tilde{\mathbf{x}}'$ is never searched beyond node n .

We may now overbound the probability that the number of computations at node j exceeds a given value N_c by

$$\Pr(C_j \geq N_c) \leq \sum_{\mathbf{x}} p(\mathbf{x}) \int_{\mathbf{y}} p(\mathbf{y}|\mathbf{x}) \mathcal{B}(|e(p(\mathbf{y}|\mathbf{x}') \geq \lambda_j)| \geq N_c) d\mathbf{y}, \quad (5.120)$$

where $e(p(\mathbf{y}|\mathbf{x}') \geq \lambda_j)$ is an error path in \mathcal{X}_j whose metric exceeds λ_j and $|\star|$ is the number of such error paths. $\mathcal{B}(\star)$ is a boolean function which equals 1 if the expression is true and 0 otherwise. The function $\mathcal{B}(\star)$ in (5.120) then simply equals 1 if there are more than N_c error paths with metric larger than λ_i and 0 otherwise.

We now proceed to overbound the indicator function analogously to Section 5.10, by realizing that $\mathcal{B}(\star) = 1$ if at least N_c error paths have a metric such that

$$L_n(\tilde{\mathbf{x}}') \geq \lambda_j, \quad (5.121)$$

and, hence,

$$\left(\frac{1}{N_c} \sum_{\mathbf{x}' \in \mathcal{X}'_j} \exp(\alpha(L_n(\tilde{\mathbf{x}}') - \lambda_j)) \right)^\rho \geq 1, \quad (5.122)$$

where α and ρ are arbitrary positive constants. Note that we have extended the sum in (5.122) over all error sequences as is customary in random coding analysis. We may now use (5.122) to overbound the indicator function $\mathcal{B}(\star)$ and we obtain

$$\Pr(C_j \geq N_c) \leq N_c^{-\rho} \sum_{\mathbf{x}} p(\mathbf{x}) \int_{\mathbf{y}} p(\mathbf{y}|\mathbf{x}) \left(\sum_{\mathbf{x}' \in \mathcal{X}'_j} \exp(\alpha(L_n(\tilde{\mathbf{x}}') - \lambda_j)) \right)^\rho d\mathbf{y} \quad (5.123)$$

and, due to (5.119),

$$\exp(-\alpha\rho\lambda_i) \leq \sum_{m=j}^{\infty} \exp(-\alpha\rho L_m(\tilde{\mathbf{x}})), \quad (5.124)$$

and we have

$$\begin{aligned} \Pr(C_j \geq N_c) &\leq N_c^{-\rho} \sum_{\mathbf{x}} p(\mathbf{x}) \int_{\mathbf{y}} p(\mathbf{y}|\mathbf{x}) \left(\sum_{\mathbf{x}' \in \mathcal{X}'_j} \exp(\alpha L_n(\tilde{\mathbf{x}}')) \right)^\rho \\ &\quad \times \sum_{m=j}^{\infty} \exp(-\alpha \rho L_m(\tilde{\mathbf{x}})) d\mathbf{y}. \end{aligned} \quad (5.125)$$

Analogously to Section 5.10, let c be the correct path and let e be the incorrect path which diverges from c at node j . Let \mathcal{E} be the set of all incorrect paths and let \mathcal{E}'_j be the set of incorrect paths (not signal sequences) corresponding to \mathbf{x}'_j . Again, \mathbf{x} and \mathbf{x}' are, strictly taken, the signal sequences assigned to the correct and incorrect path, respectively, and $\tilde{\mathbf{x}}, \tilde{\mathbf{x}}'$ are the associated partial sequences. Let then $\text{Avg}\{\Pr(C_j \geq N_c)\}$ be the ensemble average of $\Pr(C_j \geq N_c)$ over all linear-trellis codes, i.e.,

$$\begin{aligned} \text{Avg}\{\Pr(C_j \geq N_c)\} &\leq N_c^{-\rho} \overline{\sum_c p(c) \int_{\mathbf{y}} p(\mathbf{y}|\mathbf{x}) \sum_{m=j}^{\infty} \exp(-\alpha \rho L_m(\tilde{\mathbf{x}}))} \\ &\quad \times \left(\sum_{e \in \mathcal{E}'_j} \overline{\exp(\alpha L_n(\tilde{\mathbf{x}}'))} \right)^\rho d\mathbf{y}. \end{aligned} \quad (5.126)$$

Note, we have used Jensen's inequality to pull the averaging into the second sum, which restricts ρ to $0 \leq \rho \leq 1$. Since we are using time-varying random trellis codes, (5.126) becomes independent of the starting node j , which we arbitrarily set to $j = 0$ now.

Observe there are at most 2^{kn} paths e of length n in $\mathcal{E}'_j = \mathcal{E}'$. Using this and the inequality⁹

$$\left(\sum a_i \right)^\rho \leq \sum a_i^\rho, \quad a_i \geq 0, \quad 0 \leq \rho \leq 1, \quad (5.127)$$

⁹This inequality is easily shown, i.e.,

$$\frac{\sum a_i^\rho}{(\sum a_i)^\rho} = \sum \left(\frac{a_i}{\sum a_i} \right)^\rho \geq \sum \left(\frac{a_i}{\sum a_i} \right) = 1,$$

where the inequality resulted from the fact that each term in the sum is ≤ 1 and $\rho \leq 1$.

we obtain¹⁰

$$\begin{aligned} \text{Avg}\{\Pr(C_0 \geq N_c)\} &\leq N_c^{-\rho} \overline{\sum_c p(c) \int_{\mathbf{y}} p(\mathbf{y}|\mathbf{x}) \sum_{m=0}^{\infty} \exp(-\alpha\rho L_m(\tilde{\mathbf{x}}))} \\ &\quad \times \sum_{n=0}^{\infty} 2^{kn\rho} \left(\overline{\exp(\alpha L_n(\tilde{\mathbf{x}}'))} \right)^{\rho} d\mathbf{y} \end{aligned} \quad (5.128)$$

$$\begin{aligned} &= N_c^{-\rho} \sum_c p(c) \overline{\sum_{m=0}^{\infty} \sum_{n=0}^{\infty} 2^{kn\rho}} \\ &\quad \times \overline{\int_{\mathbf{y}} p(\mathbf{y}|\mathbf{x}) \exp(-\alpha\rho L_m(\tilde{\mathbf{x}})) \left(\overline{\exp(\alpha L_n(\tilde{\mathbf{x}}'))} \right)^{\rho} d\mathbf{y}}. \end{aligned} \quad (5.129)$$

Now we substitute the metrics (see (5.10))

$$L_m(\tilde{\mathbf{x}}) = \sum_{r=0}^m \log \left(\frac{p(y_r|x_r)}{p(y_r)} \right) - k, \quad (5.130)$$

$$L_n(\tilde{\mathbf{x}}') = \sum_{r=0}^n \log \left(\frac{p(y_r|x'_r)}{p(y_r)} \right) - k, \quad (5.131)$$

into the expression (5.129) and use $\alpha = \frac{1}{1+\rho}$. Now rewrite the exponentials in (5.129) as

$$2^{kn\rho} \overline{\int_{\mathbf{y}} p(\mathbf{y}|\mathbf{x}) \exp(-\alpha\rho L_m(\tilde{\mathbf{x}})) \left(\overline{\exp(\alpha L_n(\tilde{\mathbf{x}}'))} \right)^{\rho}} = \begin{cases} 2^{-(m-n)E_c(\rho)-m(E_{ce}(\rho)-k\rho)} & \text{if } m \geq n \\ 2^{-(n-m)(E_e(\rho)-k\rho)-n(E_{ce}(\rho)-k\rho)} & \text{if } n \geq m. \end{cases} \quad (5.132)$$

The exponents used above are given by

$$\begin{aligned} 2^{-E_c(\rho)} &= \int_v \sum_x q(x)p(v|x) \left(\frac{p(v|x)}{p(v)} 2^{-k} \right)^{-\frac{\rho}{1+\rho}} \\ &= 2^{k\frac{\rho}{1+\rho}} \int_v \sum_x q(x)p(v|x)^{\frac{1}{1+\rho}} p(v)^{\frac{\rho}{1+\rho}} \\ &\leq 2^{k\frac{\rho}{1+\rho}} \left(\int_v \left(\sum_x q(x)p(v|x)^{\frac{1}{1+\rho}} \right)^{1+\rho} \right)^{\frac{1}{1+\rho}} \\ &= 2^{k\frac{\rho}{1+\rho}-\frac{1}{1+\rho}E_0(\rho,\mathbf{q})} = f_c, \end{aligned} \quad (5.133)$$

¹⁰Note that it is here that we need the time-varying assumption of the codes.

where we have used Hölder's inequality above with $\beta_i = \sum_x q(x)p(v|x)^{\frac{1}{1+\rho}}$, $\gamma_i = p(v)^{\frac{\rho}{1+\rho}}$ and $\lambda = \frac{1}{1+\rho}$, and, "magically" there appears the error exponent from Section 5.10, equation (5.103)!. Analogously, we also find

$$\begin{aligned} 2^{-(E_e(\rho)-k\rho)} &= 2^{k\rho} \int_v p(v) \left(\sum_{x'} q(x') \left(\frac{p(v|x')}{p(v)} 2^{-k} \right)^{\frac{1}{1+\rho}} \right)^\rho \\ &= 2^{k\frac{\rho^2}{1+\rho}} \int_v p(v)^{\frac{1}{1+\rho}} \left(\sum_{x'} q(x') p(v|x')^{\frac{1}{1+\rho}} \right)^\rho \\ &\leq 2^{k\frac{\rho^2}{1+\rho}} \left(\int_v \left(\sum_{x'} q(x') p(v|x')^{\frac{1}{1+\rho}} \right)^{1+\rho} \right)^{\frac{\rho}{1+\rho}} \\ &= 2^{k\frac{\rho^2}{1+\rho} - \frac{\rho}{1+\rho} E_0(\rho, \mathbf{q})} = f_e, \end{aligned} \quad (5.134)$$

where $\lambda = \frac{\rho}{1+\rho}$. Finally we obtain

$$\begin{aligned} 2^{-(E_{ce}(\rho)-k\rho)} &= 2^{k\rho} \int_v \sum_x q(x) p(v|x) \left(\sum_{x'} q(x') \left(\frac{p(v|x')}{p(v|x)} \right)^{\frac{1}{1+\rho}} \right)^\rho \\ &= 2^{k\rho} \int_v \sum_x q(x) p(v|x)^{\frac{1}{1+\rho}} \left(\sum_{x'} q(x') p(v|x')^{\frac{1}{1+\rho}} \right)^\rho \\ &= 2^{k\rho - E_0(\rho, \mathbf{q})}. \end{aligned} \quad (5.135)$$

Note now that, since $1 = \frac{\rho}{1+\rho} + \frac{1}{1+\rho}$, we have

$$2^{k\rho - E_0(\rho, \mathbf{q})} = 2^{k\frac{\rho^2}{1+\rho} - \frac{\rho}{1+\rho} E_0(\rho, \mathbf{q})} 2^{k\frac{\rho}{1+\rho} - \frac{1}{1+\rho} E_0(\rho, \mathbf{q})} = f_e f_c, \quad (5.136)$$

where the two factors f_e and f_c are defined in (5.133) and (5.134).

With this we can rewrite (5.129) as

$$\text{Avg}\{\Pr(C_0 \geq N_c)\} \leq N_c^{-\rho} \sum_c p(c) \sum_{m=0}^{\infty} \sum_{n=0}^{\infty} f_e^n f_c^m. \quad (5.137)$$

The double infinite sum in (5.137) converges if $f_e, f_c < 1$ and, hence from (5.136), if $\rho k < E_0(\rho, \mathbf{q})$ and we obtain

$$\text{Avg}\{\Pr(C_0 \geq N_c)\} \leq N_c^{-\rho} \frac{1}{(1-f_e)(1-f_c)}. \quad (5.138)$$

Similarly, it can be shown [52] that there exists a lower bound on the number of computations, given by

$$\text{Avg}\{\Pr(C_j \geq N_c)\} \geq N_c^{-\rho} (1 - o(N_c)). \quad (5.139)$$

Together, (5.138) and (5.139) characterize the computational behavior of sequential decoding. It is interesting to note that if $\rho \leq 1$, the expectation of (5.138) and (5.139), i.e., the expected number of computations, becomes unbounded, since

$$\sum_{N_c=1}^{\infty} N_c^{-\rho} \quad (5.140)$$

diverges for $\rho \leq 1$ or $k \geq R_0$. Information theory therefore tells us that we cannot beat the capacity limit by using very powerful codes and resorting to sequential decoding, since what happens is that as soon as the code rate reaches R_0 , the expected number of computations per node tends to infinity. In effect our decoder fails through buffer overflow.

Further credence to R_0 is given by the observation that rates $R = R_0$ at bit error probabilities of $P_b = 10^{-5} - 10^{-6}$ can be achieved with trellis codes. This observation was made by Wang and Costello [54], who constructed random trellis codes for 8-PSK and 16-QAM constellations which achieve R_0 with constraint lengths of 15 and 16.

5.12 Some Final Remarks

As we have seen, there are two broad classes of decoders, the depth-first and the breadth-first algorithms. Many attempts have been made at comparing the respective properties of these two basic approaches; for example, [40]—or, for convolutional codes, [52]—is an excellent and inexhaustible source of information. Many of the random coding arguments in [52] for convolutional codes can be extended to trellis codes with little effort.

Where are we standing then? Sequential decoding has been popular in particular for relatively slow transmission speeds, since the buffer sizes can then be dimensioned such that buffer overflow is controllable. Sequential decoding, however, suffers from two major drawbacks. Firstly, it is a “sequential” algorithm, i.e., modern pipelining and parallelizing are very difficult, if not impossible, to accomplish. Secondly, the metric used in sequential decoding contains the “bias” term accounting for the different path lengths. This makes sequential decoding very channel dependent. Furthermore, this bias term may be prohibitively complex to calculate for other than straight channel coding applications (see, e.g., [55]).

Breadth-first search algorithms, in particular the optimal Viterbi algorithm and the popular M -algorithm, do not suffer from the metric “bias” term. These structures can also be parallelized much more readily, which makes them good candidates for VLSI

implementations. They are therefore very popular for high-speed transmission systems. The Viterbi algorithm can be implemented with a separate metric calculator for each state. More on the implementation aspects of parallel Viterbi decoder structures can be found in [15, 22, 17]. The Viterbi decoder has proven so successful in applications that it is the algorithm of choice for most applications of code decoding at present.

The M -algorithm can also be implemented exploiting inherent parallelism of the algorithm, and [46] discusses an interesting implementation which avoids the sorting of the paths associated with the basic algorithm. The M -algorithm has also been successfully applied to multi-user detection, a problem which can also be stated as a trellis (tree) search [56], and to the decoding of block codes.

However the importance of all these decoding algorithms, with the likely exception of the Viterbi algorithm, is rapidly fading in favor of the APP decoding algorithms. The impact of iterative decoding of large error control codes, which use the APP algorithms as component (see Chapters 8 and 9 on turbo coding and related topics), has been so revolutionary as to push other strategies into virtual obscurity.

Appendix 5.A

The following theorem is also known as *Gerschgorin's circle theorem*:

Theorem 5.5 *Every eigenvalue λ_i of a matrix \mathbf{P} with arbitrary complex entries lies in at least one of the circles C_i , whose centers are at p_{ii} and whose radii are $r_i = \sum_{j \neq i} |p_{ij}|$, i.e., r_i is the i -th absolute row sum without the diagonal element.*

Proof: $\mathbf{Px} = \lambda \mathbf{x}$ immediately leads to

$$(\lambda - p_{ii})x_i = \sum_{j \neq i} p_{ij}x_j. \quad (5.141)$$

Taking absolute values on both sides and applying the triangle inequality, we obtain

$$|\lambda - p_{ii}| \leq \sum_{j \neq i} |p_{ij}| |x_j| / |x_i|. \quad (5.142)$$

Now let x_i be the largest component of \mathbf{x} , then $|x_j| / |x_i| \leq 1$, and $|\lambda - p_{ii}| \leq r_i$. Q.E.D.

In the application in Section 5.9, all entries $p_{ij} \geq 0$, and hence

$$(\lambda - p_{ii}) \leq \sum_{j \neq i} p_{ij} \Rightarrow \lambda \leq \sum_j p_{ij}. \quad (5.143)$$

Now, the largest eigenvalue λ_{\max} must be smaller than the largest row sum, i.e.,

$$\lambda_{\max} \leq \max_i \sum_j p_{ij}. \quad (5.144)$$

Appendix 5.B

In this appendix we describe an efficient algorithm to produce an equivalent FSM to a given FSM \mathcal{M} , which has the minimal number of states among all equivalent FSMs. The algorithm starts out assuming all states are equivalent, and then successively partitions the sets of equivalent states until all true equivalent states are found. The algorithm terminates in a finite number of steps, since this refinement of the partitioning must end when each original state is in an equivalent set by itself. The algorithm performs the following steps:

Step 1: Form a first partition P_1 of the states of \mathcal{M} by grouping together states which produce identical sets of outputs $\delta(U \rightarrow U')$ as we go through all transitions $U \rightarrow U'$.

Step 2: Obtain the $(l + 1)$ th partition P_{l+1} from the l th partition P_l as follows: Two states u and v are in the same equivalent group of P_{l+1} if and only if

- (i) U and V are in the same equivalent group of P_l , and
- (ii) for each pair of transitions $U \rightarrow U'$ and $V \rightarrow V'$ which produce the same output, U' and V' are in the same equivalent set of P_l .

Step 3: Repeat Step 2 until no further refinement occurs, i.e., until $P_{l+1} = P_l$. P_l is the final desired partition of the states of \mathcal{M} into equivalent states. Any member of the group can now be used to represent the entire group.

The recursive nature of the algorithm quickly proves that it actually produces the desired partition. Namely, if two states are in P_l , all their *sequences* of output sets must be identical for l steps, since their successors are in P_{l-1} , etc., all the way back to P_1 , which is the first and largest group of states which produce identical sets of outputs.

Now, if $P_{l+1} = P_l$, the successors of the above two states are in P_l also, and hence their sequences of output sets are identical for $l + 1$ steps also. We conclude that their sequences of output sets are therefore identical for all time.

The above algorithm can be extended to input and output equivalence of FSM's. The interested reader will find a more complete discussion in [19, Chapter 4].

Appendix 5.C

In this appendix we calculate the vector Euclidean distance for a specific set \mathcal{C}_p of retained paths. Noting that $\boldsymbol{\delta}_n$ from (5.19) is a vector of Gaussian random variables (see also Section 2.6), we can easily write down its probability density function [57], viz.

$$p(\boldsymbol{\delta}_n) = \frac{1}{(2\pi)^{M/2} |\mathbf{R}|^{1/2}} \exp\left(-\frac{1}{2} (\boldsymbol{\mu} - \boldsymbol{\delta}_n)^T \mathbf{R}^{-1} (\boldsymbol{\mu} - \boldsymbol{\delta}_n)\right) \quad (5.145)$$

where $\boldsymbol{\mu}$ is the vector of mean values given by $\mu_i = d_i^2$, and \mathbf{R} is the covariance matrix of the Gaussian random variables $\boldsymbol{\delta}_n$ whose entries $r_{ij} = E \left[(\delta_n^{(i,c)} - \mu_i)(\delta_n^{(j,c)} - \mu_j) \right]$ can be evaluated as

$$r_{ij} = \begin{cases} 2N_0 \left(d_i^2 + d_j^2 - d_{ij}^2 \right) & \text{if } i \neq j, \\ 4N_0 d_i^2 & \text{if } i = j \end{cases} \quad (5.146)$$

and where

$$d_{ij}^2 = \left| \tilde{\mathbf{x}}^{(p_i)} - \tilde{\mathbf{x}}^{(p_j)} \right|^2 \quad \text{and} \quad d_i^2 = \left| \tilde{\mathbf{x}}^{(p_i)} - \tilde{\mathbf{x}}^{(c)} \right|. \quad (5.147)$$

The vector $\boldsymbol{\mu}$ of mean values is given by $\mu_i = d_i^2$.

Now the probability of losing the correct path at time n can be calculated by

$$\Pr(\text{CPL} | \mathcal{C}_P) = \int_{\boldsymbol{\delta}_n \leq \mathbf{0}} p(\boldsymbol{\delta}_n) d\boldsymbol{\delta}_n. \quad (5.148)$$

Equation (5.148) is difficult to evaluate due to the correlation of the entries in $\boldsymbol{\delta}_n$, but one thing we know is that the area $\boldsymbol{\delta}_n \leq \mathbf{0}$ of integration is convex. This allows us to place a hyperplane through the point closest to the center of the probability density function, $\boldsymbol{\mu}$, and overbound (5.148) by the probability that the noise carries the point $\boldsymbol{\mu}$ across this hyperplane. This results in a simple one-dimensional integral, whose value is given by (compare also (2.14))

$$\Pr(\text{CPL} | \mathcal{C}_P) \leq Q \left(\sqrt{\frac{d_l^2}{2N_0}} \right), \quad (5.20)$$

where d_l^2 , the Vector Euclidean distance, is given by

$$d_l^2 = 2N_0 \min_{\mathbf{y} \leq \mathbf{0}} (\boldsymbol{\mu} - \mathbf{y})^T \mathbf{R}^{-1} (\boldsymbol{\mu} - \mathbf{y}), \quad (5.149)$$

and \mathbf{y} is simply a dummy variable of minimization.

The problem of calculating (5.20) has now been transformed into the geometric problem of finding the point on the surface of the convex polytope $\mathbf{y} \leq \mathbf{0}$ which is closest to $\boldsymbol{\mu}$ using the distance measure of (5.149). This situation is illustrated in Figure 5.28 for a 2-dimensional scenario. The minimization in (5.149) is a constrained minimization of a quadratic form. Obviously, some of the constraints $\mathbf{y} \leq \mathbf{0}$ will be met with equality. These constraints are called the *active constraints*, i.e., if $\mathbf{y} = (\mathbf{y}^{(a)}, \mathbf{y}^{(p)})^T$ is the partitioning of \mathbf{y} into active and passive components, $\mathbf{y}^{(a)} = \mathbf{0}$. This minimum is the point \mathbf{y}_0 in Figure 5.28. The right-hand side of Figure 5.28 also shows the geometric configuration when the decorrelating linear transformation $\boldsymbol{\delta}'_n = \sqrt{\mathbf{R}^{-1}} \boldsymbol{\delta}_n$ is applied. The vector Euclidean distance

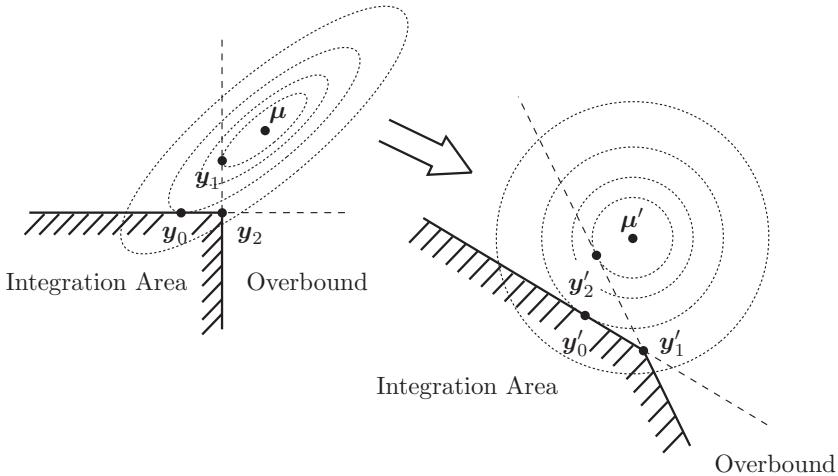


Figure 5.28: Illustration of the concept of the vector Euclidean distance with $M = 2$. The distance between \mathbf{y}_0 and $\boldsymbol{\mu}$ is d_l^2 . The right-hand side shows the space after decorrelation, and d_l^2 equals the standard Euclidean distance between \mathbf{y}'_0 and $\boldsymbol{\mu}'$.

(5.149) is invariant to such a transformation, but $E\left[(\delta_n^{(i,c)} - \mu'_i)(\delta_n^{(j,c)} - \mu'_j)\right] = \delta_{ij}$, i.e., the decorrelated metric differences are independent with unit variance each. Naturally we may work in either space. Since the random variables $\boldsymbol{\delta}'_n$ are independent, equal-variance Gaussian, we know from basic communication theory [57, Chapter 2], that the probability that $\boldsymbol{\mu}'$ is carried into the shaded region of integration can be overbounded by integrating over the half-plane not containing $\boldsymbol{\mu}'$, as illustrated in the figure. This leads to (5.20).

We now have to minimize

$$d_l^2 = 2N_0 \min_{\mathbf{y}^{(p)} \leq 0} \left(\begin{pmatrix} \boldsymbol{\mu}^{(p)} \\ \boldsymbol{\mu}^{(a)} \end{pmatrix} - \begin{pmatrix} \mathbf{y}^{(p)} \\ \mathbf{0} \end{pmatrix} \right)^T \begin{pmatrix} \mathbf{R}^{(pp)} & \mathbf{R}^{(pa)} \\ \mathbf{R}^{(ap)} & \mathbf{R}^{(aa)} \end{pmatrix}^{-1} \left(\begin{pmatrix} \boldsymbol{\mu}^{(p)} \\ \boldsymbol{\mu}^{(a)} \end{pmatrix} - \begin{pmatrix} \mathbf{y}^{(p)} \\ \mathbf{0} \end{pmatrix} \right), \quad (5.150)$$

where we have partitioned $\boldsymbol{\mu}$ and \mathbf{R} analogously to \mathbf{y} . After some elementary operations we obtain

$$\mathbf{y}^{(p)} = \boldsymbol{\mu}^{(p)} + \left(\mathbf{X}^{(pp)} \right)^{-1} \mathbf{X}^{(pa)} \boldsymbol{\mu}^{(a)} \leq \mathbf{0} \quad (5.151)$$

and

$$d_l^2 = 2N_0 \boldsymbol{\mu}^{(a)T} \left(\mathbf{X}^{(aa)} \right)^{-1} \boldsymbol{\mu}^{(a)}, \quad (5.152)$$

where¹¹

$$\begin{aligned}\mathbf{X}^{(pp)} &= \left[\mathbf{R}^{(pp)} - \mathbf{R}^{(pa)} \left(\mathbf{R}^{(aa)} \right)^{-1} \mathbf{R}^{(ap)} \right]^{-1}, \\ \mathbf{X}^{(aa)} &= \left(\mathbf{R}^{(aa)} \right)^{-1} \left[\mathbf{I} + \mathbf{R}^{(ap)} \mathbf{X}^{(pp)} \mathbf{R}^{(pa)} \right], \\ \mathbf{X}^{(pa)} &= -\mathbf{X}^{(pp)} \mathbf{R}^{(pa)} \left(\mathbf{R}^{(aa)} \right)^{-1}.\end{aligned}\quad (5.153)$$

We are now presented with the problem of finding the active components in order to evaluate (5.152). This is a combinatorial problem, i.e., we must test all $2^M - 1 = \sum_{i=1}^M \binom{M}{i}$ possible combinations of active components from the M entries in \mathbf{y} for compatibility with (5.151). This gives us the following procedure:

Step 1: Select all $2^M - 1$ combinations of active components and set $\mathbf{y}^{(a)} = \mathbf{0}$ for each.

Step 2: For each combination for which $\mathbf{y}^{(p)} \leq \mathbf{0}$, store the resulting d_l^2 from (5.152) in a list.

Step 3: Select the smallest entry from the list in Step 2 as d_l^2 .

As an example, consider again Figure 5.28. The $2^2 - 1 = 3$ combinations correspond to the points \mathbf{y}_0 , \mathbf{y}_1 , and \mathbf{y}_2 . The point \mathbf{y}_1 does not qualify, since it violates (5.151). The minimum is chosen between \mathbf{y}_0 and \mathbf{y}_2 . This process might be easier to visualize in the decorrelated space \mathbf{y}' , where all the distances are ordinary Euclidean distances, and the minimization becomes obvious.

One additional complication needs to be addressed at this point. The correlation matrix \mathbf{R} may be singular. This happens when one or more entries in \mathbf{y} are linearly dependent on the other entries. In the context of the restriction $\mathbf{y} \leq \mathbf{0}$, we have redundant conditions. The problem, again, is that of finding the redundant entries which can be dropped from consideration. Fortunately, our combinatorial search helps us here. Since we are examining all combinations of possible active components, we may simply drop any dependent combinations which produce a singular $\mathbf{R}^{(aa)}$ from further consideration without affecting d_l^2 .

¹¹These equations can readily be derived from the partitioned matrix inversion lemma:

$$\begin{pmatrix} A & B \\ C & D \end{pmatrix}^{-1} = \begin{pmatrix} E & F \\ G & H \end{pmatrix},$$

where

$$E = (A - BD^{-1}C)^{-1}; F = -EBD^{-1}; G = -D^{-1}CE.$$

and

$$H = D^{-1} + D^{-1}CEBD^{-1}.$$

Bibliography

- [1] F.L. Alvarado, "Manipulation and visualization of sparse matrices," *ORSA J. Comput.*, vol. 2, no. 2, Spring 1990.
- [2] J.B. Anderson, "Limited search trellis decoding of convolutional codes," *IEEE Trans. Inform. Theory*, vol. IT-35, Sept. 1989.
- [3] J.B. Anderson and S. Mohan, "Sequential coding algorithms: A survey and cost analysis," *IEEE Trans. Commun.*, vol. COM-32, no. 2, pp. 169–176, Feb. 1984.
- [4] J.B. Anderson and S. Mohan, *Source and Channel Coding: An Algorithmic Approach*, Kluwer Academic Publishers, Boston, MA, 1991.
- [5] T. Aulin, "Breadth first maximum-likelihood sequence detection," *IEEE Trans. Commun.*, vol. COM-47, no. 2, pp. 208–216, Feb. 1999.
- [6] T. Aulin, "Recovery Properties of the SA(B) Algorithm," Technical Report No. 105, Chalmers University of Technology, Sweden, Feb. 1991.
- [7] T. Aulin, "Study of a new trellis decoding algorithm and its applications," Final Report, ESTEC Contract 6039/84/NL/DG, European Space Agency, Noordwijk, The Netherlands, Dec. 1985.
- [8] O. Axelsson, "Milestones in the development of iterative solution methods," *J. Electr. Comp. Engr.*, vol. 2010. doi:10.1155/2010/972794.
- [9] L.R. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate," *IEEE Trans. Inform. Theory*, vol. IT-20, Mar. 1974.
- [10] K. Balachandran, "Design and performance of constant envelope and non-constant envelope digital phase modulation schemes," Ph.D. thesis, ECSE Dept. Rensselaer Polytechnic Institute, Troy, NY, Feb. 1992.
- [11] E. Biglieri, "High-level modulation and coding for nonlinear satellite channels," *IEEE Trans. Commun.*, vol. COM-32, pp. 616–626, May 1984.
- [12] E. Biglieri and P.J. McLane, "Uniform distance and error probability properties of TCM schemes," *IEEE Trans. Commun.*, vol. COM-39, pp. 41–53, Jan. 1991.
- [13] R.E. Blahut, *Principles and Practice of Information Theory*, Addison-Wesley, Reading, MA, 1987.

- [14] P.R. Chevillat and D.J. Costello, Jr., "A multiple stack algorithm for erasurefree decoding of convolutional codes," *IEEE Trans. Commun.*, vol. COM-25, pp. 1460–1470, Dec. 1977.
- [15] G.C. Clark and J.B. Cain, *Error-Correction Coding for Digital Communications*, Plenum Press, New York, 1983.
- [16] B. Classen, K. Blankenship, and V. Desai, "Turbo decoding with the constant-log-MAP algorithm," *Proc. Second Int. Symp. Turbo Codes and Related Appl.* (Brest, France), pp. 467–470, Sept. 2000.
- [17] O.M. Collins, "The subtleties and intricacies of building a constraint length 15 convolutional decoder," *IEEE Trans. Commun.*, vol. COM-40, pp. 1810–1819, Dec. 1992.
- [18] T. M. Cover and J. A. Thomas, *Elements of Information Theory*, John Wiley & Sons, New York, 1991.
- [19] P. J. Denning et al., *Machines, Language and Computation*, Prentice Hall, Englewood Cliffs, NJ, 1978.
- [20] European Telecommunications Standards Institute, "Universal mobile telecommunications system (UMTS): Multiplexing and channel coding (FDD)," *3GPP TS 125.212 version 3.4.0*, pp. 14–20, September 23, 2000.
- [21] R.M. Fano, "A heuristic discussion of probabilistic decoding," *IEEE Trans. Inform. Theory*, vol. IT-9, pp. 64–74, April 1963.
- [22] G. Feygin and P.G. Gulak, "Architectural tradeoffs for survivor sequence memory management in Viterbi decoders," *IEEE Trans. Commun.*, vol. COM-41, pp. 425–429, March 1993.
- [23] G.D. Forney, Jr. "The Viterbi algorithm," *Proc. IEEE*, vol. 61, pp. 268–278, 1973.
- [24] G.D. Forney, "Maximum-likelihood sequence estimation of digital sequences in the presence of intersymbol interference," *IEEE Trans. Inform. Theory*, vol. IT-18, pp. 363–378, May 1972.
- [25] G.J. Foschini, "A reduced state variant of maximum likelihood sequence detection attaining optimum performance for high signal-to-noise ratios," *IEEE Trans. Inform. Theory*, vol. IT-23, pp. 605–609, Sept. 1977.
- [26] R.G. Gallager, *Information Theory and Reliable Communication*, John Wiley & Sons, New-York, 1968.
- [27] J.M. Geist, "An empirical comparison of two sequential decoding algorithms," *IEEE Trans. Commun.*, vol. COM-19, pp. 415–419, Aug. 1971.
- [28] J.M. Geist, "Some properties of sequential decoding algorithms," *IEEE Trans. Inform. Theory*, vol. IT-19, pp. 519–526, July 1973.
- [29] W.J. Gross and P.G. Gulak, "Simplified map algorithm suitable for implementation of turbo decoders," *Electron. Lett.*, vol. 34, pp. 1577–1578, Aug. 6, 1998.
- [30] S. Lin and D.J. Costello, Jr., *Error Control Coding*, Prentice Hall, Englewood Cliffs, NJ, 1983.

- [31] D. Haccoun and M.J. Ferguson, "Generalized stack algorithms for decoding convolutional codes," *IEEE Trans. Inform. Theory*, vol. IT-21,, pp. 638–651, Nov. 1975.
- [32] F. Jelinek, "A fast sequential decoding algorithm using a stack," *IBM J. Res. Dev.*, vol. 13, pp. 675–685, Nov. 1969.
- [33] F. Jelinek and A.B. Anderson, "Instrumentable tree encoding of information sources," *IEEE Trans. Inform. Theory*, vol. IT-17, Jan. 1971.
- [34] L. Ma, "Suboptimal decoding strategies," MSEE thesis, University of Texas at San Antonio, May 1996.
- [35] R.J. McEliece, "On the BCJR trellis for linear block codes," *IEEE Trans. Inform. Theory*, vol. IT-42, no. 4, pp. 1072–1092, July 1996.
- [36] J.L. Massey, "Variable-length codes and the Fano metric," *IEEE Trans. Inform. Theory*, vol. IT-18, pp. 196–198, Jan. 1972.
- [37] J.L. Massey, "Coding and modulation in digital communications", *Proc. Int. Zürich Sem. Digital Commun.*, Zürich, Switzerland, pp. E2(1)–E2(4), March 1974.
- [38] H. Osthoff, J.B. Anderson, R. Johannesson, and C-F. Lin, "Systematic feed-forward convolutional encoders are better than other encoders with an M -algorithm decoder," *IEEE Trans. Inform. Theory*, vol. IT-44, no. 2, pp. 831–838, March 1998.
- [39] J.K. Omura, "On the Viterbi decoding algorithm," *IEEE Trans. Inform. Theory*, vol. IT-15, pp. 177–179, Jan. 1969.
- [40] G.J. Pottie and D.P. Taylor, "A comparison of reduced complexity decoding algorithms for trellis codes," *IEEE J. Select. Areas Commun.*, vol. SAC-7, no. 9, pp. 1369–1380, Dec. 1989.
- [41] J.G. Proakis, *Digital Communications*, McGraw-Hill, New York, 1989.
- [42] P. Robertson, P. Höher, and E. Villebrun, "Optimal and sub-optimal maximum a posteriori algorithms suitable for turbo decoding," *Eur. Trans. Telecommun.*, vol. 8, pp. 119–125, March/April 1997.
- [43] M. Rouanne and D.J. Costello, Jr., "An algorithm for computing the distance spectrum of trellis codes," *IEEE J. Select. Areas Commun.*, vol. SAC-7, no. 6, pp. 929–940, Aug. 1989.
- [44] C. Schlegel, "Evaluating distance spectra and performance bounds of trellis codes on channels with intersymbol interference," *IEEE Trans. Inform. Theory*, vol. IT-37, No. 3, pp. 627–634, May, 1991.
- [45] C.E. Shannon, "A mathematical theory of communications," *Bell Syst. Tech. J.*, vol. 27, pp. 379–423, July 1948.
- [46] S.J. Simmons, "A nonsorting VLSI structure for implementing the (M,L) algorithm," *IEEE J. Select. Areas Commun.*, vol. SAC-6, pp. 538–546, April, 1988.
- [47] S.J. Simmons and P. Wittke, "Low complexity decoders for constant envelope digital modulation," *Conf. Rec.*, GlobeCom, Miami, Florida, pp. E7.7.1–E7.7.5, Nov. 1982.
- [48] G. Strang, *Linear Algebra and Its Applications*, Harcourt Brace Jovanovich, San Diego, 1988.

- [49] G. Ungerboeck, "Channel coding with multilevel/phase signals," *IEEE Trans. Inform. Theory*, vol. IT-28, no. 1, pp. 55-67, Jan. 1982.
- [50] S. Verdú, "Minimum probability of error for asynchronous Gaussian multiple-access channels," *IEEE Trans. Inform. Theory*, vol. IT-32, pp. 85-96, Jan. 1986.
- [51] A.J. Viterbi, "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm," *IEEE Trans. Inform. Theory*, vol. IT-13, April 1969.
- [52] A.J. Viterbi and J.K. Omura, *Principles of Digital Communication and Coding*, McGraw-Hill, New York, 1979.
- [53] A.J. Viterbi and J.K. Omura, *Principles of Digital Communication and Coding*, McGraw-Hill, New York, 1979.
- [54] F.-Q. Wang and D.J. Costello, Jr., "Probabilistic construction of large constraint length trellis codes for sequential decoding," *IEEE Trans. Commun.*, vol. COM-43,, no. 9, pp. 2439-2448, Sept. 1995.
- [55] L. Wei, L.K. Rasmussen, and R. Wyrwas, "Near optimum tree-search detection schemes for bit-synchronous CDMA systems over Gaussian and two-path rayleigh fading channels," *IEEE Trans. Commun.*, vol. COM-45, no. 6, June 1997.
- [56] L. Wei and C. Schlegel, "Synchronous DS-SSMA with improved decorrelating decision-feedback multiuser detection," *IEEE Trans. Veh. Technol.*, vol. VT-43, no. 3, Aug. 1994.
- [57] J.M. Wozencraft and I.M. Jacobs, *Principles of Communication Engineering*, John Wiley & Sons, New York, 1965.
- [58] J.M. Wozencraft and B. Reiffen, *Sequential Decoding*, M.I.T. Press, Cambridge, MA, 1961.
- [59] J.M. Wozencraft and R.S. Kennedy, "modulation and demodulation for probabilistic coding", *IEEE Trans. Inform. Theory*, vol. IT-12, no. 3, pp. 291-297, July 1966.
- [60] J.M. Wozencraft and I.M. Jacobs, *Principles of Communication Engineering*, John Wiley & Sons, New York, 1965.
- [61] E. Zehavi and J.K. Wolf, "On the performance evaluation of trellis codes," *IEEE Trans. Inform. Theory*, vol. IT-33, no. 2, pp. 196-201, March 1987.
- [62] W. Zhang and C. Schlegel, "State reduction in the computation of d_{free} and the distance spectrum for a class of convolutional codes," Proceedings SICON/ICIE 93, Singapore, Sept. 1993.
- [63] W. Zhang, "Finite-state systems in mobile communications," Ph.D. dissertation, University of South Australia, June 1995.
- [64] K.Sh. Zigangirov, "Some sequential decoding procedures," *Prob. Pederachi Inform.*, vol. 2, pp. 13-25, 1966.
- [65] K.S. Zigangirov and V.D. Kolesnik, "List decoding of trellis codes," *Problems of Control and Information Theory*, no. 6, 1980.

Chapter 6

Low-Density Parity-Check Codes

6.1 Introduction

Low-density parity-check (LDPC) codes can rightfully take their stand next to turbo codes as the most powerful and most popular error control codes known. They offer a performance spectacularly close to theoretical limits when decoded using iterative soft-decision algorithms. Indeed, a rate $R = 1/2$ LDPC code with a blocklength of 10^7 bits comes to within 0.04 dB of the Shannon limit for the binary-input additive white Gaussian noise channel at a bit error rate of 10^{-6} [9]. It has a decoding threshold only 0.0045 dB away from the Shannon limit. Consequently, both LDPC and turbo codes have been called “capacity-approaching codes,” and we shall occasionally use this informal term. Both coding families are “practically” capable of solving Shannon’s channel coding challenge.

Low-Density Parity-Check (LDPC) codes and an iterative decoding algorithm for them were first introduced by Gallager more than fifty years ago [18, 19]. However, for several decades LDPC codes were largely forgotten, arguably because computers of the time could not simulate these codes with meaningful blocklengths at low error rates, and no theoretical results tying these codes to the Shannon limit could be established.

Following the discovery of turbo codes (Chapter 8), LDPC codes were rediscovered through the work of MacKay and Neal [33, 34], and quickly entered main stream interest. LDPC codes significantly differ from the more conventional trellis and “regular” block codes of the time. First, they are constructed in a random manner, well, in a controlled pseudo-random manner in most cases of interest, and second, they have a decoding algorithm whose complexity is linear in the block length of the code. This allows the decoding of large codes, which is essential in approaching the Shannon capacity. When combined with their spectacular performance, these properties makes LDPC codes a compelling class of error control codes, and their adoption in a multitude of communications applications.

In this chapter, we explore the properties and structure of low-density parity-check

codes. We begin with the fundamental construction of *regular* LDPC codes and their representation as bipartite graphs. The graphical representation of LDPC codes then leads us to the notion of *irregular* LDPC codes, which have been used by some authors to improve performance, and to the density evolution analysis technique (DE), which predicts the performance of these codes. LDPC codes and their decoders can be thought of as a prescription for selecting large random block codes, not unlike the codes suggested by Shannon's coding theorem. MacKay [33] showed, by applying random coding arguments to the random sparse generator matrices of LDPC codes, that such code ensembles can approach the Shannon limit exponentially fast in the codelength. And even at moderate codelengths, LDPCs provide impressive performance results, some of which are illustrated in Figure 6.1, and n denotes the codelength.

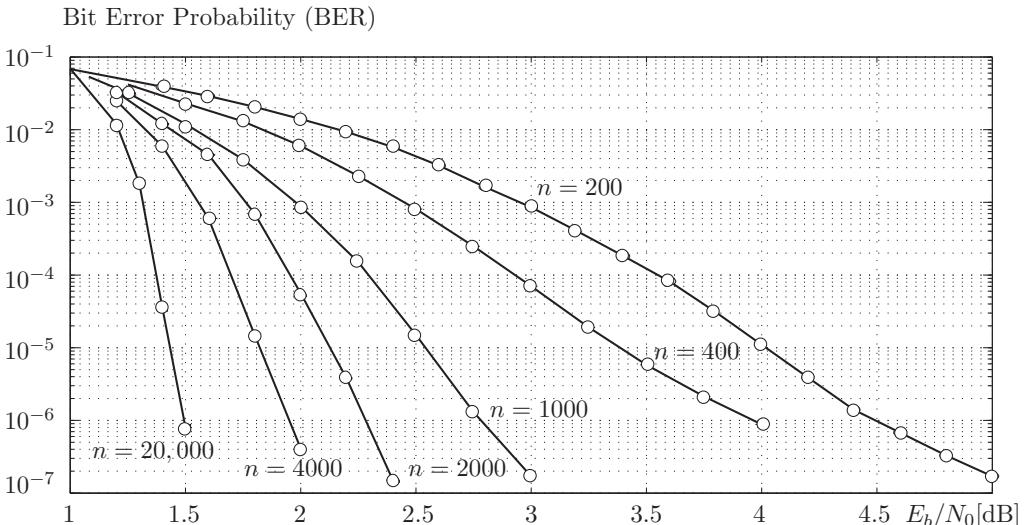


Figure 6.1: Performance results for regular (3,6) LDPC codes of rate $R = 1/2$ for varying block lengths. The capacity of the channel is at $E_b/N_0 = 0.18$ dB, and the threshold for these codes is at 1.1 dB.

Today, LDPC codes have been adopted in most recent industry standards, including wireless local-area networks (LANs), in IEEE 802.11n, WiMAX (IEEE 802.16e), digital video broadcasting (DVB-S2), 10GBase-T Ethernet over twisted-pair copper cabling (IEEE 802.3an), and the International Telecommunication Union standard for networking over power lines, phone lines, and coaxial cable (G.hn/G.9960).

6.2 LDPC Codes and Graphs

A linear block code \mathcal{C} of rate $R = k/n$ can be defined in terms of an $(n-k) \times n$ parity check matrix $\mathbf{H} = [\mathbf{h}_1, \mathbf{h}_2, \dots, \mathbf{h}_m]$, where each \mathbf{h}_j is a column vector of length $n-k$. Each entry h_{ij} of \mathbf{H} is an element of a finite field $GF(p)$. We will only consider binary codes in this chapter, so each entry is either a ‘0’ or a ‘1’ and all operations are modulo 2. The code \mathcal{C} is the set of all vectors \mathbf{x} that lie in the (right) nullspace of \mathbf{H} , i.e., $\mathbf{H}\mathbf{x} = \mathbf{0}$. Given a parity-check matrix \mathbf{H} , we can find a corresponding $k \times n$ generator matrix \mathbf{G} such that $\mathbf{G}\mathbf{H}^T = \mathbf{0}$. The generator matrix can be used as an encoder according to $\mathbf{x}^T = \mathbf{u}^T\mathbf{G}$.

In its simplest guise, an LDPC code is a linear block code with a parity-check matrix that is “sparse,” i.e., it has a small number of non-zero entries. The precise definition of sparseness is as follows: An $m \times n$ matrix has nm positions, and such a matrix is called sparse if the number of non-zero entries is of order $\mathcal{O}(n)$, where here $n > m$. That is, as we consider sequences of ever larger matrices, the number of non-zero entries grows at most as n . For very large such matrices, the majority of entries is therefore zero, and many efficient low complexity matrix manipulation algorithms are known for sparse matrices.

In [18, 19], Gallager proposed constructing LDPC codes by randomly placing 1’s and 0’s in a $m \times n$ parity-check matrix \mathbf{H} subject to the constraint that each row of \mathbf{H} had the same number d_v of 1’s and each column of \mathbf{H} had the same number d_c of 1’s. For example, the $m = 12 \times n = 24$ parity-check matrix shown in Figure 6.2 has $d_v = 6$ and $d_c = 3$ and defines an LDPC code of length $n = 24$. Codes of this form are referred to as *regular* (d_v, d_c) LDPC codes of length n . Note that such matrices are *sparse* for large n since the number of nonzero entries is $d_v n$. This is not really well visible in the rather small example of Figure 6.2.

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

Figure 6.2: Gallager-type low-density parity-check matrix \mathbf{H}_1 .

Recall that each row in a parity-check matrix represent a parity-check equation, involving some of the possible n variables. Therefore, in a (d_v, d_c) LDPC code, each information bit, that is, each variable, is involved in d_v parity-checks and each parity-check equation in turn involves d_c information bits. The fraction of 1's in the parity-check matrix of a regular LDPC code is $md_c/(mn) = d_c/n$, which approaches zero as the block length becomes large and leads to the sparseness discussed above.

As a first order of business, we wish to determine the rate of the code defined by \mathbf{H}_1 . Since the parity check matrix is randomly constructed, there is no guarantee that the rows are linearly independent and the matrix has full rank. Indeed, the rank of \mathbf{H}_1 is $13 < m = 15$ and this parity-check matrix actually defines a code with rate $R = 7/20$. In general, such randomly constructed parity-check matrices will not be full rank and $m \neq n - k$.

We could of course eliminate linearly dependent rows to find a $(n - k) \times n$ parity-check matrix, but the new matrix would no longer be regular. For LDPC codes with large n it is convenient to retain the original parity-check matrix even if it is not full rank. The maximum number of parity bits is then m , and $1 - m/n = 1 - d_v/d_c$ is the maximum rate, also called the *design rate* of the code.

Having defined a blueprint for a (d_v, d_c) regular LDPC code, we are now left to construct a particular instance of a code. To do this requires the choice of d_v , d_c , n , and k , which are constrained by the relationship that, for a regular code $md_c = nd_v$, i.e., the number of non-zero entries of \mathbf{H} must be the same whether calculated by rows or by columns. Furthermore, d_v must be less than d_c and the rate R is less than 1. Assuming that the block length n and the code rate are determined, we need to fix appropriate values for d_v and d_c . In [19], Gallager showed that the minimum distance of a “typical” regular LDPC code increases linearly with n , provided $d_v \geq 3$. Therefore, regular LDPC codes are constructed with d_v on the order of 3 or 4, subject to the above constraints. $d_v < 3$ is used in some codes, but this can cause difficulties with the iterative decoding algorithm as we will see later. For large block lengths, the random placement of 1's in \mathbf{H} such that each row has exactly d_c 1's and each column has exactly d_v 1's requires some effort, and systematic methods for doing this have been developed [33, 34]. Gallager himself used an approach based on permutation matrices, which we will also discuss in the sequel.

An important advance in the theory of LDPC codes is made possible by Tanner’s [45] method of using bipartite graphs to provide a graphical representation of the parity-check matrix. As a recognition of Tanner’s work, the bipartite graph of an LDPC code is often referred to as its *Tanner graph*. A bipartite graph is a graph in which the nodes may be partitioned into two subsets such that there are no edges connecting nodes within a subset. In the context of LDPC codes, the two subsets of nodes are the *variable* nodes and the *check* nodes. There is one variable node for each of the n bits in the code and there is one check node for each of the m rows of \mathbf{H} . An edge exists between the i th variable node and the j th check node if and only if the entry $h_{ij} = 1$ in \mathbf{H} . The bipartite graph

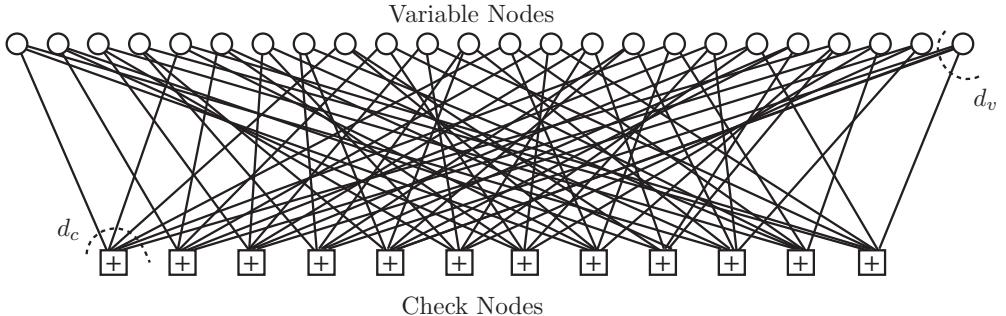


Figure 6.3: Bipartite graph for the $(3, 6)$ regular LDPC code with parity-check matrix \mathbf{H}_1 .

corresponding to \mathbf{H}_1 is shown in Figure 6.3. The number of edges incident upon a node is called the *degree* of the node. Thus, the bipartite graph of a (d_v, d_c) LDPC code contains n variable nodes of degree d_v and m check nodes of degree d_c .

It is clear that the parity-check matrix can be deduced from the bipartite graph and thus the bipartite graph can be used to define the code \mathcal{C} . We can therefore start talking about codes as defined by a set of variable nodes, a set of check nodes, and set of edges. Note that the pair (d_v, d_c) , together with the codelength n , specifies an *ensemble* of codes, rather than any particular code. This ensemble is denoted by $C^n(d_v, d_c)$. Once the degrees of the nodes are chosen, we are still free to choose which particular connections are made.

A *socket* refers to a point on a node to which an edge may be attached. For example, we say that a variable node has d_v sockets, meaning d_v edges may be attached to that node. There will be a total of nd_v sockets on variable nodes and md_c sockets on parity-check nodes. This is illustrated in Figure 6.4. Clearly the number of variable node sockets must be equal to the number of check node sockets, and a particular pattern of edge connections can be described as a permutation π from variable node sockets to check node sockets.

We have the following definition of an LDPC code as a permutation of socket numbers:

Definition 6.1 A regular LDPC code is completely defined by a permutation $\pi(i)$ of the natural numbers $1 \leq i \leq d_v n$. The index i refers to the socket number at the variable nodes, and $\pi(i)$ to the socket number at the check nodes to which socket i connects.

Selecting a random code from the ensemble $C^n(d_v, d_c)$ therefore amounts to randomly selecting a permutation of nd_v elements. Many permutations will result in a graph which

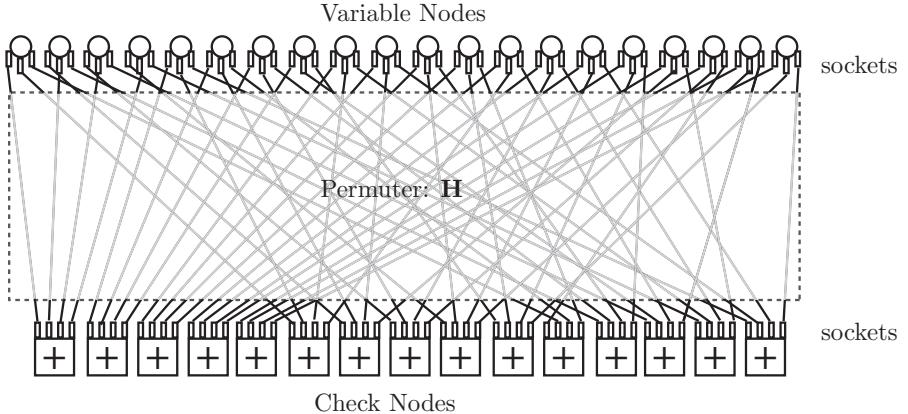


Figure 6.4: Illustration of the concept of a socket in a Tanner graph.

contains *parallel edges*, i.e., in which more than one edge join the same variable and parity-check nodes. Note that in the parity-check matrix, an even number of parallel edges will cancel. If they are deleted from the graph, then the degrees of some nodes will be changed and the code will cease to be a regular LDPC code. If they are not deleted, their presence renders the iterative decoding algorithms ineffective. We must therefore make the restriction that permutations leading to parallel edges are disallowed.

It was observed by Luby et al. [29] that allowing the degrees of the different nodes to differ can provide improved performance. They constructed $R = 1/2$ codes with a performance of up to 0.5 dB better than that of regular codes for moderate-length codes when operated on an additive white Gaussian noise channel. An *irregular* LDPC code cannot be defined in terms of the degree parameters d_v and d_c . We must instead use *degree distributions* to describe the variety of node degrees in the graph. A degree distribution $\gamma(x)$ is a polynomial in x ,

$$\gamma(x) = \sum_i \gamma_i x^{i-1}, \quad (6.1)$$

such that $\gamma(1) = 1$. That is, the coefficients γ_i can be thought of as probabilities. These coefficients are the fraction of *edges* in the graph which are connected to nodes of degree i . Note this means that γ_i/i is proportional to the fraction of *nodes* in the graph which have i edges—an important distinction—see below.

The codelength n and two degree distributions— $\lambda(x)$ and $\rho(x)$ for the variable and check nodes, respectively—are sufficient to define an *ensemble* $C^n(\lambda, \rho)$ of irregular LDPC codes, and an individual irregular code is specified by the following definition:

Definition 6.2 An irregular LDPC code is defined by a permutation $\pi(i)$ of the natural numbers $1 \leq i \leq N_e$, where $N_e = n / \sum_i \frac{\lambda_i}{i}$ is the total number of edges, and two degree distributions $\lambda(x)$ and $\rho(x)$ for the variable nodes and check nodes, respectively.

In the above definition we shall always assume that the nodes are ordered in descending degree values, which then completely specifies a given code.

The number of variable nodes in an irregular code can be computed as

$$n = N_e \sum_i \frac{\lambda_i}{i}, \quad (6.2)$$

since each node of degree i combines i edges, and N_e is the number of edges in the code. Likewise, the number of check nodes is given by

$$m = N_e \sum_i \frac{\rho_i}{i}. \quad (6.3)$$

The design rate of an irregular code is given by the dimension of the parity-check matrix, or the size of the bipartite graph as

$$R = \frac{n - m}{n} = 1 - \frac{m}{n} = 1 - \frac{\sum_i \frac{\rho_i}{i}}{\sum_i \frac{\lambda_i}{i}}. \quad (6.4)$$

Naturally, we want to rule out parallel edges also for irregular codes.

6.3 LDPC Decoding via Message Passing

One of the principal advantages of LDPC codes is that they can be decoded using an iterative decoding algorithm whose complexity grows (only) linearly with the block length of the code. Now that we have a graphical representation of LDPC codes, we can decode them using belief propagation decoding algorithms (see Chapter 6). Belief propagation is one instance of a broad class of message passing algorithms on graphs widely discussed in literature; see, for example, [26, 17, 1, 49]. The decoding algorithms for the binary erasure channel and the binary symmetric channel are discussed below in Sections 6.4.1 and 6.4.3, respectively, and the ubiquitous Gaussian channel is treated in Section 6.4.4 in detail.

All message passing algorithms must respect the following rule, which was introduced with turbo codes [6] as the extrinsic information principle:

Rule 1 (Extrinsic Information Principle)

A message sent from a node i along an edge e cannot depend on any message previously received on edge e .

The extrinsic information principle prevents that “local” incoming information is contained in the outgoing messages, which would introduce correlations in the statistical analysis presented later in this chapter. It also has the effect of slowing down convergence of the algorithms in general.

Before stating the algorithm, it is necessary to formulate the decoding problem. An LDPC code is constructed in terms of its parity check matrix \mathbf{H} , which is used for decoding. To encode an information sequence, let us call it \mathbf{u} , it is, in principle, necessary to derive a generator matrix \mathbf{G} , which has the property that $\mathbf{G}\mathbf{H}^T = \mathbf{0}$. Finding a suitable \mathbf{G} is greatly simplified if we first convert \mathbf{H} into the equivalent systematic parity check matrix $\mathbf{H}_S = [\mathbf{A}|\mathbf{I}_{n-k}]$. As is well known, the systematic generator matrix is then given by

$$\mathbf{G}_S = [\mathbf{I}_k|\mathbf{A}^T].$$

The information sequence is now encoded as $\mathbf{x}^T = \mathbf{u}^T \mathbf{G}_S$. It is worth noting that encoding by matrix multiplication following this procedure has complexity $\mathcal{O}(n^2)$, since \mathbf{G}_S is typically no longer sparse, and that therefore, in general, LDPC codes have linear decoding complexity, but apparently quadratic *encoding* complexity if we subscribe to this logic. Methods for reducing the encoding complexity are therefore of interest and will be discussed specifically Section 6.6 and throughout this chapter.

The codeword \mathbf{x} is transmitted over an additive white Gaussian noise (AWGN) channel using BPSK modulation resulting in the received sequence $\mathbf{r} = \sqrt{E_s}(2\mathbf{x} - 1) + \mathbf{n}$ (see Chapter 2), where E_s is the received energy of a single symbol (bit), and \mathbf{n} is a vector of n independent noise samples of variance $\sigma^2 = N_0/2$ each. An optimal decoder for the AWGN channel computes the posterior probabilities of the transmitted symbols and selects the symbol which has the maximum such probability. This is the *maximum a posteriori probability* (MAP) decoder, and for binary signals it simply computes the log-likelihood ratio

$$\lambda(x_r) = \log\left(\frac{P(x_n = 1|\mathbf{r})}{P(x_n = 0|\mathbf{r})}\right)$$

and makes a decision by comparing this LLR to the threshold zero. As shown in Appendix A, belief propagation on a graph, even with cycles, can closely approximate the performance of the MAP algorithm. We now state the decoding algorithm for LDPC codes on the AWGN channel as a belief propagation algorithm on the graph of the code.

The received signal r_i is associated with the variable node i . The algorithm is not directly working with r_i , but with its *log-likelihood ratio* (LLR), given by

$$\lambda_i = \log\left(\frac{P(x_i = 1|r_i)}{P(x_i = -1|r_i)}\right) = 4\frac{\sqrt{E_s}}{N_0}r_i, \quad (6.5)$$

which is a scaled version of the received signal itself.

The algorithm operates by passing messages along the edges of the bipartite graph of the code, starting with the “raw” bit LLRs λ_i , and combining these messages at the nodes by computational functions which generate new, outgoing messages. Specifically, a message from variable node i to check node j is represented by $\mu_{i \rightarrow j} \in \mathcal{R}$, and a message from check node j to variable node i is $\beta_{j \rightarrow i} \in \mathcal{R}$. Let $V_{j \setminus i}$ be the set of variable nodes which connect to check node j , excluding variable node i . Similarly, let $C_{i \setminus j}$ be the set of check nodes which connect to variable node i , excluding check node j . The decoding algorithm operates then as follows:

Algorithm for Decoding LDPC Codes on AWGN Channels:

Step 1: Initialize $\lambda_i = 4\frac{E_s}{N_0}r_i$ for each variable node.

Step 2: Variable nodes send $\mu_{i \rightarrow j} = \lambda_i$ to each check node $j \in C_i$.

Step 3: Check nodes connected to variable node i send

$$\beta_{j \rightarrow i} = 2 \tanh^{-1} \left(\prod_{l \in V_{i \setminus j}} \tanh \left(\frac{\lambda_l}{2} \right) \right). \quad (6.6)$$

Step 4: Variable nodes connected to check nodes j send

$$\mu_{i \rightarrow j} = \sum_{l \in C_{i \setminus j}} \beta_{l \rightarrow i}. \quad (6.7)$$

Step 5: When a fixed number of iterations have been completed or the estimated codeword \hat{x} satisfies the syndrome constraint $\mathbf{H}\hat{x} = \mathbf{0}$, stop. Otherwise return to Step 3.

Figure 6.5 shows simulation results for a regular LDPC code, an irregular LDPC code, and a turbo code of similar complexity on the AWGN channel [40]. This figure demonstrates the power of irregular LDPC codes in terms of performance. We will have more to say about irregularity in the next section, where the theoretical motivation and design methodology for irregular codes are discussed. The irregular node degree, however, can greatly increase the implementation complexity, especially if very high node degrees are used, and also presents a problem in the error floor region, discussed later in Chapter 7.

The reader may be tempted to conclude from Figure 6.5 that irregular LDPC codes are uniformly more powerful than other types of graph-based codes, but, in general, this is not strictly true. Numerous designs and complexity studies have failed to reveal any

inherent advantage in one structure over another. More about the actual and perceived superiority of one type of code over another will be said about in later sections.

Bit Error Probability (BER)

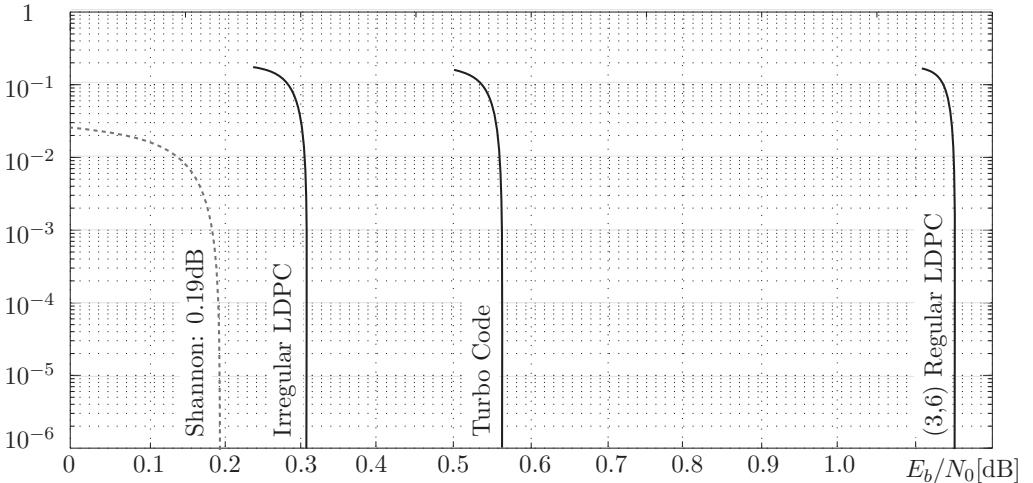


Figure 6.5: Simulation results for a regular (3,6) LDPC, an optimized irregular LDPC and a turbo code. All three codes are of rate $R = 1/2$ and have a block length of $n = 10^6$ [40].

The check node computation rule (6.6) is fairly complex. But for quantized messages it is possible to map LLRs to messages in such a way that the check node rule can be implemented with only combinational logics and an adder. Such decoders provide very good performance with only a few bits of precision. Finite-precision node processing and low-complexity implementations are discussed in Chapter 7. However, if we want to reduce complexity, one immediate possibility is to use the so-called min-sum approximation, in which we replace the check node processing rule (6.6) with

$$\beta_{j \rightarrow i} \approx \min_{l \in V_i \setminus j} (|\lambda_l|) \prod_{l \in V_i \setminus j} \text{sign}(\lambda_l). \quad (6.8)$$

Some performance loss on the order of 0.5 dB will result from this approximation, but this loss may be justified by the overall savings in decoder complexity.

6.4 Analysis Techniques

We now know how to define ensembles of LDPC codes and how to decode them via message passing, but how do we know which parameters to choose and which codes to pick for good performance? The authors of [40, 41] use an approach they term “density evolution” to compare the qualities of different ensembles of regular and irregular LDPC codes. Density evolution has its origin in the error probability evolution formulas of Gallager [18, 19] discussed in Section 6.4.3, and the modern extension to soft-output channels are, in essence, a sophisticated generalization of these early methods.

The main statement that density evolution can make about LDPC codes is the following: An LDPC code with a given degree distribution pair (λ, ρ) , operated on a channel with noise standard deviation σ has an associated *threshold* σ^* . The threshold is analogous to the Shannon capacity in that the error-probability for a randomly chosen $(\lambda(x), \rho(x))$ -code used on this channel can be made arbitrarily small for a growing size of the code if and only if $\sigma < \sigma^*$. One degree-distribution pair is said to be better than another if its threshold is closer to the channel’s capacity limit, i.e., if it can tolerate a higher noise standard deviation σ .

Threshold results for several ensembles on various channels are reported in [41]. The qualities of different ensembles of regular and irregular LDPC codes are studied and optimized via density evolution in [11, 40], where linear numerical optimization techniques are used to find optimal degree distributions.

One design methodology in designing good codes is to choose the ensemble with the best threshold, from which we select a code with the largest length which can be accommodated in a given implementation. As is typical in random coding arguments, it turns out that almost all codes in the ensemble perform equally well. Code design may therefore consist of randomly sampling a few codes from the ensemble and selecting the best among those. Actual code selection in practice, however, is strongly determined by the code’s performance in the *error floor region*, where density analysis is not applicable (see Chapter 7).

6.4.1 (Error) Probability Evolution for Binary Erasure Channels

We first illustrate the method of density evolution using the Binary Erasure Channel (BEC), shown in Figure 6.6. For regular code ensembles, it is possible to derive the exact solution for thresholds when a simple discrete message-passing algorithm is used for decoding [5]. The channel parameter for the BEC is the erasure probability ε .

Let $\mathcal{A} = \{-1, 0, +1\}$ denote the message alphabet, $r_i \in \mathcal{A}$ the received symbol at variable node i , and $d_i \in \mathcal{A}$ the decision at variable node i . A message from variable node i to check node j is represented by $\mu_{i \rightarrow j} \in \mathcal{A}$, and a message from check node j to variable node i is $\beta_{j \rightarrow i} \in \mathcal{A}$. The received signal r_i is corrupted by the BEC channel according

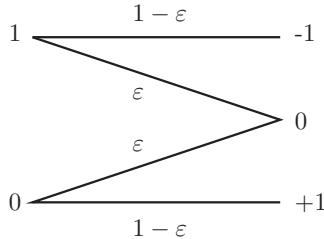


Figure 6.6: Binary erasure channel model.

to random transitions illustrated in Figure 6.6, where the labels on the edges indicate the probabilities of the edge being traversed, that is, there is a probability ϵ that a single symbol is erased. A decoding algorithm for the BEC now proceeds as follows [30]:

Algorithm for Decoding LDPC Codes on BEC Channels:

Step 1: Initialize $d_i = r_i$ for each variable node. If $r_i = 0$ then the received symbol i has been erased and variable i is said to be *unknown*.

Step 2: Variable nodes send $\mu_{i \rightarrow j} = d_i$ to each check node $j \in C_i$.

Step 3: Check nodes connected to variable node i send $\beta_{j \rightarrow i} = \prod_{l \in V_{j \setminus i}} \mu_{l \rightarrow j}$ to i . That is, if all incoming messages are different from zero, the check node sends back to i the value that makes the check consistent, otherwise it sends back a zero for “unknown”.

Step 4: If the variable i is unknown, and at least one $\beta_{j \rightarrow i} \neq 0$, the node sets $d_i = \beta_{j \rightarrow i}$ and declares variable i to be *known*. (Known variables will never have to be changed anymore.)

Step 5: When all variables are known, stop. Otherwise go to Step 2.

Under this algorithm, a check node can only output a non-zero message on an edge if all incoming messages on other edges are non-zero. If there are not enough known variables when decoding begins, then the algorithm will fail. We will have more to say about the failure of LDPC codes on BEC channels later.

Density evolution analysis for the BEC channel amounts to studying the propagation of erasure probabilities. A message is called “incorrect” if it is an erasure and has a numerical value of zero. We are now interested in determining the density of erasures among variable nodes as the number of iterations $l \rightarrow \infty$. First, we simplify the message alphabet to $\mathcal{A}' = \{0, 1\}$, where a zero denotes an erasure and a one means the variable is known. As given by the decoding algorithm, the output of a check node is known if all of its incoming messages are known, and the output of a variable node is known if one or more of its incoming (check-node) messages are known.

Let $p_v^{(l)}$ be the probability of erasures at the variable nodes at iteration l , and let $p_u^{(l)}$ be the probability, or “density” of erasures at the check nodes at iteration l . Then for a regular code, in which the node degrees are all equal, the following recursive relation can be established:

$$\begin{aligned} p_u^{(l-1)} &= 1 - [1 - p_v^{(l-1)}]^{d_c-1}, \\ p_v^{(l)} &= p_0 [p_u^{(l-1)}]^{d_v-1}, \end{aligned} \quad (6.9)$$

where $p_0 = \varepsilon$ is the initial probability of an erasure. Equation (6.9) contains an important assumption, which is that the probabilities of node u or v do not depend on themselves, that is, there are no cycles in the local subgraph which could introduce such dependencies.

This condition is best illustrated by Figure 6.7. In this local subtree, the nodes at level l do not appear in any of the lower levels. If this is the case, all probabilities which are migrated upward from below can be assumed to be independent of those at higher levels. This is the case, at least approximately, for large LDPC codes whose loops in the code’s graph are large. Hence the efforts to design LDPC codes with large loops, since otherwise the dependencies introduced in the message passing will, in general, degrade performance.

Combining the parts in (6.9), we obtain the equation of a one-dimensional iterated map for the probability that a variable node remains uncorrected as

$$p_v^{(l)} = F(p_v^{(l-1)}, p_0) = p_0 \left(1 - [1 - p_v^{(l-1)}]^{d_c-1} \right)^{d_v-1}. \quad (6.10)$$

Figure 6.8 shows the evolution of the erasure probability for the ensemble of (3,6) LDPC codes for a few input erasure values $p_0 = \varepsilon$. The staircase trajectory moving left shows the values $p_v^{(l)}$ for $\varepsilon = 0.4$ according to Equation (6.9). The solid curves express the function $p_v^{(l)} = f(p_v^{(l-1)})$, computed by combining the formulas in (6.9), and illustrate the evolution of $p_v^{(l)}$ over iterations. If $f(x, p_0) < x, \forall x \leq \epsilon$ then (6.10) will converge to $p_v^{(\infty)} = 0$. Evidently, if ε is larger than a certain value, termed the *decoding threshold*, there exists a non-zero fix-point $F(\varepsilon, x) = x$ of (6.10), and the decoded erasure probabilities no longer converge to zero. Examples of the convergence function $x - F(x, p_0)$ are shown in Figure 10.9 in Chapter 10, where we discuss spatial coupling of sparse graphs.

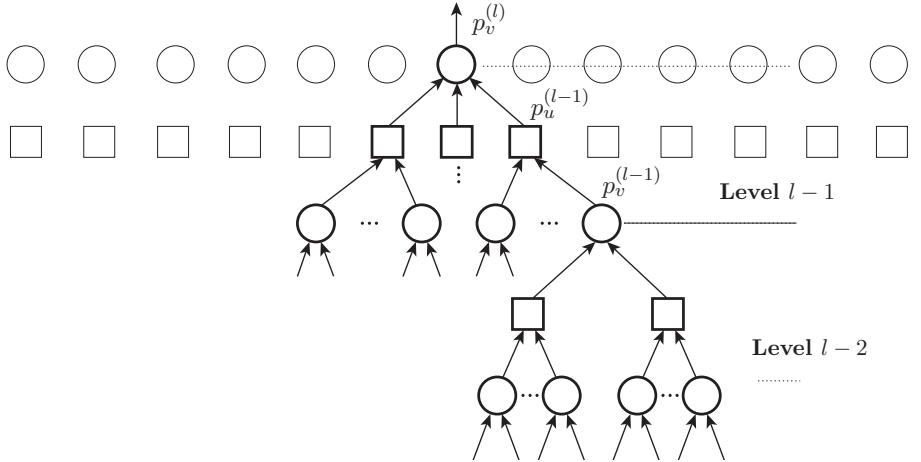


Figure 6.7: Local subtree structure of an LDPC code that is generated as the connections of a given variable node are unwrapped, allowing computation of messages through iterations.

For an irregular code with degree-distributions $\lambda(x)$ and $\rho(x)$, the degree of a particular node is a random variable. When this is taken into account, the relation (6.9) becomes [29]

$$\begin{aligned} p_u^{(l-1)} &= 1 - \sum_{i=1}^{d_c} \rho_i \left[1 - p_v^{(l-1)} \right]^{i-1} = 1 - \rho \left(1 - p_v^{(l-1)} \right), \\ p_v^{(l)} &= p_0 \sum_{j=1}^{d_v} \lambda_j \left[p_u^{(l-1)} \right]^{j-1} = p_0 \lambda \left(p_u^{(l-1)} \right). \end{aligned} \quad (6.11)$$

(6.11) can be derived by the following simple argument: The probability that a check node's output is non-zero is given by

$$\begin{aligned} P(\beta^{(l-1)} \neq 0) &= \sum_{i=1}^{d_c} P(i-1 \text{ inputs known} | \# \text{inputs} = i) P(\#\text{node degree} = i) \\ &= \sum_{i=1}^{d_c} P(\mu^{(l-1)} \neq 0)^{i-1} P(\#\text{node degree} = i), \end{aligned} \quad (6.12)$$

where we invoked the assumption that all messages are independent and identically distributed. Because $p_u^{(l)}$ and $p_u^{(l-1)}$ represent the probability that messages are *unknown*, the first equation of (6.11) is an immediate consequence.

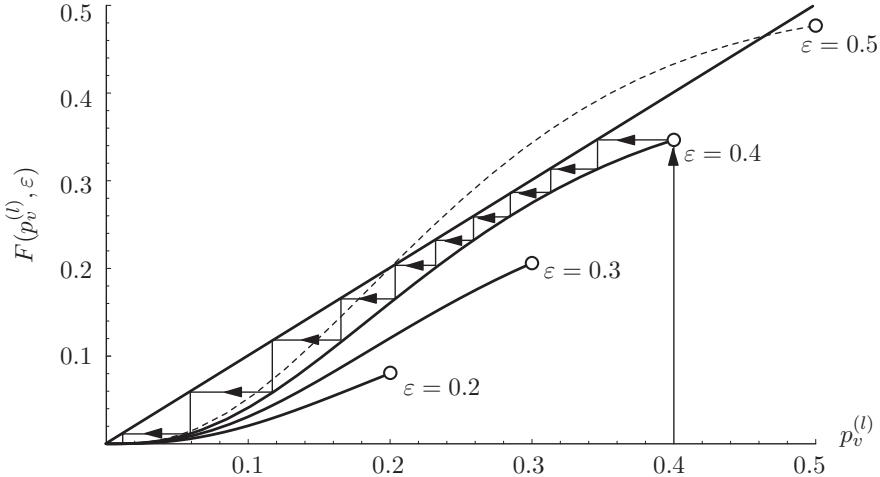


Figure 6.8: Graphical illustration of the evolution of the erasure probability $p_v^{(l)}$.

The same reasoning is used to obtain the second line of (6.11):

$$\begin{aligned} P\left(\mu^{(l-1)} = 0\right) &= \sum_{j=1}^{d_v} P(j-1 \text{ inputs unknown} \mid \text{node deg} = j) P(\text{node deg} = j) \\ &= \sum_{j=1}^{d_v} P\left(\beta^{(l-1)} = 0\right)^{j-1} P(\text{node deg} = j). \end{aligned} \quad (6.13)$$

Finally, we assume that the probability that an edge incident to a check node has degree i is equal to ρ_i . Similarly, the probability for an edge incident to a variable node to have degree j is equal to λ_j . This gives rise to the polynomial form used in the right-hand side of (6.11).

We proceed to determine the threshold for a code ensemble with parameters (λ, ρ) on the BEC, by determining the conditions under which the density of erasures converges to zero as $l \rightarrow \infty$. Based on (6.11) and appealing to Figure 6.8, this threshold happens where the probability evolution function touches the straight line at a non-zero point, i.e., where $F(x, \varepsilon) = x$. This threshold can be expressed as the supremum [5]

$$\begin{aligned} \varepsilon^* &= \sup \{\varepsilon : F(x, \varepsilon) < x, \forall x \leq \varepsilon\}, \\ \text{where } F(x, \varepsilon) &= \varepsilon \lambda (1 - \rho(1 - x)). \end{aligned} \quad (6.14)$$

Not that ε^* is simply the largest erasure probability such that for all $\varepsilon < \varepsilon^*$, the only solution to $f(\varepsilon, x) = x$ is $x = 0$. We say colloquially that "the code converges" in this case. We can express this threshold as a minimization operation as below. Formally, error-free decoding for a large code is possible if and only if

$$x = \varepsilon \lambda [1 - \rho(1 - x)] \quad (6.15)$$

has no positive solutions for $0 < x \leq \varepsilon$. Since any fixed point solution x corresponds to a unique non-zero value of ε , we study ε as a function of x :

$$\varepsilon(x) = \frac{x}{\lambda[1 - \rho(1 - x)]}, \quad (6.16)$$

which allows us to rewrite the threshold definition as

$$\varepsilon^* = \min \{ \varepsilon(x) : \varepsilon(x) \geq x \}. \quad (6.17)$$

Note that (6.17) and (6.14) are complementary, in that we approach the boundary from above in (6.17), and therefore the minimum operator can be used, that is, the condition describes values of ε for which (6.11) does *not converge*.

For a regular ensemble (d_v, d_c) , this minimum can be found analytically. Substitute $x = 1 - y$ and consider the derivative of $\varepsilon(1 - y)$, given by

$$\begin{aligned} \frac{d}{dy} \left\{ \frac{1-y}{[1-y^{d_c-1}]^{d_v-1}} \right\} = \\ [(d_c-1)(d_v-1)-1] y^{d_c-1} - (d_c-1)(d_v-1) y^{d_c-2} + 1 = 0, \end{aligned} \quad (6.18)$$

which we set to zero. By Descarte's "Rule of Signs", the number of positive real roots for a polynomial cannot exceed the number of sign changes in its coefficients. The polynomial in (6.18) therefore has no more than two roots. One of those roots is $y = 1$, the original fixpoint at $x = 0$. Dividing (6.18) by $(y - 1)$ yields

$$[(d_v - 1)(d_c - 1) - 1] y^{d_c-2} - \sum_{i=0}^{d_c-3} y^i = 0, \quad (6.19)$$

and letting s be the (only) positive real root of (6.19) the threshold for the regular LDPC code ensemble is given by

$$\varepsilon^* = \frac{1-s}{(1-s^{d_c-1})^{d_v-1}}. \quad (6.20)$$

As an example, the threshold for the $(3, 6)$ regular ensemble is $\varepsilon^* = 0.4294$ (compare Figure 6.8). This code has nominal rate $R = 0.5$. The capacity limit for the BEC at this rate is the erasure rate $\varepsilon_C = 1 - R = 0.5$.

6.4.2 Error Mechanism of LDPCs on BECs

The analysis in the preceding section assumed that the code graph could be unwrapped into a tree of depth at least equal to the number of iterations we are considering. In this sense the analysis is asymptotic, since this condition typically requires $n \rightarrow \infty$ as the number of iterations grows.

Before discussing other channels, we turn our attention to errors in finite-sized code ensembles. The knowledge gained in this will be used later in discussing the phenomenon of the error floor (discussed in Chapter 7), which is a characteristic of both LDPC and turbo codes.

Large LDPC codes are very effective at correcting erasures capable of handling erasure probabilities quite close to the maximum imposed by the Shannon capacity of the BEC channel. However, in a finite-sized code, there are specific erasure patterns that cannot be corrected. These patterns are related to what are known as *stopping sets*, [14] defined as follows:

Definition 6.3 A stopping set S is a set of variable nodes, all of whose neighboring check nodes are connected to S at least twice.

Figure 6.9 illustrates a stopping set in our example LDPC code from Figure 6.3

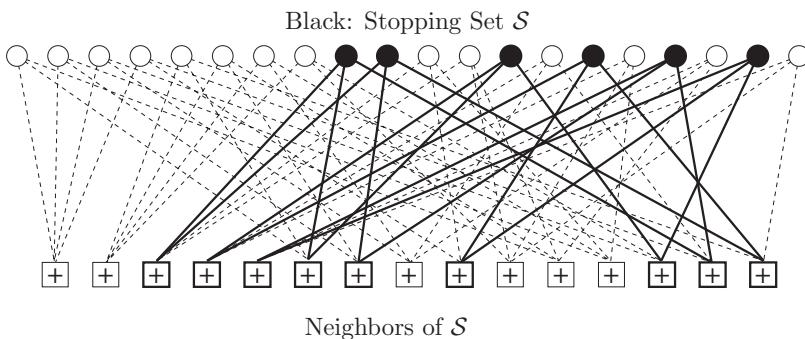


Figure 6.9: Stopping set of size six nodes in the example LDPC code.

Note that the union of two stopping sets is again a stopping set, which is straightforward to see. Therefore, any arbitrary set A of variable nodes contains a unique *maximal stopping set*, since if more than one set is contained in A , their union, which is larger, is also contained in A .

If a stopping set S is erased by the channel process, the decoding algorithm from Section 6.4.1 will fail, since in Step 3 of the algorithm, no parity-check node which is a neighbor of S can ever send out a corrected message, since all neighbors of S see at least two erasures from S , and therefore all their outputs will be erased. The nodes in S , in turn, receive messages only from those neighbors, and hence keep receiving erasures irrespective of the number of iterations performed. From this observation then, we can deduce the following lemma:

Lemma 2 *Erasure decoding will terminate at the unique maximal stopping set contained in the erased set of variables.*

While there exists a statistical analysis of stopping sets for LDPC code ensembles [14], we will mainly use the concept of the stopping set in our discussion of the error floor on the additive white Gaussian noise channel. For specific codes, the stopping sets will have to be found via combinatorial search techniques, which then, however, completely determine the code's error performance.

6.4.3 Binary Symmetric Channels and the Gallager Algorithms

The BEC is admittedly a somewhat artificial channel model, mostly popular with information theorists who are concerned with finding closed-form solutions and structural relationships which are harder to find for other channel models. A more realistic channel model is that of the binary symmetric channel (BSC) shown in Figure 6.10, which has two inputs $\{0, 1\}$ and two outputs $\{-1, 1\}$. A transmission is successful if the input equals the output, which happens with probability $1 - \varepsilon$. Conversely, an error is the inversion of a transmitted bit and happens with probability ε . This BSC has a capacity of $1 - h(\varepsilon)$, where $h(\varepsilon)$ is the binary entropy function [13].

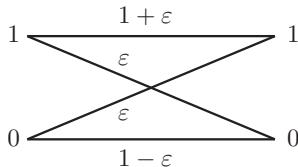


Figure 6.10: Binary symmetric channel model. This model includes the basic BPSK modulation step $1 \rightarrow 1, 0 \rightarrow -1$.

The following analysis was presented by Gallager [18, 19], but disappeared in obscurity, like LDPC codes themselves, until their rediscovery. Gallager's basic algorithm, called Gallager A, operates as follows:

Gallager's LDPC Decoding Algorithm A for BSC Channels:

Step 1: Initialize $d_i = r_i$ for each variable node.

Step 2: Variable nodes send $\mu_{i \rightarrow j} = d_i$ to each check node $j \in C_i$.

Step 3: Check nodes connected to variable node i send $\beta_{j \rightarrow i} = \prod_{l \in V_{j \setminus i}} \mu_{l \rightarrow j}$ to i . That is, the check node sends back to i the value that would make the parity check consistent.

Step 4: At the variable node i if $\lceil d_v/2 \rceil$ or more of the incoming parity check messages $\beta_{j \rightarrow i}$, disagree with d_i , change the value of variable i to its opposite value, i.e., $d_i = d_i \times (-1)$.

Step 5: Stop when no more variables are changing, or after a fixed number of iterations have been executed. Otherwise go to Step 2.

After specifying the algorithm, we can apply Gallager's probability evolution analysis to study the convergence properties of the code. Assume that the probability of error at the variable nodes is $p_v^{(l)}$, where $p_v^{(0)} = \varepsilon$. A check node will signal a correct check back if and only if an *even* number of errors occur in its checked symbols. This happens with probability¹

$$1 - p_v^{(l)} = \frac{1 + (1 - 2p_v^{(l)})^{d_c-1}}{2}. \quad (6.21)$$

In turn, at the variable nodes, an error will be corrected only if $b = \lceil d_v/2 \rceil$ or more parity checks are unsatisfied, which has probability

$$p_v^{(l)} \sum_{t=b}^{d_v-1} \binom{d_v-1}{t} \left(\frac{1 + (1 - 2p_v^{(l)})^{d_c-1}}{2} \right)^t \left(\frac{1 - (1 - 2p_v^{(l)})^{d_c-1}}{2} \right)^{d_v-1-t}, \quad (6.22)$$

where the two power terms simply express the probability of t correct parity checks and $d_v - 1 - t$ incorrect checks, respectively.

¹Equation (6.21) can be derived by considering the binomial expansions $(1 + p)^{d_c-1}$ and $(1 - p)^{d_c-1}$ and adding them.

The probability of error in a variable node at the next iteration is now given by

$$\begin{aligned} p_v^{(l+1)} = & p_v^{(l)} - p_v^{(l)} \sum_{t=b}^{d_v-1} \binom{d_v-1}{t} \left(\frac{1+(1-2p_v^{(l)})^{d_c-1}}{2} \right)^t \left(\frac{1-(1-2p_v^{(l)})^{d_c-1}}{2} \right)^{d_v-1-t} \\ & + (1-p_v^{(l)}) \sum_{t=b}^{d_v-1} \binom{d_v-1}{t} \left(\frac{1-(1-2p_v^{(l)})^{d_c-1}}{2} \right)^t \left(\frac{1+(1-2p_v^{(l)})^{d_c-1}}{2} \right)^{d_v-1-t}, \end{aligned} \quad (6.23)$$

where the first term is the probability that an erroneous variable node setting is corrected, and the second term is the probability that a correct setting is corrupted by faulty parity-check information.

The error probability evolution equation (6.23) is illustrated in Figure 6.11 for the ensemble of (3,6) LDPC codes. We note that it looks similar to the erasure probability evolution for the BEC channel, and the principle behavior of the decoder is identical.

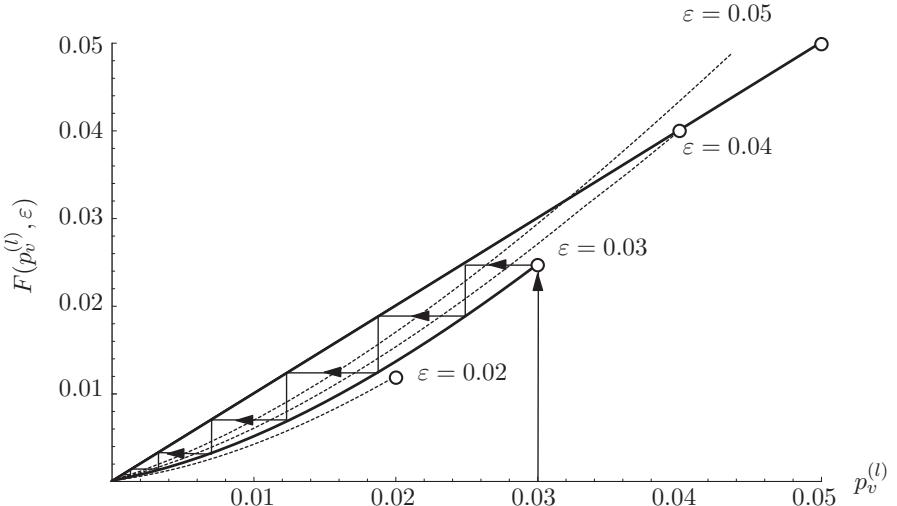


Figure 6.11: Graphical illustration of the evolution of the variable error probability $p_v^{(l)}$.

From Figure 6.11, or numerical computations, it is evident that the threshold value for a large (3,6) LDPC code on a BSC is at or near $\varepsilon^* \approx 0.04$, while the channel capacity threshold erasure rate for rate $R = 0.5$ equals $\varepsilon = 0.11$, derived from the BSC capacity formula, given by $C_{\text{BSC}} = 1 - h(R)$. It appears that it would be quite a bit harder to approach capacity on the BSC with LDPC codes.

Gallager [18, 19] has fine-tuned his algorithms by allowing a maximization over the “vote” parameter b in (6.23). This improves the threshold for the rate $R = 0.5$ LDPC codes to $\varepsilon^* = 0.052$ using the (4,8) LDPC code ensemble.

Irregular LDPC codes have become popular because they can improve on the performance of regular codes. In fact, adding irregularity to the node degrees has been one of the major constructional contributions to these codes since their inception in the early 1960s. In an irregular code, the “vote” parameter depends on the degree j of the variable node, i.e., if $b_j = \lceil j/2 \rceil$ incoming check node messages disagree with the variable node, it is changed to the new value in Step 4 in the decoding algorithm. A check node will signal a correct check back with probability

$$1 - p_v^{(l)} = \frac{1 + \rho \left(1 - 2p_v^{(l)}\right)}{2}, \quad (6.24)$$

which is a straightforward extension of (6.21) treating the node degrees as random according to $\rho(x)$.

Analogously, the probability that an erroneous variable is corrected is extended to

$$p_v^{(l)} \sum_{j=1}^{d_v} \lambda_j \sum_{t=b_j}^{j-1} \binom{j-1}{t} \underbrace{\left(\frac{1 + \rho \left(1 - 2p_v^{(l)}\right)}{2} \right)^t \left(\frac{1 - \rho \left(1 - 2p_v^{(l)}\right)}{2} \right)^{j-1-t}}_{g(t,j)}, \quad (6.25)$$

and the new iterated variable node error probability is

$$p_v^{(l+1)} = p_v^{(l)} - \sum_{j=1}^{d_v} \lambda_j \left[\sum_{t=b_j}^{j-1} \binom{j-1}{t} \left(p_v^{(l)} g(t,j) + (1 - p_v^{(l)}) g(j,t) \right) \right]. \quad (6.26)$$

The best “vote” parameter can be found to satisfy [29]

$$\frac{1 - \varepsilon}{\varepsilon} \leq \left[\frac{1 + \rho \left(1 - 2p_v^{(l)}\right)}{1 - \rho \left(1 - 2p_v^{(l)}\right)} \right]^{2b_j - j + 1}, \quad (6.27)$$

where we note that $2b_j - j + 1 = b_j - (j - 1 - b_j) = \Delta b$ is the difference between the number of check nodes that disagree with the variable node and that of those that agree. Equation (6.27) therefore states that only this difference matters and needs to be optimized.

Designing irregular graphs and numerically analyzing their thresholds reveals that better codes can be found by allowing the node degrees to vary. In [29], the authors present the improved irregular codes shown in Table 6.1.

d_v	Code 1	Code 2	Code 3	Code 4
λ_3			.123397	.093368
λ_4			.555093	.346966
λ_5	.496041	.284961		
λ_6	.173862	.124061		
λ_{16}			.321510	
λ_{21}	.077225			.159355
λ_{23}	.252871			.400312
λ_{27}		.068844		
λ_{29}		.109202		
λ_{30}		.119796		
λ_{100}		.293135		
ρ_{10}			1	
ρ_{14}	1			1
ρ_{22}		1		
ε^*	.0505	.0533	.0578	.0627

Table 6.1: Some irregular LDPC codes which outperform regular Gallager codes on the BSC at rate $R = 0.5$. The capacity threshold is at 0.111.

6.4.4 The AWGN Channel

The additive white Gaussian noise (AWGN) channel discussed in Chapter 2 is probably the most important representative of practical channels. In this section, we apply the method of density evolution to the AWGN channel. An “exact” solution for its density evolution involves several numerical steps and is quite involved. Furthermore, these give little insight into the concept of the process [41]. A close approximation can be found with less effort if messages in the decoder are assumed to have a Gaussian distribution. This is, in general not exact, but the Gaussian approximation greatly reduces the complexity and produces results very close to the exact solutions [11].

As in the rest of the chapter, we assume that binary transmission is used with unit-energy symbols from the alphabet $\{-1, +1\}$. We can also assume, without loss of generality, that the transmitted message consists only of $+1$ s corresponding to the all-one codeword, since the LDPC code is linear, and a digital “0” is mapped into $+1$ before transmission. A correct message bit in the decoded codeword is therefore one with a positive sign, and an incorrect message is one with a negative sign.

The probability density function of log-likelihood messages at the output of a matched-filter receiver is calculated from the AWGN channel equation $y = \sqrt{E_s}x + n$, where $n \sim \mathcal{N}(0, N_0/2)$ is a zero-mean Gaussian random variable with variance $\sigma^2 = N_0/2$. We

consider the log-likelihood ratio of y , given by (6.5) as

$$\lambda = \log \frac{p(y|1)}{p(y|-1)} = 4 \frac{\sqrt{E_s}}{N_0} y, \quad (6.5)$$

whose own PDF is Gaussian with mean $m_\lambda = 4E_s/N_0$ and variance $16E_s/N_0 = 2m_\lambda$, i.e.,

$$f(\lambda) = \sqrt{\frac{N_0}{16\pi E_s}} \exp\left(-\frac{N_0}{16E_s}\left(y - \frac{4E_s}{N_0}\right)^2\right), \quad (6.28)$$

as can easily be calculated from the input signal y . This log-likelihood density fulfills what is known as the *symmetry conditions*

$$f(-\lambda) = f(\lambda)e^{-\lambda}. \quad (6.29)$$

Condition (6.29), also called the *consistency condition*, implies that the density f_λ is completely determined by its mean. To simplify the analysis we now make the following

Assumption 1 *The log-likelihood messages exchanged between variable and check nodes in a large LDPC code graph have a consistent Gaussian distribution.*

This consistent Gaussian assumption allows us to consider the evolution of a single parameter, namely the mean of the distribution. Its validity has been quite well established through extensive simulations, especially at later iterations where sufficient mixing has occurred.

Since the output message is the sum of incoming messages (6.7), and noting that messages are assumed to be independent due to the local tree structure, we obtain for the mean of the variable node LLRs under probability propagation in the log domain

$$m_v^{(l)} = m_v^{(0)} + (d_v - 1)m_u^{(l-1)}, \quad (6.30)$$

where $m_v^{(l)}$ is the mean of the outgoing message from the variable node at iteration l , $m_u^{(l-1)}$ is the mean of the outgoing message from a check node at iteration $(l-1)$, and $m_v^{(0)}$ is equal to the mean of $f(\lambda)$ in (6.28), i.e., $m_v^{(0)} = 4E_s/N_0$.

The density evolution through a check node according to (6.6) is more complicated. To find the evolution of the mean, we assume the messages are independent and identically distributed (i.i.d.) and, after moving the $\tanh(\cdot)^{-1}$ to the left side, we take expectations on both sides to obtain

$$E\left[\tanh\left(\frac{U}{2}\right)\right] = E\left[\tanh\left(\frac{V}{2}\right)\right]^{d_c-1}. \quad (6.31)$$

To simplify our analysis, we define a function ϕ as

$$\begin{aligned}\phi(m_u) &= 1 - E\left[\tanh\left(\frac{U}{2}\right)\right] \\ &= 1 - \frac{1}{4\pi m_u} \int_{\mathcal{R}} \tanh\left(\frac{u}{2}\right) \exp\left[-\frac{1}{4m_u}(u - m_u)^2\right] du.\end{aligned}\quad (6.32)$$

where $m_u > 0$. The motivation for defining $\phi(\cdot)$ is primarily computational, however

$$\tilde{u} = E\left[\tanh\left(\frac{U}{2}\right)\right] = \Pr(U = 1) - \Pr(U = -1) \quad (6.33)$$

can be interpreted as a *soft bit* estimate of u , since it is the expectation $E[u|U]$, and therefore $\phi(m_u)$ is the average estimation error as per (6.32). Note that the soft bit value is always smaller or equal to unity, and this error is therefore non-negative.

The bulk of the computational difficulty occurs in the integral of (6.32), and present convenient numerical approximations to $\phi(\cdot)$ which greatly speed up computations with no significant effects on the accuracy of the results.

The mean of the check node's output message in a regular LDPC code can now be calculated from

$$\begin{aligned}\tilde{u}^{(l-1)} &= \left(\tilde{v}^{(l-1)}\right)^{d_c-1} \Rightarrow \\ \phi(m_u^{(l-1)}) &= \left(1 - \left[1 - \phi(m_v^{(l-1)})\right]^{d_c-1}\right).\end{aligned}\quad (6.34)$$

The results (6.30) and (6.34) are sufficient to compute thresholds for regular LDPC code ensembles. A code converges if $m^{(l)} \rightarrow \infty$ as $l \rightarrow \infty$, or, equivalently, if the soft bit error $\phi(m_u) \rightarrow 0$.

If we define the estimation error at iteration l as $x^{(l)} = \phi(m_v^{(l)})$, then (6.34) and (6.30) together define an iteration equation for that estimation error, given by

$$x^{(l)} = \phi\left(\frac{4E_s}{N_0} + (d_v - 1)\phi^{-1}\left(1 - \left(1 - x^{(l-1)}\right)^{d_c-1}\right)\right), \quad (6.35)$$

Succinctly, the convergence behavior of regular LPDC codes under iterative message-passing decoding is governed by the single-parameter convergence equation

$$x = \phi\left(\frac{4E_s}{N_0} + (d_v - 1)\phi^{-1}\left(1 - (1 - x)\right)^{d_c-1}\right) = F\left(x, \frac{E_s}{N_0}\right). \quad (6.36)$$

These convergence equations will play an important role again in Chapter 10 on spatial coupling. The argument x is the average estimation error on a message line leaving a

variable node and is therefore a measure of the reliability of the LLRs in the decoding process. $x \rightarrow 0$ signifies the sign of the code's LLR all converge to the correct transmitted binary bit with probability one, that is, almost surely.

The exact computation of $\phi(\cdot)$ and $\phi^{-1}(\cdot)$ is computationally expensive and slow. The computation speed can be greatly improved using a few approximations. For small m , the approximation

$$\phi(m) \approx e^{\alpha m^\gamma + \beta}, \quad (6.37)$$

with $\alpha = -.4527$, $\beta = .0218$, and $\gamma = .86$, was proposed in [11]. For larger values of m , the following upper and lower bounds become tight, so that their average can be used as a good approximation for ϕ :

$$\sqrt{\frac{\pi}{m}} e^{-\frac{m}{4}} \left(1 - \frac{3}{m}\right) < \phi(m) < \sqrt{\frac{\pi}{m}} e^{-\frac{m}{4}} \left(1 - \frac{1}{7m}\right). \quad (6.38)$$

The figure below shows a numerical comparison between the approximation

$$\phi(m) \approx \begin{cases} \exp(-0.4527m^{0.86} + 0.0218) & \text{for } m < 19.89, \\ \sqrt{\frac{\pi}{m}} \exp(-\frac{m}{4}) (1 - \frac{11}{7m}) & \text{for } m \geq 19.89 \end{cases} \quad (6.39)$$

and (6.32). The difference is also plotted separately.

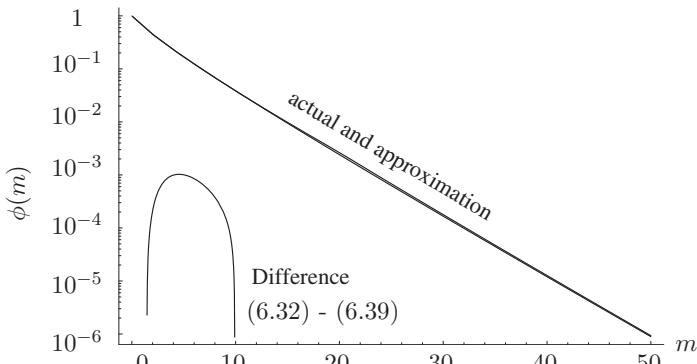


Figure 6.12: Exact function $\phi(m)$ and its approximation (6.39).

The density evolution for irregular ensembles is found in analogy with the derivation of (6.11) and the treatise above, but requires some modifications. We need to treat node degrees as random variables. The densities of the messages entering a check node are now Gaussian mixtures, with (6.30) and (6.34) as partial solutions, due to the fact that different degree variable nodes generate these messages.

Specifically, the mean value of the messages leaving a degree- i variable node is

$$m_{v,i}^{(l-1)} = (i-1)m_u^{(l-1)} + m_v^{(0)}. \quad (6.40)$$

In turn, the check node output message for a node of degree j obeys

$$E\left[\tanh\left(\frac{U}{2}\right)\right] = \prod_{i=1}^{j-1} E\left[\tanh\left(\frac{V_i}{2}\right)\right] \quad (6.41)$$

due to the independence of messages entering the check node. We now obtain

$$\begin{aligned} \phi(m_{u,j}^{(l)}) &= 1 - \prod_{i=1}^{j-1} \left[\sum_{i=1}^{d_v} \lambda_i \left(1 - \phi(m_{v,i}^{(l-1)}) \right) \right] \\ &= 1 - \left[1 - \sum_{i=1}^{d_v} \lambda_i \phi((i-1)m_u^{(l-1)} + m_v^{(0)}) \right]^{j-1}, \end{aligned} \quad (6.42)$$

and we recall that the λ_i are the probabilities that a variable node has i connecting edges.

To obtain the *average* check node output signal, we average (6.42) over the check node degrees to obtain

$$m_u^{(l)} = \sum_{j=1}^{d_c} \rho_j \phi^{-1} \left(1 - \left[1 - \sum_{i=1}^{d_v} \lambda_i \phi((i-1)m_u^{(l-1)} + m_v^{(0)}) \right]^{j-1} \right). \quad (6.43)$$

This is the recursive formula for the evolution of the check node mean m_u . Note that the check node output message does not have an exact Gaussian distribution. However, these signals are mixed by the additive variable node which produces a Gaussian distributed message with higher accuracy, especially if d_v is large, and hence working with (6.43) produces results that are quite accurate.

We now continue with the determination of the thresholds for the AWGN channel, which will inform us about the potential of LDPC codes with message passing decoding. From our assumption that the all-zero codeword is transmitted, correct bits and messages have positive signs. Decoding is error-free if the soft bit error $\phi(m_{v,i}^{(l)})$ with $m_{v,i}^{(l)}$ from (6.40) goes to zero. In this case, only messages with positive sign occur, and the probability of an incorrect bit goes to zero.

The approximate threshold for the AWGN is the limit of the parameters $m_v^{(0)} = 4E_s/N_0$ for which $\phi(m_v^{(l)}) \rightarrow 0$ as $l \rightarrow \infty$. We shall be content if the *average* soft bit error

$$x^{(l)} = \sum_{i=1}^{d_v} \lambda_i \phi(m_{v,i}^{(l)}) \quad (6.44)$$

goes to zero with the iterations $l \rightarrow \infty$. Then

$$m_u^{(l)} = \sum_{j=1}^{d_c} \rho_j \phi^{-1} \left(1 - \left[1 - x^{(l-1)} \right]^{j-1} \right) \quad (6.45)$$

from (6.43), and substituting (6.45) into (6.44) we finally obtain the iteration equation for irregular LPCD codes under message passing decoding:

$$x^{(l)} = \sum_{i=1}^{d_v} \lambda_i \phi \left[\frac{4E_s}{N_0} + (i-1) \sum_{j=1}^{d_c} \rho_j \phi^{-1} \left(1 - \left[1 - x^{(l-1)} \right]^{j-1} \right) \right], \quad (6.46)$$

where $4E_s/N_0$ is the initial channel LLR. Equation (6.46) defines the *system convergence function* for irregular codes analogous to (6.36), given as

$$F_i \left(x, \frac{E_s}{N_0} \right) \stackrel{\text{def}}{=} \sum_{i=1}^{d_v} \lambda_i \phi \left[\frac{4E_s}{N_0} + (i-1) \sum_{j=1}^{d_c} \rho_j \phi^{-1} \left(1 - [1-x]^{j-1} \right) \right]. \quad (6.47)$$

As before, decoding is error-free if the recursion $x^{(l)}$ converges to zero as $l \rightarrow \infty$. This condition is satisfied if and only if $F_i(x, E_s/N_0) < x$ for all $x \in (0, \phi(4E_s/N_0))$. The code threshold is now defined as the limiting signal-to-noise ratio

$$(E_s/N_0)^* = \inf \{ s \in \mathcal{R}^+ : F_i(x, s) - x < 0, \forall x \in]0, \phi(4E_s/N_0)] \}. \quad (6.48)$$

Again, avoiding the infimum, we can express $(E_s/N_0)^*$ as the smallest root of

$$x - F_i(x, E_s/N_0) = 0 \quad (6.49)$$

Unfortunately, even with the assumption of Gaussian message distributions made in Assumption 1, there exists no closed-form solution for (6.49), and we have to evaluate the thresholds using numerical methods.

Drawing the function $F_i(x, E_s/N_0)$, we obtain a graphical convergence diagram shown in Figure 6.13 which is analogous to those for the BEC and BSC models. As the difference $x - F_i(x, E_s/N_0)$ narrows as $x \rightarrow 0$ a “slow region” or “bottleneck” occurs, and many iterations may be required to achieve final convergence. Naturally this bottleneck becomes more pronounced the closer E_s/N_0 is to the code’s threshold.

The iterated behavior of $F_i(x, 4E_s/N_0)$ is illustrated in Figure 6.13. The curves shown are for the $d_v = 15$ degree distribution reported in Table 6.2. The curve for $E_s/N_0 = -2.52$ dB is at the threshold for this degree distribution. We see that $F_i(x, E_s/N_0) \rightarrow 0$ as the number of iterations goes to infinity. The “convergence channel” $x - F_i(x, E_s/N_0)$ is

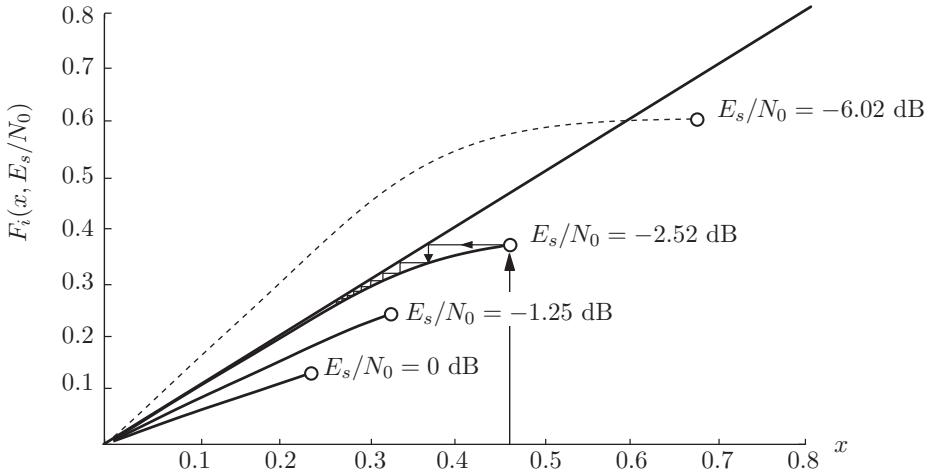


Figure 6.13: Iterative behavior of the function $h(x, E_s/N_0)$, illustrating the convergence of LDPC codes on AWGN channels.

very narrow, especially for small values of E_s/N_0 . This means a large number of iterations are required to achieve the limiting performance.

Table 6.2 show degree distributions and threshold signal-to-noise ratios for rate $R = 1/2$ irregular LDPC codes for various maximal variable node degrees. The channel capacity limit at this rate is at $E_s/N_0 = 0.198$, illustrating the excellent, “near-capacity” performance of these codes. The thresholds are computed according to both the exact [41] and the Gaussian approximative density evolution [11].

The accuracy of density evolution using the Gaussian approximation follows different trends for regular and irregular codes. For regular codes, the accuracy has been shown to improve with increasing d_v . For irregular codes, the accuracy worsens when the maximum variable node degree is increased [11]. The approximation is more accurate for regular codes, but still good for irregular codes with maximal variable node degree d_v of about ten or less.

Another result found in [11] is the *concentration theorem* for check node degree distributions. A concentrated degree distribution satisfies

$$\rho(x) = \rho_k x^{k-1} + (1 - \rho_k) x^k. \quad (6.50)$$

A distribution of this form maximizes the rate of convergence under decoding iterations.

d_v	4	8	9	10	11	12	15	20	30	50
λ_2	.38354	.30013	.27684	.25105	.23882	.24426	.23802	.21991	.19606	.17120
λ_3	.04237	.28395	.28342	.30938	.29515	.25907	.20997	.23328	.24039	.21053
λ_4	.57409			.00104	.03261	.01054	.03492	.02058		.00273
λ_5						.05510	.12015			
λ_6								.08543	.00228	
λ_7							.01587	.06540	.05516	.00009
λ_8		.41592				.01455		.04767	.16602	.15269
λ_9			.43974					.01912	.04088	.09227
λ_{10}				.43853		.01275			.01064	.02802
λ_{11}					.43342					
λ_{12}						.40373				
λ_{14}							.00480			
λ_{15}							.37627			.01206
λ_{19}								.08064		
λ_{20}								.22798		
λ_{28}									.00221	
λ_{30}									.28636	.07212
λ_{50}										.25830
ρ_5	.24123									
ρ_6	.75877	.22919	.01568							
ρ_7		.77081	.85244	.63676	.43011	.25475				
ρ_8			.13188	.36324	.56989	.73438	.98013	.64584	.00749	
ρ_9						.01087	.01987	.34747	.99101	.33620
ρ_{10}								.00399	.00150	.08883
ρ_{11}										.57497
$(E_s/N_0)^*$	-2.20	-2.56	-2.60	-2.61	-2.63	-2.64	-2.68	-2.70	-2.74	-2.76
$(E_s/N_0)_{GA}^*$	-2.164	-2.45	-2.50	-2.50	-2.50	-2.52	-2.52	-2.53	-2.55	-2.59

Table 6.2: Exact $(E_s/N_0)^*$ and approximate $(E_s/N_0)_{GA}^*$ thresholds for rate $R = \frac{1}{2}$ degree distributions. Capacity is at $E_s/N_0 = -2.823$ dB, or $E_b/N_0 = 0.188$ dB.

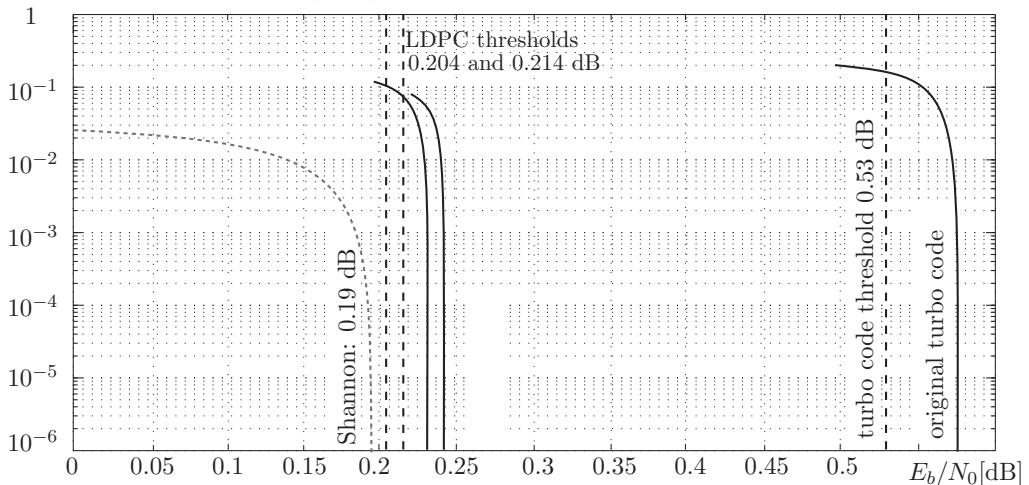
Based on the concentration theorem, Chung et al. [9] designed LDPC codes for the AWGN channel using a discretized density evolution, that is, they quantized the probability densities and propagated this quantized PDF with accuracies of up to 14 bits. The degree distributions were then obtained via linear optimization using constraint rules. The degree distributions of some of the optimized codes are given in Table 6.3.

The check node distribution is given as the single number $\rho_{av} = (1 - \rho_k)(k - 1) + \rho_k k = k - 1 + \rho_k$. With the Shannon capacity at $E_b/N_0 = 0.188$ dB, these codes come to within 0.0247, 0.0147, and 0.0045 dB of the Shannon capacity, respectively. Chung et al. [9] also provide simulation results for block lengths of $n = 10^7$, which are reproduced in Figure 6.14, illustrating the tight predictions of the density evolution method.

d_v	100	200	d_v	8000
λ_2	.170031	.153425	λ_2	.096294
λ_3	.160460	.147526	λ_3	.095393
λ_6	.112837	.041539	λ_6	.033599
λ_7	.047489	.147551	λ_7	.091918
λ_{10}	.011481		λ_{15}	.031642
λ_{11}	.091537		λ_{20}	.086563
λ_{18}		.047938	λ_{50}	.093896
λ_{19}		.119555	λ_{70}	.006035
λ_{26}	.152978		λ_{100}	.018375
λ_{27}	.036131		λ_{150}	.086919
λ_{55}		.036379	λ_{400}	.089018
λ_{56}		.126714	λ_{900}	.057176
λ_{100}	.217056		λ_{2000}	.085816
λ_{200}		.179373	λ_{3000}	.006163
λ_{6000}			λ_{6000}	.003028
λ_{8000}			λ_{8000}	.118165
$\rho_{\text{av}} = k - 1 + \rho$	10.9375	12		18.5
$(E_b/N_0)^*$ [dB]	0.212	0.194		0.192

Table 6.3: Parameters of near-capacity optimized LDPC codes.

Bit Error Probability (BER)

Figure 6.14: Simulations of LDPC and the original turbo code for block lengths $n = 10^7$.

A graphical optimization based on EXIT analysis discussed in Chapter 8 was used in [22] to optimize the convergence behavior of LDPC codes, which yielded the irregular code with an error cliff about 0.5 dB better than a regular code of the same rate at the same size of $n = 4000$. Simulations for both the degree optimized code and a regular (3,6) LDPC code are shown in Figure 6.15. The code with the error floor is the irregular design. It is clearly evident that the irregular code has a lower signal-to-noise ratio threshold. However, the error floor of the irregular code is significantly higher than the error floor of the regular code. Both codes are optimized in avoiding cycles (see next chapter), but the avoidance of cycles is substantially more successful for the regular code, where all cycles of length 8 and shorter are avoided, while in the irregular code only cycles of length 6 and shorter could be avoided.

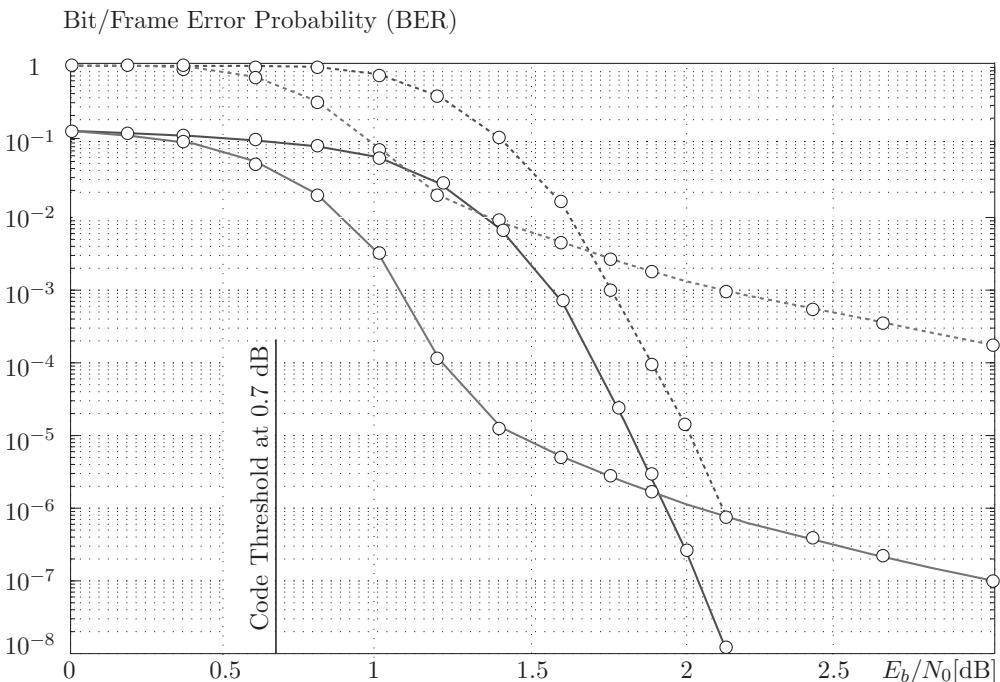


Figure 6.15: Performance results for regular and irregular cycle optimized LDPC codes of rate $R = 1/2$ for a block length of 4000 bits. Dashed lines are for the block or frame error rates, solid lines for the BER.

This leads into the question of code optimality. While degree optimization can achieve excellent code thresholds, there is no guarantee that the code will have a low error performance for larger signal-to-noise ratios. Indeed, codes with low thresholds seem to be more difficult to optimize in the error floor region as illustrated by the simple cycle-optimized codes in Figure 6.15. The issue of error floor optimization is picked up and discussed in more detail in the Chapter 7.

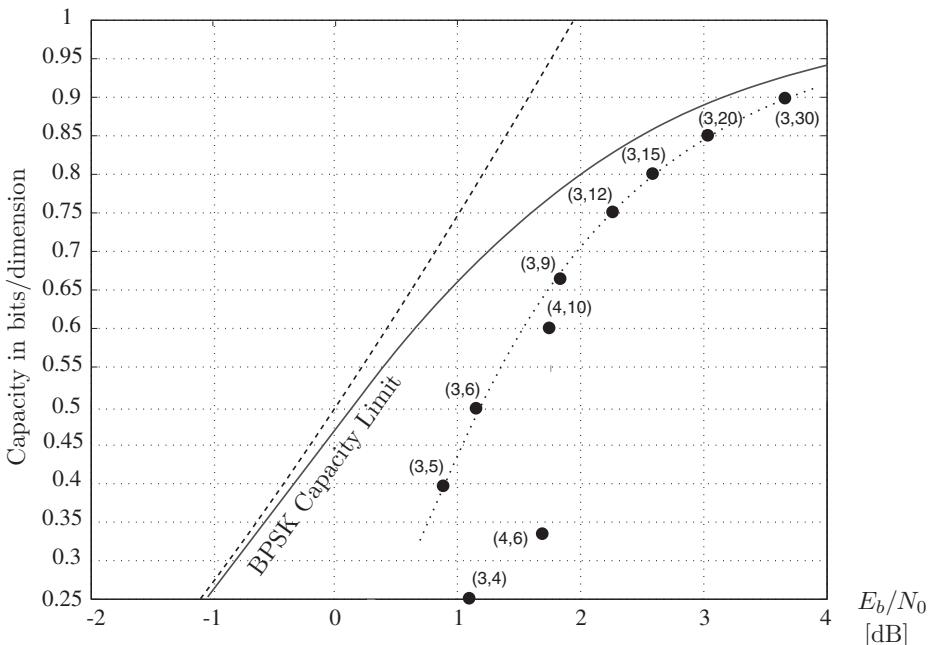


Figure 6.16: Thresholds for regular LPDC codes compared to the BPSK capacity limit.

The issue of regular versus irregular LDPC codes doesn't stop at the error floor question. As we have observed, for rate $R = 1/2$ codes, substantial gains are provided by an irregular code over a regular code, gains in the order of half a dB or more. For lower rates, even more can be gained, while the difference between regular and irregular coded performance diminishes as the rate increases. Figure 6.16 illustrates this point by comparing the thresholds of the best regular LDPC codes to the BPSK Shannon bound. High-rate regular LDPC codes have thresholds less than 0.5 dB away from the BPSK capacity limit. This accounts for the fact that in such high-rate applications, regular LDPC codes are favored, especially since they can be designed with very low error floors. The (6,32) regular code with rate $R = 0.841$ of the IEEE 802.3an standard is a case in point. This code is discussed in Section 6.5.3.

6.5 Code Families and Construction

6.5.1 Constructions with Permutation Matrices

If we take a closer look at \mathbf{H}_1 from Section 6.2, we notice that the parity-check matrix is composed of 18 sub-matrices, each of which is a permutation matrix \mathbf{P}_{ij} , that is, the operation $\mathbf{P}_{ij}\mathbf{x}$ simply rearranges the elements in \mathbf{x} like a permuter. This is illustrated in Figure 6.17. The advantage of this arrangement is twofold: First, it is easy to see that these codes are regular by construction. Second, as will be discussed later, this construction leads to simplifications of the implementation of the permuter.

We may now start by first identifying the positions of these permutation matrices within the larger parity-check matrix. This is done by the so-called *proto-matrix* whose entries are non-negative integers. For now assume we just have the entries $\{0, 1\}$ in a m', n' proto-matrix \mathbf{H}_p . To obtain the actual parity-check matrix, we expand each “1” entry into a $p \times p$ permutation matrix, creating a $m'p \times n'p$ parity-check matrix.

As an example, consider the 3×4 proto-matrix $\mathbf{H}_{p,1}$, which expands into

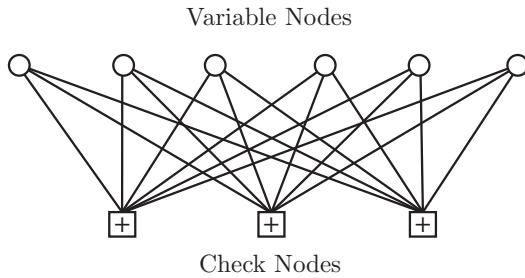
$$\begin{aligned} \mathbf{H}_{p,1} &= \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \\ \rightarrow \mathbf{H}_1 &= \begin{bmatrix} \mathbf{P}_{1,1} & \mathbf{P}_{1,2} & \mathbf{P}_{1,3} & \mathbf{P}_{1,4} & \mathbf{P}_{1,5} & \mathbf{P}_{1,6} \\ \mathbf{P}_{2,1} & \mathbf{P}_{2,2} & \mathbf{P}_{2,3} & \mathbf{P}_{2,4} & \mathbf{P}_{2,5} & \mathbf{P}_{2,6} \\ \mathbf{P}_{3,1} & \mathbf{P}_{3,2} & \mathbf{P}_{3,3} & \mathbf{P}_{3,4} & \mathbf{P}_{3,5} & \mathbf{P}_{3,6} \end{bmatrix}. \end{aligned} \quad (6.51)$$

Since the permutation matrices \mathbf{P}_{ij} in (6.51) can be of arbitrary size, we can generate large parity-check matrices from small proto-matrices. In Figure 6.17 their size is 4×4 , but in actual applications the size of these permutation matrices is typically much larger. The example presented in Section 6.5.3 uses permutation matrices of size $p = 64$.

The *proto-matrix* contains only the positions of the permutation matrices within the larger parity-check matrix, but does not specify the actual \mathbf{P}_{ij} . In the sense that the permutation matrices can be chosen arbitrarily, $\mathbf{H}_{p,1}$ specifies a family parity-check matrices. In the case of (6.51), each position is occupied by a permutation matrix, and therefore this contracted description may appear trivial. However, as we will see in the sequel, this is not always the case, and many codes have been constructed using this proto-matrix approach, or equivalent approaches [10, 16, 36, 53].

The proto-matrix has its own graphical representation, shown in Figure 6.18 and called a *protograph* of an LDPC code family. Each edge represents a permutation matrix \mathbf{P}_{ij} of size $p \times p$, and each variable and check node is a set of p stacked nodes. The protograph in Figure 6.18 corresponds to $\mathbf{H}_{p,1}$ from above.

$$\mathbf{H}_1 = \left[\begin{array}{c|c|c|c|c|c|c} & & 1 & 1 & & & \\ & 1 & & 1 & & & \\ & & 1 & & 1 & & \\ \hline 1 & & 1 & & 1 & & \\ & 1 & & 1 & & 1 & \\ & & 1 & & 1 & & \\ \hline 1 & 1 & & 1 & & 1 & \\ & & 1 & & 1 & & \\ & & & 1 & & 1 & \\ \hline 1 & & 1 & & 1 & & \\ & 1 & & 1 & & 1 & \\ & & 1 & & 1 & & \\ \hline 1 & & 1 & & 1 & & \\ & 1 & & 1 & & 1 & \\ & & 1 & & 1 & & \\ \hline 1 & & 1 & & 1 & & \\ & 1 & & 1 & & 1 & \\ & & 1 & & 1 & & \\ \hline \end{array} \right]$$

Figure 6.17: Parity-check matrix \mathbf{H}_1 as a composite of permutation matrices.Figure 6.18: Bipartite graph for the $(3, 4)$ regular LDPC code with parity-check matrix \mathbf{H}_1 .

Expanding the proto-graph into a code using size- p permutation matrices now requires replacing each entry hp_{ij} in $\mathbf{H}_{p,1}$ by $hp_{ij}\mathbf{P}_{ij}$, or, equivalently, expanding the graph in (6.18) by duplicating it p times, and rearranging the connections from check nodes $[ip, (i+1)p-1]$ to variable nodes $[jp, (j+1)p-1]$ according to \mathbf{P}_{ij} . This is illustrated in Figure 6.19 for \mathbf{H}_1 . This process is also called *graph lifting*.

Before we discuss other advantages, we briefly address the impact on the implementation of LDPC codes. While the permuter network will have to be implemented as a network of physical wires in high-speed decoders, where each node is implemented as a separate processor, slower data rates may allow a decoder to share computational resources among nodes. Specifically, a permutation matrix based code can be built with a fixed wire

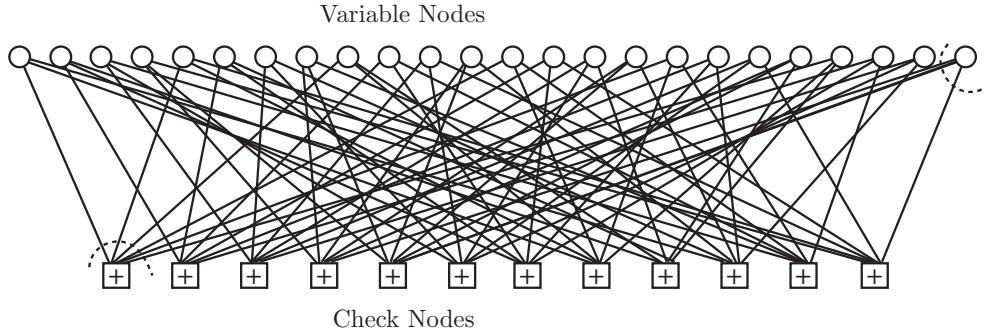


Figure 6.19: Expansion of \mathbf{H}_{p1} into \mathbf{H}_1 using permutation matrices \mathbf{P}_{ij} .

network for *only* the protograph, while permutation matrices themselves are implemented in read–write memory blocks. Such a decoder operates in partial parallel mode, processing blocks of data which correspond to the structure of the proto-matrix. This is illustrated for the code \mathbf{H}_1 in Figure 6.20.

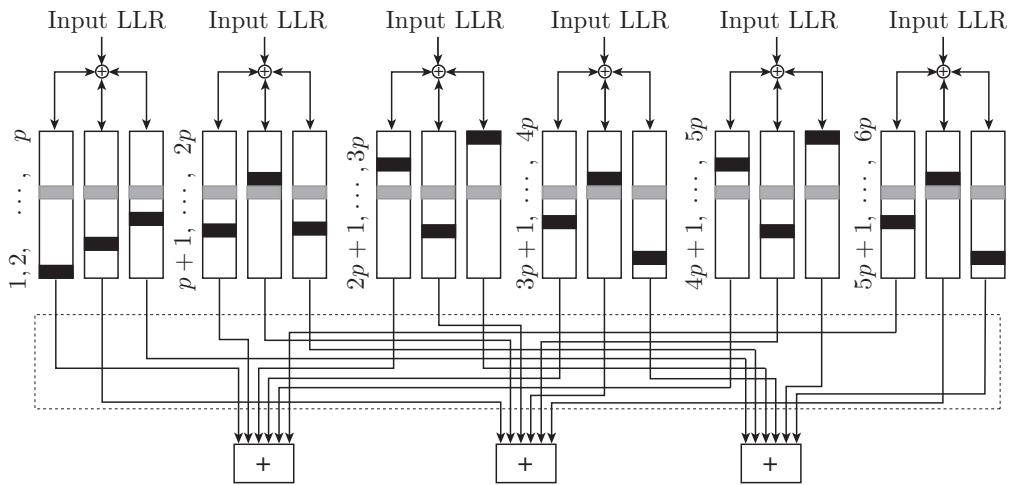


Figure 6.20: Building block for a fixed interleaver parallel implementation of \mathbf{H}_1 .

A further advantage of this structure is that read–write memory elements can be used without the danger of access collisions, since each memory block will be accessed by exactly one read and one write operation per decoding cycle, something that is not quite obvious

from the Tanner graph of the code in Figure 6.19.

Before leaving this subject, we'll discuss one more example of code construction via proto-graphs. It is also an example of a more generalized code structure, which combines turbo and LDPC code elements. The code in question is the *accumulate-repeat-accumulate* (ARA) code of [10] with the deceptively simple proto-matrix

$$\mathbf{H}_{p,2} = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}, \quad (6.52)$$

whose protograph is shown in Figure 6.21. Two points need special explanations. First, $H_{p,2}$ contains entries of 1 and 2, and the proto-graph contains parallel edges. While a regular (expanded) parity-check code cannot contain parallel edges, we recall that the edges in the protograph are placeholders for permutation matrices. An entry of “2” therefore means there are two superimposed permutation matrices that are placed into that position. After expansion, we need to make sure that the parity-check matrix does not contain overlapping entries. This is illustrated in the example expansion

$$\mathbf{H}_2 = \left[\begin{array}{cc|cc|cc} 1 & 1 & & & 1 & \\ & 1 & 1 & & 1 & \\ & & 1 & 1 & & \\ & & & 1 & 1 & \\ \hline 1 & & 1 & | & 1 & \\ 1 & & & | & 1 & 1 \\ \hline & 1 & & | & 1 & 1 \\ & & 1 & | & 1 & 1 \\ & & & | & 1 & 1 \\ & 1 & & | & 1 & 1 \\ & & 1 & | & 1 & 1 \end{array} \right] \Leftarrow \mathbf{H}_{p,2} = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}. \quad (6.53)$$

The second point concerns the differential encoders at both the input and the output. The output differential encoder will be clarified below when we introduce the *repeat-accumulate* codes. The input differential encoder is very similar. Specifically, the first edge is replaced by an identity matrix, and the parallel edge is replaced by a cyclic shift matrix, whose size $p = 5$ representative is given by

$$\mathbf{P}_5 = \begin{bmatrix} & 1 & & \\ & & 1 & \\ & & & 1 \\ 1 & & & \end{bmatrix}. \quad (6.54)$$

Figure 6.22 shows an expansion of the protograph of Figure 6.21 for a small-sized permutation matrices with $p = 5$. This illustrates the edge arrangements for the differential

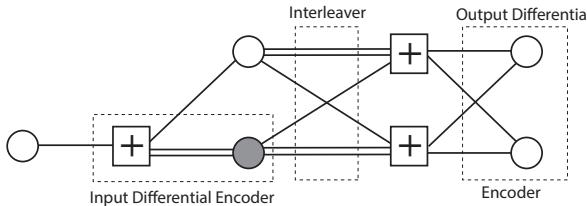


Figure 6.21: Proto-building block with parallel edges for generalized LDPC codes. The shaded variable node is punctured, i.e., they are not transmitted. It is the accumulation variable in the input differential encoder. This is the AR3A encoder of [10].

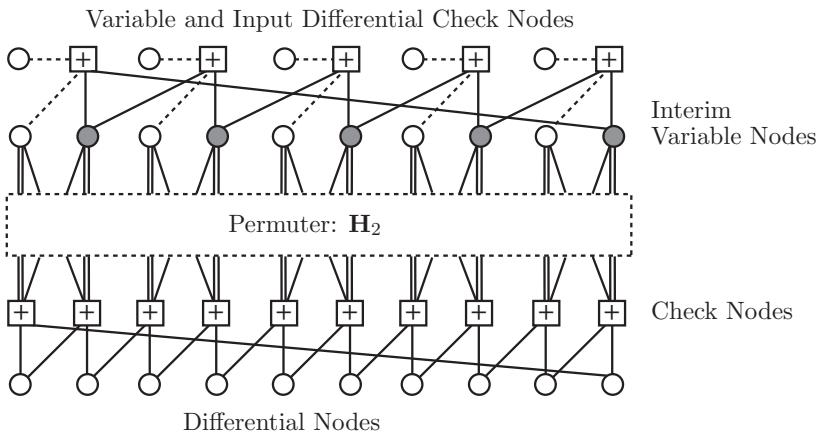


Figure 6.22: Bipartite graph for the $(3, 4)$ regular LDPC code with parity-check matrix \mathbf{H}_1 . Shaded nodes are punctured, a rate increase method we will encounter in Chapter 8.

parts of the code. Decoding proceeds following any of the algorithms discussed in this chapter, realizing that there are several layers of check and parity nodes. The order in which the messages in and out of these nodes are updated have only a minor impact on performance as long as all nodes are updated (approximately) once each iteration.

One advantage of codes such as the ARA and RA codes—discussed below—is that they can be encoded in a simple way as shown in Section 6.6. The repeat-accumulate codes also form a natural link between turbo and LDPC codes, since they can easily be interpreted as either type, and the set of differential nodes can be decoded with a trellis APP component decoder, as is customary for turbo codes.

6.5.2 Cycle Reduction Design

As we discussed earlier, cycles in the Tanner graph of the code are cause for concern, especially short cycles. First, the performance analysis discussed in Section 6.4 relies on the independence of all messages down the tree. This is only true until the shortest cycle is completed, at which time previously used messages are mixed into the message flow and the independence assumption is compromised.

More importantly however, as we will show in Chapter 7, cycles form part of the sub-graph structures that cause the error rate to remain bounded away from zero even for large values of the signal-to-noise ratio. Therefore, it was recognized early that one important design criterion would be to avoid short cycles.

Most code design rules target the elimination of short cycles [24, 48] via a variety of ad hoc search and construction algorithms. It is widely accepted that the elimination of 4-cycles and perhaps 6-cycles is essential in obtaining good performance, but the influence of higher order cycles is more debatable, and we will present a more detailed discussion of the error floor phenomenon and the impact of sub-graphs on it in Chapter 7. It seems reasonable at this point to design the interleavers with cycles as large as possible.

We concentrate on cycle-aware designs for permutation-based LDPC codes. A permutation matrix of size p can be specified by an integer s together with the equation

$$w = x + s \bmod p, \quad (6.55)$$

which relates the row index w with the column index x . Note that addressing the different check node indices in Figure 6.20 can be accomplished with a simple counter that is offset by $p - s$.

Let us start with a proto-matrix which fulfills certain conditions, for example, let the proto-matrix be cycle-4 free. We can then guarantee good interleavers by a technique presented in [36], which recursively constructs the interleaver avoiding that 6 cycles are created. Assume we have a proto-matrix that is 4-cycle free. That is, the next shortest cycle that can possibly occur is of length 6. An illustration of this is shown in Figure 6.23

Since the row index and column index are linked by (6.55), we consider the following differences: $w_1 - w_6, w_4 - w_5, w_3 - w_2$, and if

$$w_1 - w_6 + w_4 - w_5 + w_3 - w_2 = s_{1,3} - s_{2,3} + s_{2,2} - s_{3,2} + s_{1,4} - s_{3,4} \bmod p \neq 0 \quad (6.56)$$

this is an obvious sufficient condition for this 6-cycle to not be possible. It is shown in [36] that it is also a necessary condition. With this concept, [36] builds short-cycle-free LDPC codes recursively checking and ensuring (6.56) by selecting appropriate offset factors s_{ij} for each populated position in the original proto-matrix. It appears that this method not only produces large girth interleavers, but has a deceptively simple implementation since the memory addresses in Figures 6.20 can be generated by simple counters each with an offset of s_{ij} . Note that the interleavers in [44] decompose into exactly the same structure.

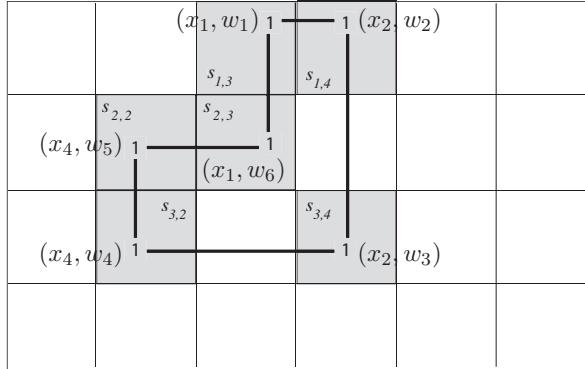


Figure 6.23: Illustration of a 6-cycle in an (LDPC) parity-check or interleaver matrix.

Many other approaches have been explored in the literature to construct large-cycle codes, many of them relying, at least in part, on computational search techniques, such as the progressive-edge growth algorithm in [24].

6.5.3 RS-based Construction

In this section, we discuss a specific example of a class of codes designed both from permutation matrices and for short cycle avoidance. These codes are the Reed–Solomon code-based LDPC code constructions from [16]. We chose this construction since it was used for the LDPC code in 10 Gbit/s IEEE 803.2 Ethernet standard. This code is one of the most cited and studied LDPC code, and we will return to it in the next chapter when we discuss the error floor.

We use symbols from the Galois field $\text{GF}(p^s)$, where p is a prime number and s is positive integer. For details on Galois fields and their application to coding theory see [27]. If α is a primitive element of $\text{GF}(p^s)$, then the powers $1 = \alpha^0, \alpha^1, \dots, \alpha^{p^s-2}$ generate all the elements. For $\rho > 0$, the Reed–Solomon code of length p^s and minimum distance $\rho - 1$ is generated by the generator polynomial

$$\begin{aligned} g(X) &= (X - \alpha)(X - \alpha^2) \cdots (X - \alpha^{\rho-2}) \\ &= g_0 + g_1 X + g_2 X^2 + \cdots + X^{\rho-2}. \end{aligned} \quad (6.57)$$

This code can be shortened to obtain a $(\rho, \text{minimum distance} = 2, \rho - 1)$ RS code with generator matrix given by

$$\mathbf{G}_{\text{RS}} = \begin{bmatrix} g_0 & g_1 & g_2 & \cdots & 1 & 0 \\ 0 & g_0 & g_1 & g_2 & \cdots & 1 \end{bmatrix}. \quad (6.58)$$

This base code will now be used to construct 4-cycle free LDPC codes in the following way. First note that since $\rho - 1$ is the minimum symbol distance of (6.58), two codewords in the code generated by \mathbf{G}_{RS} have at most *one* common symbol in same location.

We now take a codeword $\mathbf{b} = (b_1, b_2, \dots, b_\rho)$ from this code and replace each component b_j by its location vector $\mathbf{z}(b_j)$, which is a *binary* vector of length $p^s - 1$ which has a single one in position i , and i is the exponent of the symbol $b_j = \alpha^i$ expressed as a power of the primitive generator element α . The element $0 = \alpha^\infty$ may be placed at the beginning or end of this expansion. Note that $\mathbf{z}(b_j)$ has length p^s , and we construct a length ρp^s binary vector as

$$\mathbf{z}(\mathbf{b}) = (z(b_1), z(b_2), \dots, z(b_\rho)), \quad (6.59)$$

which has ρ non-zero components which equal “1” by construction. Since two codewords of the RS code have at most one common symbol in the same location, their location vectors have at most one non-zero component in common.

Now take a codeword \mathbf{b} from the RS code which has weight ρ , then the codewords $\beta\mathbf{b}$ with $\beta \in GF(p^s)$ forms a subcode with minimum distance ρ . Each coset of this subcode has the same properties of consisting of p^s sequences with minimum distance ρ also. The symbol location vectors for two codewords both taken from any of these coset therefore have *no* non-zero position in common. These binary symbol location vectors are the basis for the construction of our LDPC code parity-check matrices.

First choose any coset of the above RS subcode as the generating set. Let this be the i th coset, and note that the ordering of cosets does not matter. We first form the matrix \mathbf{A}_i over $GF(2)$ whose rows consist of the p^s symbol location vectors of the generating set. The dimensions of \mathbf{A}_i are $p^s \times \rho p^s$, and it contains a total of ρp^s non-zero entries. \mathbf{A}_i has constant row weight of ρ , and column weight of 1. In fact, \mathbf{A}_i consists of a row of ρ permutation matrices of size $p^s \times p^s$, since each symbol b_j “cycles” through all p^s elements of $GF(2^s)$, i.e.,

$$\mathbf{A}_i = [\mathbf{P}_{i1}, \mathbf{P}_{i2}, \dots, \mathbf{P}_{i\rho}], \quad \mathbf{P}_{ij} \text{ is } 2^s \times 2^s. \quad (6.60)$$

The reader may notice that we now constructed a block row of a permutation matrix-based parity-check matrix as in Figure 6.17. All that remains is to add more block rows.

This is done by adding matrices \mathbf{A}_j from other cosets to form the parity-check matrix

$$\mathbf{H}_{RS} = \begin{bmatrix} \mathbf{A}_1 \\ \vdots \\ \mathbf{A}_\gamma \end{bmatrix} = \begin{bmatrix} \mathbf{P}_{11} & \mathbf{P}_{12} & \cdots & \mathbf{P}_{1\rho} \\ \vdots & & & \\ \mathbf{P}_{\gamma 1} & \mathbf{P}_{\gamma 2} & \cdots & \mathbf{P}_{\gamma \rho} \end{bmatrix}. \quad (6.61)$$

Note that no two rows in \mathbf{A}_i have any non-zero components in the same location, and no two rows in different matrices \mathbf{A}_i and \mathbf{A}_j have more than one non-zero component in the same location. The Tanner graph of \mathbf{H}_{RS} can therefore have no 4 cycles, which can be easily seen by applying the reasoning in Figure 6.23 to this case.

\mathbf{H}_{RS} describes a (γ, ρ) regular LDPC code, and it can be shown that its minimum distance is at least $\gamma + 1$, which is, however, typically a loose bound [16]. The (6,32), rate $R = 0.841$ (2048,1723) RS-based code is generated from $\text{GF}(2^6)$ with $\rho = 32$, and $\gamma = 6$ is the code used in the IEEE 802.3 standard. Its minimum distance is at least $d_{\min} \geq 7$, but its actual $d_{\min} = 14$. We will return to this code in the next chapter. Other construction examples can be found in [16].

6.5.4 Repeat-Accumulate Codes

The near-capacity performance of low density parity-check codes and the elegance of the graphical description of the codes and the decoding algorithm led to the investigation of other graph-based coding schemes. In this section, we consider a particularly interesting variant of LDPC codes known as repeat-accumulate (RA) codes [15]. These codes are important for several reasons, but a key point is that RA codes can also be viewed as a special type of serial turbo codes (8). This is actually the more common way of treating them. They therefore serve to illustrate the close relationship between these two classes of codes, and, indeed, of all graph-based error control codes. As turbo codes they are the serial concatenation of two very simple component codes. Their relationship to LDPC codes allows them to benefit from the notion of irregularity and to be analyzed using density evolution.

Figure 6.24 shows a block diagram of a non-systematic repeat accumulate encoder in serial concatenated form. The outer encoder is a simple repetition code with rate $R = 1/q$ and the inner encoder is a rate one recursive encoder with feedback polynomial $1+D$, that is, an accumulator. The inner and outer encoders are separated by the interleaver, and the overall code rate is $R = 1/q$. Despite their apparent simplicity, these codes perform very well, and for large block lengths and rates $R = 1/q \leq 1/3$, they have thresholds within 1 dB of the Shannon limit of the AWGN channel [15]. Indeed, in Chapter 8 we will present modified versions of the RA codes which come to within 0.1 dB of capacity.

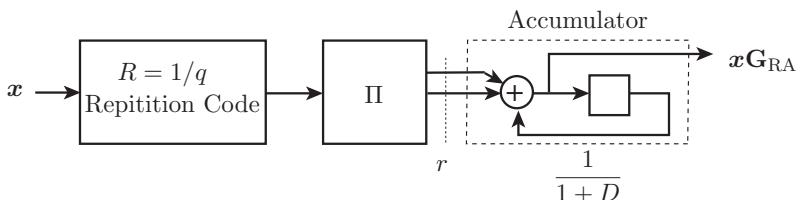


Figure 6.24: Block diagram of a repeat-accumulate code encoder.

The LDPC bipartite graphical representation of the RA code of Figure 6.24 is straightforward to find and is best illustrated by an example. Let q equal 3 and the check node

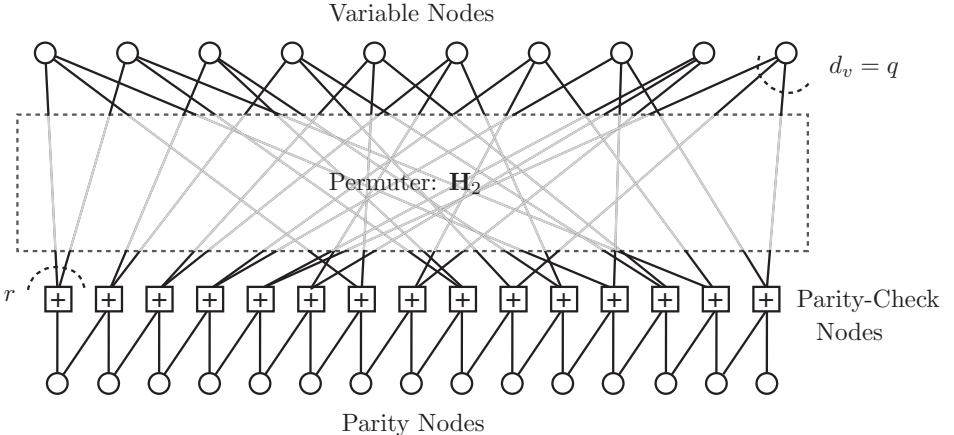


Figure 6.25: Tanner graph of an RA code with $q = 3$ and check node degree $d_c = r + 2$.

parameter r be 2, also called the *check node left degree*; then the graph of the resulting (LDPC) code is shown in Figure 6.25. This code has rate $R = r/(r+q)$. From this graph, it is clear that each variable node has degree $d_v = q = 3$ and that each parity node has degree 2 due to the accumulator, except for the rightmost parity variable node. Thus, the graph of an RA code is a regular bipartite graph with degree- q information variable nodes, degree- $r+2$ check nodes, and degree-2 parity nodes.

It is evident that RA codes can also be made irregular, this was first suggested in [25]. That is, the number of times that an information bit is repeated is varied. The resulting Tanner graph is depicted in Figure 6.26 and is parameterized by the degree distribution $(\lambda_1, \lambda_2, \dots, \lambda_J)$ of the information variable nodes and the left degree r of the check nodes. Note that in the graph of an regular RA code, contrary to what is shown in Figure 6.25, the left check node degree is typically $r = 1$.

In [25], the authors considered the subclass of systematic irregular RA codes with a fixed check node left degree of r . In a systematic RA code the graph is used to compute the parity bits, which are then simply appended to the information bits to form the codeword is $\mathbf{x} = (u_1, u_2, \dots, u_k, x_1, x_2, \dots, x_r)$. Using linear programming techniques, they obtained degree sequences for systematic irregular RA codes and then computed exact and approximate thresholds. The results for codes of rate $R \approx 1/3$ are shown in Table 6.4. These codes are very potent and come to within approximately 0.1 dB of the capacity limit.

RA codes form an important link between LDPC and turbo code approaches, unifying these apparent different methods. They are also representatives of (systematic) serially concatenated codes, a somewhat neglected coding strategy which, however, in no way lags

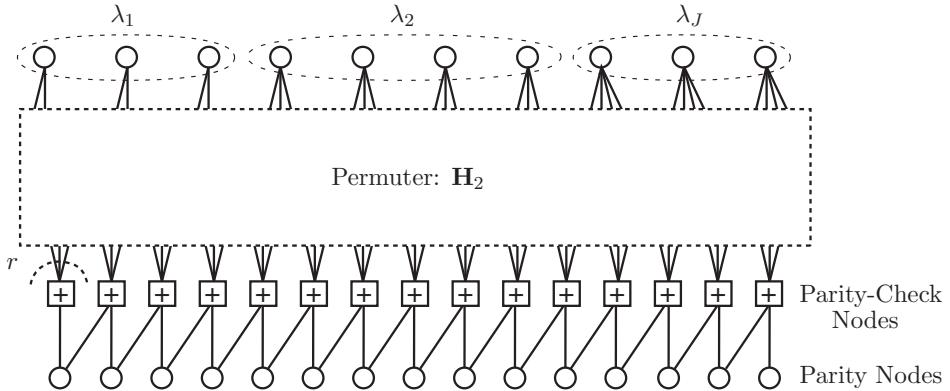


Figure 6.26: Tanner graph of an irregular RA code.

a	2	3	4
λ_2	.139025	.078194	.054485
λ_3	.222155	.128085	.104315
λ_5		.160813	
λ_6	.638820	.036178	.126755
λ_{10}			.229816
λ_{11}			.016484
λ_{12}		.108828	
λ_{13}		.487902	
λ_{27}			.450302
λ_{28}			.017842
Rate	0.333364	0.333223	0.333218
$(E_b/N_0)^*$ [dB]	0.191	-0.250	-0.368
(E_b/N_0) [dB]	0.293	-0.117	-0.255
Capacity (dB)	-0.4953	-0.4958	-0.4958

Table 6.4: Degree profiles for optimized systematic irregular repeat accumulate codes [25].

behind the turbo and LDPC codes in terms of performance or implementation efficiency.

6.6 Encoding of LDPC Codes

The decoding operations discussed in this chapter are all very efficient in terms of computational complexity, that is, their complexity grows only linear with the length n of the

code, and the number of iterations i . This is the minimal complexity with a decoding effort per bit roughly constant and independent of the size of the code, proportional only to the number of iterations needed.

At first glance, however, the complexity of encoding an LDPC code may be more complex. First, we need to find the encoding matrix, such that $\mathbf{G}\mathbf{H}^T = \mathbf{0}$. While finding \mathbf{G} is quite straightforward [27], \mathbf{G} is, in general, not sparse, and the number of non-zero entries in \mathbf{G} grows as n^2 . With it, the encoding operation $\mathbf{u}^T\mathbf{G}$ requires $\mathcal{O}(n^2)$ operations, which may be very significant for large n .

In this remainder of this chapter we now discuss three approaches to reduce this apparent encoding complexity to algorithms of order $\mathcal{O}(n)$. All of these use triangulation of \mathbf{H} in one way or other.

6.6.1 Triangular LDPC Codes

One way around this problem is to enforce a triangular structure of the parity-check matrix and see if LDPC codes with good performance can still be constructed. Due to the correlations enforced by the structural constraint, a random performance analysis is not strictly possible anymore. However, constructing LDPC codes with triangular parity-check matrices does produce codes with equivalent performance to those with completely random parity-check matrices. This is illustrated in Figure 6.27 for a large code design with $n = 36,993$. As can be seen from the BER performance, there is little difference in performance between a triangular and a completely random code.

If \mathbf{H}_p is lower triangular, then the BEC decoding algorithm from Section 6.4.1 can effectively be used to perform iterative encoding in at most $m = N - K$ steps. Consider the following simple example:

$$\mathbf{H} = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 1 \end{bmatrix}. \quad (6.62)$$

The constraint graph for this matrix is shown in Figure 6.28.

The information bits at the top of the graph are initialized with ± 1 and the parity nodes with 0. When the BEC decoding algorithm is applied, the first round of message passing will cause parity-bit 1 to become known. In the second round, a non-zero message from parity-bit 1 allows parity-bit 2 to become known. Similarly, parity bit 3 becomes known in the third iteration. Encoding therefore takes place in $m = 3$ iterations, in fact, successively going through each row constraint in the parity-check matrix. In other words, the set of parity bit variables is never a stopping set.

The authors of [20] found that encoding can also be performed by means of iterated approximation. Suppose a codeword has the structure $\mathbf{x} = [\mathbf{x}_u \mathbf{x}_p]$, where \mathbf{x}_u is the

information sequence and \mathbf{x}_p is the parity sequence. We can then split \mathbf{H} into $[\mathbf{H}_u \mid \mathbf{H}_p]$, giving the equation

$$\mathbf{H}_p \mathbf{x}_p^T = \mathbf{H}_u \mathbf{x}_u^T, \quad (6.63)$$

which we need to solve $\mathbf{H}_p \mathbf{x}_p^T = \mathbf{H}_u \mathbf{x}_u^T$ for \mathbf{x}_p^T . To do so, we first select an initial guess for \mathbf{x}_p , which we'll denote by \mathbf{i}_0 . We will then use an iterative method to obtain subsequent approximations, denoted \mathbf{x}_k . These approximations will converge to \mathbf{x}_p using the following iteration:

$$\mathbf{x}_{i+1}^T = (\mathbf{H}_p + I) \mathbf{x}_i^T + \mathbf{H}_u \mathbf{x}_u^T, \quad (6.64)$$

provided \mathbf{H}_p is non-singular. Correct encoding results after $i' \geq i$ iterations if and only if $(\mathbf{H}_p + \mathbf{I})^{i'} = 0$. To apply the algorithm, we can split the code's graph into two parts as shown in Figure 6.28: an information-bit section and a parity-bit section. The arrows in the graph indicate which edges will be enabled to transmit edges from check nodes in the encoding algorithm.

The top half of the graph computes the bits in \mathbf{b} . The bottom part only represents the constraint in \mathbf{H}_p , which is an $m \times m$ matrix, so the number of parity nodes (labeled v_i) is the same as the number of check nodes (labeled c_i) for this part of the graph. The encoding algorithm can then be implemented with the following algorithm:

Step 1: Set all $v_i = +1$. Initialize information nodes to ± 1 as appropriate, and propagate messages to yield b_i .

Step 2: Variable nodes send $\mu_{ij} = b_i$ to all $j \in C_i \setminus i$.

Step 3: Check nodes answer with $\beta_{jj} = \prod_{l \in V_j \setminus j} \mu_{lj}$, sent to v_j only.

Step 4: Variable nodes update $v_i = \beta_{ii}$. Return to Step 2 and repeat for the prescribed number of iterations.

This result allows us to construct a code which is guaranteed to be graphically encodable in a fixed number of iterations.

In [20] a method is presented for constructing “reversible” LDPC codes, which can be encoded in two iterations. We first note that $(\mathbf{H}_p \oplus \mathbf{I})^2 = \mathbf{I} \rightarrow \mathbf{H}_p^2 = \mathbf{I}$. A reversible code may then be constructed by selecting a 2×2 matrix \mathbf{A}_0 for which $\mathbf{A}_0^2 = \mathbf{I}$. \mathbf{H}_p can be obtained by recursive application of one of the following rules:

$$\mathbf{A}_k = \begin{bmatrix} \mathbf{A}_{k-1} & \mathbf{I} \\ \mathbf{0} & \mathbf{A}_{k-1} \end{bmatrix} \quad \text{or} \quad \mathbf{A}_k = \begin{bmatrix} \mathbf{A}_{k-1} & \mathbf{0} \\ \mathbf{0} & \mathbf{A}_{k-1} \end{bmatrix}. \quad (6.65)$$

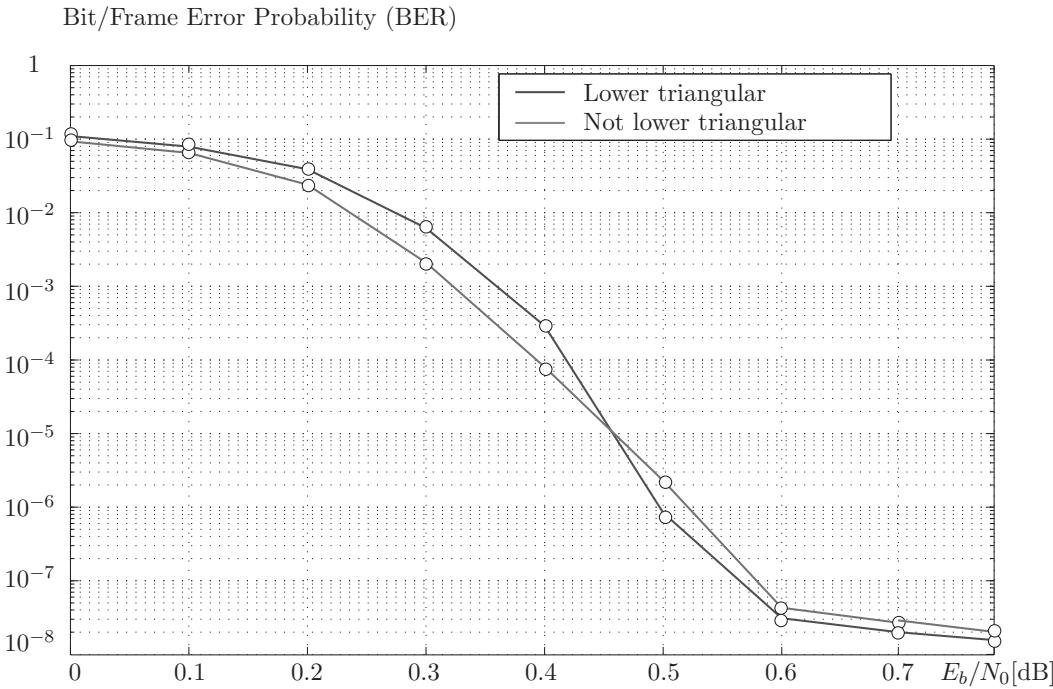


Figure 6.27: Performance results for a triangular and a non-triangular (random) LDPC codes of rate $R = 1/3$ for a block length of 36,993 bits.

As an example, we can construct the matrix

$$\mathbf{A}_0 = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \Rightarrow \mathbf{H}_p = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (6.66)$$

and concatenate it with

$$\mathbf{H}_u = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 \end{bmatrix}. \quad (6.67)$$

The resulting code is an [8,4] extended Hamming code, whose graph is shown in Figure 6.29. This graph can be used for decoding as well as encoding. It may also be used to encode with the earlier BEC algorithm.

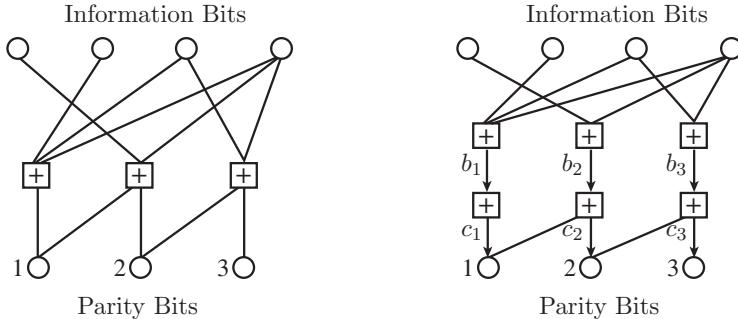


Figure 6.28: Constraint graph for the matrix in (6.62) on the left-hand side, and splitting the code into two parts to apply iterative encoding on the right.

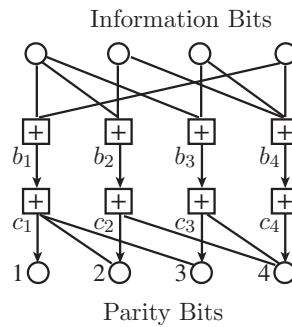


Figure 6.29: Graph of the [8,4] extended Hamming code.

6.6.2 Specialized LDPC Codes

Most special LDPC codes which are easily encodable rely on a triangular structure as well. In this section we will discuss only the RA codes discussed Section 6.5.4. Linear complexity encoding is already obvious if we refer to the code block diagram in Figure 6.24 instead of its Tanner graph.

If we draw out the parity-check matrix of the RA code \mathbf{H}_2 from Figure 6.25, we note that the parity section of \mathbf{H}_2 is diagonal with a single lower diagonal, which allows linear-time encoding via the BEC decoding algorithm, or simple back-substitution.

$$\mathbf{H}_2 = \left[\begin{array}{ccc|ccc|ccc} 1 & & & 1 & 1 & 1 & 1 & 1 & 1 \\ & 1 & & & 1 & 1 & 1 & 1 & 1 \\ & & 1 & & & 1 & 1 & 1 & 1 \\ & & & 1 & & & 1 & 1 & 1 \\ & & & & 1 & 1 & & 1 & 1 \\ \hline 1 & & & 1 & & & & 1 & 1 \\ & 1 & & & 1 & & & & 1 & 1 \\ & & 1 & & & 1 & & & 1 & 1 \\ & & & 1 & & & 1 & & 1 & 1 \\ & & & & 1 & & & & & 1 & 1 \\ \hline 1 & & & 1 & & & & & 1 & 1 \\ & 1 & & & 1 & & & & & 1 & 1 \\ & & 1 & & & 1 & & & & & 1 & 1 \\ & & & 1 & & & 1 & & & & & 1 & 1 \\ & & & & 1 & & & & & & & & 1 & 1 \\ \hline 1 & & & 1 & & & & & & & & & 1 & 1 \\ & 1 & & & 1 & & & & & & & & & 1 & 1 \\ & & 1 & & & 1 & & & & & & & & & 1 & 1 \\ & & & 1 & & & 1 & & & & & & & & & 1 & 1 \end{array} \right]. \quad (6.68)$$

All other codes based on this same kind of construction, including the ARA code discussed in Section 6.5.1, are equally encodable with linear complexity, which is best seen from their turbo code viewpoint.

6.6.3 Approximate Triangularization

In general, when we try to find a generator matrix for an LDPC code via the relation $\mathbf{G}\mathbf{H}^T = 0$, \mathbf{G} will no longer be sparse. Again, suppose a codeword has the structure $\mathbf{x} = [\mathbf{x}_u \mathbf{x}_p]$, where \mathbf{x}_u is the information sequence and \mathbf{x}_p is the parity sequence, and split \mathbf{H} into $[\mathbf{H}_u \mid \mathbf{H}_p]$, giving the equation

$$\mathbf{H}_p \mathbf{x}_p^T = \mathbf{H}_u \mathbf{x}_u^T; \quad (6.69)$$

$$\Rightarrow \mathbf{x}_p^T = \mathbf{H}_p^{-1} \mathbf{H}_u \mathbf{x}_u^T. \quad (6.70)$$

Note also that \mathbf{H}_u and \mathbf{H}_p are square matrices. There are a few methods available to make this computation efficient. Instead of forcing the parity-check matrix to be triangular as in the previous cases, [42] suggest placing the matrix in “approximate lower triangular” form in which the upper right corner is populated with 0 as shown in Figure 6.30.

A given parity-check matrix \mathbf{H} may be placed into this form using only row and column permutations. The distance g is referred to as the “gap.” The gap can be thought of as a measure of the distance from H to a true lower-triangular matrix. We then split \mathbf{x}_p into

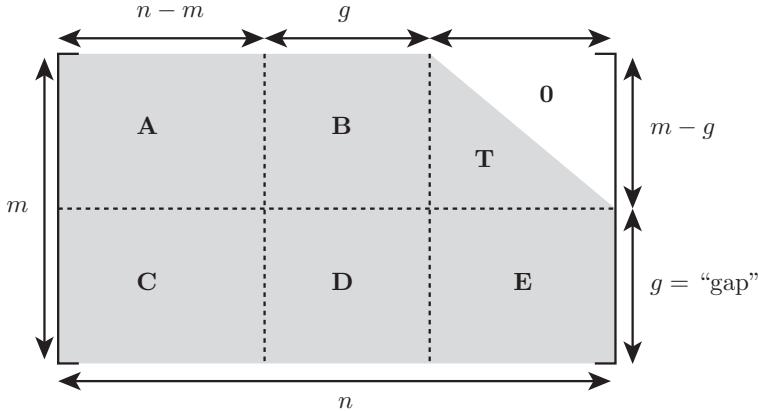


Figure 6.30: Conversion of the parity-check matrix of an LDPC code into approximate lower triangular form.

$[p_1, p_2]$, where p_1 has length g and p_2 has length $m - g$, and multiply \mathbf{H} from the left by

$$\begin{bmatrix} \mathbf{I} & \mathbf{0} \\ -\mathbf{ET}^{-1} & \mathbf{I}, \end{bmatrix}, \quad (6.71)$$

giving

$$\begin{bmatrix} \mathbf{A} & \mathbf{B} & \mathbf{T} \\ \mathbf{C} - \mathbf{ET}^{-1} \mathbf{A} & \mathbf{D} - \mathbf{ET}^{-1} \mathbf{B} & \mathbf{0} \end{bmatrix}. \quad (6.72)$$

The parity-check rule then gives the following equations

$$\begin{aligned} \mathbf{Ax}_u^T + \mathbf{Bp}_1^T + \mathbf{Tp}_2^T &= \mathbf{0}, \\ (\mathbf{C} - \mathbf{ET}^{-1} \mathbf{A}) \mathbf{x}_u^T + (\mathbf{D} - \mathbf{ET}^{-1} \mathbf{B}) \mathbf{p}_1^T &= \mathbf{0}. \end{aligned} \quad (6.73)$$

Now define $\mathbf{P} = \mathbf{D} - \mathbf{ET}^{-1} \mathbf{B}$, and assume \mathbf{P} is non-singular, and we arrive at

$$\begin{aligned} \mathbf{p}_1^T &= \mathbf{P}^{-1} (\mathbf{C} - \mathbf{ET}^{-1} \mathbf{A}) \mathbf{x}_u^T, \\ \mathbf{p}_2^T &= -\mathbf{T}^{-1} (\mathbf{Ax}_u^T + \mathbf{Bp}_1^T). \end{aligned} \quad (6.74)$$

A judicious arrangement of these computations gives a sequence of complexity $O(n)$. Only the computation $\mathbf{P}^{-1} (\mathbf{Cx}_u^T - \mathbf{ET}^{-1} \mathbf{Ax}_u^T)$ has complexity $O(g^2)$, because \mathbf{P}^{-1} is a dense $g \times g$ matrix. The authors of [42] also show that, for sufficiently large n , $g \leq O(\sqrt{n})$ with high probability for a randomly chosen LDPC code. This algorithm therefore also provides encoding with linear complexity in the codelength n . In fact, this method can be used, for example, with the RS-based (2048,1723) code in Section 6.5.3 giving a matrix \mathbf{P} of size only 2×2 .

Appendix 6.A: Factor Graphs

Factor Graphs: Introduction and History

Factor graphs are graphical representations of complex functions, that is, representations of how variables of a global function are grouped together locally. These graphical methods have arisen as a description of the structure of probabilistic networks, where one of their primary purpose is to provide a simple visualization of probabilistic dependencies.

The second purpose of factor graphs is to describe the network of connections among variables by edges between nodes, along which probabilities can be exchanged locally with the purpose of calculating global probabilistic functions. In that sense, the factor graph represents a blueprint for a computational machine that evaluates the global function via local processing.

Factor graphs are bipartite graphs; that is, their nodes fall into just two groups such that there are no connections inside each group, but only between nodes in different groups. In the case of factor graphs, these groups are variable nodes and function nodes.

In 1981, Tanner [45, 46] defined block and convolutional error-correcting codes in terms of bipartite graphs with codeword bit nodes and constraint or subcode nodes, which often represent parity-check constraints. These graphs were used as an effective model for code description, construction, and decoding. Tanner also presented the fundamentals of iterative decoding on graphs.

Wiberg et al. [50] introduced analogous graphs for trellis codes by adding hidden state variables to Tanner’s graphs. This addition extended the use of bipartite graphs to trellises and turbo codes. Wiberg’s work [51] established such bipartite graphs, which he termed “Tanner graphs,” as a model applicable to a wide variety of decoding algorithms, including the Viterbi [27] or min-sum algorithm, and the BCJR [3], forward–backward [4], or APP algorithm, discussed in Chapter 7.

The artificial intelligence community also developed graphical methods of solving probabilistic inference problems, termed *probability propagation*. Pearl [38] presented an algorithm in 1986 under the name *Belief Propagation* in *Bayesian Networks* for use in acyclic or cycle-free graphs and applied it to models of human reasoning. An independent but equivalent algorithm was developed by Lauritzen and Spiegelhalter [35] in 1988.

Belief or probability propagation has application in any field where probabilistic inference is used, hence also in error control decoding. There the probabilistic value of transmitted information symbols needs to be inferred from noisy received symbols. We have encountered probability propagation under the name APP decoding in Chapter 7. Information theorists noticed this connection and have shown that the turbo decoding algorithm [6], which is based on the APP algorithm, is an application of belief propagation to a graph with cycles [37].

In [26], Kschischang et al. introduced factor graph formalism, showing that belief prop-

agation and many algorithms used in digital communications and signal processing are all representations of a more general message-passing algorithm, the sum-product algorithm, operating on factor graphs. This algorithm computes marginals of a global probabilistic function in terms of local functions. The factor graph of a code is a visual expression of this factorization into local probabilistic functions. Aji and McEliece [1] present an equivalent but alternative formulation with their generalized distributive law (GDL). We will discuss the sum–product algorithm in Section 6.6.3 after examining graphical models for probabilistic inference.

Graphical Function Representation

Figure 6.31 shows the Bayesian network and the factor graph for a simple four-state Markov chain, that is, for the probability $P(A, B, C, D) = P(A)P(B|A)P(C|B)P(D|C)$. The nodes represent the variables A, B, C , and D , and the edges between the nodes indicate probabilistic, or functional, dependencies. In a Bayesian network, probabilistic conditioning is indicated by the direction of arrows between variable nodes, while the factor graph takes a more general point of view: A small node, called a *function node*, represents the function whose arguments are the variables connected to it. In Figure 6.31 these functions are $P(A), P(B|A), P(C|B)$, and $P(D|C)$ for our Markov chain example, but in general the functions do not have to be conditional probabilities. We will see that this more general formulation will be very useful in the description of the factor graphs of error control codes.

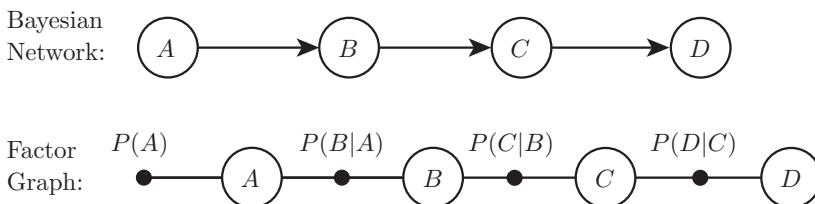


Figure 6.31: Graphical models for a four-state Markov chain.

Both graphical methods visually express global functions in terms of local functions. Edges between nodes indicate all dependencies; there are no hidden dependencies. Belief propagation, as originally introduced by Pearl [38, 39] uses the Bayesian network representation. Graphical decoding via sum-product decoding [45, 50], functionally equivalent to belief propagation, utilizes factor graphs or their bipartite precursors. Both represen-

tations are equivalent; however, the factor graph is more general.

Factor graphs form the computational matrix for evaluating global functions through the communication of local messages. In order to see how this works, let us return to the example of the Markov chain in Figure 6.31. We wish to calculate the marginal probability distribution of D , given by

$$\begin{aligned} P(D) &= \sum_A \sum_B \sum_C P(A, B, C, D) \\ &= \sum_A \sum_B \sum_C P(A)P(B|A)P(C|B)P(D|C), \end{aligned} \quad (6.75)$$

whose “brute-force” evaluation requires $n_A \times n_B \times n_C \times n_D$ summations, where n_A, n_B, n_C , and n_D are, respectively, the numbers of values A, B, C , and D can assume. The factorization in (6.75) allows us to execute the summations locally:

$$P(D) = \sum_C P(D|C) \underbrace{\sum_B P(C|B)}_{\underbrace{P(B)}_{P(C)}} \underbrace{\sum_A P(B|A)P(A)}_{P(A)}. \quad (6.76)$$

This local factored evaluation has only $n_A n_B + n_B n_C + n_C n_D$ summations. These savings are, of course, due to factorization in (6.75). It is obviously computationally more efficient to work with local distributions.

With this, we now wish to calculate the conditional probability $P(C|D = d)$. Node D plays the role of an observed value in our Markov chain, and the conditional, or a posteriori, probability of C is the quantity of interest. Again, using basic probability, we calculate

$$P(C|D = d) = \frac{\sum_{A,B} P(A, B, C, D = d)}{P(D = d)} = K \times P(C, D = d), \quad (6.77)$$

where we need to sum out the intermediate variables A and B . We now factor the global joint distribution into local distributions and obtain

$$\begin{aligned} P(C, D = d) &= \sum_A P(A) \sum_B P(B|A)P(C|B)P(D = d|C) \\ &= \underbrace{P(D = d|C)}_{\mu_{Z \rightarrow C}} \sum_B P(C|B) \underbrace{\sum_A P(B|A) P(A)}_{\underbrace{\mu_{W \rightarrow A}}_{\underbrace{\mu_{X \rightarrow B}}_{\mu_{Y \rightarrow C}}}}. \end{aligned} \quad (6.78)$$

As illustrated in Figure 6.32, Equation (6.78) can be visualized in the factor graph of the Markov chain as passing messages between nodes. The messages $\mu_{\text{function} \rightarrow \text{variable}}$ are

passed from function nodes to variable nodes. For our simple Markov chain example, the variable nodes simply copy the message from the incoming edge to the outgoing edge.

At node C , the incoming messages are multiplied to form $P(C, D = d) = \text{BEL}(C) = \mu_{Y \rightarrow C} \mu_{Z \rightarrow C}$. In an analogous manner, any joint probability can be found by calculating forward (a priori) and backward (a posteriori) messages and multiplying them to produce the belief (BEL) of a variable node.

Note that $\text{BEL}(C)$ is a vector whose size depends on the alphabet size of C . So for binary C , $\text{BEL}(C) = [\text{BEL}(C = -1, D = d), \text{BEL}(C = 1, D = d)]$, for $C \in \{-1, 1\}$. Each of the incoming messages is also a vector, or a list, of probability values, whose length equals the cardinality or size of the message alphabet of the receiving variable node. Thus $\mu_{X \rightarrow B} = P(B) = [P(B = -1), P(B = 1)]$ for $B \in \{-1, 1\}$.

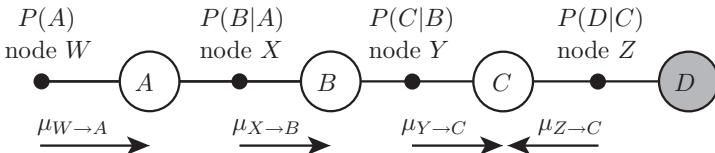


Figure 6.32: Illustration of message passing in a four-state Markov chain.

At each function node, the incoming message is multiplied by that function and a summation over the variable on the incoming side is performed to produce the outgoing marginalized message. For example, at node X the summation over A generates

$$\mu_{X \rightarrow B} = \sum_A P(B|A) \mu_{W \rightarrow A}. \quad (6.79)$$

No summation at W is performed, since there is no incoming message. Likewise, at node Z no summation is performed because node D is observed, and its value is fixed to $D = d$.

In the Markov chain, the variable nodes simply pass messages from the input to the output and back to the input, as they only have two connecting edges per node. One edge serves as pathway for the incoming message, while the other edge provides a path for the outgoing message. The function nodes multiply the incoming message with their function and sum over the variable sending the incoming message.

In a more general network, where the variable or function nodes may have multiple incoming edges (more than two connecting edges to a node), the incoming messages are multiplied at the function node to generate the outgoing messages. We discuss this general algorithm in the next section.

The Sum–Product Algorithm

The sum-product algorithm implements probability calculations as message-passing in a factor graph. It operates on *local distributions*, computing results at each node and passing messages on to connecting nodes. In a general network, there are two types of messages:

1. Function-to-Variable: At the function nodes, incoming variable-to-function messages are multiplied with that function and summed over the variables whose nodes send them, i.e., the node variable is marginalized out to produce the outgoing messages. For example, in Figure 6.32, the message passed from node X to node B is

$$P(B) = \sum_A P(B|A)\mu_{W \rightarrow A}.$$

2. Variable-to-Function Messages: The messages sent from the function nodes are in the form of a priori probability or a posteriori likelihood lists. An example of the former is $\mu_{W \rightarrow A} = P(A)$, and that of the latter is $\mu_{Z \rightarrow C} = P(D = d|C)$. At a variable node, incoming messages are multiplied to generate the outgoing messages.

For an acyclic graph, i.e., for a tree graph which contains no cycles, all nodes need to send and receive messages only once, after which the marginalized probabilities of all variables can be calculated by multiplying all incoming messages at that variable node to form $\text{BEL}(\text{node}) = \prod \mu_{\text{function} \rightarrow \text{variable}}$. We will show later that $\text{BEL}(\text{node})$ is the exact marginalized probability for acyclic graphs, while for cyclic graphs it is only approximate as the algorithm becomes iterative along loops. Furthermore, the order of the message passing, called the *schedule*, is not important for a tree graph.

Let us now study the more complex factor graph of the [8,4] extended Hamming code. This code has the systematic parity-check matrix [32]

$$\mathbf{H}_{[8,4]} = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}. \quad (6.80)$$

The code has 4 information bits, which we associate with variable nodes U_1, \dots, U_4 , and 4 parity bits which are associated with the variable nodes X_5, \dots, X_8 . There are also 8 observed output symbols, nodes Y_1, \dots, Y_8 , which are the noisy received variables U_1, \dots, X_8 . The function nodes V_1, \dots, V_4 are the four parity-check equations described by (6.80), given as boolean truth functions. For example, the function at node V_1 is $U_1 \oplus U_2 \oplus U_3 \oplus X_5 = 0$ and corresponds to the first row of $\mathbf{H}_{[8,4]}$. The complete factor graph of the parity-check matrix representation of this code is shown below in Figure 6.33.

This network has loops, e.g., $U_1 - V_1 - U_2 - V_2 - U_1$, and thus the sum–product algorithm has no well-defined forward–backward schedule, but many possible schedules of passing messages between the nodes. A sensible message passing schedule will involve all nodes

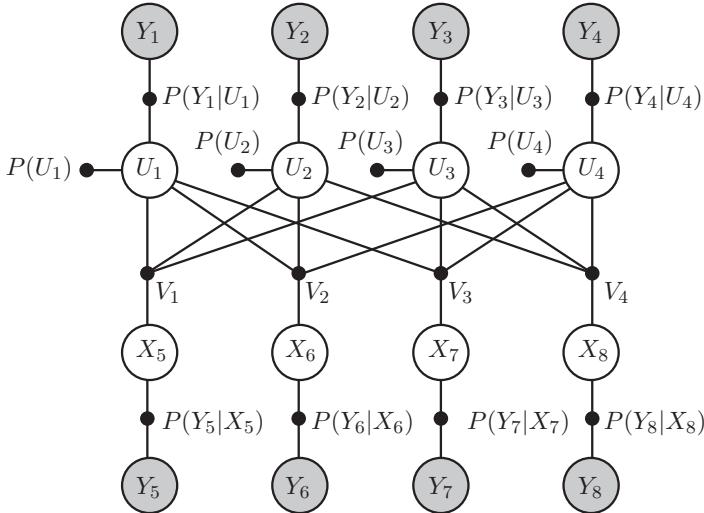


Figure 6.33: [8,4] Hamming code factor graph representation.

which are not observation nodes in a single sweep, called an iteration. Such iterations can then be repeated until a desired result is achieved. The messages are passed exactly according to the rules of the sum-product algorithm. More precisely, iterative probability propagation proceeds according to the following sum-product algorithm:

Step 1: Initialize the observation nodes Y_k to the observed values $Y_k = y_k$.

Step 2: Send the likelihood messages $\mu_{Y_k \rightarrow U_k} = P(Y_k = y_k | U_k)$ from Y_k to U_k and $\mu_{Y_j \rightarrow X_j} = P(Y_j = y_j | X_j)$ from Y_j to X_j . For an AWGN channel, $P(Y_k = y_k | U_k = u) = \frac{1}{\sqrt{N_0}} \exp(-(y_k - u)^2 / N_0)$.

Step 3: Send the a priori probability messages $\mu_{U_k} = P(U_k)$ to the nodes U_k . If no a priori information is available, assume the uniform distribution $P(U_k = 1) = P(U_k = -1) = 0.5$.

Step 4: Send the variable to function messages $\mu_{X_j \rightarrow V_j} = P(Y_j = y_j | X_j)$ from the variable nodes X_j to the function nodes V_j .

Steps 1–4 are executed only once at the beginning of the decoding process and “stored” permanently on the edges. When required, the sent messages are reused during subsequent iterations of the inner nodes. During decoding, the evaluation of the a posteriori probabilities of the nodes U_k now proceeds iteratively via the following steps which involve only the inner nodes in the network, as shown in Figure 6.34:

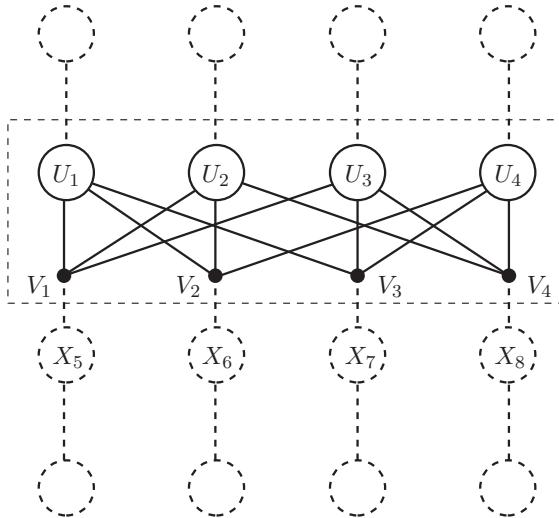


Figure 6.34: Inner nodes participating in the iterative part of the decoding process.

Step 5: Send the messages $\mu_{U_k \rightarrow V_j}$ from the nodes U_k to the nodes V_j with which they are connected. These messages are given by

$$\mu_{U_k \rightarrow V_j} = \mu_{U_k} \mu_{Y_k \rightarrow U_k} \prod_{i \neq j} \mu_{V_i \rightarrow U_k}. \quad (6.81)$$

In the first iteration, no messages from the inner function nodes have been received yet, and $\mu_{U_k \rightarrow V_j} = \mu_{U_k} \mu_{Y_k \rightarrow U_k}$.

Step 6: For each outgoing edge of V_j , multiply the incoming messages and sum over all variables which sent messages to V_j . Send the results as messages $\mu_{V_j \rightarrow U_k}$ from V_j to U_k and X_k . No messages need to be sent back to the nodes X_j as long as we are not interested in their belief probabilities.

Step 7: Calculate the belief of the variables U_k as the product

$$\begin{aligned} \text{BEL}(U_k) &= \mu_{U_k} \mu_{Y_k \rightarrow U_k} \prod_j \mu_{V_j \rightarrow U_k} = P(U_k, Y_1, \dots, Y_8) \\ &\propto P(U_k | Y_1, \dots, Y_8). \end{aligned}$$

Step 8: End the iterations or go to Step 5. The iterations are ended by choosing a *stopping criterion*, such as reaching a desired accuracy ϵ $\text{BEL}(U_{k,i}) - \text{BEL}(U_{k,i-1}) \leq \epsilon, \forall U_k$, or a maximum number of iterations $i = i_{\max}$.

The Factor Graph of Trellises

In this section we forge the connection between factor graphs and the trellis representations of codes discussed earlier in this book. Not only will this lead us to more efficient factor graphs in the sense of decoding efficiency, but it will also establish the equivalence between the APP algorithm discussed in Chapter 5 (Section 5.7) and the sum-product algorithm. This will establish factor graphs as a general and convenient high-level description of trellis codes.

Figure 6.35 shows the trellis of the code used in Chapter 5 to discuss the APP algorithm. In the light of factor graph terminology, we now realize that the states S_1, \dots, S_8 are random variables, and we represent these random variables by variable nodes as shown in the equivalent factor graph representation 6.36. These state nodes are no longer binary random variables as was the case in the factor graph of the [8,4] Hamming code, but have cardinalities corresponding to the size of the state space. The function nodes V_r now express the trellis constraints relating the previous state S_r , the new state S_{r+1} and the transmitted symbol(s) X_r . The function V_r encapsulates the two functions $X_r = \tau(U_r, S_r), S_{r+1} = \sigma(U_r, S_r)$. Note that the information bit U_r is implicit in the trellis representation, determining the upper or lower path emanating from each state. Also, the received symbols Y_r are not shown in the trellis representation. It becomes evident that the factor graph is a higher level view of the trellis code, the details of which are hidden in the function nodes V_j .

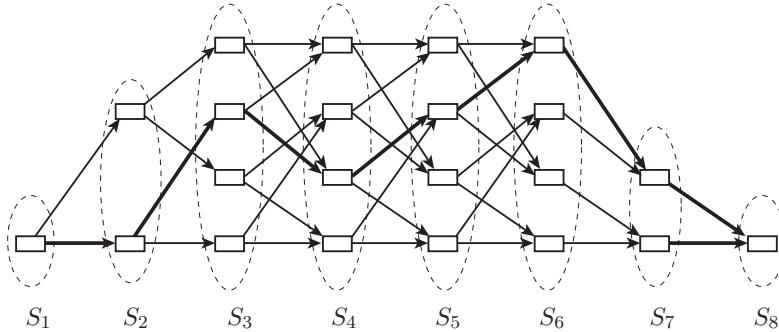


Figure 6.35: Example trellis of the short trellis code from Chapter 5.

We now establish the equivalence of the APP algorithm with the sum-product algorithm. The first three steps in the latter, according to Page 303 are preparatory in nature and identical to the APP algorithm, if a forward-backward update schedule is used. That is, updating starts at the state variable node S_1 and proceeds along the chain to state node S_8 and then back again. The peripheral nodes X_r and U_r send and receive messages as required by this schedule.

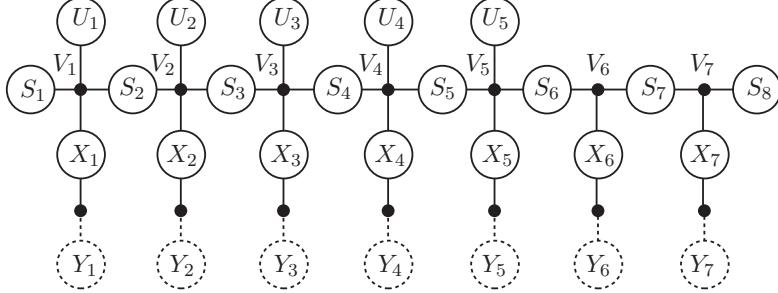


Figure 6.36: Factor graph of the trellis code in Figure 6.35.

Step 4 in the sum–product algorithm generates the messages

$$\mu_{U_r \rightarrow V_r} = P(U_r = u_r), \quad (6.82)$$

$$\mu_{X_r \rightarrow V_r} = P(Y_r = y_r | X_r), \quad (6.83)$$

$$\text{and } \mu_{S_r \rightarrow V_r} = P(S_r, Y_1, \dots, Y_{r-1}) = [\alpha_r(0), \dots, \alpha_r(S-1)], \quad (6.84)$$

from the variable nodes U_r , X_r , and S_r , respectively, to the function node V_r . All of these messages are vectors. For example, for a rate 1/2 binary convolutional code, (6.82) has two entries for $u_r = \{0, 1\}$, (6.83) has four entries for $\{00, 01, 10, 11\}$, and (6.84) has S entries, an α -value for each of the S states. The function node V_r contains the probabilistic function $P(S_{r+1}, X_r | S_r, U_r)$ relating all connecting variable nodes.

At Step 5 in the sum–product algorithm, the incoming “forward” messages are marginalized at the function node V_r over the sending node variables, multiplied, and forwarded to the next state variable node S_{r+1} as

$$\begin{aligned} \alpha_{r+1} &= [\alpha_r(0), \dots, \alpha_r(S-1)] \\ &= \sum_{U_r, S_r} \mu_{S_r \rightarrow V_r} \mu_{X_r \rightarrow V_r} \mu_{U_r \rightarrow V_r} P(S_{r+1} | S_r, U_r, X_r), \end{aligned} \quad (6.85)$$

where

$$\alpha_{r+1}(j) = \sum_i \alpha_r(i) \sum_{U_r} P(Y_r = y_r | X_r) P(U_r) P(S_{r+1} = j, X_r | S_r = i, U_r). \quad (6.86)$$

In the general case, there would also be a sum over x_r , but due to the simple trellis, U_r and S_r uniquely determine X_r . Carrying out the sum over U_r in the above equation, we

obtain the forward recursion for α , equivalent to (5.28):

$$\alpha_{r+1}(j) = \sum_i \alpha_r(i) \gamma_{r+1}(i, j), \quad (6.87)$$

where $\gamma_{r+1}(i, j) = P(S_{r+1} = j, Y_r = y_r | S_r = i)$ as for APP decoding.

Figure 6.37 illustrates the processing that takes place at function node V_r .

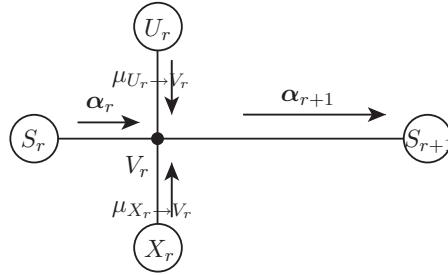


Figure 6.37: Function node processing during the “forward” phase.

Analogously, on the backward sweep of algorithm, the incoming messages are marginalized, multiplied, and forwarded to the previous state variable S_r as

$$\beta_r = [\beta_r(0), \dots, \beta_r(S-1)], \quad (6.88)$$

where

$$\beta_r(j) = \sum_i \beta_{r+1}(i) \sum_{U_r} P(Y_r = y_r | X_r) P(U_r) P(S_{r+1}, X_r | S_r, U_r). \quad (6.89)$$

As before, we can replace the last terms with γ_{r+1} to get

$$\beta_r(j) = \sum_i \beta_{r+1}(i) \gamma_{r+1}(j, i). \quad (6.90)$$

This is the backward recursion for β developed in Chapter 5 for APP decoding (5.29).

According to Step 7, the belief or conditional probability of state S_r is calculated as the product of the incoming messages:

$$P(S_r = i, Y_1 = y_1, \dots, Y_L = y_L) = \text{BEL}(S_r = i) = \alpha_r(i) \beta_r(i). \quad (6.91)$$

From the derivation in Chapter 5 for (5.27), we know that this conditional probability is exact. Beliefs, i.e., conditional probabilities, of the other variables U_r and X_r are calculated completely analogously, resulting in (5.44) and (5.45).

Exactness of the Sum–Product Algorithm for Trees

As we have seen, the sum–product algorithm for a trellis network is equivalent to the APP algorithm of Section 5.7, and thus exact. That is, it generates the exact a posteriori probabilities of the hidden variables. We are now going to show that this is true for any network whose factor graph can be drawn as a tree.

Figure 6.38 shows an edge in a general factor graph, connecting a variable node X with a function node V . The function node V is a function of the variables X, Z_1, \dots, Z_N , and the variable X is an argument in the functions V, W_1, \dots, W_I . The nodes Z_n send the messages $\mu_{Z_n \rightarrow V}$, which represent variable probabilities, in the form of lists to the function node V . At the function node, these incoming messages are multiplied componentwise, marginalized, and forwarded to the variable node X as

$$\mu_{V \rightarrow X} = \sum_{Z_1, \dots, Z_N} f_V(X, Z_1, \dots, Z_N) \prod_{n=1}^N \mu_{Z_n \rightarrow V}. \quad (6.92)$$

From the right-hand side, the function nodes W_i send the messages $\mu_{W_i \rightarrow X}$ to the function node X . These messages can be thought of as conditional probabilities $P(X|\mathcal{W}_i)$, where \mathcal{W}_i represents all the variables “below” the function node W_i . At node X , these incoming messages are multiplied componentwise and forwarded to the function node V as

$$\mu_{X \rightarrow V} = \prod_{i=1}^I \mu_{W_i \rightarrow X}. \quad (6.93)$$

Note that both $\mu_{V \rightarrow X}$ and $\mu_{X \rightarrow V}$ are L_X -valued message lists, where L_X is the number of values the random variable X can assume. Finally, the belief of X is calculated as

$$\text{BEL}(X) = \mu_{V \rightarrow X} \prod_{i=1}^I \mu_{W_i \rightarrow X} = \mu_{V \rightarrow X} \mu_{X \rightarrow V}, \quad (6.94)$$

where the product is again performed componentwise.

We will now show that the belief calculated in (6.94) is the exact joint probability $P(X, \text{observation})$ if the network’s factor graph is a tree, where *observation* is the set of observed random variables. The exact a posteriori probability $P(X|\text{observation})$ is easily derived from the joint probability through normalization. Without loss of generality, assume that the node X of interest is moved to the “top” of the inverted tree as shown in Figure 6.39. Furthermore, all observed nodes Y_k are located at the bottom. Subnetworks below the observed nodes can be ignored, since the observed nodes isolate these lower parts, that is, if $P(QY|S) = P(Q|Y)P(Y|S)$, then observation of $Y = y$ leads to $P(Q|Y = y)P(Y = y|S)$ and Q is irrelevant for S and can be ignored.

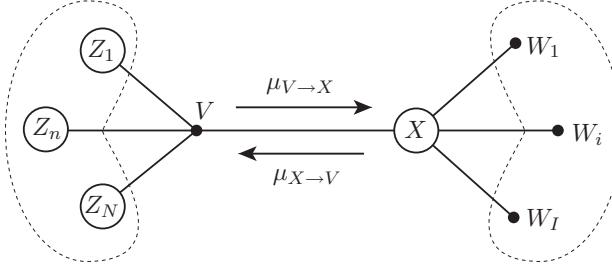


Figure 6.38: General edge in a factor graph.

In the first step of the sum–product algorithm, the observed nodes Y_k send $P(Y_k = y|S_l)$ to their immediate higher variables nodes S_l . The message that S_l forwards is, according to (6.93), the likelihood $\prod_k P(Y_k = y_k|S_l) = P(\text{observation}|S_l)$. S_l is in turn connected to the next higher variable node, denoted by T_r , through the function node $P(S_l|T_r)$ which marginalizes the messages of all connected nodes S_l and generates

$$\begin{aligned} \sum_{S_1, \dots, S_L} & P(S_1|T_r)P(\text{observation}|S_1) \cdots P(S_L|T_r)P(\text{observation}|S_L) \\ &= P(\text{observation}|T_r). \end{aligned} \quad (6.95)$$

The T -level is now formally identical to the S -level, and, for a tree, this process can be continued recursively up the branches to the function node W_i , which passes the message $\prod_r \sum_{T_r} P(\text{observation}, T_r|X)$ to node X .

On the other side of the tree, the a priori side, the process starts with the free nodes U_1, \dots, U_L who pass a priori information to node R_h , which forwards the message

$$\sum_{U_1, \dots, U_L} P(U_1)P(R_h|U_1) \cdots P(U_L)P(R_h|U_L) = P(R_h) \quad (6.96)$$

to the function node $P(Z_n|R_h)$, which marginalizes out all connected R_h . Again, in a tree this process continues up to function node V_j which passes the message $\mu_{V_j \rightarrow X} = \sum_{Z_1 \dots Z_N} P(X|Z_1, \dots, Z_N) \prod_n P(Z_n)$ to node X . Node X in turn multiplies the incoming messages and calculates

$$\text{BEL}(X) = \prod_i \mu_{W_i \rightarrow X} \prod_j \mu_{V_j \rightarrow X}, \quad (6.97)$$

$$\begin{aligned}
\text{BEL}(X) &= \prod_i \prod_{r=1}^R \sum_{T_r} P(\text{observation}, T_r | X) \\
&\quad \times \prod_j \sum_{Z_1} \cdots \sum_{Z_N} P(X | Z_1, \dots, Z_N) \prod_{n=1}^N P(Z_n) \\
&= \prod_i \prod_j \prod_{r=1}^R \sum_{T_r} \sum_{Z_1} \cdots \sum_{Z_N} P(\text{observation}, T_r, Z_1, \dots, Z_N, X) \\
&= P(X, \text{observation}),
\end{aligned}$$

which, after normalization, yields the desired conditional probability.

Note that we are not restricted to having observation nodes only on one side of the tree. The sum-product algorithm also works exactly with observation nodes on the a priori side of the tree in addition to the observed nodes Y_k . Consider a set of observations $R_1 = r_1, R_2 = r_2$ at the R_h level, which we call observation O . The remaining R_h nodes are not observed. The message forwarded to node Z_n becomes

$$\begin{aligned}
&\sum_{R_3 \cdots R_H} P(Z_n | R_1 = r_1) P(Z_n | R_2 = r_2) P(Z_n | R_3) P(R_3) \cdots P(Z_n | R_H) P(R_H) \\
&= P(Z_n | R_1 = r_1, R_2 = r_2) = P(Z_n | O),
\end{aligned} \tag{6.98}$$

and the conditioning on observation O simply proceeds up to node X .

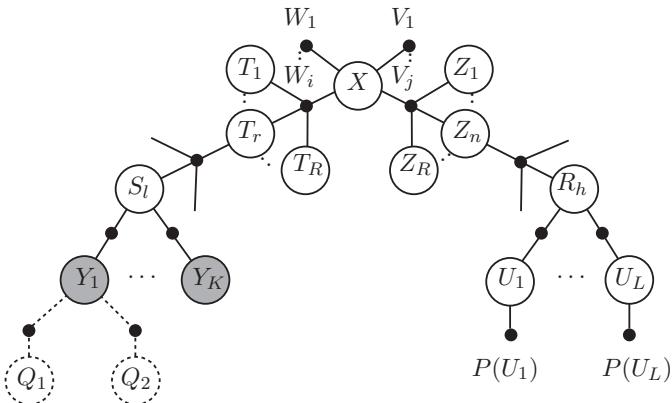


Figure 6.39: Structure of a tree factor graph.

The sum-product algorithm does not produce exact a posteriori probabilities for loopy networks, i.e., networks with cycles. Exact probability propagation in a loopy network is known to be NP-complete [12]. Nonetheless, the sum-product algorithm is used successfully in loopy networks to perform error control decoding as shown earlier. This has proven particularly successful in decoding turbo codes. The algorithm has also been applied successfully to interference cancellation and multiple access joint detection, both usually described by loopy networks.

The passing of messages can occur in any arbitrary order. This does have an effect on the result, but very little is currently known about how to choose an efficient update schedule. In a tree, of course, the order of the updates is irrelevant. The convergence behavior of this algorithm in loopy networks is also little understood and the topic of much current research interest.

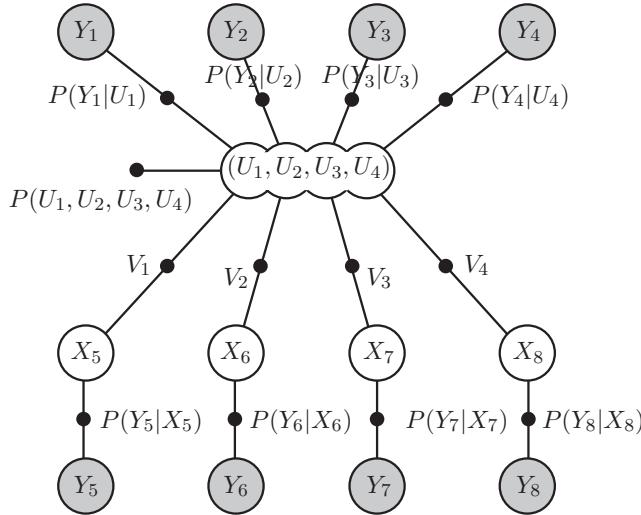


Figure 6.40: [8,4] Hamming code factor graph representation as a tree.

Note that any given system has a large number of possible factor graphs, and one can always generate a tree graph for any system through node contraction. That is, nodes which form a cycle are simply lumped together and treated as a single node. This is illustrated for the factor graph of the [8,4] Hamming code in Figure 6.40, where we have contracted the four information nodes into a single 16-valued node. Of course, decoding now becomes more difficult, not easier as it might seem, since passing messages through the large node is in effect the same as an exhaustive search over all 16 codewords, i.e.,

more complex than the simple binary factor graph of Figure 6.33.

If the variables are binary, i.e., two-valued with $X \in [0, 1]$, we can simplify the messages. Instead of communicating the probabilities $[p_0, p_1]$, we may use $p_0 + p_1 = 1$ to simplify our representation into a single parameter. A commonly used representation is the log-likelihood ratio (LLR) $\lambda = \ln(p_0/p_1)$. The messages change as follows when using LLR representations.

LLR Variable Node Messages:

Referring to Figure 6.38, variable node X receives LLR messages from the function nodes W_i in the form

$$\mu_{W_i \rightarrow X} = \ln(p_0/p_1) = \lambda_{W_i}.$$

For two nodes, the probability that $X = 0$ equals $\frac{p_0 q_0}{p_0 q_0 + p_1 q_1}$, where p_0, p_1 are the probabilities of W_1 , and q_0, q_1 those of W_2 . The fraction expresses the conditional probability $P(X = 0)$, given that both incoming messages refer to the same variable X . However, the message format is now $\lambda_X = \ln(P(X = 0)/P(X = 1))$, and hence

$$\begin{aligned} P(X = 0) &= P(X = 0|O_{B,C}) = \frac{p_0 q_0}{p_0 q_0 + p_1 q_1}, \\ P(X = 1) &= P(X = 1|O_{B,C}) = \frac{p_1 q_1}{p_0 q_0 + p_1 q_1}, \\ \lambda_X &= \ln\left(\frac{p_0 q_0}{p_1 q_1}\right) = \ln(p_0/p_1) + \ln(q_0/q_1). \end{aligned} \quad (6.99)$$

The outgoing message from node X to function node V is now

$$\mu_{X \rightarrow V} = \ln(x_0/x_1) = \ln(p_0/p_1) + \ln(q_0/q_1) = \mu_{W_1 \rightarrow X} + \mu_{W_2 \rightarrow X}, \quad (6.100)$$

i.e., the LLR messages are simply added. This can easily be extended to more than two incoming messages, and we obtain the summation of LLR rule used in (6.7).

LLR Check Node Messages:

Assume now the function at node V represent a parity check of all incoming message. Node V receives LLR messages from variable nodes Z_1 and Z_2 of the form

$$\mu_{Z_1 \rightarrow V} = \ln(p_0/p_1) = \lambda_{z_1}, \quad \mu_{Z_2 \rightarrow V} = \ln(q_0/q_1) = \lambda_{z_2}. \quad (6.101)$$

The node enforces the parity-check constraint $f(x, z_1, z_2) = [X \oplus Z_1 \oplus Z_2 = 0]$, and we calculate that $z_0/z_1 = \frac{p_0 q_0 + p_1 q_1}{p_0 q_1 + p_1 q_0}$, and the outgoing LLR message from check node V to variable X is

$$\mu_{V \rightarrow X} = \ln(x_0/x_1) = \ln\left(\frac{p_0 q_0 + p_1 q_1}{p_0 q_1 + p_1 q_0}\right). \quad (6.102)$$

However, we want to be able to express the outgoing LLR message in terms of the incoming LLR messages λ_{z_1} and λ_{z_2} . This requires some trigonometric manipulations. Noting that $p_0/p_1 = e^{\lambda_{z_1}}$ and $q_0/q_1 = e^{\lambda_{z_2}}$ and dividing both x_0 and x_1 by $p_1 q_1$, we obtain

$$\mu_{V \rightarrow X} = \ln \left(\frac{e^{\lambda_{z_1}} e^{\lambda_{z_2}} + 1}{e^{\lambda_{z_1}} + e^{\lambda_{z_2}}} \right). \quad (6.103)$$

Factoring out $e^{\frac{\lambda_{z_1}}{2}} e^{\frac{\lambda_{z_2}}{2}}$ from numerator and denominator gives

$$\mu_{V \rightarrow X} = \ln \left(\frac{\cosh\left(\frac{\lambda_{z_1} + \lambda_{z_2}}{2}\right)}{\cosh\left(\frac{\lambda_{z_1} - \lambda_{z_2}}{2}\right)} \right). \quad (6.104)$$

Using a hyperbolic expansion for $\cosh(x \pm y)$ and dividing numerator and denominator by the common term $\cosh\left(\frac{\lambda_{z_1}}{2}\right) \cosh\left(\frac{\lambda_{z_2}}{2}\right)$ we obtain

$$\mu_{V \rightarrow X} = \ln \left(\frac{1 + \tanh\left(\frac{\lambda_{z_1}}{2}\right) \tanh\left(\frac{\lambda_{z_2}}{2}\right)}{1 - \tanh\left(\frac{\lambda_{z_1}}{2}\right) \tanh\left(\frac{\lambda_{z_2}}{2}\right)} \right), \quad (6.105)$$

as used in [8]. With the relation $\tanh^{-1}(z) = \frac{1}{2} \ln\left(\frac{1+z}{1-z}\right)$, we further obtain

$$\mu_{V \rightarrow X} = 2 \tanh^{-1} \left(\tanh\left(\frac{\lambda_{z_1}}{2}\right) \tanh\left(\frac{\lambda_{z_2}}{2}\right) \right). \quad (6.106)$$

Again, this derivation can be extended to multiple input messages and we obtain the check node message function (6.6).

The check node LLR message can be approximated by using $\ln(\cosh(x)) \approx |x| - \ln(2)$ for $x \gg 1$. Thus,

$$\mu_{V \rightarrow X} \approx \left| \frac{\lambda_{z_1} + \lambda_{z_2}}{2} \right| - \left| \frac{\lambda_{z_1} - \lambda_{z_2}}{2} \right| = \text{sgn}(\lambda_{z_1}) \text{sgn}(\lambda_{z_2}) \min(|\lambda_{z_1}|, |\lambda_{z_2}|), \quad (6.107)$$

the approximation discussed in (6.8).

To summarize: For variable nodes the outgoing LLR message is simply the sum of all incoming LLR messages. If variable node X has degree i , the message it sends to function node V will be

$$\mu_{X \rightarrow V} = \sum_{\substack{j=1 \\ (j \neq V)}}^i \lambda_j, \quad (6.108)$$

For parity-check nodes the incoming LLR messages multiply together through the tanh transformation. Check node V of degree i sends to variable node X the message

$$\mu_{V \rightarrow X} = 2 \tanh^{-1} \left(\prod_{\substack{j=1 \\ (j \neq X)}}^i \tanh \left(\frac{\lambda_j}{2} \right) \right). \quad (6.109)$$

Binary factor graphs thus allow for a significant simplification of the computations required at the nodes. Nonetheless, note that the outgoing messages are true LLR values given the incoming messages; that is, if the incoming messages are exact, so are the outgoing messages. In this sense the operations (6.108) and (6.109) are *locally optimal*.

It is interesting to muse that the APP algorithm used in iterative turbo decoding can itself be viewed as an instance of belief propagation between nodes which represent individual trellis sections. The analogy between LDPC and turbo codes can further be strengthened using Forney's normal graphs [17], which leads to the comparison illustrated in Figure 6.41, showing the factor graphs for LDPC codes and standard turbo codes. Both codes (and most others) can be represented as graph structures joined by a randomly selected permutation Π . This graphical representation also exposes the turbo decoding algorithm as an instance of message-passing decoding on code graphs.

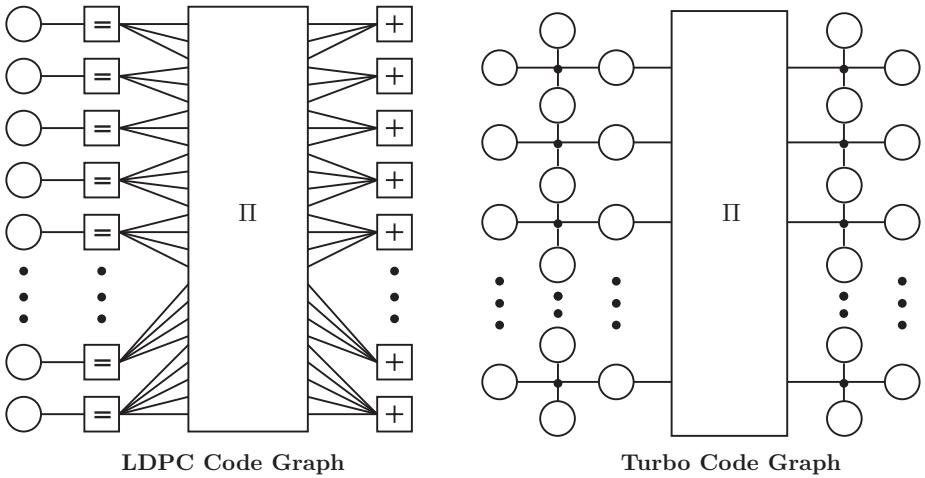


Figure 6.41: Factor graphs for LDPC and turbo codes. The nodes in the turbo code graph are the trellis sections of the constituent component codes—see Chapter 8.

Bibliography

- [1] S.M. Aji and R.J. McEliece, “The generalized distributive law,” *IEEE Trans. Inform. Theory*, pp. 325–343, March 2000.
- [2] A. Anastasopoulos, “A comparison between the sum-product and the min-sum iterative detection algorithms based on density evolution,” *IEEE Global Telecommunications Conference 2001*, GLOBECOM ’01, Vol. 2, pp. 1021–1025, Nov. 25–29, 2001.
- [3] L.R. Bahl, J. Cocke, F. Jelinek, and J. Raviv, “Optimal decoding of linear codes for minimizing symbol error rate,” *IEEE Trans. Inform. Theory*, pp. 284–287, March 1974.
- [4] L.E. Baum and T. Petrie, “Statistical inference for probabilistic functions of finite-state Markov chains,” *Ann. Math. Stat.*, vol. 37, pp. 1559–1563, 1966.
- [5] L. Bazzi, T.J. Richardson, and R. Urbanke, “Exact thresholds and optimal codes for the binary symmetric channel and Gallager’s decoding algorithm A,” *IEEE Trans. Inform. Theory*, vol. 50, no. 9, pp. 2010–2021, 2004.
- [6] C. Berrou and A. Glavieux, “Near optimum error correcting coding and decoding: Turbo-codes,” *IEEE Trans. Commun.*, vol. 44, no. 10, pp. 1261–1271, Oct. 1996.
- [7] A.J. Blanksby and C.J. Howland, “A 690-mW 1-Gb/s 1024-b, rate-1/2 low-density parity-check code decoder,” *IEEE Journal Solid-State Circuits*, vol. 37, no. 2, pp. 404–412, March 2002.
- [8] G. Battail, M.C. Decouvelaere and P. Godlewski, “Replication decoding,” *IEEE Trans. Inform. Theory*, vol. IT-25, pp. 332–345, May 1979.
- [9] S. Y. Chung, G. D. Forney, 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. Lett.*, vol. COMM-5, no. 2, pp. 58–60, Feb. 2001.
- [10] D. Divsalar, S. Dolinar, J. Thorpe, and C. Jones, “Constructing LDPC codes from simple loop-free encoding modules,” *IEEE J. Select. Areas Commun.*, vol. 27, no. 6, pp. 876–888, 2009.
- [11] S.-Y. Chung, T.J. Richardson, and R. Urbanke, “Analysis of sum-product decoding of low-density parity-check codes using a Gaussian approximation,” *IEEE Trans. Inform. Theory*, pp. 657–670, Feb. 2001.

- [12] G.F. Cooper, "The computational complexity of probabilistic inference using Bayesian belief networks," *Artificial Intell.*, vol. 42, pp. 393–405, 1990.
- [13] T.M. Cover and J.A. Thomas, *Elements of Information Theory*, Wiley Series in Telecommunications, John Wiley & Sons, New York, 1991.
- [14] C. Di, D. Proietti, E. Telatar, T. Richardson, and R. Urbanke, "Finite length analysis of low-density parity-check codes on the binary erasure channel," *IEEE Trans. Inform. Theory*, vol. 48, pp. 1570–1579, June 2002.
- [15] D. Divsalar, H. Jin and R. J. McEliece, "Coding theorems for ‘turbo-like’ codes," *Proceedings of the 1998 Allerton Conference*, pp. 928–936, Oct. 1998.
- [16] I. Djurdjevic, J. Xu, K. Abdel-Ghaffar, and S. Lin, "A class of low-density parity-check codes constructed based on Reed–Solomon codes with two information symbols," *IEEE Commun. Lett.*, vol. 7, no. 7, pp. 317–319, 2003.
- [17] G.D. Forney, "Codes on graphs: normal realizations," *IEEE Trans. Inform. Theory*, pp. 520–548, Feb. 2001.
- [18] R. G. Gallager, "Low-density parity-check codes," *IRE Trans. on Inform. Theory*, pp. 21–28, vol. 8, No. 1, Jan. 1962.
- [19] R.G. Gallager, *Low-Density Parity-Check Codes*, MIT Press, Cambridge, MA, 1963.
- [20] D. Haley, A. Grant, and J. Buetefuer, "Iterative encoding of low-density parity-check codes," *Proc. IEEE Globecom 2002*, Oct. 2002.
- [21] D. Haley, C. Winstead, A. Grant, V. Gaudet, and C. Schlegel, "An analog/digital mode-switching LDPC Codec," Proceedings of the ISCAS 2005, Kobe, Japan.
- [22] S. Howard, S. Zeinoddin, C. Schlegel, "EXIT Analysis to optimize irregular low-maximum-variable-degree LDPCs of rate 0.5," HCD internal report, July 2004.
- [23] S. Howard, R. Hang, V. Gaudet, C. Schlegel, S. Bates, "Degree-matched check node approximation to sum-product decoding of LDPCs," *Proc. 2005 IEEE International Symposium on Information Theory*, ISIT 2005, Adelaide, Australia, Sept. 4–9, 2005.
- [24] X-Y. Hu, E. Eleftheriou, and D.M. Arnold, "Regular and irregular progressive edge-growth tanner graphs," *IEEE Trans. Information Theory*, vol. 51, no. 1, pp. 386–398, 2005.
- [25] J. Jin, A. Khandekar and R. J. McEliece, "Irregular repeat-accumulate codes," *Proceedings of the Second International Conference on Turbo Codes*, pp. 125-127, Sept. 2000.
- [26] F.R. Kschischang, B.J. Frey, and H.-A. Loeliger, "Factor graphs and the sum-product algorithm," *IEEE Trans. Inform. Theory*, Feb. 2001.
- [27] S. Lin and D. Costello, Jr., *Error Control Coding*, 2nd edition, Prentice Hall, Englewood Cliffs, 2004.
- [28] M. Luby, M. Mitzenmacher, A. Shokrollahi, and D. Spielman, "Improved low-density parity-check codes using irregular graphs," *IEEE Trans. Inform. Theory*, vol. 47, no. 2, pp. 585–598, 2001.

- [29] M. Luby, M. Mitzenmacher, A. Shokrollahi, and D. Spielman, "Analysis of low density codes and improved designs using irregular graphs," *Proceedings 30th ACM (Association for Computing Machinery) STOC (Symposium on Theory of Computing)*, May 23–26, 1998.
- [30] M. Luby, M. Mitzenmacher, A. Shokrollahi, D. Spielman, and V. Stemann, "Practical loss-resilient codes," *Proc. 29th Annual ACM Symp. Theory of Computing*, pp. 150–159, 1997.
- [31] D.J.C. MacKay, *Information Theory, Inference, and Learning Algorithms*, Cambridge University Press, ISBN 0-521-64298-1, 2003.
- [32] F.J. MacWilliams and N.J.A. Sloane, *The Theory of Error-Correcting Codes*, Vol. 16, North-Holland Mathematical Library, North-Holland, Amsterdam, 1988.
- [33] D. J. C. MacKay and R. M. Neal, "Near Shannon limit performance of low density parity check codes," *IEE Electron. Lett.*, vol. 32, no. 18, pp. 1645–1646, Aug. 1996.
- [34] D. J. C. MacKay, "Good error-correcting codes based on very sparse matrices," *IEEE Trans. Inform. Theory*, vol IT-45, no. 2, pp. 399–431, March 1999.
- [35] S.L. Lauritzen and D.J. Spiegelhalter, "Local computations with probabilities on graphical structures and their application to expert systems," *J. Royal Stat. Soc. B*, vol. 50, pp. 157–224, 1988.
- [36] J. Lu and J.M.F. Moura, "Partition-and-shift LDPC codes," *IEEE Trans. Magnetics*, vol. 41, no. 10, pp. 2977–2979, Oct. 2005.
- [37] R.J. McEliece, D.J.C. MacKay and J.-F. Cheng, "Turbo decoding as an instance of Pearl's 'belief propagation' algorithm," *IEEE J. Select. Areas Commun.*, vol. SAC-16, pp. 140–152, Feb. 1998.
- [38] J. Pearl, "Fusion, propagation, and structuring in belief networks," *Artificial Intell.*, vol. 29, pp. 241–288, 1986.
- [39] J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*, Morgan Kaufmann Publishers, Burlington, MA 1988.
- [40] T.J. Richardson, M. Shokrollahi, and R. Urbanke, "Design of capacity-approaching irregular low-density parity-check codes," *IEEE Trans. Inform. Theory*, pp. 619–637, Feb. 2001.
- [41] T.J. Richardson and R. Urbanke, "The capacity of low-density parity-check codes under message-passing decoding," *IEEE Trans. Inform. Theory*, pp. 599–618, Feb. 2001.
- [42] T.J. Richardson and R. Urbanke, "Efficient encoding of low-density parity-check codes," *IEEE Trans. Inform. Theory*, pp. 638–656, Feb. 2001.
- [43] C. Schlegel and L. Perez, *Trellis and Turbo Coding*, IEEE/Wiley 2004, Hoboken, NJ, ISBN 0-471-22755-2.
- [44] J. Sun and O. Takeshita, "Interleavers for turbo codes using permuation polynomials over integer rings," *IEEE Trans. Inform. Theory*, vol. 51, no. 1, Jan. 2005.
- [45] R. M. Tanner, "A recursive approach to low complexity codes," *IEEE Trans. Inform. Theory*, vol. 74, no. 2, pp. 533–547, Sep. 1981.

- [46] R.M. Tanner, Patent No. 4,295,218, "Error-correcting coding system," Oct. 13, 1981.
- [47] S. ten Brink, "Convergence behavior of iteratively decoded parallel concatenated codes," *IEEE Trans. Commun.*, vol. 49, no. 10, pp. 1627–1737, Oct. 2001.
- [48] T. Tian, C.R. Jones, J.D. Villasenor, and R.D. Wesel, "Selective avoidance of cycles in irregular LDPC code construction," *IEEE Trans. Commun.*, vol. 52, no. 8, pp. 1242–1247, Aug. 2004.
- [49] N. Wiberg, H.-A. Loeliger, and R. Kötter, "Codes and iterative decoding on general graphs," *Eur. Trans. Telecommun.*, pp. 513–525, Sept./Oct. 1995.
- [50] N. Wiberg, H.-A. Loeliger and R. Koetter, "Codes and iterative decoding on general graphs," *Eur. Trans. Telecommun.*, vol. 6, pp. 513–525, Sept./Oct. 1995.
- [51] N. Wiberg, *Codes and Decoding on General Graphs*, PhD thesis, Linköping University, Linköping, Sweden, 1996.
- [52] M. Yang, W. E. Ryan, and Y. Li, "Design of efficiently encodable moderate-length high-rate irregular LDPC codes," *IEEE Trans. Commun.*, vol. 54, no. 2, pp. 564–571, April 2004.
- [53] J. Xu, L. Chen, L. Zheng, L. Lan, and S. Lin, "Construction of low-density parity-check codes by superposition," *IEEE Trans. Commun.*, vol. 53, no. 2, pp. 243–251, Feb. 2005.
- [54] H. Tang, J. Xu, and S. Lin, "Codes on finite geometries," *IEEE Trans. Inform. Theory*, vol. 51, no. 2, pp. 572–596, Feb. 2005.

Chapter 7

Error Floors

7.1 The Error Floor Problem

As observed with the original turbo codes, LDPC codes, primarily those which are constructed randomly, suffer from an error floor. The error floor is the region of SNR where the slope of the bit error curves exhibits to relatively shallow grade as a function of the SNR. For large codes, the steep bit-error rate (BER) function near the code's threshold quite abruptly changes to flat behavior. For moderate to large-sized LDPC codes, this error floor typically sets in around bit error rates of $P_b = 10^{-5}$ or 10^{-6} ; see Figure 7.1. In the error floor region an increase of SNR has only moderate effect in lowering the BER, and for many practical purposes, this onset of the floor therefore also signifies the error correction limit of the code. There are many low-BER applications, however, that require very low *frame error rates*, and this error floor is a nuisance and must be controlled.

In this chapter we discuss analytical and experimental approaches to understand and control the error floor. While the causes of the error floor between LDPC and turbo codes are somewhat different, the mechanisms to lower it both invariably start with a careful design of the interleaver, and, in the case of the LDPC codes, a proper design of the decoding algorithm, especially its numerical operations. In both cases we find that certain subgraphs of the graph of the interconnect network of the code are responsible for the error floor, and designing this network carefully is therefore a primary approach to controlling the error floor.

Due to the importance of error floors in low-BER applications, several authors have studied the error floor phenomenon since the early 2000's [8, 7, 9, 12, 5, 1, 6], and modifications of the decoding algorithm have been studied to alleviate the problem and lower the error floor, specifically targeting absorbing sets [4, 12, 5, 9, 6], which have been implicated as main mechanisms for the error floor as discussed below. As we will show, the onset of the error floor on Gaussian channels is very strongly related to the behavior of

the algorithm on binary symmetric channels, and the dynamics of the absorbing sets fully explain why and when the error floor becomes dominant in a given code.

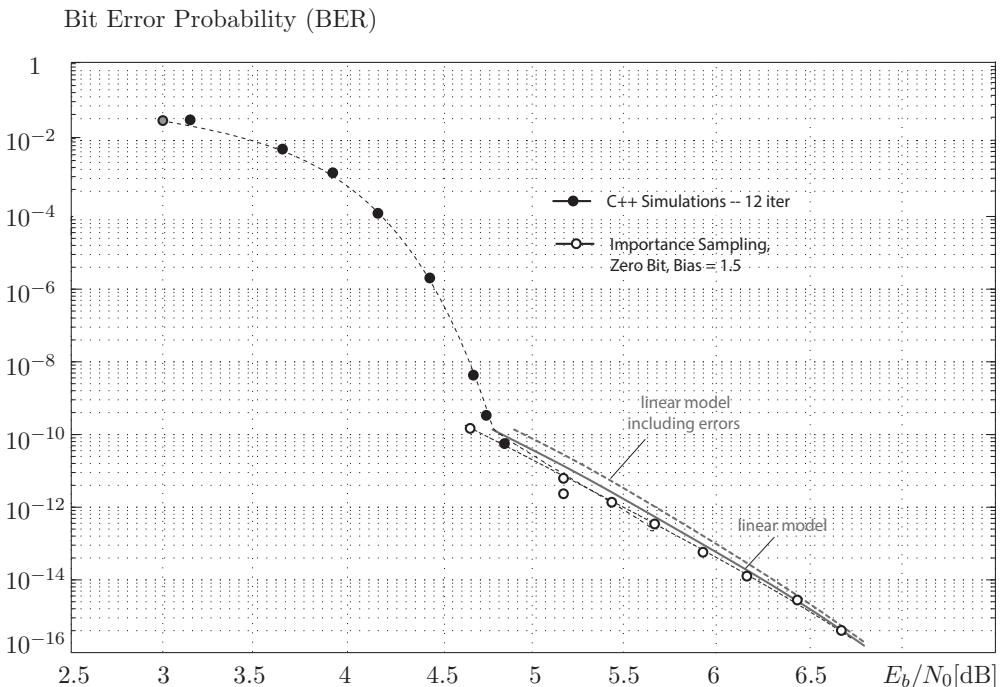


Figure 7.1: Error floor phenomenon illustrated with the IEEE 802.3an LDPC from Section 6.5.3, illustrating the onset of the error floor at about $P_b = 10^{-10}$.

We also will take a computational look at the problem of the error floor and investigate how number representations, number ranges, and mathematical operations affect the error floor in practical decoder settings where computational resources are a key concern. We will see that the error floor depends strongly on the number representation used for the internal log-likelihood messages circulated in the decoder, and we show that the growth rate of the error patterns in the absorbing set can be balanced by the growth of the LLRs external to the absorbing set and that the manifestation of the error floor can be

delayed significantly or completely avoided. This makes the error floor, at least partially, a computation phenomenon.

This effect can be quantified both analytically and computationally using importance sampling guided simulations, which we will discuss later in this chapter. We verify the hypothesis that, quite unlike in the case of binary erasure or binary symmetric channels, the error floor phenomenon for LDPC codes on AWGN channels can essentially be classified as an “artifact” of inexact decoding, and not an inherent weakness of the code, and, in the end, the only solid cause for an error floor due to the code itself are its low-weight codewords. The main contribution of coding theory to confronting the error floor is therefore a good code design which strives to generate an adequate minimum distance in the code. We begin our discussion with a look at the binary-erasure channel mechanisms, but wish to caution the reader that the conclusion from this discourse have to be used with care and that the results are not universally generalizable.

We have already discussed the concept of the *stopping set* in conjunction with decoding on the binary erasure channel. These stopping sets are fixed points of the decoding algorithm and are error patterns that cannot be corrected. The concept of stopping sets can be extended to other channel models, and we first discuss the so-called *absorption sets* which are responsible for the persistent error patterns of the decoder for signaling on the binary symmetric channel of Section 6.4.3. The are defined as follows:

Definition 7.1 An absorption set A is a set of variable nodes, such that the majority of each variable node’s all neighboring check nodes are connected to A an even number of times.

Figure 7.2 illustrates an absorption set in our example LDPC code from Figure 6.3.

We can now show that the Gallager algorithms from Section 6.4.3 will fail to correct an absorption set, and that the decoder gets trapped in such a set. Assume that the black nodes in Figure 7.2 are in error, e.g., they are logical one’s, while all other nodes are logical zeros. In Step 3 of the algorithm in Section 6.4.3, the first four neighbor nodes from the left will return logical one’s to the absorption set, while the three neighbor nodes to the right return zero’s. However, as is easily checked, at each variable node in the absorption set, the *majority* of the incoming messages are logical one’s, and the nodes will therefore retain their value of one. This cycle now continues and a stable error pattern is established.

This is, however, where the similarities between absorption and stopping sets end. For example, unlike in the case of the BEC, the final absorption set in a decoding cycle does not necessarily need to be a subset of the initial error set from the BSC channel. Very little work on the analysis of LDPC codes over the BSC exists, since the LDPC codes do

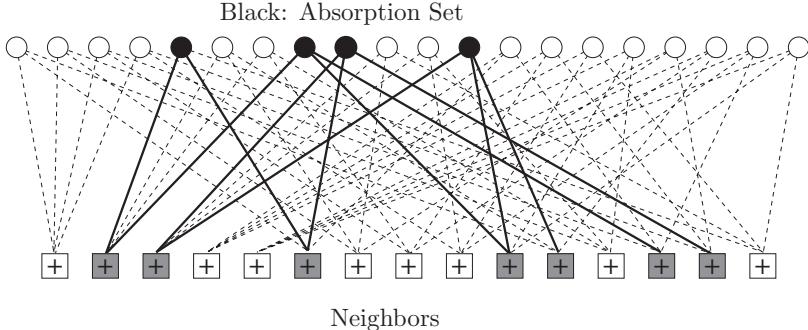


Figure 7.2: Absorption set of size four nodes in an example LDPC code. Note that four of the check nodes that are neighbors to the absorption set are doubly connected to the set, and three nodes are singly connected—see definition above.

not perform particularly well on the BSC. However, absorption sets are important to us since they are closely linked to the failure mechanism of the codes on AWGN channels, the most important channel model for practical coding applications.

Before we delve deeper into the importance of absorption sets, let us recall that the density and probability evolution algorithms discussed in the last chapter depended on the assumption of independent signals along a tree-like network as in Figure 6.7. If the code's Tanner graph has loops, then nodes at level l will have appeared previously a number of levels lower in the tree. In fact, due to the bipartite nature of the Tanner graph, only cycles of even length exist, and cycles of length 4 are the shortest possible cycles since we disallow parallel edges between variable and check nodes.

Since short cycles in the graph of the code cause dependencies among the message that are assumed independent, they are also implicated with the error floor. Avoiding such short cycles has therefore become a primary design objective and proven quite effective. The shortest cycle in the Tanner graph of a code is called the *girth* of the Tanner graph, or simply of the code.

However, simply increasing the girth of a code has limited effect on the error floor, and it is not the magical solution to low error floors, in particular since creating large girths require very large code sizes. Also, it is not just cycles which play a role, but other closely connected subgraphs. This is where we return to the absorption sets, noting that each absorption set contains at least one cycle, which can be seen by the following simple argument: Depart at an arbitrary node in the absorption set \mathcal{A} , travel to a check node that is connected to the set an even number of times, and return to \mathcal{A} . Repeat this process

until you return to the original node and complete a cycle. Before the cycle is completed, it is always possible to return to a check node with an even number of connections since the nodes in \mathcal{A} must have a *majority* of connections to such neighbors.

Figure 7.3 illustrates the *dominant* absorption set in the IEEE 802.3an code from Section 6.5.3, and we have isolated the unsatisfied, singly connected neighboring check nodes since they play a special role in the dynamics of the set under message passing decoding. Note also that only the check and variable nodes involved in the absorption sets are drawn. The nomenclature used for absorption sets is typically using two parameters a and b : a indicates the number of variable nodes in the set, and b indicates the number of connections to unsatisfied check nodes. In this terminology the set in Figure 7.3 is an (8,8) absorption set.

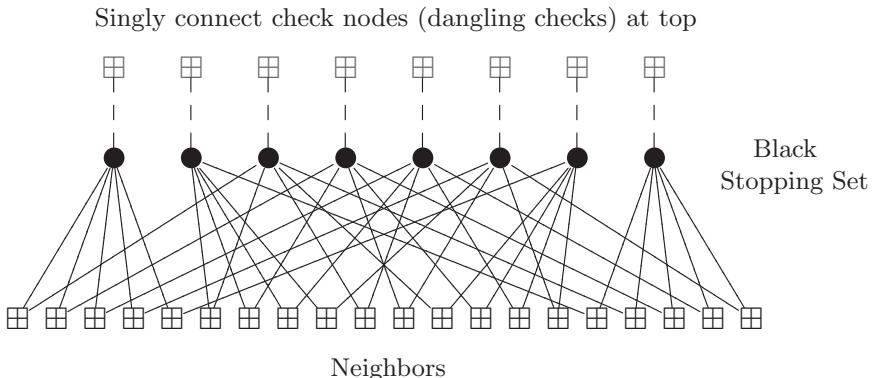


Figure 7.3: Size-8 absorption set of the IEEE 802.3 LDPC code with 8 unsatisfied checks.

7.2 Dynamics of the Absorption Sets

The (8,8) absorption set in Figure 7.3 is dominant in the sense that it is responsible for the error floor of the code in Figure 7.1. There are other absorption sets in the code, as we will see later, but those are not “visible” in the sense that their contribution to the error floor is much smaller than that of the dominant set.

In this section we present an analysis which will allow us to explain the effects of the absorption set. The analysis is approximative, but leads to very accurate results and has excellent predictive power—all that we really need from a theory. We start with the illustration of eight log-likelihood (LLR) values of the absorption set variable nodes during

a particular decoding cycle. These LLR values are shown as the eight traces in Figure 7.4. We can distinguish an initial phase where the LLRs fluctuate apparently randomly, but undergo a growth process. This phase is followed by a “bit flipping” process once the limit values of the LLR number range have been reached. In this example, this limit is set to an arbitrary, but quite typical, value of $|\lambda_{\max}| = 10$.

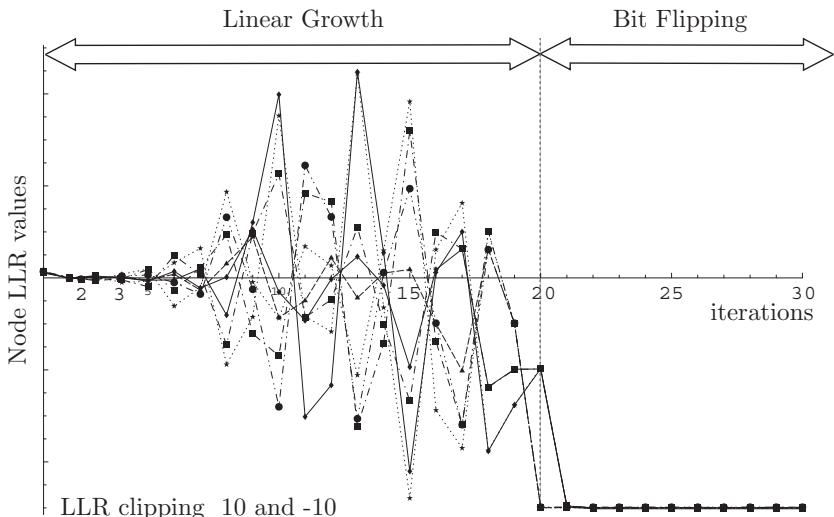


Figure 7.4: Trajectories of the LLR values of the variable nodes in the absorption set of Figure 7.3 during a specific decoding cycle.

We now understand the importance of the absorption sets in the decoding of LDPC code even over the Gaussian channel. Once the limit values of the LLRs are reached, a signal clipping occurs. The effect of this is that only the sign of the LLR value participates in the message passing, since all values are either plus or minus the maximum. In effect, the decoder has migrated to simple bit flipping, which is identical to the decoding algorithm for the BSC channel discussed in Section 6.4.3. An absorption set caught in the bit flipping stage of the decoding algorithm will never correct itself.

The reader may wonder about the impact of the clipping value on this phenomenon, and we will later see that large variations in the error floor can be generated simply by changing this clipping threshold. However, some form of clipping always occurs, even in a floating point computer version of the algorithm. However, the subtle influence the clipping on the error floor may account for the large disagreements by different researchers about where the floor lies for a particular code. The facts are that this floor depends on a

number of parameters which are decoder related, but not code related, and it is therefore possible to steer the error floor over wide ranges with a proper decoder design.

In order to develop an analytical methodology, we denote the *outgoing edge values* from the variable nodes in Figure 7.3 by x_i , i.e., x_1, x_2, \dots, x_5 leave variable node $v = 0$, x_6, x_7, \dots, x_{10} variable node $v = 1$, etc. Collect the x_i in the length-40 column vector \mathbf{x} , which is the vector of outgoing variable edge values in the absorption set. Likewise, and analogously, let \mathbf{y} denote the *incoming edge values* to the variable nodes, such that y_j corresponds to the reverse-direction message. Now, at iteration $i = 0$

$$\mathbf{x}_0 = \boldsymbol{\lambda}, \quad (7.1)$$

where the initial input is the vector $\boldsymbol{\lambda} = [\lambda_1, \dots, \lambda_1, \lambda_2, \dots, \lambda_2, \dots, \lambda_8, \dots, \lambda_8]^T$ of channel intrinsics copied onto the first outgoing messages. It undergoes the following operation at the check node:

$$\mathbf{y}_0 = \mathbf{C}\mathbf{x}_0, \quad (7.2)$$

where \mathbf{C} is a permutation matrix that exchanges the absorption set signals. This is a valid approximation, since we conjecture that the messages within the absorption set converge much slower than those external to the set. Therefore, even though the check nodes in the code have 32 edges each, the minimum message will dominantly affect the outputs, and this is the message from within the absorption set. We will make this approximation more precise in what follows.

At iteration $i = 1$ we now obtain

$$\mathbf{x}_1 = \mathbf{V}\mathbf{C}\boldsymbol{\lambda} + \boldsymbol{\lambda} + \boldsymbol{\lambda}_1^{(\text{ex})}, \quad (7.3)$$

where \mathbf{V} is the variable node function matrix, i.e., each output is the sum of the other four inputs from the check nodes plus the intrinsic input. The extrinsic inputs from the remainder of the code graph are contained in $\boldsymbol{\lambda}_1^{(\text{ex})}$. Following our linear model, at iteration $i = j$ extrinsic signals are injected into the absorption set as

$$\boldsymbol{\lambda}_j^{(\text{ex})} = [\lambda_{j1}^{(\text{ex})}, \dots, \lambda_{j1}^{(\text{ex})}, \lambda_{j2}^{(\text{ex})}, \dots, \lambda_{j2}^{(\text{ex})}, \dots, \lambda_{j8}^{(\text{ex})}, \dots, \lambda_{j8}^{(\text{ex})}]^T$$

via the extrinsic check nodes. By induction we obtain at iteration $i = I$

$$\mathbf{x}_I = \sum_{i=0}^I (\mathbf{V}\mathbf{C})^i \boldsymbol{\lambda} + \sum_{i=1}^I (\mathbf{V}\mathbf{C})^{I-i} \boldsymbol{\lambda}_i^{(\text{ex})}. \quad (7.4)$$

We can now compute the largest eigenvalue of $\mathbf{V}\mathbf{C}$ and its eigenvector, since they dominate the power of the matrix $(\mathbf{V}\mathbf{C})^i$. We find the following result:

Lemma 7.1 *The largest eigenvalue of $\mathbf{V}\mathbf{C}$ for the $(8, 8)$ set is $\mu_{\max} = d_v - 2 = 4$, and its associated eigenvector is $\mathbf{v}_{\max} = [1, 1, \dots, 1]^T$.*

Proof: First write $\mathbf{VC} = 4\mathbf{M}$. By inspection \mathbf{M} is a probability matrix, i.e., the sum of each row equals unity. As a special case of the Perron–Frobenius theorem, it is known that the largest eigenvalue of a probability matrix is 1; therefore the largest eigenvalue of \mathbf{VC} equals 4. By inspection, $\mathbf{VC}[1, 1, \dots, 1]^T = 4[1, 1, \dots, 1]^T$. Q.E.D.

We can now use the spectral theorem

$$(\mathbf{VC})^i \boldsymbol{\lambda} \xrightarrow{i \rightarrow \infty} \mu_{\max}^i (\boldsymbol{\lambda}^T \mathbf{v}_{\max}) \mathbf{v}_{\max} \quad (7.5)$$

to obtain an estimate of the behavior of \mathbf{x}_I . If error indicator $\beta = \mathbf{x}_I \cdot \mathbf{1} \leq 0$ for $I \rightarrow \infty$, then the absorption set is in error and cannot be corrected. Combining (7.5) with (7.4), we find

$$\beta = \boldsymbol{\lambda}^T \mathbf{v}_{\max} + \sum_{j=1}^I \frac{(\boldsymbol{\lambda}_j^{(\text{ex})} + \boldsymbol{\lambda})^T \mathbf{v}_{\max}}{\mu_{\max}^j} \leq 0 \quad (7.6)$$

or, in the case of our (8, 8) absorption set

$$\beta = \sum_{i=1}^8 \left(\lambda_i + \sum_{j=1}^I \frac{\lambda_{ji}^{(\text{ex})} + \lambda_i}{\mu_{\max}^j} \right) \leq 0. \quad (7.7)$$

The eigenvalue $\mu_{\max} = d_v - 2$ is the *gain* of the absorption. It determines how rapidly the set-extrinsic information is suppressed by the dynamics of the set in favor of the set-internal signals.

Exact knowledge of $\lambda_{ji}^{(\text{ex})}$ is typically not available to the analysis, since these values depend on the actual received signals. However, assuming that the code structure extrinsic to the absorption set operates “regularly,” we may substitute average values for the $\lambda_{ji}^{(\text{ex})}$. By “regular” in this context, we mean that statistics of $\lambda_{ji}^{(\text{ex})}$ follow those predicted by density evolution, and we further assume that $\lambda_{ji}^{(\text{ex})}$ has a *consistent* Gaussian distribution [2, 27], with a mean equal to $2\sigma^2$. We therefore only need the mean of $\lambda_{ji}^{(\text{ex})}$, which we find via Gaussian density evolution discussed in Chapter 6, calculated from (6.30) and (6.34) as

$$m_{\lambda^{(\text{ex})}}^{(i)} = \phi^{-1} \left(1 - \left[1 - \phi \left(m_{\lambda} + (d_v - 1)m_{\lambda^{(\text{ex})}}^{(i-1)} \right) \right]^{d_c - 1} \right),$$

where $m_{\lambda} = 2E_b/\sigma^2$ is the mean of λ_i , $m_{\lambda^{(\text{ex})}}^{(i)}$ is the mean of the extrinsic signal $\lambda_{ji}^{(\text{ex})}$.

Since all the components in (7.7) are Gaussian, we can compute the probability of β

not exceeding zero after some tedious but straightforward computations as

$$\begin{aligned} P_{\text{AS}} &= \Pr(\beta \leq 0) \\ &= \Pr\left(\frac{\text{Mean}(\beta)}{\sqrt{\text{Variance}(\beta)}}\right) \end{aligned} \quad (7.8)$$

$$= Q\left(\frac{2m_\lambda + 2 \sum_{j=1}^I \frac{m_{\lambda^{(\text{ex})}}^{(j)} + m_\lambda}{\mu_{\max}^j}}{\sqrt{\left(1 + \sum_{j=1}^I \frac{1}{\mu_{\max}^j}\right)^2 m_\lambda + \sum_{j=1}^I \frac{m_{\lambda^{(\text{ex})}}^{(j)}}{\mu_{\max}^{2j}}}}\right). \quad (7.9)$$

Equation (7.9) expresses the probability of an absorption set falling in error as an exponential error function, that is, analogous to a regular decision error probability. The difference is that the remainder of the code affects this error probability via its injection of the extrinsic messages.

Before we can study this equation in more detail, Schlegel and Zhang [9] found that a small but important refinement to the formula can decisively enhance its accuracy. The exchange of extrinsics through the matrix \mathbf{C} is an approximation in two ways, First, during the initial iterations, the $d_c - 2 = 30$ inputs to the check node from the set-external part of the code are relatively small, and the entries of \mathbf{C} are strictly less than unity, and, in case one or an odd number of these set-external extrinsic messages has an erroneous sign, the returned signal to the absorption set switches polarity. The first issue is addressed as follows. Using a Taylor series approximation, we can show that the quintessential check node operation

$$\tanh^{-1}\left(\tanh(x) \prod_{i=1}^{d_c-2} \tanh(x_i)\right) = \prod_{i=1}^{d_c-2} \tanh(x_i) x + O[x^3],$$

where $\prod_{i=1}^{d_c-2} \tanh(x_i)$ can be interpreted as a “check node gain.” Since we have no access to the actual check messages, we again use the mean $m_{\mu^{(\text{ex})}}^{(i)}$ of the signals $\mu^{(\text{ex})}$ from the

variable to the check nodes for x_i , to compute an average gain as

$$\begin{aligned} g_i &= E \left[\prod_{i=1}^{d_c-2} \tanh \left(\frac{m_{\mu^{(\text{ex})}}^{(i)}}{2} \right) \right] \\ &= E \left[\tanh \left(\frac{m_{\mu^{(\text{ex})}}^{(i)}}{2} \right) \right]^{d_c-2} \\ &= \left(1 - \phi \left(m_{\mu^{(\text{ex})}}^{(i)} \right) \right)^{d_c-2}, \end{aligned} \quad (7.10)$$

where the last equality results from the definition of the density evolution function $\phi(\cdot)$ in (6.32). With this the probability in (7.9) is modified to the somewhat unwieldy-looking expression

$$P_{\text{AS}} = Q \left(\frac{2m_\lambda + 2 \sum_{j=1}^I \left(\frac{m_{\lambda^{(\text{ex})}}^{(j)} + m_\lambda}{\mu_{\max}^j} \prod_{l=1}^j \frac{1}{g_l} \right)}{\sqrt{\left(1 + \sum_{j=1}^I \prod_{l=1}^j \frac{1}{g_l \mu_{\max}} \right)^2 m_\lambda + \sum_{j=1}^I m_{\lambda^{(\text{ex})}}^{(j)} \left(\prod_{l=1}^j \frac{1}{g_l \mu_{\max}} \right)^2}} \right). \quad (7.11)$$

In order to obtain an estimate of the influence the absorption set has on the bit error rate of the code, we need the multiplicity of this set. A restricted exhaustive search discussed in [9] computes this value as 14,272 and also presents tables of the multiplicities per variable node for all of the variable nodes. This multiplicity ranges from $A_{8,8} = 37\text{--}75$. We now apply the union bound to obtain an upper bound of the impact of this absorption set on the error floor. Given that the (8,8) absorption set is dominant in this code, this is also an estimate of the error rate in the error floor, given as

$$P_b \approx A_{8,8} P_{\text{AS}}. \quad (7.12)$$

This estimate is plotted in Figure 7.1 and shows very close agreement with simulation results, as well as with the *importance sampling* techniques discussed in Section 7.5.

The second issue of erroneous messages entering the check nodes can be handled with similar arguments, and the details are outlined in [9]. However, the impact on the error formula of this effect is quite minor and we therefore do not discuss this further here. Figure Figure 7.1 includes this effect in the error curve labeled “linear model including errors.”

One last issue that needs to be highlighted is that of the clipping threshold. While the generic algorithm on Page 257 does not have any obvious computational limits, such a limit exists in all practical implementations. First, for computational reasons the number representation along the message lines must be limited to a few bits, typically anywhere between 4 and 10. This imposes both a maximum resolution as well as a maximum value that can be represented numerically. As the LLRs grow in the course of decoding, numerical values will be limited, or clipped at this threshold λ_{\max} . For most of the bit error range, λ_{\max} is quite unimportant, however, in the error floor it has a very strong impact. $\lambda_{\max} = 10$ was chosen for both the calculations as well as simulations in Figure 7.1.

The second reason is one of numerical precision. The regular generic check node processor computes $m_{\lambda^{(\text{ex})}}^{(i)}$, which will grow to infinity as long as the code operates above its threshold. However, using

$$\lambda_{j \rightarrow i} = 2 \tanh^{-1} \left(\prod_{l \in V_j \setminus \{i\}} \tanh \left(\frac{\lambda_{l \rightarrow j}}{2} \right) \right) \quad (7.13)$$

in the decoding algorithm effectively clips $\lambda_i^{(\text{ex})}$ due to numerical limitations of computing the inverse hyperbolic tangent of a number close to unity. This is observed in [1] as saturation. More specifically, if p bits are available to represent the argument of $\tanh^{-1}(\cdot)$, then the largest output value is approximated by (see [1])

$$\lambda = (p + 2) \ln 2. \quad (7.14)$$

This equals 38.1230949 the 64-bit IEEE 754 double-precision floating-point format with $p = 53$ bits of precision.

Therefore, instead of (7.13), the corrected min-sum algorithm [3] computes the check-to-variable LLR as

$$\lambda_{j \rightarrow i} = \left(\min_{l \in V_j \setminus \{i\}} (|\lambda_{l \rightarrow j}|) + \text{CT} \right) \prod_{l \in V_j \setminus \{i\}} \text{sign}(\lambda_{l \rightarrow j}), \quad (7.15)$$

where CT represents the correction term that is introduced to mitigate the loss of the min-sign approximation introduced in (6.8). This specific correction factor was derived in [3], but other such approaches are possible. It is shown in [3] that this correction leads to a performance of no observable loss for a variety of LDPC codes. The factor is dependent on the check node degree and given as

$$\text{CT} = \begin{cases} -\ln(d_c - 1)/4, & \text{if } \min_{l \in V_j \setminus \{i\}} |\lambda_{l \rightarrow j}| \geq 3 \ln(d_c - 1)/8, \\ 0, & \text{otherwise,} \end{cases} \quad (7.16)$$

Using (7.15) effectively avoids the clipping due to numerical instability and is the preferred practical implementation since it also offers lower computational complexity than the original check node computation (7.13).

We summarize this section with the two steps required to compute the error floor of an LDPC code. First, we need a procedure to identify the absorption sets which are most influential on the BER. This requires a search procedure. The authors of [9] have looked at the IEEE 802.3an LDPC code discussed here, and identified the small absorption sets in Table 3.1, of which the (8,8) set in Figure 7.3 is the dominant absorption set. It is shown in [9] that the (8,8) absorption set causes the highest BER contribution and that the next “lower” set has a BER impact that is about an order of magnitude less.

a	b	Existence	Multiplicity	Gain
< 5		No		
5	10	No		
6	6, 8, 10, 12	No		
	0, 2, 4, 6, 8, 10	No		
7	12	Yes	65, 472 ¹	3.29
	14	Yes	14, 720	3
	0, 2, 4, 6	No		
	8	Yes	14, 272	4
8	10	No		
	12	Yes	44, 416	3.5
	14	Yes	?	3.25
	16			3
	0, 2, 4	No		
	6, 8, 10	Yes	?	
9	12			3.67
	14			3.44
	16			3.22
	18			3
	0, 2, 4, 6, 8			
	10	Yes	> 192	4
10	12	Yes	?	3.8
	14			3.6
	16			3.4
	18			3.2
	20			3

Table 3.1: Absorption sets of IEEE 802.3an LDPC code.

¹Only sets not contained in (8,8) absorption sets are counted — see [9].

The second step is to apply the computational procedure outlined in this section to calculate the impact of each absorption set to the error floor. This requires computation of μ_{\max} and v_{\max} numerically using the set topology, and then it uses (7.6), in general, to ascertain the error probability of each set. The dominant set and its multiplicity typically produce a good estimate of the error floor via the union bound.

7.3 Code Design for Low Error Floors

The need to address the error floor problem was of course recognized early on, before a comprehensive theory of it was available. In this section we discuss a couple of approaches which have led to successful designs, and are worth discussing.

One such successful method is to realize that not all variable nodes have the same error rates and that an irregular LDPC code has an unequal bit error rate. In fact, due to (6.40), nodes with a higher degree will have a more rapid accumulation of their LLR value and thus will be more reliable. We can therefore associate nodes of high variable degrees with information nodes, which need high error protection, and those of lower degrees with parity nodes, whose error rates are, in principle, irrelevant. In decoding, we are typically only interested the error rate of the information bits, and the parity bits are discarded. This is the approach taken in the design of the so-called *extended irregular repeat accumulate codes (eIRA)* studied in [11].

The parity-check matrix of these codes has a special structure, an example of which is

$$\mathbf{H} = \left[\begin{array}{cccc|ccccc} 1 & 1 & 1 & & & 1 & & & \\ & & & 1 & 1 & 1 & & & \\ & & & 1 & 1 & 1 & 1 & & \\ & & & 1 & 1 & 1 & & 1 & 1 \\ & & & 1 & & & & 1 & 1 \\ \hline 1 & & & 1 & & 1 & & 1 & 1 \\ 1 & & & 1 & & 1 & & 1 & 1 \\ 1 & 1 & & 1 & & & & 1 & 1 \\ 1 & & 1 & 1 & & & & 1 & 1 \\ 1 & & 1 & & 1 & & & 1 & 1 \\ 1 & & & 1 & & & & 1 & 1 \end{array} \right], \quad (7.17)$$

where the parity section is characterized by the dual diagonal which allows very efficient recursive encoding; see Section 6.6.

Yang et al. [11] have constructed eIRA codes using optimized degree profiles. The construction of the parity-check matrix simply ensured that there were no 4-cycles. The authors noted that if they biased the distributions such that no degree-3 or no degree-4 information variable nodes existed, they could trade off the error floor and the convergence

threshold. Figure 7.5 shows the performance of three such codes of rate $R = 0.82$ of size (4161,3430) [11] whose degree distributions are given by

Code 1:

$$\begin{aligned}\lambda(x) &= 0.00007 + 0.1014x + 0.5895x^2 + 0.1829x^6 + 0.1262x^7 \\ \rho(x) &= 0.3037x^{18} + 0.6963x^{19}\end{aligned}$$

Code 2:

$$\begin{aligned}\lambda(x) &= 0.0000659 + 0.0962x + 0.9037x^3 \\ \rho(x) &= 0.2240x^{19} + 0.7760x^{20}\end{aligned}$$

Code 3:

$$\begin{aligned}\lambda(x) &= 0.0000537 + 0.0784x + 0.9215x^4 \\ \rho(x) &= 0.5306x^{24} + 0.4694x^{25}\end{aligned}$$

Another approach proposed in [10] is to lower the so-called *approximate cycle extrinsic message degree*. The *extrinsic message degree* (EMD) is a precursor of the more complete absorption set, which considers only cycles instead of the entire set. While computationally of little value, the concept is easier to incorporate into the design of a code, since, primarily, the designer works only with the short cycles of the codes, which are a topologically much simpler structure.

The idea is similar to those we have seen affecting the error performance of the absorption set, namely, a larger number of messages entering the set from the surrounding code network lower the probability that the set falls in error—see (7.7). This concept is, however, applied to cycles instead. Consider the cycle on the left in Figure 7.6. It is actually a stopping set—remember that each stopping or absorption set contains at least one cycle. As a stopping set, it has an extrinsic message degree (EMD) of zero; there are no outside message lines connecting to the variable nodes of this set. We have come across the concept of the extrinsic message degree in the previous section, so let us define it formally:

Definition 7.2 *The extrinsic message degree (EMD) of a set is defined as the number of connections from the variable nodes of the set to outside (check) nodes.*

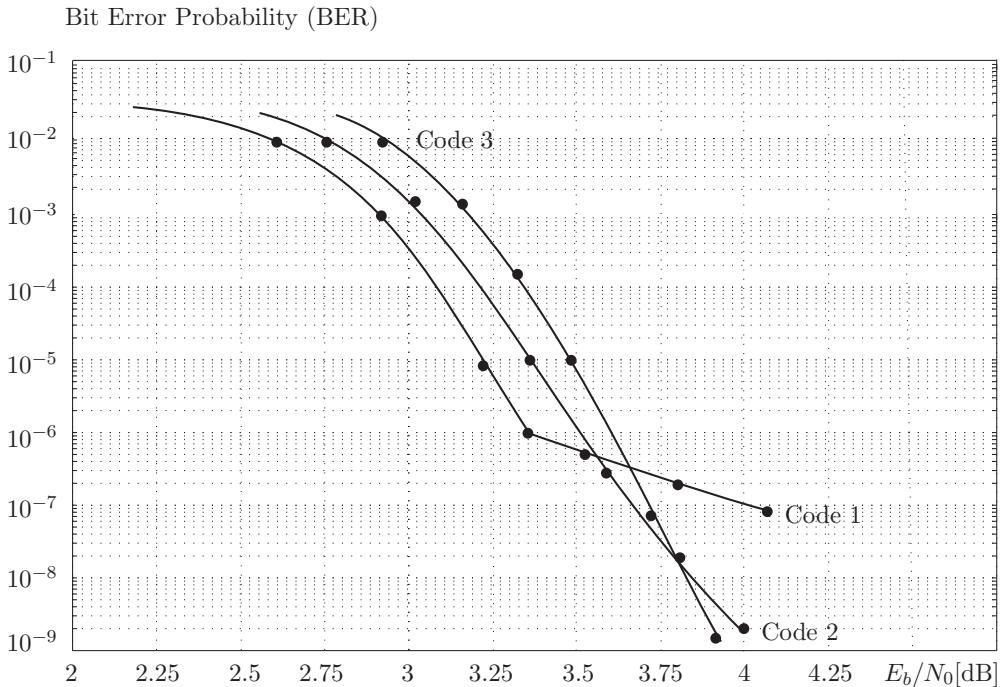


Figure 7.5: Performance of optimized eIRA codes of rate $R = 0.82$ with low error floors.

The EMD of a reduced set derived from the stopping set is also shown in Figure 7.6. It has an EMD of 3. This illustrates the somewhat arbitrary nature of the definition. Nonetheless, this definition has proved quite useful. Note that a stopping set always has an EMD of zero, and an (a,b) absorption set has an EMD equal to b .

Working with the EMD of sets is combinatorially complex as we have seen in our description of absorption sets, and a computationally more practical measure is needed to make it useful for code construction. Tian et al. [10] proposed to consider the EMD only of cycles, ignoring possible connection inside the cycle. For example, their definition ignores the central check node in the set on the right-hand set of Figure 7.6 and gives the associated circle an approximate EMD of 5. In general then, the approximate EMD of a circle of length $2d$ simply equals

$$\text{ACE} = \sum_{i=1}^d (\lambda_i - 2) \quad (7.18)$$

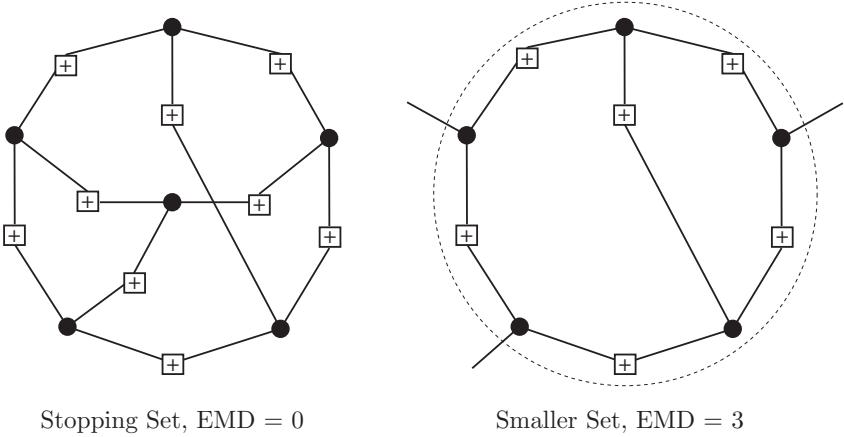


Figure 7.6: Extrinsic message degree of two sets.

and is thus not significantly more complex to compute than finding the cycles in a code.

On the basis of this *Approximate Cycle EMD*, called the ACE of a cycle, Tian et al. [10] proposed the construction of codes with parameters d_{ACE} and η_{ACE} , thus defined:

Definition 7.3 An LDPC code has parameters $(d_{\text{ACE}}, \eta_{\text{ACE}})$ if all cycles of length $d \leq 2d_{\text{ACE}}$ have $\text{ACE} \geq \eta_{\text{ACE}}$.

This definition opens avenues to the construction of LDPCs with many different sets of parameters. That this procedure can produce good codes is illustrated in Figure 7.7 [10], which compares two codes of rate $R = 0.5$ and length 10K.

As we have seen in the preceding section, the error mechanism and its impact on the error floor are well understood, and are based on complex topological structures which form part of the code's interconnect network. While construction methods such as the ones presented in this section have been quite successful in reducing the number of such sensitive structural weaknesses, they can never be completely avoided in codes of finite size. We therefore turn our attention now to the importance of the decoding algorithm itself as a contributor to the error floor of LDPC codes.

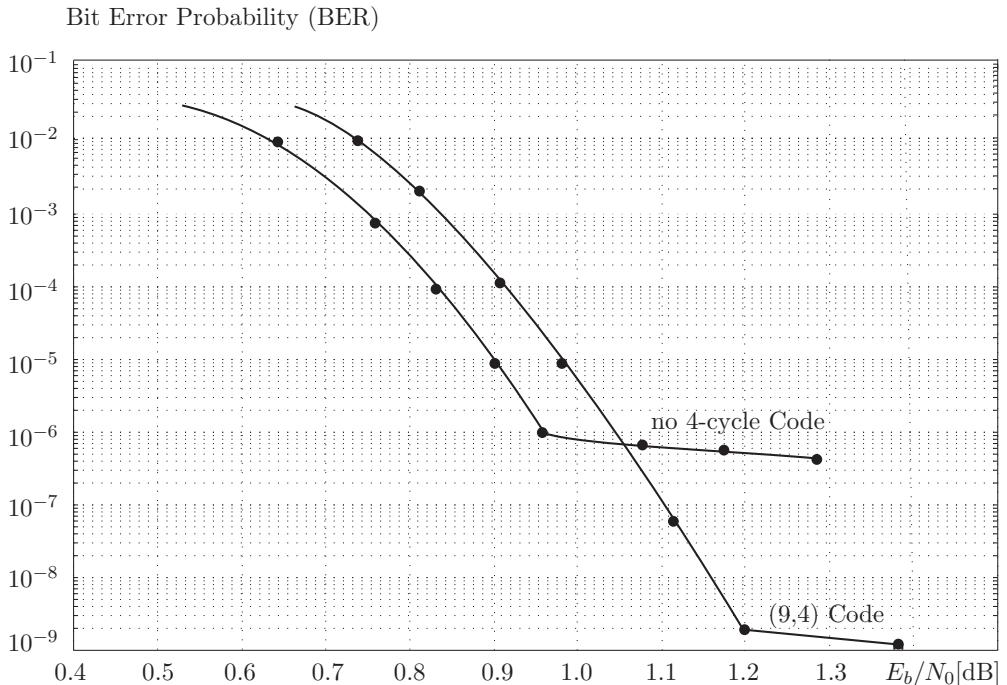


Figure 7.7: Performance of codes of rate $R = 0.82$ and length $N = 10,000$ constructed with the ACE criterion. The first code contains no 4-cycles, while the code is constructed such that each cycle of length 18 or less has an EMD of at least 4.

7.4 Impact of the Decoding Algorithm

The error floor equation (7.11) can give clues to how the actual bit error rate will behave as a function of the different parameters. Primarily, the absorption set itself determines the magnitude of the gain μ_{\max} , which is determined by the structure of the alone. It influences how rapidly the effect of the extrinsic information $m_{\lambda^{(ex)}}$ diminishes.

In an ideal code using floating point arithmetic, $m_{\lambda^{(ex)}}$ itself has exponential growth as determined by density evolution and will balance the geometric growth of the effect of the absorption set gain. We glimpse the possibility that if the growth of $m_{\lambda^{(ex)}}$ can be sustained, the argument of (7.11) can be made arbitrarily large and hence $P_{\text{AS}} \rightarrow 0$. That is, in this case the impact of an absorption set, even with a strong gain, μ_{\max} can be made to vanish.

As we discussed in Section 7.2, for a variety of reasons, usually due to complexity reasons, the log-likelihood messages are limited to a finite range, and therefore the growth of μ_{\max} cannot keep up with the growth of μ_{\max} . As a consequence, the set-extrinsic messages have an effect on P_{AS} only for a few initial iterations, after which their contribution quickly vanishes. If the set is not corrected during these initial iterations, the error will persist no matter how many total iterations are applied to the algorithm. This effect is well known and caused by the decoder converting to effective bit flipping decoding.

If we are using (7.15) and (7.16) the limitation on λ_{ex} is completely under the designer's control and not influenced by numerical issues as in (7.13). Figure 7.8 shows the error rates due to the dominant absorption set of the IEEE 802.3an LDPC code in its error floor region. The dashed curves plot (7.11) for $I = 10$ and m_{λ} and $m_{\lambda^{(\text{ex})}}^{(i)}$ are bounded by 10, which is the clipping threshold. The circles are the numerical results of importance sampling (IS) utilizing (7.15) with the same I and LLR clipping threshold. When the threshold is increased to 100, the error rate decreases as predicted by this analysis. Also shown are results for a clipping value of 1,000.

The error formula (7.11) is quite sensitive to numerical issues, however, despite this sensitivity, importance sampling simulations and analytical results agree to within a few tenths of a dB over the entire range of E_b/N_0 values of interest. Reference [13] discusses further details and presents additional code examples.

7.5 Importance Sampling (IS)

Software-, and even hardware-based simulation of very low error rates is often infeasible. For example, even at a bit rate of 1 Gb/s, it would take over 11 days to produce a single error at an error rate of 10^{-15} , which is a typical error rate requirement for optical transmission systems. This is the domain of *importance sampling*, which elevates the occurrence of the important event, in our case the errors.

We begin with the following general formulation of the error computation problem. Let P_e be the quantity we wish to compute by integrating the probability density function $f(x)$ over the domain Ω which is associated with the “error,” i.e.,

$$P_e = \int_{\Omega} f(x) dx, \quad \Omega \text{ is the domain of integration.} \quad (7.19)$$

This situation is illustrated in Figure 7.9 for a one-dimensional variable and domain, in which case the integral can of course be performed numerically. However, if the domain Ω spans many dimensions, the describing the boundary of the domain Ω can become exceedingly complex, which makes an analytical computation of (7.19) practically impossible.

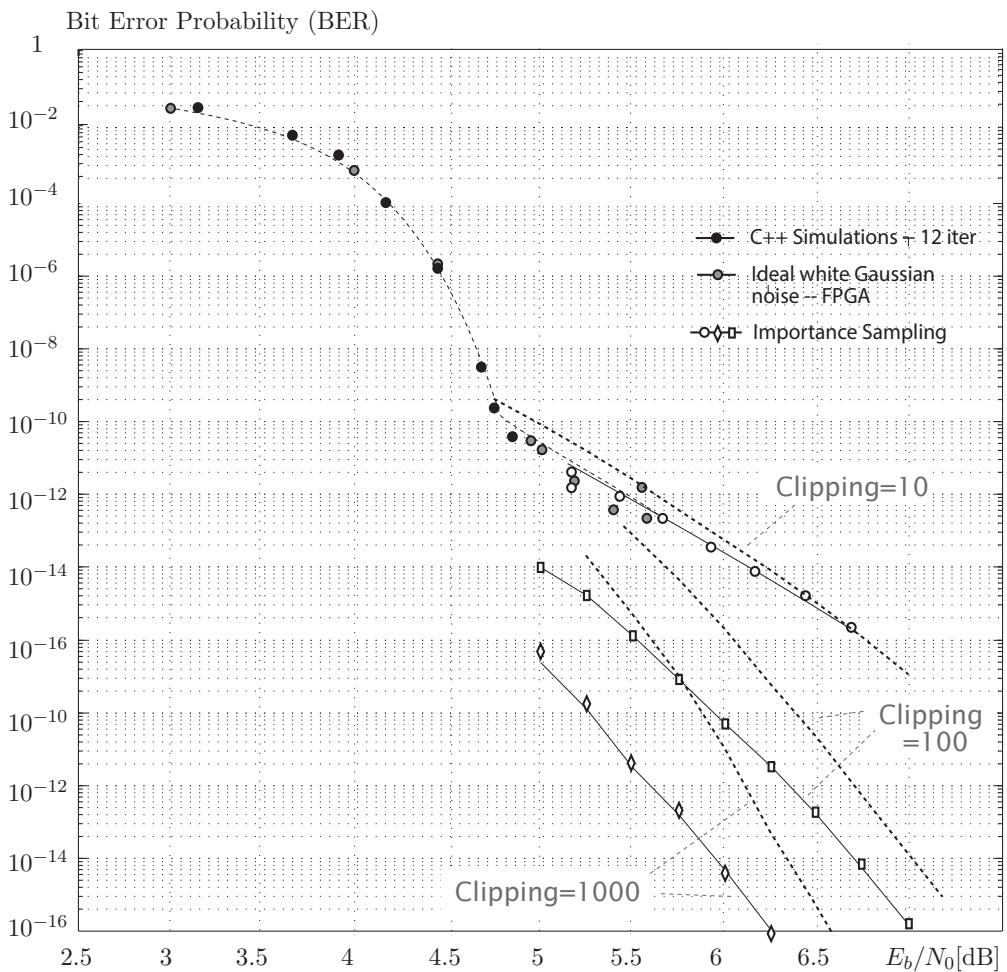


Figure 7.8: Bit error rates due to the dominant absorption set of the IEEE 802.3an LDPC code using both formula (7.11) and importance sampling (IS) with LLR clippings at 10, 100, and 1000, respectively. The iteration number is set to 10.

This is the situation encountered in computing error probabilities for many communications applications and is a situation which is addressed by importance sampling. IS

evaluates this integral (7.19) as

$$P_e = \int_{\Omega} f(x) dx = \int_{\Omega} \frac{f(x)}{\rho(x)} \rho(x) dx = \int_{\Omega} w(x) \rho(x) dx, \quad (7.20)$$

where the weighting function $w(x) = f(x)/\rho(x)$ changes the distribution of the samples over Ω .

Of course, (7.19) is no easier to compute analytically since the boundary problem is the same. However, often it is straightforward to determine whether a point sample lies within the domain of integration or outside—for example, if it is an error event of a decoder.

Using sample counts, we therefore approximate

$$P_e \approx \frac{1}{N_s} \sum_{i=1}^{N_s} w(x_i) I_{\Omega}(x_i) \quad (7.21)$$

where the new random samples are drawn according to $\rho(x)$, however, instead of the original distribution $f(x)$. The indicator function $I_{\Omega}(x_i) = 1$ if $x_i \in \Omega$ and zero otherwise formalizes that only samples in the domain of integration are counted. The count N_s denotes the number of samples used in the approximation (7.21). The goal of importance sampling is now to distort the original density function in such a way that N_s can be minimized for a given accuracy of the estimate, and therefore simulations can be accelerated.

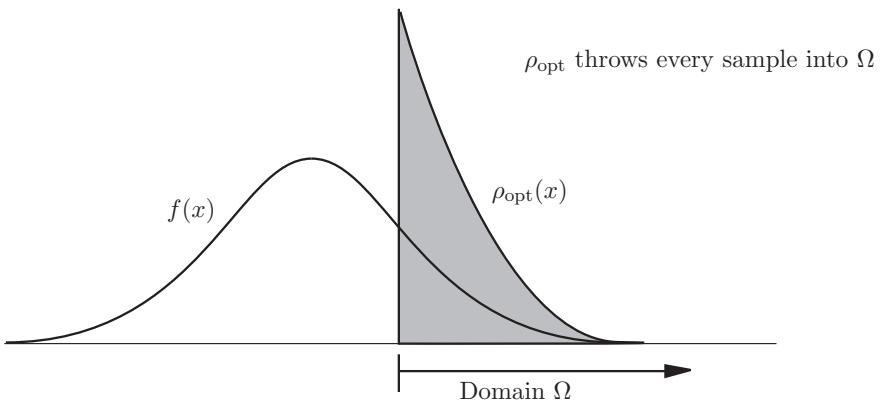


Figure 7.9: Illustration of importance sampling for a one-dimensional function.

It can easily be seen that the optimal distortion of $f(x)$ would use

$$\rho_{\text{opt}}(x) = \frac{|f(x)|}{\int_{\Omega} |f(x)| dx}; \quad x \in \Omega, \quad (7.22)$$

and lead to the constant weighting function $w(x) = \int_{\Omega} |f(x)| dx$ —which would require only a single sample. Clearly (7.22) is not of practical interest since it requires that we can compute the integral that we originally tried to avoid.

Once we fix a $\rho(x)$ —to be discussed below—the error is estimated via IS as

$$\tilde{P}_e = \frac{1}{N_s} \sum_{i=1}^{N_s} w(x_i) I_{\Omega}(x_i). \quad (7.23)$$

It can be seen easily that \tilde{P}_e is an unbiased estimator of P_e , that is $E[\tilde{P}_e] = P_e$, and its variance is computed as

$$\begin{aligned} \sigma_s^2 &= \frac{1}{N_s^2} E \left[\sum_{i=1}^{N_s} w(x_i) I_{\Omega}(x_i) \sum_{j=1}^{N_s} w(x_j) I_{\Omega}(x_j) \right] - P_e^2 \\ &= \frac{1}{N_s} E \left[\sum_{i=1}^{N_s} w^2(x_i) I_{\Omega}(x_i) \right] + \frac{N_s - 1}{N_s} P_e^2 - P_e^2 \\ &= \frac{1}{N_s} E \left[\sum_{i=1}^{N_s} w^2(x_i) I_{\Omega}(x_i) \right] - \frac{P_e^2}{N_s}. \end{aligned} \quad (7.24)$$

Of course, the variance itself needs to be estimated since the actual error P_e is the unknown quantity. An unbiased estimator for the variance is found as

$$\tilde{\sigma}_s^2 = \frac{1}{N_s} \left(\frac{1}{N_s} \sum_{i=1}^{N_s} w^2(x_i) I_{\Omega}(x_i) - \left(\frac{1}{N_s} \sum_{i=1}^{N_s} w(x_i) I_{\Omega}(x_i) \right)^2 \right), \quad (7.25)$$

The performance advantage of IS is often expressed as the *gain* it attains over direct Monte-Carlo simulation, i.e., $\rho(x) = 1$, and is expressed as

$$G_s = \frac{\sigma_{MC}^2}{\sigma_s^2}. \quad (7.26)$$

The key is to ensure that the gain $G_s \gg 1$ order to save on the number of simulation runs.

The variance of Monte-Carlo simulations is computed assuming that each run is independent from other runs and has the same error probability P_e . The variance σ_{MC}^2 is therefore the variance of a binary 0/1 random variable and found by setting $w(x_i) = 1$ in (7.24), or (7.25), respectively, to obtain

$$\sigma_{MC}^2 = \frac{P_e(1 - P_e)}{N_s}. \quad (7.27)$$

This is also how the error bars for error rate simulations are derived, by finding the standard deviation $\hat{\sigma}_s$ according to (7.25).

The gain G_s is therefore a measure of how much the reliability of an estimate is improved, as well as how much computational gain is achieved over standard Monte-Carlo sampling for the same reliability. Figure 7.10 shows the this gain for the case of the error rate of the [7,4] single-error correction Hamming code using maximum-likelihood decoding in an additive white Gaussian channel to be discussed in Section 7.6. Clearly, massive speed gains are possible with importance sampling if done correctly.

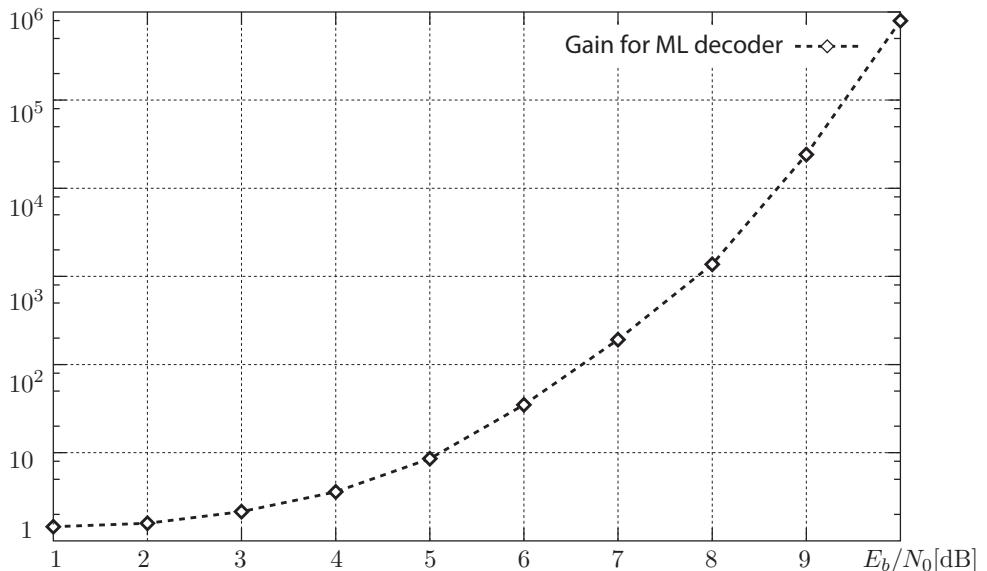


Figure 7.10: Acceleration provided by importance sampling in simulation the bit error probability of the [7,4] Hamming code.

7.6 Computing Error Rates via Importance Sampling

We now apply the method of importance sampling to the computation of error rates of error control decoders. A few additional caveats need to be carefully addressed and avoided

to obtain accurate results. If these are not properly dealt with, the results can easily be skewed and can become useless.

Figure 7.11 shows the generic setup encountered by an error control decoder. However, the reader should be cautioned that the simplicity of the situation in the figure does not reflect how complex the situation is in general, and that such a simple description of the decoding geometry is, in general, not possible.

We wish to compute P_0 the probability that an error occurs if codeword \mathbf{x}_0 is transmitted. This happens when the received signal vector \mathbf{y} falls outside the *decision region* \mathcal{D}_0 for \mathbf{x}_0 , i.e., the region in received signal space for which the decoder outputs \mathbf{x}_0 as its decoded codeword. The decision region can be the maximum likelihood decision region discussed in Chapter 2, or any other decoder decision region. In fact, in general we do not have a precise description of the boundary of the decision regions, and this is precisely why we use simulation—in particular, IS. By simulating the decoder, we do not need to know precisely where these boundaries are.

Formally, however, the error probability in question is obtained by integrating the conditional channel probability density function $p(\mathbf{y}|\mathbf{x}_0)$ over the complement of the decision region \mathcal{D}_0 of \mathbf{x}_0 , which can be expressed as the sum over the decision regions for all other codewords \mathbf{x}_i .

$$P_0 = \int_{\bigcup_{i \neq 0} \mathcal{D}_i} p(\mathbf{y}|\mathbf{x}_0) d\mathbf{y} = \sum_{i=1}^M \int_{\mathcal{D}_i} p(\mathbf{y}|\mathbf{x}_0) d\mathbf{y} \quad (7.28)$$

Generally, we can closely approximate P_0 by concentrating on the most probable error neighborhoods by restricting the explored error regions to those in the immediate proximity of \mathbf{x}_0 as illustrated in Figure 7.11. Caution must be exercised, however, to make sure that these dominant neighbors are sufficiently well understood. This is not trivial in any typical-sized modern error control code. In fact, the absence of such knowledge is precisely what makes importance sampling such a sensitive technique in practice.

In general now, the purpose of IS is to bias the noise towards producing significantly more errors, so the number of simulations can be reduced. This can be accomplished in a number of ways:

- **Excision:** Certain received signal vectors are recognized as not causing an error and can be discarded without simulating their effect on the decoder. E.g., if simple sign decoding causes all of the bits to be correct in that they produce a codeword, the decoder will also generate the same codeword and does not need to be evoked. Excision can sometimes be used to speed up a simulation, but will fail to help in simulating the error floor in our case.
- **Variance Scaling:** The noise variance is simply increased and thus causes more

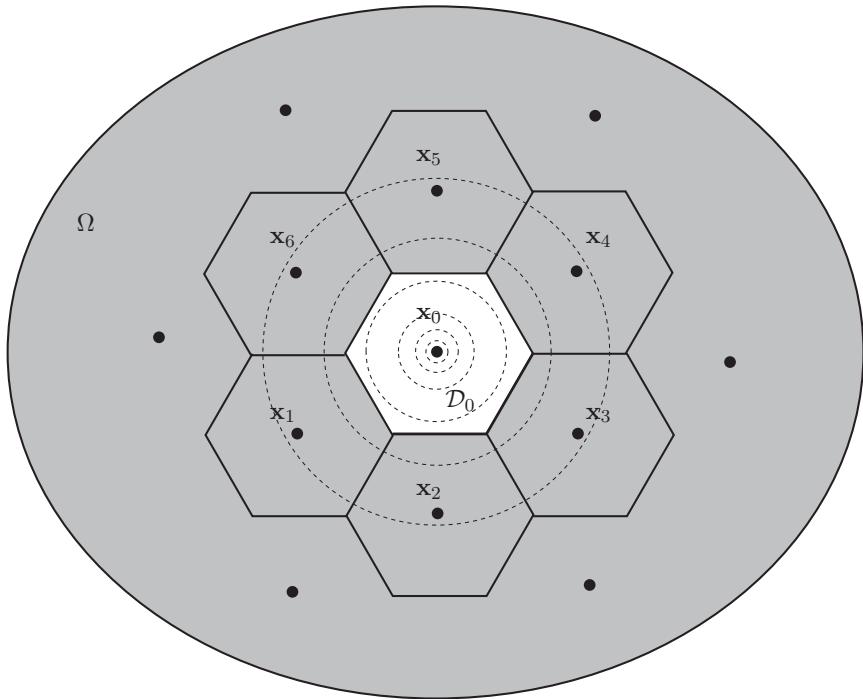


Figure 7.11: Illustration of the decision regions followed by a decoder.

errors. One would apply the following weight function in

$$w(\mathbf{y}) = \frac{\sigma_B}{\sigma} \exp\left(-|\mathbf{y} - \mathbf{x}_0|^2 \frac{\sigma_B^2 - \sigma^2}{\sigma_B^2 \sigma^2}\right) \approx \exp\left(-\frac{|\mathbf{y} - \mathbf{x}_0|^2}{\sigma^2}\right). \quad (7.29)$$

$w(\mathbf{y})$ is exponential in the SNR. However, variance scaling does not work well in our case. The reasons are simple. If variance scaling worked, it would be possible to deduce a code's error performance at high signal-to-noise ratios from its performance in the low signal-to-noise ratio regime. This is typically not feasible, since different error mechanisms are responsible for the errors in the two regimes.

- **Mean Translation:** The most promising method, and the only one which is useful in our case, is where the biased samples are generated according to $p^*(\mathbf{y}) = p(\mathbf{y} - \mu)$,

where μ is a shift value towards the **decision boundary** of interest. We obtain

$$\begin{aligned} P_{i0} &= \int_{\mathcal{D}_i} \frac{p(\mathbf{y}|\mathbf{x}_0)}{p(\mathbf{y}-\mu|\mathbf{x}_0)} p(\mathbf{y}-\mu|\mathbf{x}_0) d\mathbf{y} \\ \implies P_{i0} &\approx \frac{1}{N_s} \sum_{j=1}^{N_s} \frac{p(\mathbf{y}-\mu|\mathbf{x}_0)}{p(\mathbf{y}_j-\mu|\Omega\mathbf{x}_0)} p(\mathbf{y}_j-\mu|\mathbf{x}_0) I(\mathbf{y}_i). \end{aligned} \quad (7.30)$$

Typically, in mean-shift importance sampling, the noise is shifted towards the decision boundary. This shift may be approximate since the boundary may not be exactly known, but will be somewhere in the “vicinity” of the midpoint between two codewords, i.e.,

$$\mu = \frac{\mathbf{x}_0 + \mathbf{x}_i}{2}.$$

Another issue that presents itself immediately with mean-shift IS is which boundary to shift towards. As illustrated in Figure 7.12, there are typically a number of qualifying nearest neighbors, and one really needs to separately shift towards each of them and then combine the error probabilities into a complete estimate of P_e . This is done by obtaining IS estimates of the probability that codeword \mathbf{x}_0 is decoded as \mathbf{x}_i , and then sum all these contributions over the candidates that are examined, i.e.,

$$\tilde{P}_e = \sum_{i:\text{neighbors}} \tilde{P}_{0 \rightarrow i}. \quad (7.31)$$

The difficulty here is to know where those dominant neighboring decision regions are and to have an approximate estimate of their boundaries with the codeword region to be simulated. As cautioned repeatedly, this requires a fairly detailed understanding of the topology of the code which is to be examined.

In the example case of the [7,4] Hamming code, fortunately the geometry of the codewords is known exactly, since the code is *perfect*. Concentrating on its 4 nearest neighbors at Hamming distance $d_H = 3$, we were able to obtain the results in Figure 7.10.

As we have cautioned the reader several times, knowledge of the codeword topologies is vital for a proper use of importance sampling, and IS results are only reliable to extend that this knowledge is complete. For instance, in our importance sampling simulations for the IEEE 802.3an LDPC code in Figures 7.1 and 7.8, the implicit assumption was that the dominant absorption set from Figure 7.3 and its multiplicities are primarily dominating the error probabilities. This, however, may not correct for all cases, such as very large clipping values in 7.3, and the actual minimum-distance codewords with $d_{\min} = 14$ may cause the error floor. Unfortunately, the multiplicity of the minimum distance neighbors is unknown for this code, and hence we can only put in a coarse estimate of where this error floor may lie—see again Figure 7.8. For a clipping value of 10, however, we can be

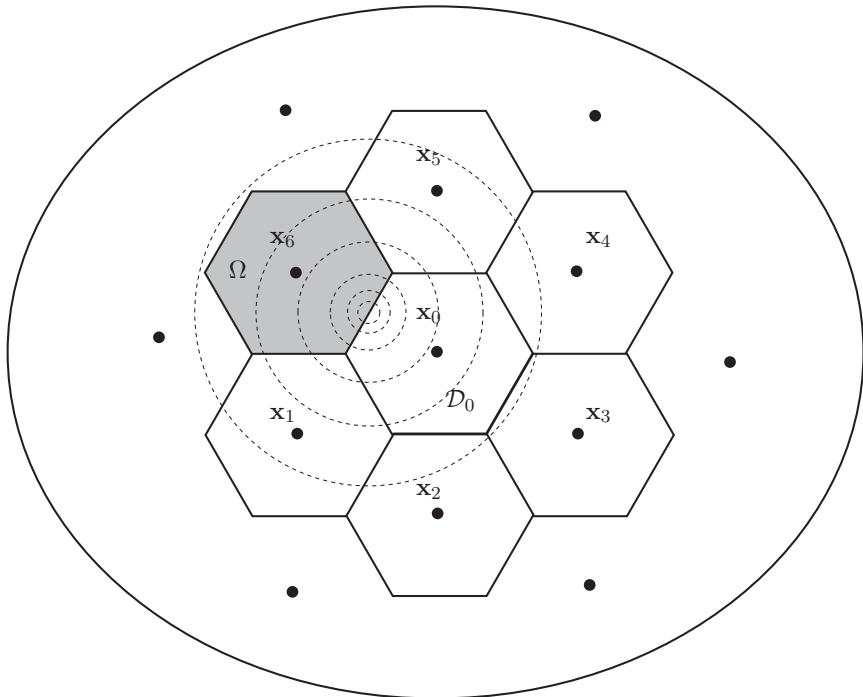


Figure 7.12: Illustration of biasing the noise around codeword x_0 towards codeword x_6 to generate an importance sampling error estimate $\tilde{P}_{0 \rightarrow 6}$.

assured that the correct error mechanism has been identified since both simulations and actual performance bit error rates coincide.

As a final example we present IS results for the $(16,11)^2$ product code constructed from two $(16,11)$ Hamming codes. Product codes are discussed in Chapter 9. The well-structured geometries of the product codes not only allow the application of a very regular decoding algorithm, its codeword topology is also easy to derive from the topology of the component code, in this case a $(16,11)$ extended Hamming code with $A_{\min} = 15$ nearest codewords at Hamming distance $d_H = 4$. In particular, the minimum distance of the product code is $d_{rmH_p} = d_H^2 = 16$, and its multiplicity equals $A_{H_p} = A_H^2 = 225$. This information is sufficient to compute the bit error rate caused by minimum-distance error events, which is the dominating mechanism for this code in the error floor regime.

This conclusion is further substantiated by the fact that IS simulation, measured hardware results, and the minimum distance approximation

$$P_e \approx A_{\text{HP}} Q\left(\sqrt{\frac{2d_{\text{HP}}}{N_0}}\right) \quad (7.32)$$

all agree, as shown in Figure 7.13.

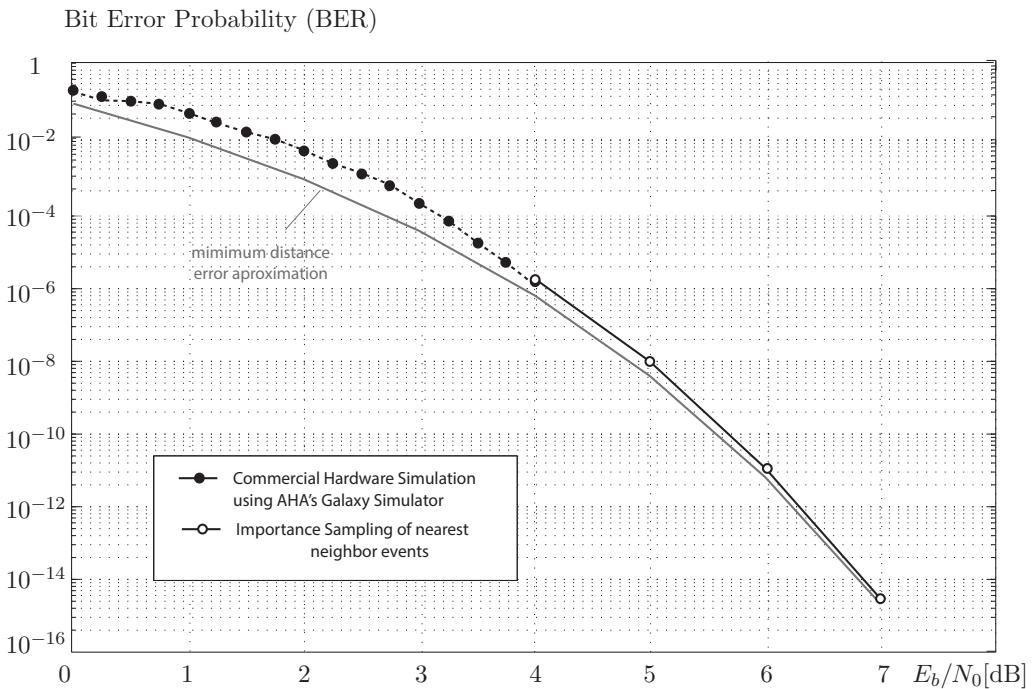


Figure 7.13: Low bit error evaluation for the $(16,11)^2$ product code using importance sampling.

The reader may argue that in this case the IS simulations offer no additional information; however, this is not so. The error approximation (7.32) implicitly assumes the use of a maximum-likelihood decoder which perfectly respects the ideal decision boundaries.

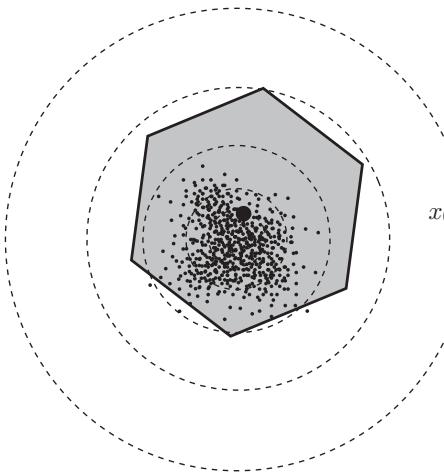


Figure 7.14: Illustration of the problem of overbiasing in applying importance sampling to the low-error evaluation problem.

However, the actual implementation of product codes does not use ML decoding for complexity reasons, but rather uses a form of iterative decoding and message passing between the row and column product codes. In this case then, since the importance sampling simulations are run with an iterative decoder, the IS results, in fact, confirm that the decoder will behave like a maximum-likelihood decoder even in the very low bit error regime.

Finally, we wish to discuss a couple of common pitfalls when applying importance sampling to the evaluation of bit and frame error rates. First, and most obvious, is the problem of *overbiasing*. This situation is graphically illustrated in Figure 7.14. What happens in this situation is that the center of the noise source is shifted too far into the erroneous decision region, highlighted as the grey hexagon in the figure. While in theory, and with sufficiently many samples, this does not pose a problem, in practice the estimate of the error probability becomes biased, since the majority of the IS samples are far from the decision boundary. That is, the area which produces the dominant errors and typically constitutes a very small sliver near the decision boundary is not explored by sufficient IS samples. The resultant error probability estimate is usually substantially smaller than the correct value.

There is typically no clear way of finding out the correct bias value; however, if we explore a few different bias values, it usually becomes evident very easily where the overbiasing phenomenon sets in. If a new bias value suddenly produces a smaller error estimate

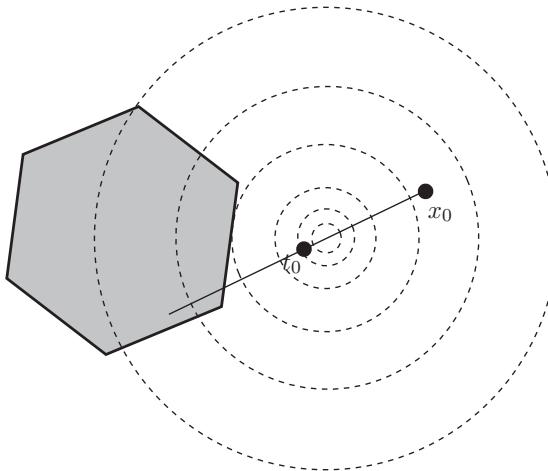


Figure 7.15: Illustration of linear continuous biasing.

than a previous bias value, we have entered the overbiasing situation. Finding a correct bias value which causes no error in the estimate and still produces a sufficient speed-up of the simulation is more of an art form than exact science and requires an attentive experimenter.

Some researchers [8] have alleviated this issue by biasing along a continuous line. That is, a conceptually infinite number of biasing points is explored. This is shown in Figure 7.15 where the bias is located along the line t connecting \mathbf{x}_0 and \mathbf{t}_0 . This method was first used by Richardson to quantify the effect of the so-called *trapping sets* of LDPC codes.

Since we can possibly use all the bias points along the line t , a finite number of points are generated according to the probability distribution of the noise component along the line t . Since the projection of a multi-dimensional Gaussian random variable onto a single dimension produces a standard 1-dimensional Gaussian random variable, we can therefore write

$$P_e = \frac{1}{\sqrt{2\pi\sigma^2}} \int_{-\infty}^{\infty} P_{e|s=x} \exp\left(-\frac{x^2}{2\sigma^2}\right) dx \quad (7.33)$$

The integral (7.33) has to be evaluated numerically, which produces \hat{P}_e as the IS estimate with various shifts as discussed above. Finding the point \mathbf{t}_0 is analogous to finding the direction of bias discussed earlier, i.e., some knowledge about the decision boundaries is required.

As a final point, we wish to emphasize the issue of “biasing away” from the dominant

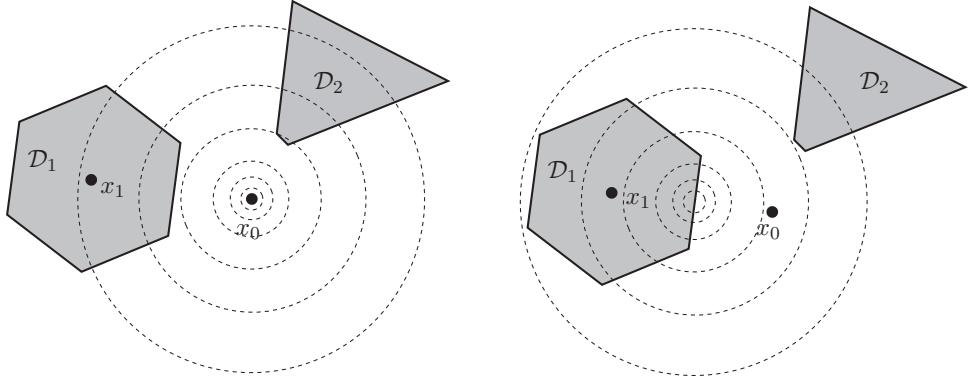


Figure 7.16: Illustration of the problem of biasing away from the dominant error region, which now sees a much reduced number of error events.

event. This is illustrated in Figure 7.16. The reader should note that the 2-dimensional rendition in the figure is a poor representation of what really happens; however, it does capture the idea of the problem. Assuming that there are two error regions \mathcal{D}_1 and \mathcal{D}_2 , a cursory examination may only identify \mathcal{D}_1 , even though \mathcal{D}_2 will dominate the error floor for higher values of the SNR, since its closest tip has a smaller Euclidean distance from the correct codeword \mathbf{x}_0 . However, if we now bias towards \mathcal{D}_1 , the reader will easily see that the probability of seeing any \mathcal{D}_2 errors is greatly diminished, and the experimenter may be fooled into thinking that the dominant error event has been found and identified. This underlines, once again, the importance of knowing the code's topology adequately in order to produce an accurate estimate of the error probabilities in the error floor regime. Recall our discussion around Figure 7.8 in this context.

All of these effects are visible only with a *finite* number of samples. The biasing theory assumes an infinite number of samples and produces no biased estimates as such. However, the purpose of importance sampling is to reduce the number of simulation runs, and therefore effects due to finite numbers of samples have to be taken seriously and carefully.

While the error mechanisms in turbo codes follow similar dynamics in principle, the fact that the node processors are stronger constituent codes than the simple repetition and parity-check codes in LDPC systems, we observe that non-codeword errors do not appear to make up a measurable portion of the error events. Turbo code design, therefore, has concentrated primarily on increasing the minimum free distance of the codes. This will be discussed in Chapters 8 and 9.

Bibliography

- [1] B.K. Butler and P.H. Siegel, “Error floor approximation for LDPC codes in the AWGN channel,” *IEEE Trans. Inform. Theory*, vol. 60, no. 12, pp. 7416–7441, Dec. 2014.
- [2] S.-Y. Chung, T.J. Richardson, and R. Urbanke, “Analysis of sum–product decoding of low-density parity-check codes using a Gaussian approximation,” *IEEE Trans. Inform. Theory*, pp. 657–670, Feb. 2001.
- [3] S. L. Howard, C. Schlegel, and V. C. Gaudet, “Degree-matched check node decoding for regular and irregular LDPCs,” *IEEE Trans. Circuits Syst. II: Express Briefs*, vol. 53, no. 10, pp. 1054 –1058, Oct. 2006.
- [4] M. Ivković, S. K. Chilappagari, and B. Vasić, “Eliminating trapping sets in low-density parity-check codes by using tanner graph covers,” *IEEE Trans. Inf. Theory*, vol. 54, no. 8, pp. 3763–3768, Aug. 2008.
- [5] J. Kang, Q. Huang, S. Lin, and K. Abdel-Ghaffar, “An iterative decoding algorithm with backtracking to lower the error-floors of LDPC codes,” *IEEE Trans. Inf. Theory*, vol. 59, no. 1, pp. 64–73, Jan. 2011.
- [6] G. B. Kyung and C.-C. Wang, “Finding the exhaustive list of small fully absorbing sets and designing the corresponding low error-floor decoder,” *IEEE Trans. Commun.*, vol. 60, no. 6, pp. 1487–1498, June 2012.
- [7] J. Sun, O. Y. Takeshita, and M. P. Fitz, “Analysis of trapping sets for LDPC codes using a linear system model,” in *42nd Annual Allerton Conference on Communications, Control and Computing*, Oct. 2004.
- [8] T.J. Richardson, “Error floors of LDPC codes,” *Proc. 2003 Allerton Conference on Communications, Control and Computing*, pp. 1426–1435, Oct. 2003.
- [9] C. Schlegel and S. Zhang, “On the dynamics of the error floor behavior in (regular) LDPC codes,” *IEEE Transactions on Information Theory*, vol. 56, pp. 3248–3264, June 2010.

- [10] T. Tian, C.R. Jones, J.D. Villasenor, and R.D. Wesel, “Selective avoidance of cycles in irregular LDPC code construction,” *IEEE Trans. Commun.*, vol. 52, no. 8, pp. 1242–1247, Aug. 2004.
- [11] M. Yang, W. E. Ryan, and Y. Li, “Design of efficiently encodable moderate-length high-rate irregular LDPC codes,” *IEEE Trans. Commun.*, vol. 54, no. 2, pp. 564–571, April 2004.
- [12] L. Dolecek, Z. Zhang, V. Anantharam, M. Wainwright, and B. Nikolic, “Analysis of absorbing sets and fully absorbing sets for array-based LDPC codes,” *IEEE Trans. Inf. Theory*, vol. 56, no. 1, pp. 181–201, Jan. 2010.
- [13] S. Zhang and C. Schlegel, “Controlling the error floor in LDPC decoding,” *IEEE Trans. Commun.*, vol 61, no. 9, pp. 3566–3575, Sept. 2013.

Chapter 8

Turbo Coding: Basic Principles

8.1 Introduction

The discovery of turbo codes introduced a paradigm shift in the error control coding discipline, which fundamentally changed the way it will be thought of and practiced. This shift moved focus away from maximum-likelihood decoding approaches and their approximations to message-passing decoding in the code's connectivity network, often represented by the code's graphical representation or the *Tanner graph* discussed earlier in this book. The presentation, at the International Conference on Communications in 1993, of implementable error control codes that achieve Shannon's bound to within half a dB of signal-to-noise ratio was met with a degree of skepticism by a community accustomed to the view that the Shannon bound represented an ideal, yet unachievable, limit.

The “near-capacity” performance of turbo codes and their novel iterative decoding algorithm thus stimulated an explosion of research efforts to understand this new coding scheme. From this emerged two fundamental questions regarding turbo codes. First, assuming optimum or near optimum decoding can be performed, why did turbo codes perform so well, and how does the iterative decoder work [23, 52, 80]?

In practical systems, coding gains have always been limited by the technology available for implementation. That is, while powerful error control codes at the time of the earliest applications of FEC coding were known, it was not feasible to implement them at the time. As technology advanced, so did the ability to implement complex coding schemes as demonstrated by the *Big Viterbi Decoder* (BVD), built by the Jet Propulsion Laboratory, which uses 2^{14} states [70], and represented the most complex decoder realized in hardware at that date. While the use of increasingly more complex coding schemes did reduce the gap between real system performance and theoretical limits (see Sections 1.5 and 1.6), these gains were not as dramatic as expected and a “law of diminishing returns” was suspected to be at work.

The significance of turbo codes thus can be appreciated upon observing their performance. Figure 8.1 shows simulation results of the original rate $R = 1/2$ turbo code presented in [19], along with simulation results of a maximum free distance (MFD) $R = 1/2$, memory $\nu = 14$ ((2, 1, 14)) convolutional code with Viterbi decoding. The convolutional code belongs to the family of codes chosen for the Galileo mission and decoded using the BVD. The turbo code outperforms the (2, 1, 14) code by 1.7 dB at a bit error rate (BER) of 10^{-5} . This comparison is stunning, especially since a detailed complexity analysis reveals that the complexity of the turbo decoder is, in fact, smaller than that of the BVD. The comparison of these simulation results raises two burning questions regarding the performance of turbo codes. What are the underlying processes that allow a turbo code to have a BER that achieves 10^{-5} at a signal-to-noise ratio (SNR) of 0.7 dB—a mere 0.5 dB away from the Shannon capacity for the BPSK constellation used—only to then flare out into an “error floor,” where it hardly improves even with massively higher signal power? While the turbo code is large, we will see that the implementation complexity of its decoder is nonetheless quite moderate and significantly smaller than that for the large convolutional code.

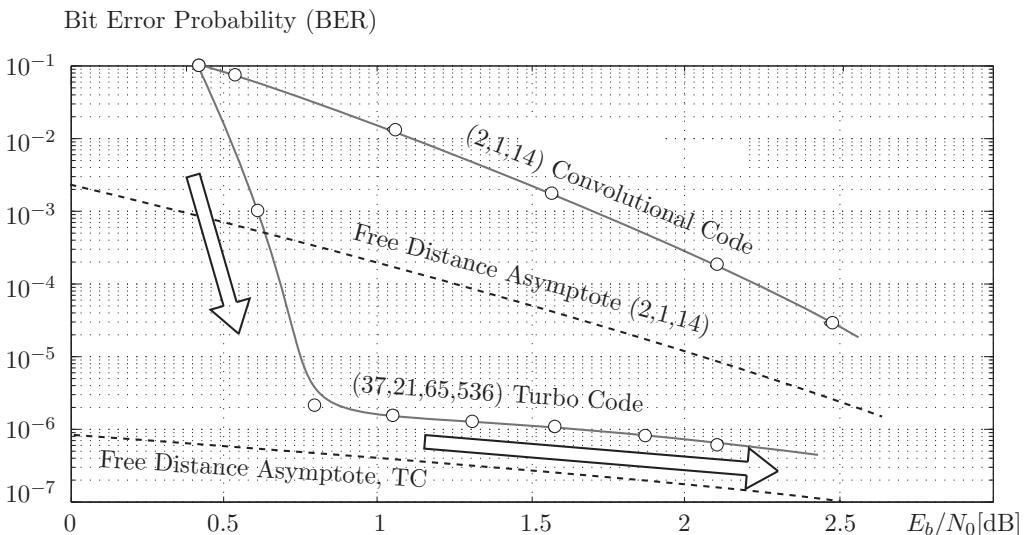


Figure 8.1: Simulated performance of the original turbo code and the code’s free distance asymptote and the simulated performance of the (2, 1, 14) MFD convolutional code and its free distance asymptote.

This chapter addresses turbo codes, or parallel concatenated convolutional codes (PCCCs), in detail. We begin with a simple example of a turbo encoder in order to introduce the fundamental structure of parallel concatenated convolutional codes. This is followed by a structural comparison of the convolutional $(2, 1, 14)$ code with a specific turbo code based on the original Berrou code. This comparison leads us to the conclusion that the distance spectrum, and in particular the free distance, discussed in Chapter 3 for trellis codes, is the appropriate tool to understand the behavior of turbo codes in the error floor region [55]. This effort leads to an interpretation that applies not only to turbo codes, but also lends insight into designing codes in general.

We then present a more general analysis of turbo codes using transfer functions and the notion of a probabilistic “uniform” interleaver. These techniques for analyzing the performance of turbo codes using transfer functions were pioneered in [9], [12], and [30]. This analysis then reveals basic design rules for good turbo codes.

Having addressed the performance in the error floor region, we then proceed to the study of the “turbo cliff” region. The performance of the codes in the turbo cliff region, in particular, the onset of the turbo cliff, requires an entirely different analysis tool. It is based on the statistical behavior of the component codes and is as such bound to the iterative decoding algorithm. Thus, at that point we take a break from analysis and discuss the details of the iterative decoding algorithm that makes these codes implementable. The iterative behavior of the component decoders will be captured by measuring their improvement of the mutual information at each iteration. These measurements will be quantified by the *extrinsic information exchange* (EXIT) charts [74]–[76], discussed later in this chapter in Section 8.7. Finally, we introduce serial concatenation as a turbo coding alternative and discuss the application of turbo coding methods to band-limited signaling using larger signal dimensions.

8.2 Parallel Concatenated Convolutional Codes

An encoder for a classical turbo code consists of the *parallel concatenation* of two or more, usually identical, rate $R = 1/2$ convolutional encoders, realized in recursive systematic or systematic feedback form,¹ and a pseudorandom interleaver as illustrated in Figure 8.2. This encoder structure is called a parallel concatenation because the two encoders operate on the same block of input bits as compared to traditional serial concatenation where the second encoder operates on the *output* bits of the first encoder. Let us consider turbo encoders with two identical component convolutional encoders, though this encoding method

¹The recursive systematic form is a special case of the controller canonical form while the systematic feedback form is the observer canonical form. See Sections 4.4 and 4.9.

can easily be extended to the case where different encoders or three or more encoders are used [29]. However, the major conclusions and performance results are achievable with only two component encoders.

The interleaver in Figure 8.2 is used to permute the input bits such that the two encoders are operating on the same block of input bits, but in a different order. The first encoder directly receives the input bit u_r and produces the output pair $(u_r, v_r^{(1)})$ while the second encoder receives the input bit u'_r and produces the output pair $(u'_r, v_r^{(2)})$, of which u'_r is discarded. The sequence of input bits are grouped into finite length blocks whose length, N , equals the size of the interleaver. Since both encoders are systematic and operate on the same set of input bits, it is only necessary to transmit the input bits once, and the overall code has rate $R = 1/3$. In order to increase the rate of the code to $R = 1/2$, the two parity sequences, $v^{(1)}(D)$ and $v^{(2)}(D)$, can be punctured, typically by alternately deleting $v_r^{(1)}$ or $v_r^{(2)}$. We will refer to a turbo code whose component encoders have parity-check polynomials $h_0(D)$ and $h_1(D)$, and whose *interleaver* is of length N , as an (h_0, h_1, N) turbo code, where h_0 and h_1 are the octal representations of the connector polynomials. Note that in the literature N is frequently called the block length of the code even though this terminology is inconsistent with traditional definitions of block length. In fact, the codelength $n = N/R$.

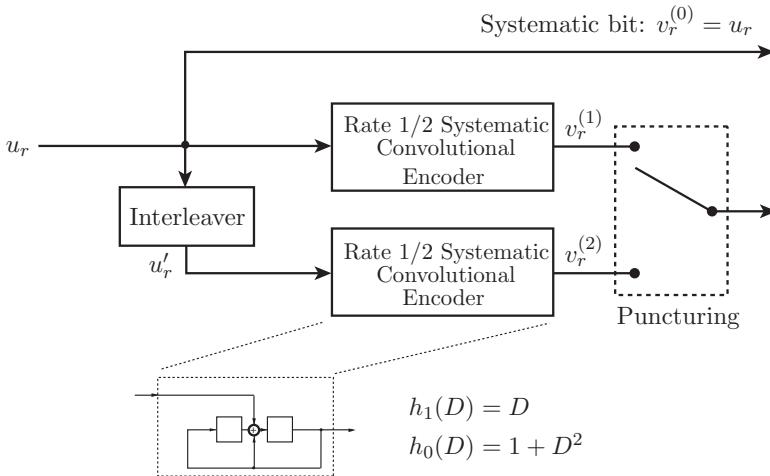


Figure 8.2: Block diagram of a turbo encoder with two component encoders and an optional puncturing device to increase the code rate above the base rate of $R = 1/3$.

In order to illustrate the basic principles of parallel concatenation, consider the encoder shown in Figure 8.2 with $(2, 1, 2)$ constituent encoders with parity-check polynomials $h_0(D) = 1 + D^2 \equiv 5$ and $h_1(D) = D \equiv 2$. For purposes of illustration, assume a small pseudorandom interleaver of size $N = 16$ bits which generates a $(5, 2, 16)$ turbo code. This interleaver can be realized as a length-16 buffer which is filled sequentially, with the input bits u_r . Once the buffer has been filled, the input bits to the second encoder, u'_r , are obtained by reading the interleaver in a pseudorandom manner until each bit has been read once and only once. The pseudorandom nature of the interleaver in our example is represented by the permutation $\Pi_{16} = \{15, 10, 1, 12, 2, 0, 13, 9, 5, 3, 8, 11, 7, 4, 14, 6\}$, which implies $u'_0 = u_{15}$, $u'_1 = u_{10}$, and so on.

If $\{u_0 \dots u_{15}\} = \{1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0\}$ is the input sequence, and the interleaver is represented by the permutation Π_{16} , then the sequence $\mathbf{u}' = \Pi_{16}(\{u_0 \dots u_{15}\}) = \{0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0\}$ is the input to the second encoder. The trellis diagrams for both constituent encoders with these inputs are shown in Figure 8.3. The corresponding unpunctured parity sequences are $\{0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0\}$ for the first encoder and $\{0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1\}$ for the second encoder. The resulting codeword has Hamming weight $d = w(\mathbf{u}) + w(\mathbf{v}^{(1)}) + w(\mathbf{v}^{(2)}) = 4 + 3 + 3 = 10$ without puncturing. If the code is punctured up to rate $R = 1/2$ beginning with $v_0^{(1)}$, then the resulting codeword has weight $d = 4 + 3 + 2 = 9$. If, on the other hand, the puncturing begins with $v_0^{(2)}$, then the punctured codeword has Hamming weight $4 + 0 + 1 = 5$.

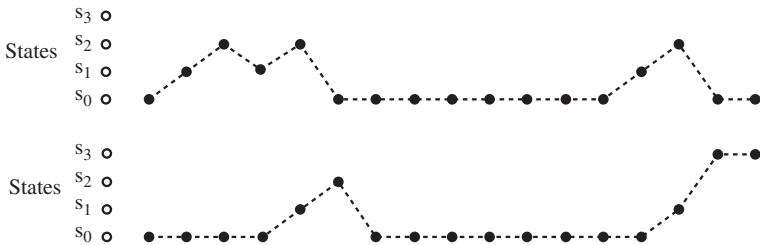


Figure 8.3: Examples of detours in the constituent encoders.

This simple example illustrates several salient points concerning the structure of the codewords in a turbo code. First, because the pseudorandom interleaver permutes the input bits, the two input sequences \mathbf{u} and \mathbf{u}' are almost always different, though of the same weight, and the two encoders will (with high probability) produce parity sequences of different weights. Second, it is easily seen that a codeword may consist of a number of distinct detours in each encoder, as illustrated in Figure 8.3. Since the constituent

encoders are realized in recursive systematic form a non-zero input bit is required to return the encoder to the all-zero state and thus all detours are associated with information sequences of weight 2 or greater, at least 1 for departure and 1 for the merger. Finally, with a pseudorandom interleaver it is highly unlikely that both encoders will be returned to the all zero state at the end of the codeword even when the last ν bits of the input sequence \mathbf{u} may be used to force the first encoder back to the all zero state, i.e., the trellis is terminated.

If neither encoder is forced to the all zero state, i.e., no tail bits are used to terminate the first encoder, then the sequence consisting of $N - 1$ zeroes followed by a one is a valid input sequence \mathbf{u} to the first encoder. For some interleavers, this sequence will be permuted to itself and $\mathbf{u}' = \mathbf{u}$. In that case, the maximum weight of the codeword with puncturing, and thus the free distance of the code, will be only two! For this reason, it is common to terminate the first encoder in the all-zero state. The ambiguity of the final state of the second encoder has been shown by simulation to result in negligible performance degradation for large interleavers [1, 57]. Special interleaver structures that result in both encoders returning to the all zero state are discussed in [3], [4], and [47].

Finding the free distance and distance spectrum of a turbo code is complicated by the fact that parallel concatenated convolutional codes are, in general, time-varying codes due to the interleaver. That is, for the shifted sequence² $Du(D)$ the first parity sequence is $Dv^{(1)}(D)$, but the input sequence to the second component encoder is *not* $Du'(D)$ (with high probability) due to the interleaver. Thus the second parity sequence is *not* $Dv^{(2)}(D)$. Continuing with the example in Figure 8.3, if $Du(D)$ is the input sequence, then $\Pi_{16}(Du(D)) = \{1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0\}$ and the second parity sequence is $\{0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0\}$. Thus, time shifting the input bits results in codewords that differ in both bit position and overall Hamming weight. In the next section we will see that this simple observation has a profound effect.

8.3 Distance Spectrum Analysis of Turbo Codes

In Chapter 5, the bounds on the performance of trellis codes were developed assuming an infinite-length trellis. The result is the bound of Equation (5.71) which gives the average information bit error rate *per unit time*. In order to make clear the distinction between turbo codes and convolutional codes, it is useful to consider both codes as block codes and to redevelop the performance bounds from this point of view. To this end, the information sequences are restricted to length N . With finite-length information sequences of length N , a $(2, 1, \nu)$ convolutional code may be viewed as a block code with 2^N codewords of length $2(N + \nu)$. The last ν bits input to the encoder are used to force the encoder back

²We will use the D -transform notation for sequences whenever more convenient, such as to describe shifting of sequences.

to the all zero state, that is, to *terminate* the encoder, and are referred to as the *tail*.

The bit error rate performance of a finite-length convolutional code with maximum likelihood (ML) decoding and binary antipodal signaling on an additive white Gaussian noise (AWGN) channel with an SNR of E_b/N_0 is upper bounded by

$$P_b \leq \sum_{i=1}^{2^N} \frac{w_i}{N} Q\left(\sqrt{d_i \frac{2RE_b}{N_0}}\right), \quad (8.1)$$

where w_i and d_i are the information weight and total Hamming weight, respectively, of the i th codeword. Collecting codewords of the same total Hamming weight and defining the average information weight per codeword as

$$\tilde{w}_d = \frac{w_d}{N_d},$$

where w_d is the total information weight of all codewords of weight d and N_d is the number, or multiplicity, of codewords of weight d , yields

$$P_b \leq \sum_{d=d_{\text{free}}}^{\infty} \frac{N_d \tilde{w}_d}{N} Q\left(\sqrt{d \frac{2RE_b}{N_0}}\right),$$

where d_{free} is the free distance of the code.

If a convolutional code has N_d^0 codewords of weight d caused by information sequences $u(D)$ whose first one occurs at time 0, then it also has N_d^0 codewords of weight d caused by the information sequences $Du(D)$, N_d^0 codewords of weight d caused by the information sequences $D^2u(D)$, and so on. Thus, for $d_{\text{free}} \leq d \leq 2d_{\text{free}} - 1$, we have

$$\lim_{N \rightarrow \infty} \frac{N_d}{N} = N_d^0$$

and

$$\lim_{N \rightarrow \infty} \tilde{w}_d = \lim_{N \rightarrow \infty} \frac{w_d}{N_d} = \frac{w_d^0}{N_d^0} \triangleq \tilde{w}_d^0,$$

where w_d^0 is the total information weight of all codewords with weight d which are caused by information sequences whose first one occurs at time 0. For $d \geq 2d_{\text{free}}$, the situation is somewhere more complicated due to the fact that codewords can consist of multiple error events. That is, as $N \rightarrow \infty$, $N_d = N \cdot N_d^0 + N_d(m)$ where $N_d(m)$ is the number of codewords of weight d made up of m error events. As we are primarily interested in low weight codewords, we will not try to develop more precise expressions for these multiplicities.

The bound on the BER of a finite length terminated convolutional code with ML decoding becomes

$$\begin{aligned} P_b &\leq \sum_{d=d_{\text{free}}}^{2 \cdot d_{\text{free}} - 1} N_d^0 \tilde{w}_d^0 Q\left(\sqrt{d \frac{2RE_b}{N_0}}\right) + \sum_{d=2 \cdot d_{\text{free}}}^{\infty} \frac{N_d \tilde{w}_d}{N} Q\left(\sqrt{d \frac{2RE_b}{N_0}}\right) \\ &= \sum_{d=d_{\text{free}}}^{2 \cdot d_{\text{free}} - 1} w_d^0 Q\left(\sqrt{d \frac{2RE_b}{N_0}}\right) + \sum_{d=2 \cdot d_{\text{free}}}^{\infty} \frac{N_d \tilde{w}_d}{N} Q\left(\sqrt{d \frac{2RE_b}{N_0}}\right), \end{aligned} \quad (8.2)$$

which is similar to the standard union bound for ML decoding as developed in Chapter 5. For this reason, efforts to find good convolutional codes for use with ML decoders have focused on finding codes that maximize the free distance d_{free} and minimize the number of free distance paths N_{free}^0 for a given rate and total encoder memory.

The performance of a turbo code with maximum likelihood decoding is also bounded by (8.1). Collecting codewords of the same total Hamming weight, the bound on the BER for turbo codes also becomes

$$P_b \leq \sum_{d=d_{\text{free}}}^{\infty} \frac{N_d \tilde{w}_d}{N} Q\left(\sqrt{d \frac{2RE_b}{N_0}}\right). \quad (8.3)$$

However, in the turbo encoder the pseudorandom interleaver maps the input sequence $u(D)$ to $u'(D)$ and the input sequence $Du(D)$ to the sequence $(Du'(D))'$ that is different from $Du'(D)$ with very high probability. Thus, unlike convolutional codes, the input sequences $u(D)$ and $Du(D)$ produce codewords with different Hamming weights. As a consequence of this, $N_d \tilde{w}_d$ is much less than N for low-weight codewords. This is due to the pseudorandom interleaver which maps low-weight parity sequences in the first component encoder to high weight parity sequences in the second component encoder. Thus, for low-weight codewords we typically find

$$\frac{\tilde{w}_d N_d}{N} \ll 1,$$

where

$$\frac{N_d}{N} \quad (8.4)$$

is called the *effective multiplicity* of codewords of weight d .

8.4 The Free Distance of a Turbo Code

For moderate and high signal-to-noise ratios, it is well known that the free distance term in the union bound on the bit error rate performance dominates the bound. Thus, the

asymptotic performance of convolutional and turbo codes with ML decoding approaches

$$P_b \approx \frac{N_{\text{free}} \tilde{w}_{\text{free}}}{N} Q\left(\sqrt{d_{\text{free}} \frac{2RE_b}{N_0}}\right), \quad (8.5)$$

where N_{free} is the error coefficient and \tilde{w}_{free} is the average weight of the information sequences causing free distance codewords. The right-hand side of Equation (8.5) and its associated graph is called the *free distance asymptote*, P_{free} , of a code.

As the error floor in Figure 8.1 occurs at moderate to high SNRs, we endeavor to find the free distance of turbo codes. Algorithms for finding the free distance of turbo codes are described in [69] and [37]. These algorithms were applied to a turbo code with the same component encoders, puncturing pattern, and interleaver size N as in [19] and a *particular* pseudorandom interleaving pattern. The parity-check polynomials for this code are $h_0 = D^4 + D^3 + D^2 + D + 1$ and $h_1 = D^4 + 1$, or $h_0 = 37$ and $h_1 = 21$ using octal notation. This (37, 21, 65, 536) code was found to have $N_{\text{free}} = 3$ paths with weight $d_{\text{free}} = 6$. Each of these paths was caused by an input sequence of weight 2 and thus $\tilde{w}_{\text{free}} = 2$. Though this result was for a particular pseudorandom interleaver, a similar result occurs for most pseudorandom interleavers with $N = 65,536$.

For this particular turbo code, the free distance asymptote is given by

$$P_{\text{free}} = \frac{3 \cdot 2}{65,536} Q\left(\sqrt{6 \frac{2 \cdot 0.5 \cdot E_b}{N_0}}\right),$$

where the rate loss due to the addition of a 4-bit tail to terminate the first encoder is ignored and

$$\frac{N_{\text{free}}}{N} = \frac{3}{65,536}$$

is the effective multiplicity. The free distance asymptote is shown plotted in Figure 8.1 along with simulation results for this code using the iterative decoding algorithm with 18 iterations, from which it can clearly be seen that the simulation results do in fact approach the free distance asymptote for moderate and high SNRs. Since the slope of the asymptote is essentially determined by the free distance of the code, it can be concluded that the error floor observed with turbo codes is due to the fact that they have a relatively small free distance and consequently a relatively flat free distance asymptote.

Further examination of expression (8.5) reveals that the manifestation of the error floor can be manipulated in two ways. First, increasing the length of the interleaver while preserving the free distance and the error coefficient will lower the asymptote without

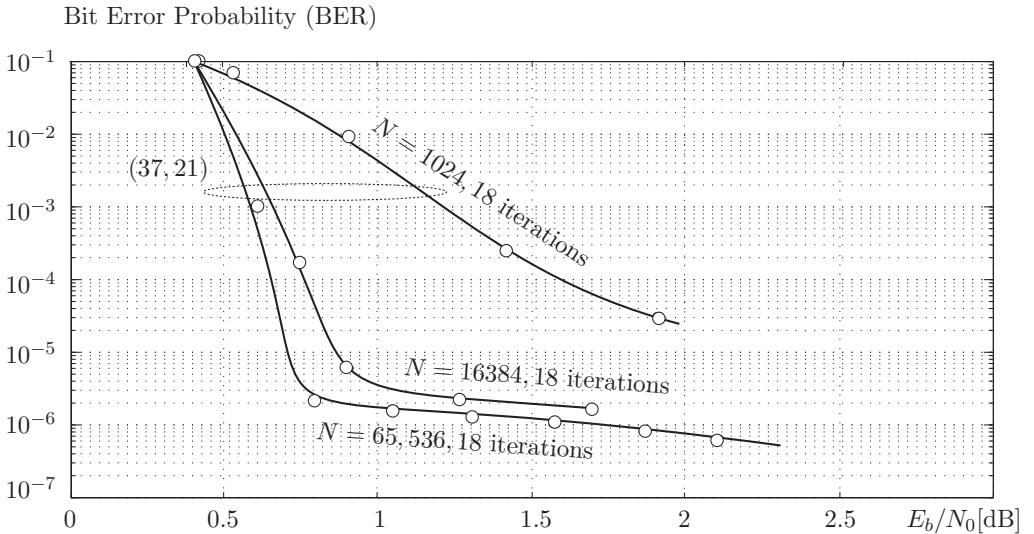


Figure 8.4: Simulations illustrating the effect of the interleaver size on the performance of a turbo code with random interleaving.

changing its slope by reducing the effective multiplicity. In this case, the performance curve of turbo codes does not flatten out until slightly higher SNRs and lower BERs are reached. Conversely, decreasing the interleaver size while maintaining the free distance and error coefficient results in the error floor being raised and the performance curve flattens at slightly lower SNRs and higher BERs. This can be seen in simulation results for the turbo code with varying N shown in Figure 8.4. Due to the steepness of the performance curve in turbo cliff region, very small changes in the SNR result in its onset.

If the first constituent encoder is not forced to return to the all-zero state and the weight 2 codewords described at the end of Section 8.2 are allowed, then the error floor is raised to the extent that the code performs poorly even for large interleavers. If the size of the interleaver is fixed, then the error floor can be modified by increasing the free distance of the code while preserving the multiplicity coefficient. This changes the slope of the free-distance asymptote. That is, increasing the free distance increases the slope of the asymptote and decreasing it decreases that slope. Thus, one cannot completely disregard free distance when constructing turbo codes. Techniques for increasing the free distance of turbo codes are discussed in the next section, and primarily depend on the interleaved and its design, which will be discussed in detail in the next chapter.

Note: If we compare the error floor of turbo codes to those for LDPC codes discussed in Chapter 6, we find that the mechanics of the former are much simpler. Even though the iterative decoding discussed shortly is not a maximum-likelihood method in the strict sense, the error events causing the error floor in turbo codes are nonetheless the low-weight codewords. This is in marked contrast to the LDPC codes, where the error floor was caused by a combination of structural and computational effects.

The role that the free distance and effective multiplicity play in determining the asymptotic performance of a turbo code is further clarified by examining the asymptotic performance of a convolutional code. The free distance asymptote of a convolutional code is given by the first term in the union bound of Equation (8.2). The maximum free distance $(2, 1, 14)$ code whose performance is shown in Figure 8.1 has $d_{\text{free}} = 18$, $N_{\text{free}}^0 = 18$, and $w_{\text{free}}^0 = 137$ [24]. Thus, the free distance asymptote for this code is

$$P_{\text{free}} = 137 \cdot Q\left(\sqrt{18 \frac{2 \cdot 0.5 \cdot E_b}{N_0}}\right),$$

which is also shown in Figure 8.1.

As expected, the free distance asymptote of the $(2, 1, 14)$ code is much steeper than the free distance asymptote of the turbo code due to the increased free distance. However, because the effective multiplicity of the free distance codewords of the turbo code, given by (8.4), is much smaller than the multiplicity of the $(2, 1, 14)$ code, the two asymptotes do not cross until the SNR is much greater than $E_b/N_0 = 2.5$ dB, and the BER of both codes is lower than the targeted BER of many practical systems. Thus, even though the $(2, 1, 14)$ convolutional code is asymptotically much better than the $(37, 21, 65, 536)$ turbo code, the latter substantially outperforms the former at error rates and SNR values at which many systems operate.

To emphasize the importance of using a pseudorandom interleaver with turbo codes, we now consider a turbo code with a regular rectangular interleaver. The same constituent encoders and puncturing pattern as in [19] are used in conjunction with a 120×120 rectangular interleaver. This rectangular interleaver is realized as a 120×120 matrix into which the information sequence \mathbf{u} is written row by row. The input sequence to the second encoder \mathbf{u}' is then obtained by reading the matrix column by column. A 120×120 rectangular interleaver implies an interleaver size of $N = 14,400$ and thus this is a $(37, 21, 14, 400)$ turbo code.

Using the algorithm described in [69], this code was found to have a free distance of $d_{\text{free}} = 12$ with a multiplicity of $N_{\text{free}} = 28,900!$ For this code, each of the free distance paths is caused by an information sequence of weight 4, so $\tilde{w}_{\text{free}} = 4$. The free distance asymptote for this code is thus given by

$$P_{\text{free}} = \frac{28,900 \cdot 4}{14,400} Q\left(\sqrt{12 \frac{2 \cdot 0.5 \cdot E_b}{N_0}}\right).$$

The free distance asymptote is plotted in Figure 8.5 along with simulation results using the iterative decoding algorithm of [19] with 18 iterations. This figure clearly shows that the free distance asymptote accurately estimates the performance of the code for moderate and high SNRs.

This code achieves a bit error rate of 10^{-5} at an SNR of 2.7 dB and thus performs 2 dB worse than the $(37, 21, 65, 536)$ turbo code with a pseudorandom interleaver even though it has a much larger free distance. The relatively poor performance of the $(37, 21, 14, 400)$ turbo code with a rectangular interleaver is due to the large multiplicity of d_{free} paths. This results in an effective multiplicity of

$$\frac{N_{\text{free}}}{N} = \frac{28,900}{14,400} \approx 2,$$

which is much larger than the effective multiplicity of the random $(37, 21, 14, 400)$ turbo code. We now show that the large multiplicity is a direct consequence of the use of the rectangular interleaver and that, furthermore, increasing the size of the interleaver does not result in a significant reduction in the effective multiplicity of the free distance codewords.

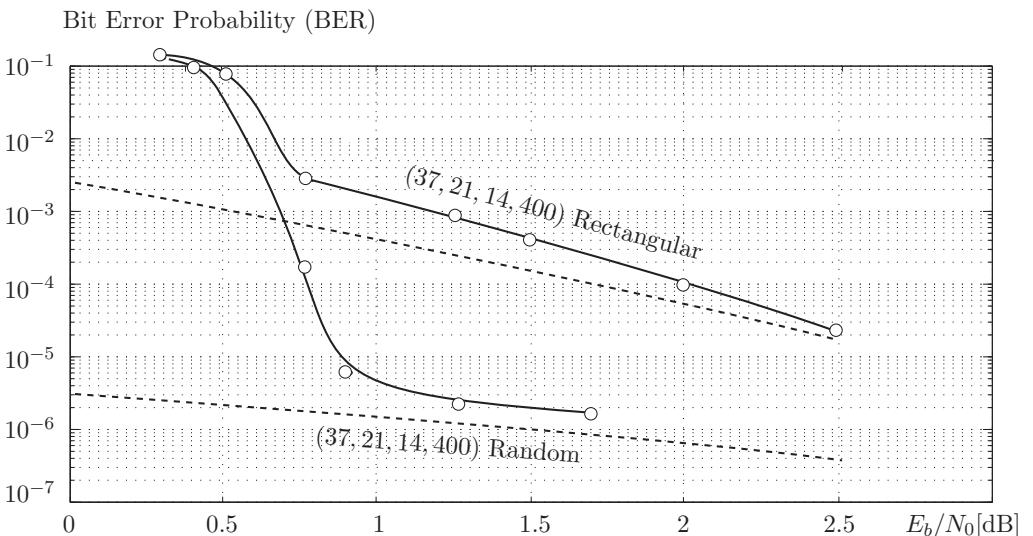


Figure 8.5: Comparison of simulated performance and free distance asymptote for a turbo code with a rectangular interleaver with that of a turbo code with a random interleaver.

The free distance paths in the turbo code with the rectangular interleaver are due to four basic information sequences of weight 4. These information sequences are depicted in Figure 8.6 as they would appear in the rectangular interleaver. The “square” sequence in Figure 8.6a depicts the sequence $\mathbf{u} = 1, 0, 0, 0, 0, 1, 0_{594}, 1, 0, 0, 0, 0, 1, 0_\infty$, where 0_{594} denotes a sequence of 594 consecutive zeroes and 0_∞ represents a sequence of zeroes that continues to the end of the information sequence. In this case, the rectangular interleaver maps the sequence \mathbf{u} to itself and therefore $\mathbf{u}' = \mathbf{u}$. The sequence \mathbf{u} results in a parity sequence $\mathbf{v}^{(1)}$ from the first constituent encoder which, after puncturing, has weight 4. Similarly, the input sequence $\mathbf{u}' = \mathbf{u}$ results in a parity sequence $\mathbf{v}^{(2)}$ from the second constituent encoder which, after puncturing, also has weight 4. The weight of the codeword is then $d_{\text{free}} = 4 + 4 + 4 = 12$. Since the “square” sequence in Figure 8.6a can appear in $(\sqrt{N} - 5) \times (\sqrt{N} - 5) = 13,225$ distinct positions in the rectangular interleaver, and in each case $\mathbf{u}' = \mathbf{u}$ and a codeword of weight $d_{\text{free}} = 12$ results, this results in 13,325 free distance codewords. Note that for every occurrence of the “square” sequence to result in a codeword of weight 12 the weight of both parity sequences must be invariant to which is punctured first.

The rectangular patterns in Figures 8.6b and 8.6c also result in weight 12 codewords. For the corresponding two input sequences, the weight of one of the parity sequences is affected by whether or not it is punctured first and only every other position in which the rectangular patterns appear in the interleaver results in a codeword of weight $d_{\text{free}} = 12$. Thus, the input sequences for the patterns in Figure 8.6b and Figure 8.6c each result in $0.5(\sqrt{N} - 10) \times (\sqrt{N} - 5) = 6,325$ free distance codewords. For the sequence in Figure 8.6d, the weight of both parity sequences is affected by which is punctured first and only one out of four positions in which the rectangular pattern appears in the interleaver results in a codeword of weight $d_{\text{free}} = 12$. Consequently, this sequence results in $0.25(\sqrt{N} - 10) \times (\sqrt{N} - 10) = 3,025$ free distance codewords. Summing the contributions of each type of sequence results in a total of $N_{\text{free}} = 28,900$ codewords of weight $d_{\text{free}} = 12$.

It is tempting to try to improve the performance of a turbo code with a rectangular interleaver by increasing the size of the interleaver. However, all of the information sequences shown in Figure 8.6 would still occur in a larger rectangular interleaver, so the free distance cannot be increased by increasing N . Also, since the number of free distance codewords is on the order of N , increasing the size of the interleaver results in a corresponding increase in N_{free} such that the effective multiplicity N_{free}/N does not change perceptibly. We thus have the somewhat strange situation that even if $N \rightarrow \infty$, the performance of the code in terms of bit error rate is practically unchanged. Without the benefit of a reduced effective multiplicity, the free distance asymptote, and thus the error floor, of turbo codes with rectangular interleavers is not lowered enough for them to manifest the excellent performance of turbo codes with pseudorandom interleavers. Attempts to design interleavers for turbo codes generally introduce structure to the interleaver and thus can destroy the very randomness that results in such excellent performance at low SNRs.

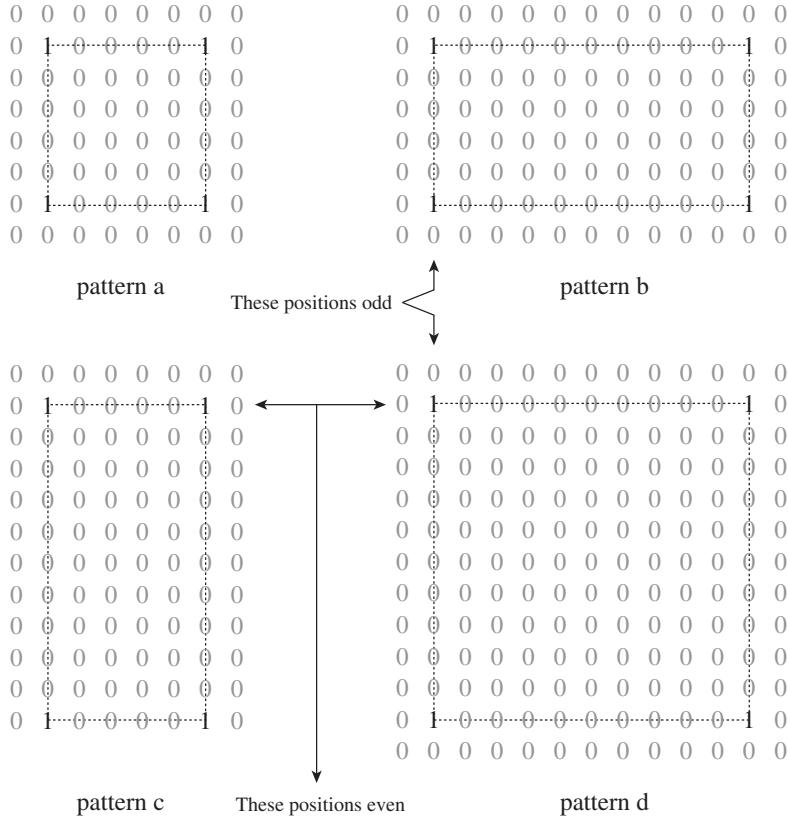


Figure 8.6: Input information sequences causing d_{free} codewords in a turbo code with a rectangular interleaver.

8.5 Weight Enumerator Analysis of Turbo Codes³

In this section we take a closer look at the distance spectrum of turbo codes and its effect on their performance in the error region of interest. We will be using the method of weight enumerator functions introduced in Section 5.9. For specific codes the distance

³This section can be omitted at first reading without loss of continuity.

spectrum has to be obtained via computer search. Though illuminating, finding even limited distance spectrum data is computationally expensive even for moderate interleaver lengths. In addition, the results for a particular turbo code do not easily generalize to the entire class of codes. In this section, the observations made concerning the distance spectrum are formalized using a weight enumerating function analysis. In order to simplify the notation and discussion, the turbo codes are used with identical component codes and without puncturing. The extension to punctured codes is straightforward [69]. As with convolutional codes and trellis codes, the primary result of this analysis will be insight into the design of the codes rather than computable bounds on the performance.

The bound of (8.3) requires knowledge of the complete distance spectrum of a turbo code. For parallel concatenated codes, it will be convenient to rewrite the distance spectrum information in the form of the *Input Output Weight Enumerating Function* (IOWEF)

$$A(W, X) = \sum_{d=d_{\text{free}}}^{\infty} \sum_w A_{w,d} W^w X^d, \quad (8.6)$$

where $A_{w,d}$ is the number of codewords of weight d caused by information sequences of weight w . Finding the IOWEF for conventional convolutional codes is frequently done via transfer function methods as was discussed in Chapter 3. Unfortunately, finding the exact transfer function of a turbo code is infeasible for all but trivial turbo codes due to the pair-wise state complexity introduced by the interleaver.

In order to obtain a more manageable technique, we will compute an approximate WEF based on the notion of a *probabilistic uniform interleaver*, which was introduced in [9, 11]. The fundamental idea of uniform interleaving is to consider the performance of a turbo code averaged over all possible pseudorandom interleavers of a given length. For a given N , there are $N!$ possible pseudorandom interleavers and, assuming a uniform distribution, each occurs with probability $\frac{1}{N!}$. Let a particular interleaver map an information sequence \mathbf{u} of weight w to an information sequence \mathbf{u}' , also of weight w . Then there are a total of $w!(N-w)!$ interleavers in the ensemble of $N!$ interleavers that perform this same mapping. Thus, the probability that such a mapping and, hence, that the codeword that results from the input sequences \mathbf{u} and \mathbf{u}' occurs is

$$\frac{w!(N-w)!}{N!} = \frac{1}{\binom{N}{w}}. \quad (8.7)$$

Note that, as was the case in the random coding arguments used to prove the Shannon capacity for trellis codes, the assumption of a uniform distribution of the interleavers is chosen in the random interleaving argument. The derivations that follow, however, do not constitute an existence argument as in the random coding bounds, since using different interleavers for different weight input sequences is not excluded in the analysis.

This probabilistic interleaving argument is now used to connect the codewords of the first component encoder in the turbo encoder to the codewords in the second component encoder. As (8.7) is a function of the weight of the *information* sequence, and parallel concatenation demands this, some additional weight enumeration functions are necessary.

Following [12], define the *Input Redundancy Weight Enumerating Function* (IRWEF) of a systematic component encoder as

$$A^C(W, Z) = \sum_w \sum_z A_{w,z}^C W^w Z^z,$$

where $A_{w,z}^C$ is the number of codewords of weight $d = w + z$ generated by input sequences of weight w and with parity sequences of weight z . Given the IRWEF of an encoder, the IOWEF is easily found by adding the information and parity weights.

Restricting attention to information sequences of a particular weight w results in the *Conditional Weight Enumerating Function* (CWEF)

$$A_w^C(Z) = \sum_z A_{w,z}^C Z^z.$$

$A_w^C(Z)$, then, is a polynomial representation of that portion of the distance spectrum of the component code which is due to input sequences of weight w . Given the CWEFs of a component encoder, its IRWEF is found as

$$A^C(W, Z) = \sum_w W^w A_w^C(Z).$$

The goal is now to develop a relationship between the CWEFs of the component encoders and the CWEF $A_w(Z)$ for the overall turbo code, which may be used to bound the bit error rate of the turbo code by

$$P_b \leq \sum_{w=w_{\min}}^N \frac{w}{N} W^w A_w(Z) \Big|_{W=Z=e^{-RE_b/N_o}}, \quad (8.8)$$

where w_{\min} is the minimum weight information sequence that generates a codeword in the *terminated convolutional* component code.

Using the uniform interleaver, the overall CWEF of a turbo code can now be computed as

$$A_w(Z) = \frac{A_w^C(Z) \cdot A_w^C(Z)}{\binom{N}{w}} = \frac{[A_w^C(Z)]^2}{\binom{N}{w}},$$

where, again, we have assumed identical component encoders. The CWEF can in turn be used to compute the transfer function bound described in Chapter 5 on the performance of

a *specific* code. However, if we seek insight into turbo code design and other more general results, additional manipulations are necessary.

This additional manipulation is primarily a consequence of the differences between viewing a convolutional code as having an infinite trellis and viewing a terminated convolutional code as a block code. In the former, codewords are frequently interpreted as error events, i.e., a path that diverges from the all zero state at a specific time and remerges for the first time some number of branches later. As discussed in Section 8.2, the latter viewpoint results in codewords that are the concatenation of detours. This is the viewpoint that we will find useful in the subsequent development.

In order to obtain a more useful expression for $A_w^C(Z)$, we begin by developing an approximate expression for the CWEF of the component encoders. Let

$$A_w^{(n)}(Z) = \sum_z A_{w,z}^{(n)} Z^z$$

be the n -event enumerating function of a component encoder. Here, $A_{w,z}^{(n)}$ represents the number of codewords consisting of the concatenation of n detours with total information weight w and total parity weight z . Unlike in Section 8.2, we further constrain concatenation to not allow strings of zeros between the detours. Finally, assuming that the length of the detours of interest are short compared to the interleaver length N , the CWEF of the component encoder can be approximated as

$$A_w^C(Z) \approx \sum_{n=1}^{n_{\max w}} \binom{N}{n} A_w^{(n)}(Z),$$

where $n_{\max w}$ is the maximum number of detours in a weight- w information sequence. These detours can now be positioned by an interleaver of length N to arbitrary locations.

The overall CWEF of the turbo code with uniform interleaving can now be approximated by the combination of the pieces from both encoders as

$$A_w(Z) \approx \sum_{n_1=1}^{n_{\max w}} \sum_{n_2=1}^{n_{\max w}} \frac{\binom{N}{n_1} \binom{N}{n_2}}{\binom{N}{w}} A_w^{(n_1)}(Z) A_w^{(n_2)}(Z).$$

This approximation can be further simplified by approximating the binomial coefficient by $N^n/n!$ and by approximating each term in the summations by the maximum values of n_1 and n_2 , respectively. Doing this yields

$$A_w(Z) \approx \frac{w!}{(n_{\max w}!)^2} N^{2n_{\max w}-w} [A_w^{n_{\max w}}]^2.$$

Finally, using this approximation in the bound of (8.8) leads to

$$P_b \approx \sum_{w=w_{\min}}^N W^w \cdot w \cdot \frac{w!}{(n_{\max_w}!)^2} N^{2n_{\max_w}-w-1} [A_w^{n_{\max_w}}(Z)]^2 \Big|_{W=Z=e^{-RE_b/N_o}} \quad (8.9)$$

The approximation (8.9), although unwieldy, is now in a form that can be used to develop design rules for turbo codes.

We begin by considering the effect of the choice of the component encoder realization on the performance of the overall code. If the component encoders are realized in non-systematic feedforward form, then $w_{\min} = 1$ and the maximum number of distinct error events is $n_{\max_w} = w$. Using this along with the approximation $A_w^{(w)}(Z) \approx [A_1^{(1)}(Z)]^w$ in (8.9) yields [12]

$$P_b \approx \sum_{w=1}^N W^w \frac{1}{(w-1)!} N^{w-1} [A_1^{(1)}(Z)]^{2w} \Big|_{W=Z=e^{-RE_b/N_o}}.$$

This expression reveals that for non-recursive component encoders, the exponent of the interleaver length N is always non-negative. That is, the effective multiplicity loses the factor of $1/N$, and one of the primary reasons for the low bit error rates of turbo codes is lost. This is intuitively satisfying since information sequences of weight 1 do, in fact, get interleaved to delayed versions of themselves regardless of the interleaver type and thus the resulting codewords will have high multiplicity. In addition, unlike the example in Section 8.2, the first and second encoders will then always generate the same parity sequence.

The result is quite different if recursive component encoders are used. In this case, $w_{\min} = 2$ and $n_{\max_w} = \lfloor w/2 \rfloor$. Following [12], the cases for w odd and even are treated separately. For $w = 2j + 1$ odd, it is straightforward to show that the summand in (8.9) becomes

$$(2j+1)(j+1) \cdot \binom{2j+1}{j} N^{-2} \cdot W^{(2j+1)} [A_{2j+1}^{(j)}(Z)]^2,$$

which is negligible for large interleavers due to the factor N^{-2} , and we will no longer consider odd weight sequences. For $w = 2j$, the summand becomes

$$(2j) \cdot \binom{2j}{j} N^{-1} \cdot W^{(2j)} [A_{2j}^{(j)}(Z)]^2,$$

which has an interleaver gain factor of N^{-1} . Thus, recursive component encoders manifest a reduced multiplicity and an interleaver gain for all input sequences. This leads to the first, well-accepted design rule for turbo codes.

Rule 1: Choose component encoders with feedback.

In order to develop the other design rules, attention is now focused only on those information sequences of even weight since they have the smallest interleaver gain. Approximation of (8.9) then becomes

$$P_b \approx \sum_{j=1}^{\lfloor N/2 \rfloor} (2j) \cdot \binom{2j}{j} N^{-1} \cdot W^{2j} [A_{2j}^{(j)}(Z)]^2 \Big|_{W=Z=e^{-RE_b/N_0}}. \quad (8.10)$$

The analysis of this expression is further simplified through use of the approximation $A_{2j}^{(j)}(Z) \approx A_2^{(1)}(Z)^j$, which reduces the problem to a consideration of codewords associated with information sequences of weight-2 only. It is now necessary to briefly digress and explore the structure of codewords generated by weight 2 information sequences in recursive encoder realizations.

In recursive encoders, codewords generated by weight 2 information sequences must correspond to detours with a first 1 required to leave the all-zero state and a second 1 required to return to the all-zero state. It is well known from the theory of linear feedback shift registers that blocks of additional intermediate zeroes must traverse a cycle, that is, a path beginning and ending in the same nonzero state, in the encoder state diagram. Denote by z_{cycle} the parity weight gained by traversing such a cycle. Then the overall parity weight of a codeword generated by a weight 2 information sequence is of the form

$$k \cdot z_{\text{cycle}} + t, \quad (8.11)$$

where k is an integer and t is parity weight achieved on the diverging and remerging transitions caused by the input 1s. Clearly, the minimum value of (8.11) is

$$z_{\min} = z_{\text{cycle}} + t,$$

which occurs when $k = 1$.

Returning to our analysis, we can now write

$$\begin{aligned} A_2^{(1)} &\approx Z^{z_{\min}} + Z^{2z_{\min}-t} + Z^{3z_{\min}-2t} \dots \\ &= \frac{Z^{z_{\min}}}{1 - Z^{z_{\min}-t}}, \end{aligned}$$

and substituting this expression into (8.10) yields

$$\begin{aligned} P_b &\approx \sum_{j=1}^{\lfloor N/2 \rfloor} (2j) \cdot \binom{2j}{j} N^{-1} \cdot W^{2j} [A_2^{(1)}(Z)]^{2j} \Big|_{W=Z=e^{-RE_b/N_0}} \\ &= \sum_{j=1}^{\lfloor N/2 \rfloor} (2j) \cdot \binom{2j}{j} N^{-1} \cdot W^{2j} \left[\frac{Z^{z_{\min}}}{1 - Z^{z_{\min}-t}} \right]^{2j} \Big|_{W=Z=e^{-RE_b/N_0}}. \end{aligned}$$

Finally letting $W = Z$ results in

$$P_b \approx \sum_{j=1}^{\lfloor N/2 \rfloor} (2j) \cdot \binom{2j}{j} N^{-1} \frac{(Z^{2+2z_{\min}})^j}{(1 - Z^{z_{\min}-t})^{2j}} \Big|_{Z=e^{-RE_b/N_o}}, \quad (8.12)$$

determining the quantity

$$d_{\text{eff}} = 2 + 2z_{\min},$$

which has been termed the *effective free distance* of the turbo code [12].

The expression (8.12) demonstrates that for large interleavers, the effective free distance is an important parameter affecting performance in the error floor, and it is desirable to maximize d_{eff} , which is equivalent to maximizing the minimum cycle weight z_{\min} . The desire to maximize z_{\min} results in two additional design rules:

Rule 2: Choose the feedback polynomial of the component encoders to be primitive.

This rule is based on the observation that the minimum cycle weight is likely to be larger for longer cycles. Borrowing once more from the theory of linear feedback shift registers, it is known that selecting the feedback polynomial to be primitive of degree ν results in the maximum possible cycle length of $2^\nu - 1$. A simple proof of this rule can be found in [12].

The third design rule is based on the observation that the effective free distance occurs when a weight 2 information sequence that generates parity weight z_{\min} in the first encoder is interleaved to an information sequence that generates parity z_{\min} in the second encoder. In order for this to occur, the original and interleaved information sequences must correspond to the shortest possible weight 2 sequence that generates a complete codeword in both component encoders. This situation can be avoided by designing the interleaver such that two 1's entering the first encoder which are separated by less than S positions are interleaved such that they are separated by more than T positions, where $T > S$. More precisely, let n_1 and n_2 be the indices of two 1s in the original information sequence and let $\pi(n_1)$ and $\pi(n_2)$ be the respective indices in the interleaved sequence. An interleaver is said to have *spreading factors* (S, T) if $(n_2 - n_1) < S$ implies $\pi(n_2) - \pi(n_1) \geq T$. If such an interleaver can be found and $T > z_{\min}$, then this can ensure that the resulting free distance is greater than effective minimum distance. Hence:

Rule 3: Choose an interleaver with a large spreading factor.

This design rule presents somewhat of a dilemma. As demonstrated in Section 8.3, interleavers with too much regularity, such as the ubiquitous row-column interleaver, may have good spreading factors, but actually result in poor overall code performance due to large multiplicities. On the other hand, a pseudorandom interleaver may produce a “thin” distance spectrum but have poor spreading factors and result in a code with a small free distance. The solution may be found in the so-called $S - \epsilon$ -random interleavers described in [32] which are algorithmically constructed. These interleavers begin as pseudorandom interleavers and are manipulated to try to achieve a desired set of spreading factors. More on interleavers and their design will be said in Chapter 9.

We now briefly compare the performance of some turbo codes to illustrate the design rules just developed. All of the codes considered are based on $(2, 1, 4)$ component codes with an interleaver length of $N = 5000$ and an overall code rate of $R = 1/2$. The same puncturing pattern was used for all the codes. The first code is a $(37, 21, 5000)$ code with a pseudorandom interleaver. This code uses the same component encoders as the original turbo code [19], which has nonprimitive feedback polynomials. This code has a free distance of $d_{\text{free}} = 6$ with a multiplicity of 1. The second code is a $(23, 33, 5000)$ code with the same pseudorandom interleaver as the first code. In this case, the component encoder uses a primitive feedback polynomial and has $d_{\text{free}} = 9$ with a multiplicity of 1. The last code is a $(23, 33, 5000)$ code with a spread interleaver designed following [8]. This code has a free distance of $d_{\text{free}} = 18$ with a multiplicity of 1.

The performance of all three of these codes is shown in Figure 8.7. Interestingly, for low SNRs the $(37, 21, 5000)$ code outperforms the other two codes by nearly 0.15 dB until it manifests the error floor expected from a code with $d_{\text{free}} = 6$. The performance of the $(23, 33, 5000)$ codes is similar for low and moderate SNRs and both codes outperform the $(37, 21, 5000)$ code for BERs below 10^{-5} . This is consistent with the higher free distance of these codes. The code with the spread interleaver eventually outperforms the code with the pseudorandom interleaver due to its larger free distance.

These results remind us that the design rules derived in this section, particularly Rule 2 and Rule 3, are based on approximations that are only valid for high SNRs and long interleaver lengths. The design rules and analysis to this point clearly do not tell the whole story. In particular, we wish to understand the turbo cliff and what causes this precipitous drop of the bit error rate that is the hallmark of turbo codes. To do this, we must first study the decoding algorithm, the subject of the next section.

8.6 Iterative Decoding of Turbo Codes

One quickly realizes that optimal decoding of turbo code is far too complex, and the discoverers of turbo codes never attempted either optimal decoding or an approximation

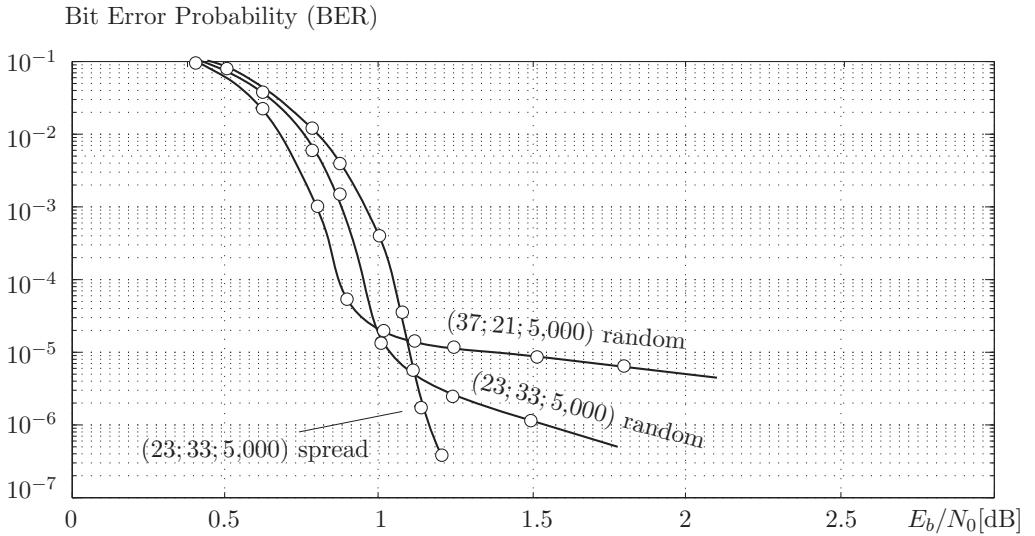


Figure 8.7: Performance of three different $R = 1/2$ turbo codes with an interleaverlength of $N = 5,000$ and 20 decoder iterations.

thereof, but instead proposed a novel iterative decoder based on the a posteriori probability (AAP) decoding algorithm of the component codes described in Chapter 5. Empirical evidence quickly suggested that this iterative decoding algorithm performed remarkably well in the sense that it almost always would find the optimum decoding solution.

Even though the optimal decision rule following the maximum a posteriori decoding principle is infeasible to implement for any code of reasonable size, it can nonetheless be used to motivate the iterative decoding procedure invented by Berrou and Glavieux [20]. To initiate the discussion, let us start with the maximum a posterior rule, which would select

$$\hat{u}_r = \max_{u=\{0,1\}} P[u_r = u | \mathbf{y}^{(0)}, \mathbf{y}^{(1)}, \mathbf{y}^{(2)}],$$

where $\mathbf{y}^{(0)}$ is the received systematic bit sequence and $\mathbf{y}^{(m)}$, $m = 1, 2$, are the received parity sequences corresponding to the m th constituent encoder.

This formula can be developed as follows:

$$\begin{aligned}
 P[u_r = u | \mathbf{y}^{(0)}, \mathbf{y}^{(1)}, \mathbf{y}^{(2)}] &\equiv P(\mathbf{y}^{(0)}, \mathbf{y}^{(1)}, \mathbf{y}^{(2)} | u_r = u) P[u_r] \\
 &= \sum_{\substack{\mathbf{u} \\ (u_r=u)}} P(\mathbf{y}^{(0)}, \mathbf{y}^{(1)}, \mathbf{y}^{(2)} | \mathbf{u}) P[\mathbf{u}] \\
 &= \sum_{\substack{\mathbf{u} \\ (u_r=u)}} P(\mathbf{y}^{(2)} | \mathbf{u}) P(\mathbf{y}^{(0)}, \mathbf{y}^{(1)} | \mathbf{u}) P[\mathbf{u}].
 \end{aligned} \tag{8.13}$$

The conditioning in the second term in the summation of (8.13) can be reversed using Bayes' rule to obtain

$$P(\mathbf{y}^{(0)}, \mathbf{y}^{(1)} | \mathbf{u}) P[\mathbf{u}] = P(\mathbf{y}^{(0)}, \mathbf{y}^{(1)}, \mathbf{u}) \equiv P[\mathbf{u} | \mathbf{y}^{(0)}, \mathbf{y}^{(1)}] \tag{8.14}$$

Clearly, computing (8.14) for all \mathbf{u} is far too complex, and we simply the expression using the *distribution separation*

$$P[\mathbf{u} | \mathbf{y}^{(0)}, \mathbf{y}^{(1)}] \approx \prod_{r=1}^L P[u_r | \mathbf{y}^{(0)}, \mathbf{y}^{(1)}]$$

where the values

$$P[u_r | \mathbf{y}^{(0)}, \mathbf{y}^{(1)}]$$

are seen to be the soft information produced by the first decoder which has access only to $\mathbf{y}^{(0)}, \mathbf{y}^{(1)}$. These are now used as *a priori probability information* in (8.13) delivered by decoder #1, to which decoder #2 adds the part from received sequence $\mathbf{y}^{(2)}$. The APP calculation of (8.13) can now be approximated by

$$\begin{aligned}
 P[u_r = u | \mathbf{y}^{(0)}, \mathbf{y}^{(1)}, \mathbf{y}^{(2)}] &\approx \sum_{\substack{\mathbf{u} \\ (u_r=u)}} P(\mathbf{y}^{(2)} | \mathbf{u}) \prod_{l=1}^L P[u_l | \mathbf{y}^{(0)}, \mathbf{y}^{(1)}] \\
 &= \sum_{\substack{\mathbf{u} \\ (u_r=u)}} P[\mathbf{u} | \mathbf{y}^{(2)}] \prod_{l=1}^L P[u_l | \mathbf{y}^{(0)}, \mathbf{y}^{(1)}] P[u_l]
 \end{aligned}$$

In Section 5.7, it was shown that this is precisely the quantity that the APP decoder for code #2 computes, using $P[u_l | \mathbf{y}^{(0)}, \mathbf{y}^{(1)}] P[u_l] \rightarrow P[u_l]$ as a priori probability of information bit u_l .

To proceed further it is useful to consider log-likelihood ratios of the probabilities in question, defined by

$$\Lambda(u_l) = \log \frac{P[u_l = 1 | \mathbf{y}^{(0)}, \mathbf{y}^{(1)}]}{P[u_l = 0 | \mathbf{y}^{(0)}, \mathbf{y}^{(1)}]}, \quad \Lambda_s(u_l) = \log \frac{P[u_l = 1]}{P[u_l = 0]}.$$

Furthermore, according to (5.27)

$$L(u_r) = \log \frac{P[u_r = 1 | \mathbf{y}^{(0)}, \mathbf{y}^{(1)}, \mathbf{y}^{(2)}]}{P[u_r = 0 | \mathbf{y}^{(0)}, \mathbf{y}^{(1)}, \mathbf{y}^{(2)}]}$$

can be calculated as

$$L(u_r) = \log \frac{\sum_{i,j \in A(u_r=1)} \gamma_r(j, i) \alpha_{r-1}(i) \beta_r(j)}{\sum_{i,j \in A(u_r=0)} \gamma_r(j, i) \alpha_{r-1}(i) \beta_r(j)}, \quad (8.15)$$

where the transition probability $\gamma_r(i, j)$ is given as

$$\gamma_r(i, j) = P[u_r | \mathbf{y}^{(0)}, \mathbf{y}^{(1)}] P(y_r^{(2)} | x_r^{(2)}), \quad (8.16)$$

where u_r is the information bit causing the transition from $i \rightarrow j$. Substituting (8.16) into (8.15) and factoring yields

$$\begin{aligned} L(u_r) &= \log \frac{\sum_{\substack{i,j \in A(u_r=1) \\ i,j \in A(u_r=0)}} P(y_r^{(2)} | x_r^{(2)}) \alpha_{r-1}(i) \beta_r(j)}{\sum_{i,j \in A(u_r=0)} P(y_r^{(2)} | x_r^{(2)}) \alpha_{r-1}(i) \beta_r(j)} + \log \frac{P[u_r = 1 | \mathbf{y}^{(0)}, \mathbf{y}^{(1)}]}{P[u_r = 0 | \mathbf{y}^{(0)}, \mathbf{y}^{(1)}]} \\ &= \Lambda_{e,r}^{(2)} + \Lambda_r, \end{aligned}$$

where $\Lambda_{e,r}^{(2)}$ is the *extrinsic information* contributed by the second decoder and Λ_r is the *a posteriori* log-likelihood ratio from the first decoder.

Applying the same decomposition to the *a posteriori* probabilities produced by the first decoder, we obtain

$$L(u_r) = \Lambda_{e,r}^{(2)} + \Lambda_{e,r}^{(1)} + \Lambda_s, \quad (8.17)$$

where

$$\Lambda_s = \log \frac{P(\mathbf{y}_r^{(0)} | u_r = 1)}{P(\mathbf{y}_r^{(0)} | u_r = 0)}$$

is the *a posteriori* probability of the systematic bits, which are conditionally independently distributed.

A block diagram of an iterative turbo decoder is shown in Figure 8.8, where each APP decoder corresponds to a constituent code and generates the corresponding extrinsic information $\Lambda_{e,r}^{(m)}$ for $m = 1, 2$ using the corresponding received sequences. The interleavers are identical to the interleavers in the turbo encoder and are used to reorder the sequences so that they are properly aligned at each decoder.

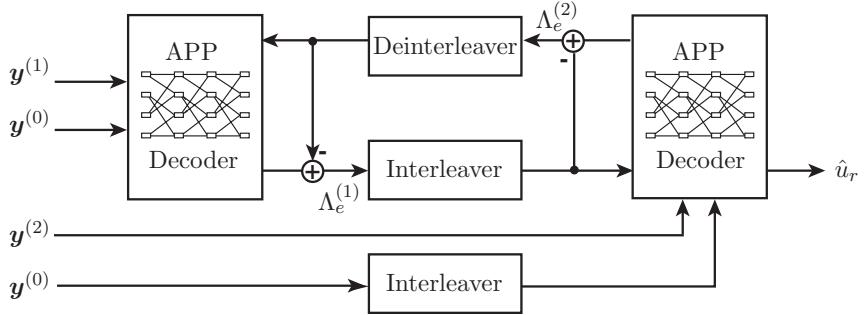


Figure 8.8: Block diagram of a turbo decoder with the two constituent code decoders.

Since the separation assumption destroyed optimality of the equation, the algorithm is iterated several times through the two decoders, each time the component decoder uses the currently calculated a posteriori probability as input. However, direct use of (8.17) would lead to an accumulation of “old” extrinsic information by calculating

$$L_r^{(1)\prime} = \Lambda_{e,r}^{(1)\prime} + \underbrace{\Lambda_{e,r}^{(2)} + \Lambda_{e,r}^{(1)}}_{L_r^{(1)}} + \Lambda_s .$$

Therefore the decoders are constrained to exchanging extrinsic information only, which is accomplished by subtracting the input values to the APP decoders from the output values as shown in Figure 8.8.

The extrinsic information is a reliability measure of each component decoder’s estimate of the transmitted information symbols based on the corresponding received component parity sequence only. Since each component decoder uses the received systematic sequence directly, the extrinsic information allows the decoders to share information without biasing. The efficacy of this technique can be seen in Figure 8.9, which shows the performance of the original (37; 21; 65,536) turbo code parameterized by decoder iterations. It is impressive that the performance of the code with iterative decoding continues to improve up to 18 iterations (and possibly beyond). In the next section we discuss a statistical analysis tool that will help shed light on the dynamics of iterative decoding of concatenated coding systems.

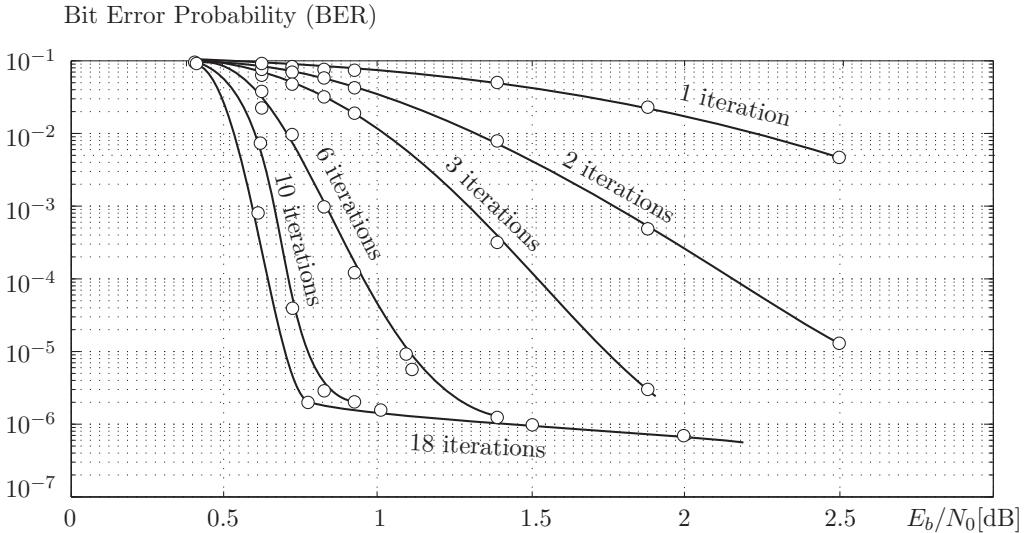


Figure 8.9: Performance of the (37;21;65,536) turbo code as function of the number of decoder iterations.

8.7 EXIT Analysis

As we have seen, the distance spectrum and the minimum distance of turbo codes can explain the error floor and, to some extent, the threshold behavior of these codes. The behavior of turbo codes at the onset of the turbo cliff is, however, better understood by a statistical analysis called the *extrinsic information transfer* (EXIT) analysis, first presented by ten Brink [74, 75, 76]. The EXIT analysis is related to similar methods which are based on the transfer of variances [33], as well as to density evolution analysis discussed in Chapter 6.

The basic philosophy EXIT is to view the component decoders of the turbo decoder (Figure 8.8) as a statistical processor which transforms an input value, i.e., the extrinsic LLR of the information symbols, into an output value, the recomputed extrinsic LLR. The component decoder is therefore seen as a nonlinear LLR processor, as indicated in Figure 8.10. The input LLR is denoted by $\Lambda_e^{(A)}$ since it takes on the function of an a priori probability, and the output extrinsic LLR is denoted by $\Lambda_e^{(E)}$.

Clearly we expect the component decoder to improve the extrinsic LLR in the course of the iterations, such that $\Lambda_e^{(E)}$ is better than $\Lambda_e^{(A)}$ in some sense, since otherwise iterations

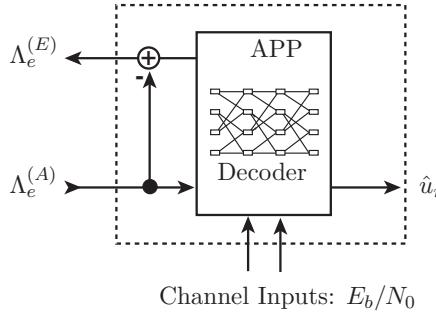


Figure 8.10: Component decoder seen as extrinsic LLR transformer.

will lead nowhere. The question is then how to measure the quality of the extrinsic LLR value. Experimental measurements of $\Lambda_e^{(m)}$ show that it is approximately Gaussian, i.e.,

$$\Lambda_e^{(m)} = \mu_\lambda u + n_\lambda, \quad n_\lambda \sim \mathcal{N}(0, \sigma_\lambda^2), \quad (8.18)$$

where $u \in \{-1, 1\}$ is the information bit whose LLR is expressed by $\Lambda_e^{(m)}$, μ_λ is a mean shift, and n_λ is a zero-mean independent Gaussian random variable with variance σ_λ^2 .

The question is now how to measure the reliability of $\Lambda_e^{(m)}$. The mutual information $I(u, \Lambda_e^{(m)})$ between $\Lambda_e^{(m)}$ and u has proven to be the most accurate and convenient measure, but the normalized variance has also been used in several papers. The mutual information measure has the following immediate advantages:

- The measure is bounded $0 \leq I(u, \Lambda_e^{(m)}) \leq 1$.
- The upper limit indicates high reliability, that is, the variance $\sigma_\lambda^2 \rightarrow 0$.
- The lower bound indicates low reliability, and $\sigma_\lambda^2 \rightarrow \infty$.
- The measure is monotonically increasing (with reliability).

In order to capture the input–output behavior of the component decoder, simulated extrinsic values $\Lambda_e^{(A)}$ are generated at its input, according to the Gaussian distribution (8.18) with independent realizations. This independence models the effect of the interleaver, which, ideally, destroys the statistical dependence between successive input LLR values that was introduced by the preceding component decoder.

In the case of Gaussian modelled input LLR values, the mutual information measure, in bits, can be calculated as

$$I_A = I(u; \Lambda_e^{(A)}) = \frac{1}{\sqrt{2\pi}\sigma_\lambda} \int_{-\infty}^{\infty} \exp\left(-\frac{(\lambda - \mu_\lambda)^2}{2\sigma_\lambda^2}\right) (1 - \log_2 [1 + \exp(-\lambda)]) d\lambda.$$

The mutual information of the output extrinsic LLR and the information bit u is more complex to evaluate since $\Lambda_e^{(E)}$ is not exactly Gaussian. Furthermore, the values of $\Lambda_e^{(E)}$ corresponding to $\Lambda_e^{(A)}$ have to be found via simulating the component decoder. Other than in very simple cases, such as the rate $R = 1/2$ repetition code, no closed form or even analytical formulas for the output extrinsic mutual information exist to date. The output extrinsic information I_E is therefore an empirical function of the component decoder, the input I_A , and the channel signal-to-noise ratio at which the decoder operates, formally given by the EXIT function

$$I_E = T(I_A, E_b/N_0).$$

To evaluate this function, the following steps are executed. First a Gaussian modelled input LLR with mutual extrinsic information value I_A is generated and then fed into the decoder, and the output extrinsic LLRs $\Lambda_e^{(E)}$ are captured. Second, a numerical evaluation of mutual information between the information symbols u and the captured extrinsic output LLRs $\Lambda_e^{(E)}$ yields I_E , calculated as

$$I_E = \frac{1}{2} \sum_{u=\pm 1} \int_{-\infty}^{\infty} p_E(\xi|u) \log_2 \left(\frac{2p_E(\xi|u)}{p_E(\xi|U=-1) + p_E(\xi|U=1)} \right) d\xi,$$

where $p_E(\xi|u)$ is the empirical distribution of $\Lambda_e^{(E)}$ as measured at the output of the APP decoder. Clearly, the disadvantage of this method lies in the fact that the component decoder needs to be simulated and that we are working with empirical distributions.

Figure 8.11 shows the EXIT function $T(I_A, E_b/N_0)$ for a memory 4, 16-state, rate $R = 2/3$ component convolutional code with $h_0(D) = 1 + D^3 + D^4$ and $h_1(D) = 1 + D + D^2 + D^3 + D^4$ (see Section 4.3). The numbers on the different trajectories are the E_b/N_0 values in dB.

Figure 8.12 shows the EXIT function $T(I_A, E_b/N_0)$ for several encoder memory sizes for a fixed signal-to-noise ratio of $E_b/N_0 = 0.8$ dB. It is interesting that no one memory order, i.e., no one code, is superior over the entire range of values of the mutual information. As a general trend, weaker codes are better when the input mutual information is poor, for I_A approximately less than 0.4, while stronger codes are better when the input I_A is already good. As we will see, this sheds light on why it is quite useless to try to build stronger turbo codes by using stronger component codes.

In a complete turbo decoder the extrinsic information is now exchanged between the component decoders, where output extrinsic LLRs become input extrinsic LLRs for the next decoder. In our analysis these iterations are captured by a sequence of applications of component decoder EXIT functions, that is,

$$0 \xrightarrow{T_1} I_E^{(1)} = I_A^{(2)} \xrightarrow{T_2} I_E^{(2)} = I_A^{(2)} \xrightarrow{T_1} I_E^{(1)} = I_A^{(2)} \xrightarrow{T_2} \dots ,$$

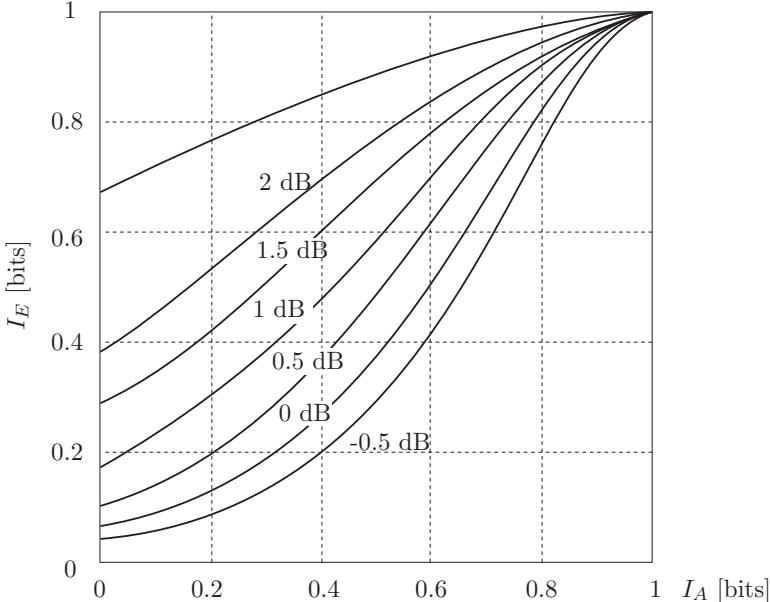


Figure 8.11: EXIT function of a 16-state convolutional code.

where T_1 is the EXIT function of decoder #1 and T_2 is that of decoder #2.

The behavior of this exchange can be visualized in the EXIT chart, where the EXIT functions of both component decoders are plotted. However, the transfer curve of decoder #2 is reflected about the 45° line to reflect that I_E is its input mutual information, and I_A is its output mutual information. As shown in Figure 8.13, for sufficiently large values of E_b/N_0 (in this case for $E_b/N_0 = 0.8$ dB), these two curves leave open a channel, and they have no intersect point other than (1,1). Since the iterations start with zero LLRs, i.e., with zero mutual information at point (0,0) in the chart, the iterations progress by reflecting from one curve to the other as illustrated in Figure 8.13. The first iteration through decoder #1 takes an input value of $I_A = 0$ and produces an output value of $I_E = 0.18$. This value then acts as the input to decoder #2, and after one complete iteration the mutual information has a value of 0.22, at point (1) in the plot. Analogously, the iterations proceed through points (2), (3), etc., until they reach $I_A = I_E = 1$ after about 11 iterations. The curves are drawn for the 16-state component decoders of the original turbo decoder, of which we know that it converges at $E_b/N_0 \approx 0.7$ dB, see Figure 8.9.

The trajectory plotted in Figure 8.13 is a measured characteristic of a single decoding

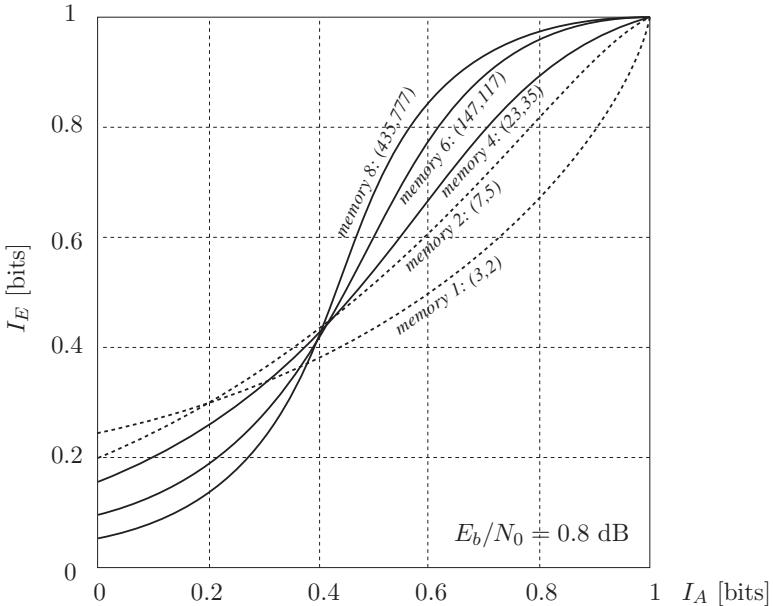


Figure 8.12: EXIT function for various convolutional codes.

cycle of the turbo decoder, and the accuracy with respect to the prediction of the EXIT chart is impressive, testifying to the robustness of the method. The growing discrepancies between the measured characteristic and the EXIT chart starting at iteration 7 are a result of the finite-sized interleaver, in this case $L = 60,000$ [74]. The larger the interleaver chosen, the more accurate the EXIT chart becomes as a prediction tool of the convergence properties of the code (see Figure 8.14).

It is worth noting that most of these iterations occur at very high bit error rates, i.e., $P_b > 10^{-2}$, in fact, $I_E \approx 0.9$ corresponds to $P_b = 10^{-2}$. This is interesting since it teaches us that the question of whether the decoder converges or not is decided early on, and that even though very little improvement may be seen in terms of bit error rates for many iterations, the decoder will converge to a low bit error rate eventually. It is clear from this analysis that the EXIT chart is a tool to evaluate component decoders, their cooperative statistical behavior such as onset of the turbo cliff, number of iterations to convergence, etc., but delivers no information about the error floor performance of a code and thus its ultimate operational performance.

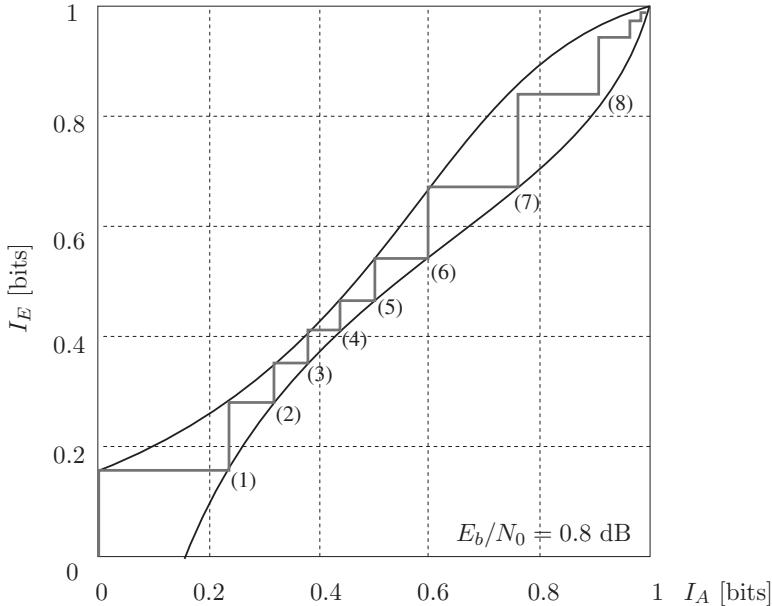


Figure 8.13: EXIT chart combining the EXIT functions of the two decoders involved in a turbo decoder for $E_b/N_0 = 0.8 \text{ dB}$.

As an example of using the EXIT chart to predict the onset of the turbo cliff behavior, we reduce the signal-to-noise ratio E_b/N_0 for the codes discussed until the open channel closes at $E_b/N_0 = 0.53 \text{ dB}$, which represents the *pinch-off* signal-to-noise ratio of this, the original turbo code combination of decoders. The sharpness of this pinch-off signal-to-noise ratio depends on the size of the interleaver, since finite-sized interleavers always leave some residual correlation, in particular for larger iteration numbers, which is in violation of the independence assumption used to generate the EXIT functions of the component decoders. Figure 8.14 shows the dramatic turbo cliff effect through simulations with different interleaver sizes using the 16-state component codes. It becomes evident that for very large interleaver sizes, the pinch-off signal-to-noise ratio accurately represents the onset of the turbo cliff.

ten Brink [74] has used this method to find the pinch-off signal-to-noise ratios for turbo codes using the component codes whose EXIT functions are given in Figure 8.12. He presented the following table of pinch-off values for the component code combinations:

ν	C_1/C_2	(3,2)	(7,5)	(13,15)	(23,37)	(67,45)	(147,117)
1	(3,2)	>2 dB					
2	(7,5)	1.49 dB	0.69 dB				
3	(13,15)	1.14 dB	0.62 dB	0.62 dB			
4	(23,37)	1.08 dB	0.65 dB	0.64 dB	0.68 dB		
5	(67,45)	0.86 dB	0.62 dB	0.66 dB	0.70 dB	0.77 dB	
6	(147,117)	0.84 dB	0.63 dB	0.67 dB	0.72 dB	0.81 dB	0.84 dB

Table 8.1: Table of pinch-off signal-to-noise ratios for combinations of component codes.

Table 8.1 illustrates some interesting facts. First, it clearly demonstrates that using larger codes, $\nu > 4$ does not give stronger turbo codes, in fact, a combination of two strong codes has a worse pinch-off value than that of the best combinations. Second, it illustrates the judicious choice of component codes chosen by the inventors of turbo codes, who chose the $\nu = 4$, (37,21) component codes which have a pinch-off signal-to-noise ratio of 0.53 dB (see Figure 8.14). Much searching [74] has yielded some code combinations with slightly better pinch-offs, for example, using two (22,37) codes with a pinch-off at 0.48 dB, or two (110,141), $\nu = 6$ codes with a pinch-off at 0.42 dB, but these are small improvements. Research into asymmetric code combinations has been carried out as well; see [14].

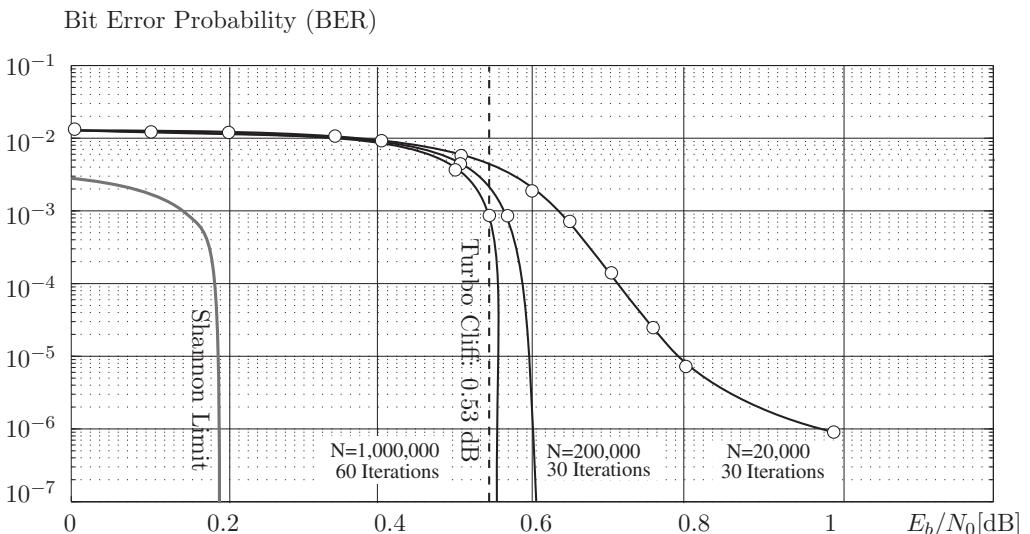


Figure 8.14: Simulations of the turbo code using 16-state component decoders and different sized interleavers related to theoretical prediction of the turbo cliff.

8.8 Serial Concatenation

So far we have shown that the parallel concatenation of convolutional codes coupled with an iterative decoding algorithm could perform near the capacity of an AWGN channel at virtually all practical error rates. This required careful selection of the component codes and the interleaver. Here we now investigate the performance of the historically more common serial concatenation of codes in which the output of one encoder becomes the input to the next encoder. We will see that the lessons and techniques learned from parallel concatenation can be applied to serial concatenation with a few minor adaptations.

Seriously concatenated codes were first considered in detail by Forney [36] as a means of constructing codes with long block lengths and decoding algorithms which could be built with the technology at the time. In classic serial concatenation, the second decoder typically operated on the hard decision output of the first decoder with no reliability (soft) information. This limited the performance of these codes, and the second decoder was frequently constrained to the role of “cleaning up” the residual errors of the first decoder. In fact, classic serial concatenation is one approach to mitigating the error floor of parallel concatenation [53], which is also used in the modern DVB-S2 standard [34] for broadband satellite application. Perhaps the most successful example of classic serial concatenation is the CCSDS standard which used a maximum free distance (MFD) (2, 1, 6) convolutional code as the inner code and various Reed–Solomon block codes as outer codes. Though still of considerable importance, this chapter will not discuss classic serial concatenation and the reader is referred to [36, 6] for a discussion thereof.

This chapter will discuss the design and performance of serial concatenated codes for decoding with an iterative soft-decision decoder. This coding and decoding scheme was first investigated in [38] and came to full fruition in the outpouring of work that followed the discovery of turbo codes. We begin with a description of serial concatenated convolutional codes (SCCCs) and the analysis of their performance assuming uniform interleaving and maximum-likelihood (ML) decoding. Next, a description is given of the modifications required to the iterative decoding algorithm of Section 8.6 for its use in a serially concatenated system. This is followed by a performance comparison of parallel and serial concatenated codes. Finally, we consider alternative serial concatenations, including repeat accumulator (RA) codes, whose outstanding performance at low signal-to-noise ratios (SNRs) is predicted by EXIT analysis.

8.9 Cascaded Convolutional Codes

Before we begin with the formal analysis of serial concatenated codes, it is worth considering a simple example of *cascaded* convolutional codes, which is where serial concatenation started. This will provide us with some intuition into the more analytical results that fol-

low, as well as regarding the differences between SCCC and other forms of concatenation. The block diagram of a simple SCCC is shown in Figure 8.15. The outer code is a rate $R_o = 1/2$ code with generator matrix

$$G_o(D) = [1 + D^2 \quad 1 + D + D^2], \quad (8.19)$$

which specifies a 4-state code with $d_{\text{free}}^o = 5$. The inner code is a rate $R_i = 2/3$ code with generator matrix

$$G_i(D) = \begin{bmatrix} 1 + D & D & 1 \\ 1 + D & 1 & 1 + D \end{bmatrix}, \quad (8.20)$$

which specifies a 4-state code with $d_{\text{free}}^i = 3$. As with the parallel concatenated codes of earlier in this chapter, the two encoders are separated by an interleaver.

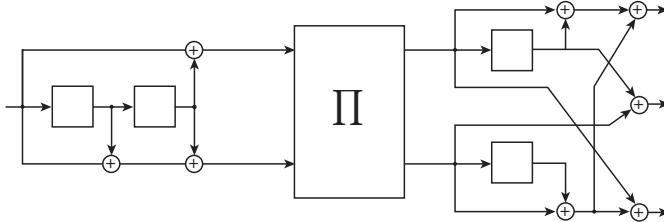


Figure 8.15: Block diagram of a simple serial concatenated convolutional code (SCCC2).

The first observation to make is that without an interleaver; i.e., if the interleaver is the identity permutation, the concatenated code simply becomes a convolutional code with overall generator matrix

$$G(D) = G_o(D)G_i(D) = [D + D^2 \quad 1 + D^2 + D^3 \quad D^2 + D^3], \quad (8.21)$$

which specifies an 8-state, rate $R = 1/3$ code with $d_{\text{free}} = 7$ that can be decoded using the maximum likelihood decoding algorithm described in Chapter 5. Concatenated codes constructed in this manner are referred to as *cascaded convolutional codes*. Such codes were studied before the discovery of turbo codes and had a particularly interesting application on the Galileo mission [31, 51].

Though the current chapter does not address cascaded codes, the cascaded code of (8.21) does give some insight into the distance structure of the more general class of serial concatenated convolutional codes. In particular, for a cascaded code the outer encoder serves as a “filter” that eliminates many input sequences to the inner encoder. In the current example, the outer encoder, by nature of its free distance, eliminates all input sequences to $G_i(D)$ of weight less than $d_{\text{free}}^o = 5$ and, by nature of its rate, eliminates

$2^N - 2^{N/2}$ of the possible input sequences of length N . Thus, the input sequences that generate many of the low-weight codewords in the inner encoder are eliminated by the outer encoder. This filtering operation should guarantee that the overall code will have a reasonable free distance and that free distance will increase as the component code complexity increases. However, the simple cascading of maximal-free distance codes by itself does not typically result in maximal-free distance concatenated codes; the corresponding maximal free distance of such codes is often rather modest.

Since the filtering operation is not affected by the presence of an interleaver, it is reasonable to anticipate that SCCC will not suffer from the low free distance and corresponding error floors that trouble turbo codes with random interleavers. This does not imply, however, that the interleaver has no role. Without the interleaver, serial concatenation is a technique for generating complex convolutional codes with relatively large, but not maximal, free distance. As shown in Chapter 5, large convolutional codes are asymptotically good but do not offer near-capacity performance at practical error rates. In serial concatenation, the interleaver achieves “spectral thinning” as well as introducing the randomness necessary for effective iterative decoding. In order to fully understand the role of the interleaver in the design of SCCC, we now embark upon a more detailed analysis again employing the concept of uniform interleaving.

8.10 Weight Enumerator Analysis of SCCC

A block diagram of a generic serial concatenated convolutional encoder is shown in Figure 8.16. The outer encoder is of rate $R_o = k/m$ with encoder memory ν_o , and the inner encoder is of rate $R_i = m/n$ with encoder memory ν_i . The interleaver is of length N and it will be assumed for clarity of notation that N is divisible by m . The overall rate of the SCCC is thus $R = R_o \times R_i = k/n$ if we ignore the effects of trellis termination. As with parallel concatenated convolutional codes (PCCCs), finding the exact weight enumerating function (WEF) for a SCCC is infeasible due to the complexity of the resulting hypertrellis and it will be sufficient for our purposes to find an approximation [16].

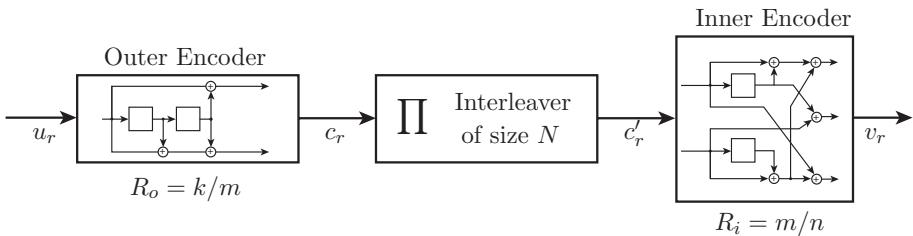


Figure 8.16: Block diagram of a serial concatenated convolutional code.

Recall from Section 8.5 that the Input–Output Weight Enumerating Function (IOWEF) of a code may be written as

$$A(W, X) = \sum_{d=d_{\text{free}}}^{\infty} \sum_w A_{w,d} W^w X^d, \quad (8.6)$$

where $A_{w,d}$ was the number of codewords of weight d caused by information sequences of weight w . Considering only codewords caused by information sequences of a certain weight results in the Conditional Weight Enumerating Function (CWEF)

$$A_w(X) = \sum_d A_{w,d} X^d,$$

and the probability of an information bit error may then be expressed as

$$P_b \leq \sum_{w=w_{\min}}^{NR_o} \frac{w}{NR_o} A_w(X) \Big|_{X=e^{-RE_b/N_o}}, \quad (8.22)$$

where w_{\min} is the minimum information weight and NR_o is the maximum information weight generating a codeword in the overall code. For serial concatenated codes, it is useful to rewrite this bound as

$$P_b \leq \sum_{d=d_{\text{free}}}^{N/R_i} \sum_{w=w_{\min}}^{NR_o} \frac{w}{NR_o} A_{w,d} e^{-dRE_b/N_o}, \quad (8.23)$$

where d_{free} is the free distance of the overall code and N/R_i is the length of the codewords and thus the largest possible codeword weight. Note that in these expressions the weight of the codeword is not separated into a parity weight and an information weight as was required in the case of parallel concatenation.

In order to derive the CWEF of the overall serial concatenation in terms of the CWEFs of the component codes, we will once again make use of the uniform interleaver. In parallel concatenation, a particular CWEF of the overall code is related to the CWEFs of the two component codes for the same information weight. In serial concatenation, however, the CWEFs of the two component codes are linked by the weight of the output of the outer encoder, which becomes the input to the inner encoder. Thus, assuming a uniform interleaver, we obtain

$$A_{w,d} = \sum_{l=d_{\text{free}}^o}^N \frac{A_{w,l}^o \cdot A_{l,d}^i}{\binom{N}{l}} \quad (8.24)$$

and

$$A_w(X) = \sum_{l=d_{\text{free}}^o}^N \frac{A_{w,l}^o \cdot A_l^i(X)}{\binom{N}{l}},$$

where d_{free}^o is the free distance of the outer code and l is the weight of the codeword out of the outer encoder. Given the CWEF of the overall code, a bound on the bit error rate of the code may be obtained using (8.22).

As with parallel concatenated codes, we will find it convenient to derive an approximate expression for these weight enumerators. Let

$$A_w^{(n)}(X) = \sum_d A_{w,d}^{(n)} X^d$$

be the n -event weight enumerating function of the equivalent block code of a terminated convolutional code, where $A_{w,d}^{(n)}$ is the number of codewords of weight d due to information sequences of weight w and which consist of n distinct detours. For large interleaver lengths N , the number of codewords of weight d due to information sequences of weight w may be approximated as

$$A_{w,d} \approx \sum_{n=1}^{n_{\max w}} \binom{N}{n} A_{w,d}^{(n)}, \quad (8.25)$$

where $n_{\max w}$ is the maximum number of detours due to information sequences of weight w that can be concatenated in a block of length N .

Applying the approximation of (8.25) to the outer and inner code of the serial concatenated system yields

$$A_{w,l}^o \approx \sum_{n^o=1}^{n_{\max w}^o} \binom{N/m}{n^o} A_{w,l}^{(n^o)}$$

and

$$A_{l,d}^i \approx \sum_{n^i=1}^{n_{\max l}^i} \binom{N/m}{n^i} A_{l,d}^{(n^i)},$$

respectively. Substituting these two approximations into (8.24) leads to

$$A_{w,d} = \sum_{l=d_{\text{free}}^o}^N \sum_{n^o=1}^{n_{\max w}^o} \sum_{n^i=1}^{n_{\max l}^i} \frac{\binom{N/m}{n^o} \binom{N/m}{n^i}}{\binom{N}{l}} A_{w,l}^{(n^o)} A_{l,d}^{(n^i)}.$$

We will once again make use of the approximation

$$\binom{N}{n} \approx \frac{N^n}{n!}, \quad (8.26)$$

which results in

$$A_{w,d} = \sum_{l=d_{\text{free}}^o}^N \sum_{n^o=1}^{n_{\max_w}^o} \sum_{n^i=1}^{n_{\max_l}^i} N^{n^o+n^i-l} \frac{l!}{m^{n^o+n^i} n^o! n^i!} A_{w,l}^{(n^o)} A_{l,d}^{(n^i)}. \quad (8.27)$$

Finally, using (8.27) in the bound of (8.23) results in

$$P_b \leq \sum_{d=d_{\text{free}}}^{N/R_i} \sum_{w=w_{\min}}^{N \cdot R_o} \sum_{l=d_{\text{free}}^o}^N \sum_{n^o=1}^{n_{\max_w}^o} \sum_{n^i=1}^{n_{\max_l}^i} N^{n^o+n^i-l-1} \frac{l! A_{w,l}^{(n^o)} A_{l,d}^{(n^i)}}{m^{n^o+n^i-1} n^o! n^i!} \frac{w}{k} e^{-dR E_b / N_o}. \quad (8.28)$$

A judicious examination of this bound yields considerable insight into the design of SCCC s for moderate and high SNRs. Indeed, (8.28) is the key to understanding the design rules for SCCC s.

First, we notice that the inner four summations in (8.28) only effect the multiplicity of codewords of weight d . The outer summation and exponential term are a function of the distance properties of the code and are independent of the inner four summations. As the SNR becomes very large, only the d_{free} term is of consequence and it has the form of the decaying exponential times a multiplicity that is dominated by $N^{n^o+n^i-l-1}$. We begin by considering the maximum value of the exponent for the d_{free} term

$$\alpha(d_{\text{free}}) = \max_{w,l} \{n^o + n^i - l - 1\}$$

for interleavers of length N . Making the reasonable assumption that the free distance codeword of the overall code is a single error event in the inner code and that this error event is caused by a codeword from the outer encoder of minimum weight d_{free}^o , then $n^i = 1$, $n^o = 1$ and $l = d_{\text{free}}^o$ and

$$\alpha(d_{\text{free}}) = 1 - d_{\text{free}}^o.$$

Thus, provided that $d_{\text{free}}^o \geq 2$, the SCCC will exhibit an interleaver gain as the SNR gets large and the outer code should simply be chosen to have large free distance.

As was observed with parallel concatenated codes, asymptotic design rules and large free distance do not fully capture the characteristics of a good code and this is true of serial concatenation as well. It was observed in [16] and [18] that the bound of (8.28) becomes loose compared to simulation results at higher SNRs as the interleaver length

N increases. That is, for a given N there exists an SNR at which the multiplicity of higher-weight codewords is more significant than the free distance event. As N increases and the code becomes spectrally dense, this threshold SNR moves away from the capacity of the channel.

This is consistent with observations made earlier on concerning the performance of maximum free distance convolutional codes with increasing memory. In order to characterize this effect, the authors in [16] consider finding the maximum value of the exponent of N regardless of codeword weight. This may be expressed as

$$\alpha_{\max} = \max_d \{n^o + n^i - l - 1\} = \max_{w,l} \{n^o + n^i - l - 1\}$$

and we are primarily concerned with whether or not α_{\max} is greater than or less than zero.

Following [16], we first consider the case of inner encoders that do not employ feedback. For these encoder realizations, an error event may be caused by an information sequence of weight 1 and thus a sequence of weight l from the outer encoder may cause l distinct detours in the inner code. When this occurs, $n^i = l$ and the exponent is maximized by $n^o = n_{\max}^o$, which results in

$$\alpha_{\max}(\text{no feedback}) = n_{\max}^o + l - l - 1 = n_{\max}^o - 1 \geq 0, \quad (8.29)$$

and the corresponding multiplicity increases rapidly as N increases. The bound of (8.28) will then diverge from the free distance asymptote at higher SNRs, which suggests that the code will perform poorly at low and moderate SNRs.

If the outer encoder generates a codeword with weight d_{free}^o , then $n^o = 1$, $l = d_{\text{free}}^o$ and, for random interleavers, $n^i = d_{\text{free}}^o$. The corresponding exponent for N is then

$$\alpha_{d_{\text{free}}}(\text{feedforward}) = 1 + d_{\text{free}}^o - d_{\text{free}}^o - 1 = 0 \quad (8.30)$$

and the interleaver does not affect the multiplicity due to $N^{\alpha_{\max}} = 1$. Notice that in this situation the concatenated code generates a codeword of weight $d = d_{\text{free}}^o d_{\text{free}}^i$ and the concatenated code is similar to a product code! Indeed, for interleavers on the order of several hundred bits, a search has verified that the free distance of this code combination is equal to the product distance. Thus, the interleaver provides a significant improvement in free distance compared to cascaded convolutional codes, but no “turbo code effect” yet.

For encoders utilizing feedback, the minimum weight of an information sequence that generates a finite-length error event is two. Consequently, a sequence of weight l out of the outer encoder can cause at most $\lfloor l/2 \rfloor$ error events in the inner encoder and the exponent becomes

$$\alpha_{\max} = \max_{w,l} \left\{ n^o + \left\lfloor \frac{l}{2} \right\rfloor - l - 1 \right\}.$$

For an outer code with free distance d_{free}^o it is clear that the maximum number of error events in a codeword of weight l is bounded by

$$n_{\max}^o \leq \left\lfloor \frac{l}{d_{\text{free}}^o} \right\rfloor$$

and the exponent may accordingly be bounded by

$$\alpha_{\max} \leq \max_l \left\{ \left\lfloor \frac{l}{d_{\text{free}}^o} \right\rfloor - \left\lfloor \frac{l+1}{2} \right\rfloor - 1 \right\}, \quad (8.31)$$

which is now a function of only l . In order to perform the maximization in (8.31), we observe that the weight l of a codeword produced by the outer encoder must satisfy

$$id_{\text{free}}^o \leq l < (i+1)d_{\text{free}}^o \quad (8.32)$$

for some integer i . The maximum exponent occurs when $l = d_{\text{free}}^o$ and for this case

$$\alpha_{\max}(\text{feedback}) = - \left\lfloor \frac{d_{\text{free}}^o + 1}{2} \right\rfloor, \quad (8.33)$$

which implies that the multiplicities of all codewords see an interleaver gain provided the free distance of the outer code is greater than two and the inner encoder utilizes feedback. This yields the first design rule for SCCCs.

Rule 1: Choose the inner encoder should be in feedback form.

Further insight into the design of SCCCcs may be gained by considering the minimum weight of the codeword, denoted by d_α , associated with α_{\max} . The derivation of (8.33) required that the number of error events in the inner code be maximized. For d_{free}^o even, this means that there are $d_{\text{free}}^o/2$ error events, each due to a weight 2 input. Let $d_{\min}^{(2)}$ be the minimum weight of codewords in the inner code that are due to weight 2 input sequences. Then the minimum weight of the codeword associated with the maximum multiplicity exponent is

$$d_\alpha = \frac{d_{\text{free}}^o d_{\min}^{(2)}}{2}. \quad (8.34)$$

If d_{free}^o is odd, then there are $(d_{\text{free}}^o - 3)/2$ error events due to weight 2 inputs and one error event due to a weight 3 input. In this case

$$d_\alpha = \frac{(d_{\text{free}}^o - 3)d_{\min}^{(2)}}{2} + d_{\min}^{(3)}, \quad (8.35)$$

where $d_{\min}^{(3)}$ is the minimum weight of codewords in the inner code due to weight 3 input sequences. Clearly it is desirable to have an inner code with large $d_{\min}^{(2)}$, i.e., a large effective free distance. From parallel concatenation we know that this is accomplished by choosing the feedback polynomial to be primitive, and this becomes the second design rule.

Rule 2: *Maximize inner encoder effective free distance by choosing its feedback polynomial to be primitive.*

These two design rules are the same rules used to choose good component encoders for parallel concatenated convolutional codes, and thus encoders optimized for turbo codes are also excellent candidates for inner codes in serially concatenated systems. In choosing the outer encoder, the primary concern is simply the free distance d_{free}^o and thus traditionally good convolutional codes are suitable. It is well known that with traditional convolutional codes, non-recursive encoder realizations have a slight advantage in bit error rate performance for large SNRs due to the mapping from information sequences to codewords, and this holds for the outer encoder in serially concatenated systems as well.

A final design guideline is based on careful consideration of (8.35) which shows that d_α is based on both the effective free distance of the inner code and $d_{\min}^{(3)}$ when d_{free}^o is odd. If the inner code can be chosen such that no odd weight input sequences generate finite length codewords, then $d_{\min}^{(3)} = \infty$ and $d_\alpha = \infty$. This is easily achieved by choosing the feedback polynomial of the inner encoder to have a factor of $(1 + D)$. It should be noted that choosing the feedback polynomial in this manner means that it cannot be primitive and thus this competes with the second design rule. As with turbo codes, the choice between a primitive or non-primitive feedback polynomial is a trade-off between performance at low and high SNRs, respectively [46].

In order to illustrate the first design rule, we consider the performance of two $R = 1/3$ serial concatenated convolutional codes with pseudorandom interleavers. Both codes use a rate $R_o = 1/2$, memory $\nu = 2$ outer code realized in non-systematic feedforward form. The encoder for this code is

$$G_o(D) = [\begin{array}{cc} 1 + D^2 & 1 + D + D^2 \end{array}], \quad (8.36)$$

which describes a 4-state $(2, 1, 2)$ code with $d_{\text{free}}^o = 5$. The first SCCC uses a rate $R_i = 2/3$ inner code with total encoder memory $\nu = 2$, which has a generator matrix of

$$G_1(D) = \left[\begin{array}{cc} 1 & 0 & \frac{1+D^2}{1+D+D^2} \\ 0 & 1 & \frac{1+D}{1+D+D^2} \end{array} \right]$$

and is realized in systematic feedback form.⁴ Note that the feedback polynomial is primitive. Simulation results for the resulting rate $R = 1/3$ concatenated code, denoted SCCC1 and shown in Figure 8.17, are shown in Figure 8.18 for several interleaver lengths. This code has $d_{\text{free}}^o = 5$ and from (8.33) the maximum exponent is given by

$$\alpha_{\max} = - \left\lceil \frac{d_{\text{free}}^o + 1}{2} \right\rceil = - \left\lceil \frac{5 + 1}{2} \right\rceil = -3 \quad (8.37)$$

and we would expect an interleaver gain of N^{-3} . The simulation results do indeed manifest this gain with increasing N .

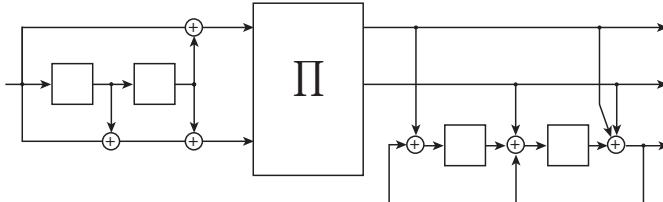


Figure 8.17: Block diagram and encoders for SCCC1.

The second serial concatenated convolutional code uses a rate $R_i = 2/3$ inner code with total encoder memory $\nu = 2$ with generator matrix

$$G_2(D) = \begin{bmatrix} 1+D & D & 1 \\ 1+D & 1 & 1+D \end{bmatrix}$$

when realized in nonsystematic feedforward form. Simulation results for the resulting rate $R = 1/3$ concatenated code, denoted SCCC2 and shown in Figure 8.15, are given in Figure 8.19 for several interleaver lengths. This code clearly performs much poorer than SCCC1.

For increasing N this code exhibits an error floor despite its relatively large free distance. In fact, simulation results suggest that the performance curves for $N = 1,000$ and $N = 10,000$ will converge for high SNR. This is not unexpected! For both codes, it is highly likely that the free distance $d_{\text{free}} = d_{\text{free}}^o d_{\text{free}}^i = 15$ and from (8.30) there is no interleaver gain associated with the free distance multiplicity. Thus, the free distance

⁴Note that for encoders with rate $R = \frac{k}{k+1}$ and $k > 1$, the recursive systematic realization is generally not a minimal realization, but the systematic feedback realization is.

asymptotes of these two codes are essentially identical. For large N , however, the pseudo-random interleaver does have a spectral thinning effect on some higher-weight codewords, which results in the improved performance at moderate SNRs.

In order to illustrate the second design rule, we construct a third code, SCCC3, using a non-primitive feedback inner encoder. SCCC3 uses a rate $R_i = 2/3$ inner code with total encoder memory $\nu = 2$ and generator matrix

$$G_3(D) = \begin{bmatrix} 1 & 0 & \frac{D^2}{1+D^2} \\ 0 & 1 & \frac{1+D+D^2}{1+D^2} \end{bmatrix}$$

when realized in systematic feedback form. The feedback polynomial for this code, $1+D^2 = (1+D)(1+D)$, is not primitive. The performance of this code is shown in Figure 8.20 for a variety of block lengths. The asymptotic performance of SCCC3 is clearly inferior to that of SCCC1, though it does have a small advantage at high error rates.

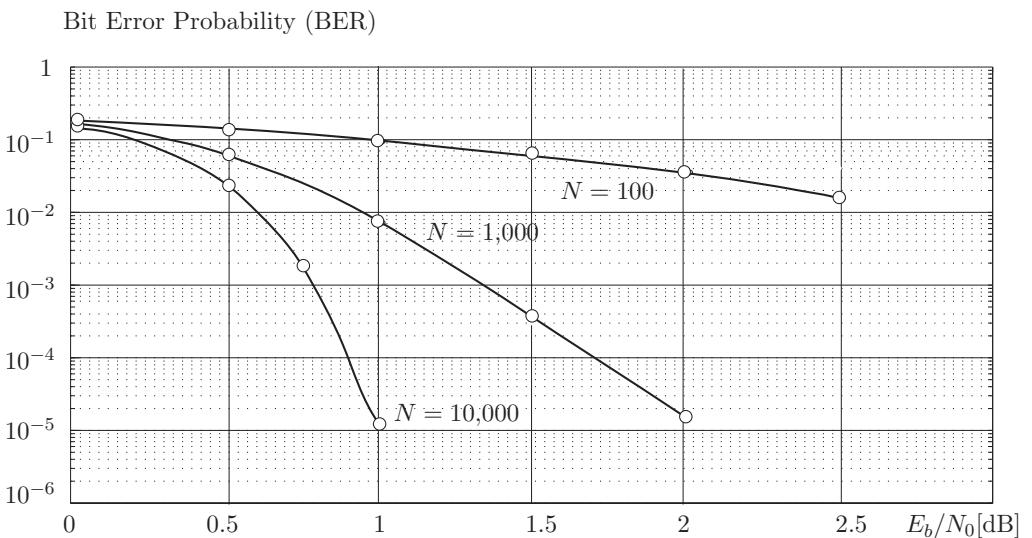


Figure 8.18: Simulation results for the $R = 1/3$ code SCCC1 (inner encoder with feedback) as a function of interleaver length.

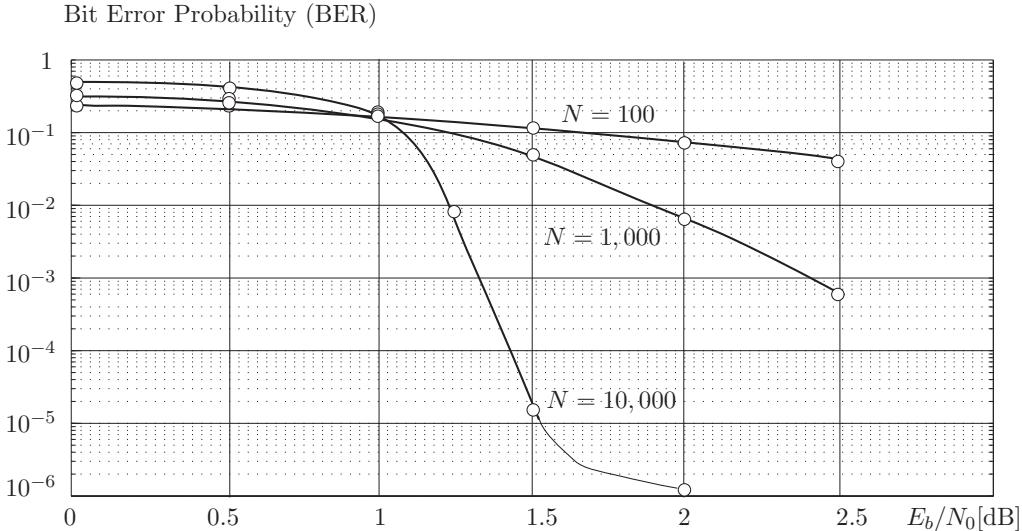


Figure 8.19: Simulation results for the $R = 1/3$ code SCCC2 (inner encoder without feedback) as a function of interleaver length.

8.11 Iterative Decoding and Performance of SCCCs

Seriously concatenated codes may be iteratively decoded in a manner very similar to that of parallel concatenated codes. A block diagram of the iterative decoder for a serial concatenated code is shown in Figure 8.21. The APP decoders are the same as those described in Chapter 5 and Section 8.6. The essential difference lies in the fact that with serial concatenation the outer decoder does not have access to channel observations and must rely on information received from the inner decoder.

To be precise, the inner APP decoder receives the channel log-likelihood ratio (LLR) and the extrinsic information from the outer decoder and computes a new log-likelihood. The output LLR of the inner APP decoder is

$$\begin{aligned} L_r^{(1)} &= \log \frac{P[c'_r = 1 | \mathbf{y}]}{P[c'_r = 0 | \mathbf{y}]} \\ &= \log \frac{P[\mathbf{y} | c'_r = 1]}{P[\mathbf{y} | c'_r = 0]} + \log \frac{P[c'_r = 1]}{P[c'_r = 0]} = \Lambda_{e,r}^{(1)'} + \Lambda_r^{(1)'}, \end{aligned}$$

of which only $\Lambda_{e,r}^{(1)'}$ is passed to the second decoder, again for reasons of avoiding the

Bit Error Probability (BER)

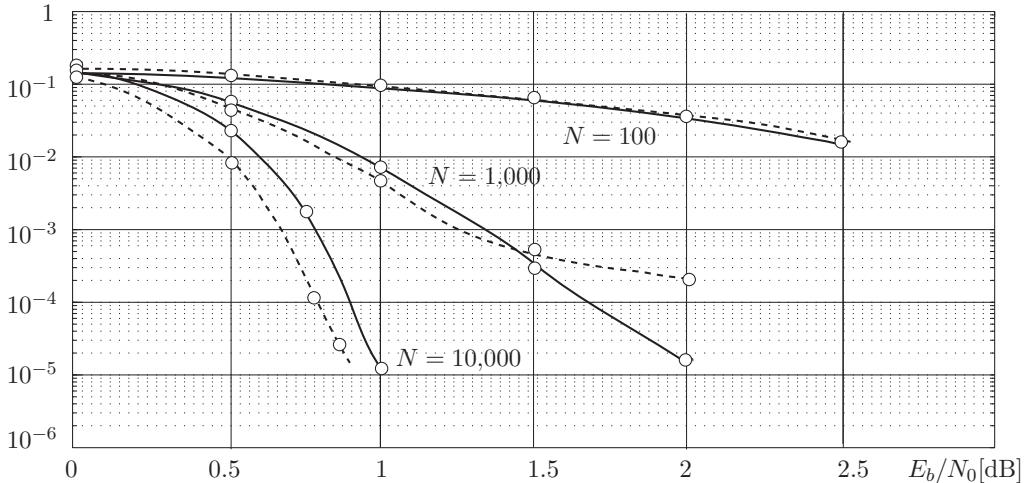


Figure 8.20: Performance of serial concatenated convolutional codes with a primitive inner encoder (SCCC1, solid line) and a nonprimitive inner encoder (SCCC3, dashed line).

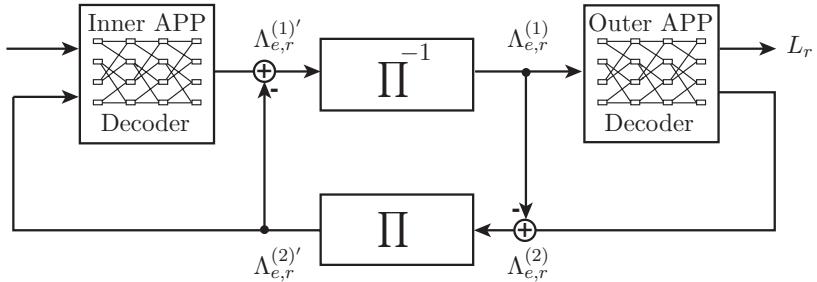


Figure 8.21: Block diagram of the iterative decoder for serial concatenated codes.

accumulation of dated information. The outer APP decoder now computes its LLR using the correctly deinterleaved extrinsic information from the first decoder as its only input.

The outer decoder computes an information bit log-likelihood and a codeword bit

log-likelihood for iterative processing by the inner decoder and thus generates

$$\begin{aligned} L_r^{(2)} &= \log \frac{P[c_r = 1 | \mathbf{\Lambda}_r^{(1)}]}{P[c_r = 0 | \mathbf{\Lambda}_r^{(1)}]} \\ &= \log \frac{P[\mathbf{\Lambda}_r^{(1)} | c_r = 1]}{P[\mathbf{\Lambda}_r^{(1)} | c_r = 0]} + \log \frac{P[c_r = 1]}{P[c_r = 0]} = \Lambda_{e,r}^{(2)} + \Lambda_{e,r}^{(1)} \end{aligned}$$

and passes $\Lambda_{e,r}^{(2)}$ to the first decoder. In this manner, serial concatenated codes can be iteratively decoded even though they do not share the same symbol sequence. This is based on the capability of the APP algorithms to incorporate *any* probabilistic prior information whether it comes in the form of received channel data, prior information on the information symbols, or prior information on the code symbols.

The natural question to ask at this point is how do serial concatenated convolutional codes compare to parallel concatenated convolutional codes? We address this question using both analysis and simulation. The analytical comparison uses the results from Sections 8.5 and 8.10, and the reader is reminded that these results assumed maximum likelihood decoding and large SNRs.

For parallel concatenated convolutional codes with recursive encoders, it was shown that for large SNRs the bit error rate could be approximated by

$$P_b \approx \sum_{j=1}^{\lfloor N/2 \rfloor} (2j) \cdot \binom{2j}{j} N^{-1} \left. \frac{(Z^{2+2z_{\min}})^j}{(1 - Z^{z_{\min}-t})^{2j}} \right|_{Z=e^{-RE_b/N_o}}, \quad (8.12)$$

The quantity $d_{\text{eff}} = 2 + 2z_{\min}$ determines the performance of the parallel turbo code and is generated by a weight-2 information sequence. The interleaver gain of a PCCC is fixed at N^{-1} , and d_{eff} is, in general, quite small.

For serial concatenated convolutional codes, the situation is markedly different. If we assume that the free distance of the outer code, d_{free}^o , is even, then substituting (8.33) and (8.34) into (8.28) and keeping only the term corresponding to α_{\max} results in

$$P_b \approx KN^{-d_{\text{free}}^o/2} \exp \left[-\frac{d_{\text{free}}^o d_{\min}^{(2)}}{2} \frac{RE_b}{N_o} \right], \quad (8.38)$$

where K is a constant proportional to the multiplicity [16]. From this expression, the SCCC to have an interleaver gain of $N^{-d_{\text{free}}^o/2}$, and the dominant distance is $d_{\text{free}}^o d_{\min}^{(2)}/2$, both of which increase with the outer code free distance. This suggests that SCCC should have a distinct advantage over PCCCs.

Figure 8.22 shows simulation results for a serial concatenated convolutional code and a parallel concatenated code for two different interleaver lengths. The serial code is

SCCC1 and the PCCC is a rate $R = 1/3$ code based on $(2, 1, 2)$ component codes with $h_1(D) = 1 + D^2$ and $h_0(D) = 1 + D + D^2$. In this, N_u corresponds to the length of the information sequence, so both codes have the same overall block length. The simulation results shown are for seven iterations of the iterative decoding algorithm. The results shown in Figure 8.22 clearly show the advantage of the serial concatenated code for large SNR. The increased interleaver gain is also evident. However, the PCCC has a distinct advantage at very low SNRs. This is an aspect of the performance that the union-bound analysis cannot capture.

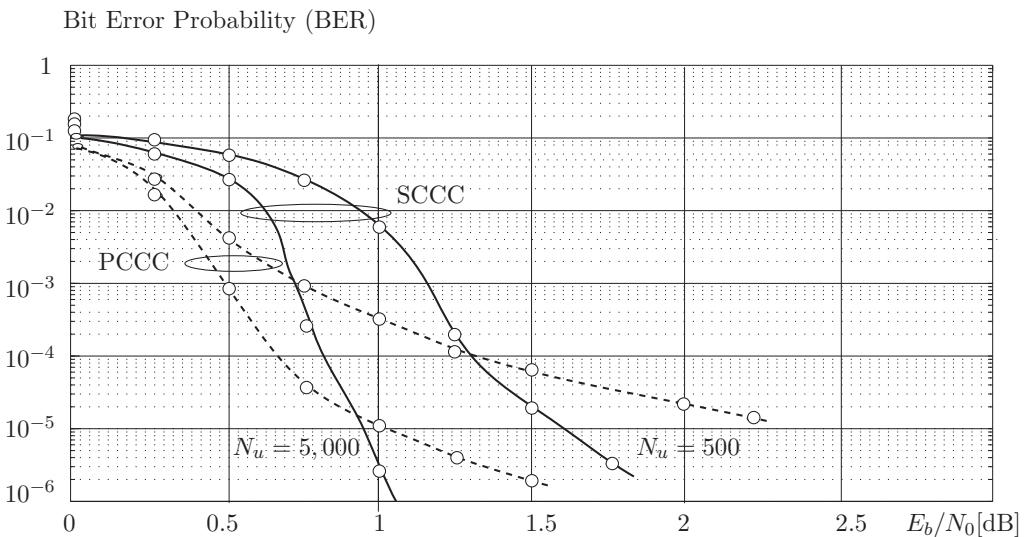


Figure 8.22: Comparison of a serial concatenated turbo code and a parallel concatenated code with the same overall block lengths and seven decoder iterations.

8.12 EXIT Analysis of Serially Concatenated Codes

As with turbo codes, the weight enumerator analysis of serial concatenated convolutional codes provides insight into code performance for moderate to high SNRs. In order to gain additional insight into the performance at low SNRs, we will once again rely on the EXIT

analysis of Section 8.7, which is used again to predict the onset of the turbo cliff for SCCC₁ and leads us to a particularly simple serial scheme with outstanding performance [75].

There are two fundamental differences between the EXIT analysis of parallel concatenated codes and serial concatenated codes. First, for serial concatenated codes the inner decoder passes extrinsic information concerning its code symbols, rather than information symbols, to the outer decoder. Thus, the EXIT function of the inner decoder,

$$I_E^i = T_i(I_A, E_b/N_0),$$

is different. The EXIT function of the rate $R_i = 2/3$, memory-2 inner code used in SCCC₁ is given in Figure 8.23 as a function of E_b/N_0 . Notice the large starting value of I_E^i .

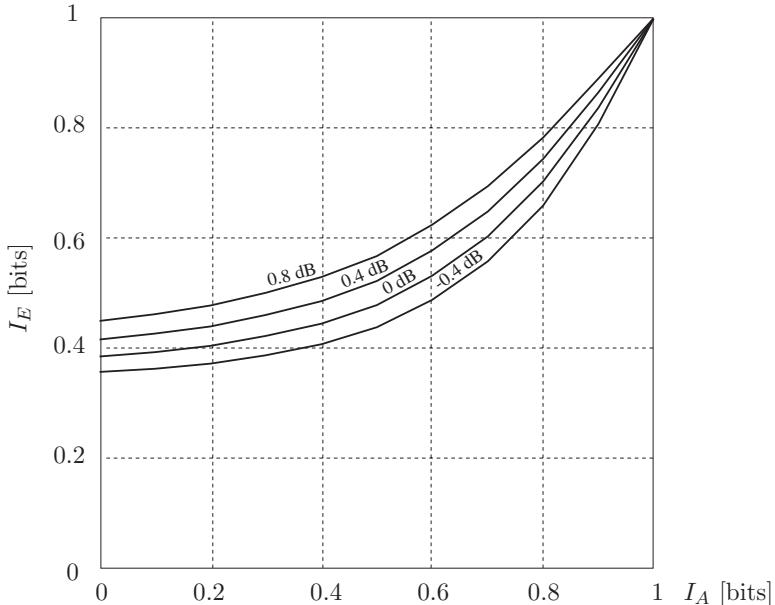


Figure 8.23: EXIT function of the rate $R_i = 2/3$ inner code used in SCCC₁.

Second, in serial concatenation the outer code does not see the channel outputs. Instead, it sees the extrinsic information output of the inner encoder and consequently its EXIT function,

$$I_E^o = T_o(I_E^i),$$

is not a function of the SNR. The EXIT functions of several $R_o = 1/2$ inner codes are given in Figure 8.24 as functions of I_A^i .

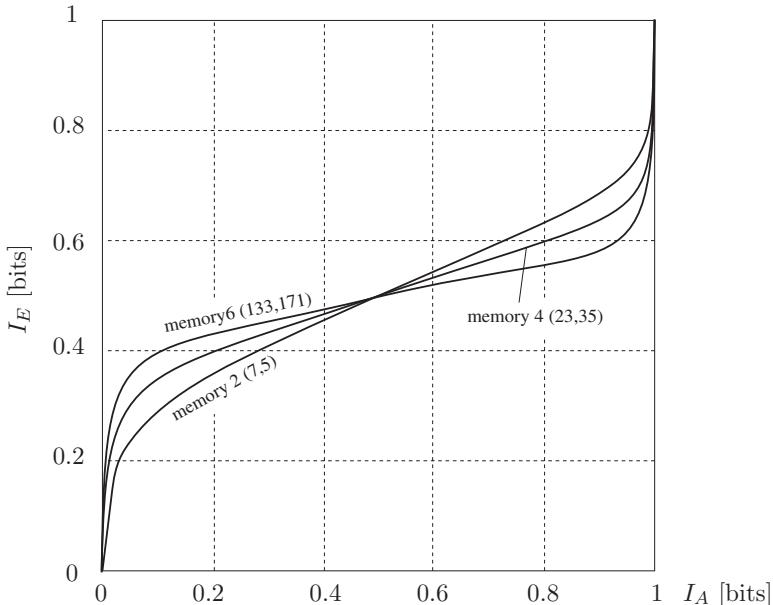


Figure 8.24: EXIT function of several $R_o = 1/2$ outer codes.

As with turbo codes, the EXIT functions of the inner and outer encoders can be combined to form an EXIT chart that predicts the behavior of the iterative decoder. For serial concatenated codes the inner and outer codes are not identical and the EXIT chart of the overall code is not symmetric with respect to the 45-degree line as in Figure 8.13. The EXIT chart for SCCC1 with $N = 10,000$ is shown in Figure 8.25. As with turbo codes, the accuracy of this method is impressive and this technique can be used to calculate a pinch-off signal-to-noise ratio for serial concatenated codes.

The EXIT analysis of serial concatenated codes may be used to analyze two particularly clever alternative schemes, the so-called *doped repeat scramble* (DRS) codes [78] and *repeat-accumulate* (RA) codes [13].⁵ DRS codes are effective for rate $R = 1/2$ while RA codes are generally effective for rate $R = 1/3$ or lower. Indeed, RA codes have been shown to approach the capacity of the AWGN channel as the rate approaches 0. Both of these concatenated coding schemes have the same basic structure consisting of an outer repetition code with rate $R_o = 1/m$ and an inner rate $R_i = 1$ scrambler/accumulator.

⁵RA codes were discussed from an LDPC perspective in Section 6.5.4, since they can most easily be viewed as belonging to either class of codes.

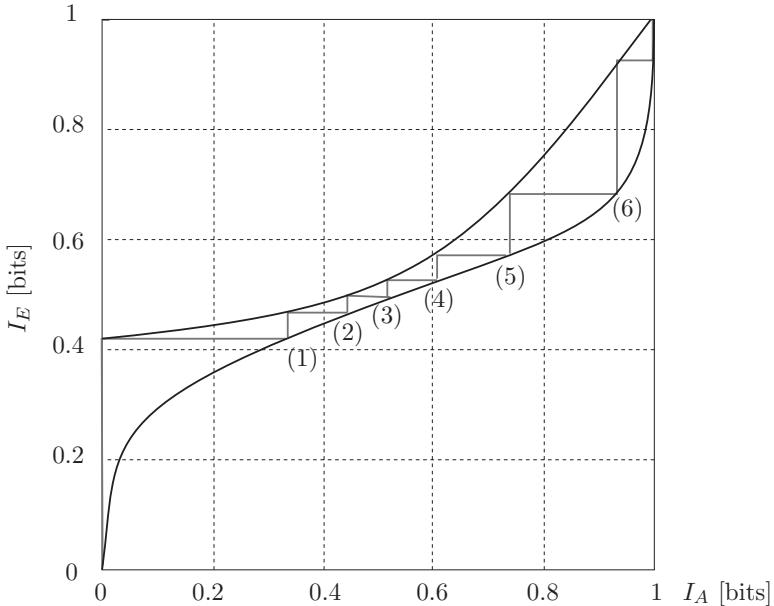


Figure 8.25: EXIT exchange chart for SCCC1 with decoder iterations for $E_b/N_0 = 0$ dB.

The overall rate of the code is determined solely by the outer code.

The inner accumulator is simply a rate-1/2 recursive systematic convolutional encoder that is punctured to rate 1. For DRS codes the puncturing pattern is typically such that one out every 100 bits is the systematic information bit, thus giving rise to the term “doped” codes. For RA codes no systematic information bits are used and the inner encoder is a rate-1 recursive encoder with feedback polynomial $h_0(D) = 1 + D$ and feedforward polynomial $h_1(D) = 1$.

The EXIT chart of the rate $R = 1/2$ repetition code used in the DRS codes is deceptively simple and given by the diagonal line

$$I_E^o = T_o(I_E^i) = I_E^i,$$

which is a consequence of the fact that its extrinsic soft decoder simply swaps the LLR values, regardless of SNR. The code design problem is then to find the appropriate inner accumulator and puncturing pattern that result in an EXIT function $T_i(I_A)$ that stays above this line for the lowest possible signal-to-noise ratio.

For DRS codes, extensive analysis and simulation [75] revealed that an inner code with feedback polynomial $h_0(D) = 1 + D + D^2 + D^3$ and feedforward polynomial

$h_1(D) = D + D^2 + D^3$ and a doping ratio of 1 in 100 resulted in a pinch-off of 0.28 dB. The EXIT exchange chart for this code is shown in Figure 8.26. This figure shows that as the signal-to-noise ratio is decreased, the EXIT function of the inner code gets closer and closer to that of the outer repetition code and thus more and more iterations are required for decoding. In order to achieve performance near the pinch-off with an interleaver length of $N = 10,000$, 100 decoder iterations are required.

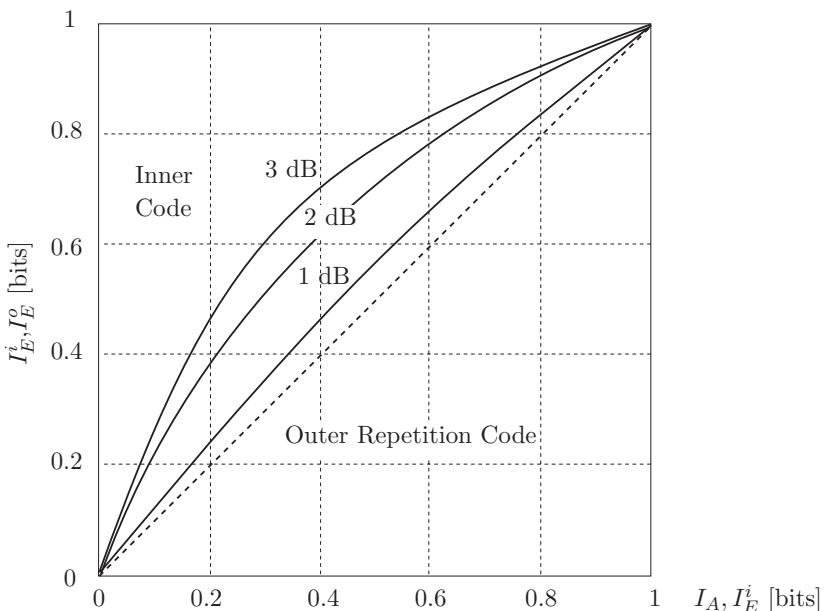


Figure 8.26: EXIT chart for a doped-repeat-scramble code with inner code $(h_0(D), h_1(D)) = (1 + D + D^2 + D^3, D + D^2 + D^3)$ and a doping ratio of 1 in 100.

8.13 Viewpoint

Turbo codes represent a paradigm shift in the 60 years of searching for the ultimate error control codes promised by the publication of Shannon's mathematical theory of communication. The new codes' introduction in 1993 has shaken a community that had come to believe that near-Shannon performance was "nearly" impossible, and now turbo codes with simple iterative decoders allow decoding at signal-to-noise ratios virtually arbitrarily close to the capacity limit [25, 75], at least for low-rate codes with $R < 1$.

While it has taken the scientific community a few years to come to terms with the new paradigm, iterative decoding is now seen as the key to achieving limit performance, and applications have spread to other fields such as equalization and multiple access interference resolution. Turbo coding and iterative decoding have changed forever how we approach error control coding. Gone are the waterfall error curves that gradually decline with increasing signal-to-noise ratios. The new methods of graph-based codes and iterative message passing teach pinch-off signal-to-noise ratios and turbo cliff behaviors. A data link with coding either works perfectly or does not work at all. This has far-reaching consequences on all parts of the complex communications infrastructure, as error control coding has become an integral part of practically all data communications in space or time (storage).

8.14 Turbo-Trellis-Coded Modulation

As trellis-coded modulation is an extension of binary coding methods to larger signal constellations, so is turbo-coded modulation the extension of turbo coding principles to include larger signal constellations for operation in spectrally limited regimes. As we will see, there are very few changes necessary to accommodate higher signaling alphabets, and higher spectral efficiencies. Mainly we are dealing with multi-level symbols now, rather than binary symbols, and the concept of log-likelihood ratios, which served us so well with binary codes, needs to be extended.

While *trellis-turbo-coded modulation* (TTCM), which can be seen as the direct extension of parallel concatenation, has found only limited interest, serial concatenation of binary error control codes with TCM is experiencing a surge of interest, in particular in the context of *space-time coding*, where the larger signal alphabets are comprised of matrices of complex symbols, as discussed in Chapter 2.

To start out with the analog of binary parallel concatenation, consider Figure 8.27, which shows the encoder for a TTMC system proposed by Robertson and Wörz [58]–[61]. One difference is that blocks of n -coded bits are treated as input symbols. The interleaver is symbol-oriented, and the component encoders are trellis encoders, such as those discussed in Chapter 3; here for $n = 3$. The final sequence of transmitted symbols is generated by selecting symbols alternately from the two encoders, i.e., $\mathbf{x} = [x_0^{(1)}, x_1^{(2)}, x_2^{(1)}, \dots]$. In this fashion the output symbols are punctured in the ratio 1:2 to maintain a rate $R = 2$ bits/symbol.

Since the encoded symbols no longer allow a separation of information (systematic) bits and parity bits, the encoded symbols of the second component encoder are deinterleaved before transmission. While this process could just as well be carried out at the receiver, deinterleaving before transmission simplifies synchronization. It has the effect that the n bits which make up the information portion of the symbols x_r are transmitted in order,

irrespective from which encoder the symbols originate.

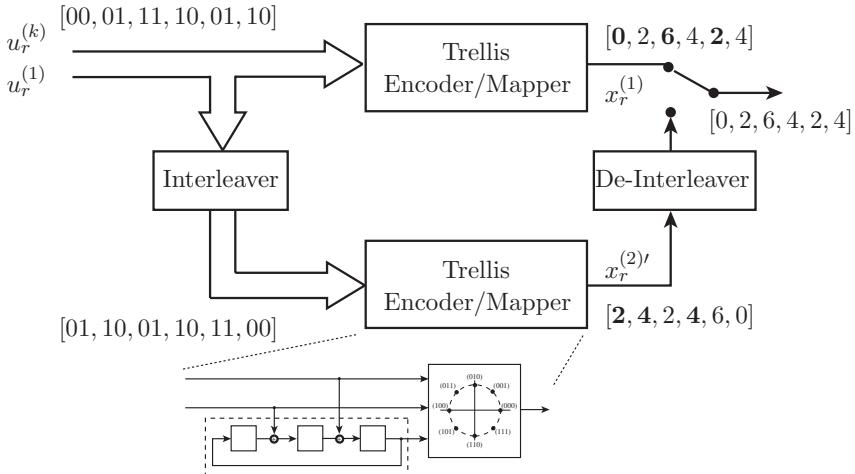


Figure 8.27: Block diagram of a turbo-trellis encoder with two constituent encoders.

As an example, assume that a block of $L = 6$ pairs of symbols is encoded in a TTCM system with 8-PSK trellis component codes, as shown in Figure 8.27. If the sequence of pairs of input bits $[u_r^{(2)}, u_r^{(1)}]$ is $[00, 01, 11, 10, 01, 10]$, then the sequence of output symbols from the first component encoder is $[0, 2, 6, 4, 2, 4]$ (see state-transition diagram in Figure 3.2). The interleaver permutes the bit pairs $[u_r^{(2)}, u_r^{(1)}]$ into the sequence $[01, 10, 01, 10, 11, 00]$, which is encoded by the second component encoder into the 8-PSK symbol sequence $[2, 4, 2, 4, 6, 0]$. Only the bold-faced symbols are transmitted, the others are punctured, and the final transmitted sequence is $[0, 2, 6, 4, 2, 4]$. Of course L needs to be chosen large enough for this system to have a turbo effect and measurable coding gain.

The decoder is also structured analogously to the iterative decoder of a parallel concatenated turbo-coded system and is shown in Figure 8.28. The sequence of received symbols \mathbf{y} is fed alternately to the first or the second APP decoder, in accordance with the puncturing order at the transmitter. If no symbol is available to be fed to the component decoder, a one is entered in lieu of the conditional channel probability $\Pr(y_r|x_r)$. This gives uniform weight to all symbols from the channel input and represents our lack of channel information for these symbols.

Decoding proceeds analogously to standard turbo decoding, but the component APP decoders need to be altered to accommodate the symbol structure of the received signal. In essence, we need to extract the conditional probability of each trellis branch from the

received signal. As discussed in Chapter 5, the only place where the received symbol affects the APP decoder is in the calculation of the branch metric

$$\begin{aligned} \gamma_r(j, i) &= \Pr(s_{r+1} = j, y_r | s_r = i) \\ &= \underbrace{\Pr[s_{r+1} = j | s_r = i]}_{\text{transition probability}} \left\{ \begin{array}{ll} 1 & \text{if symbol is punctured,} \\ \underbrace{\Pr(y_r | x_r)}_{\text{channel probability}} & \text{otherwise.} \end{array} \right. \quad (5.30) \end{aligned}$$

While the *transition probability* is simply the a priori probability $\Pr(u_r) = \Pr_a^{(k)}(u_r)$ of the symbol u_r causing the transition from state i to state j , the *channel probability* equals the actual conditional probability $\Pr(y_r | x_r)$ if the received symbol y_r originated from the corresponding component decoder, or is set to 1 if that symbol was punctured, and hence y_r delivers no information on the transmitted symbol for this component decoder.

The APP decoders calculate the a posteriori probabilities of all $M = 2^n$ symbol vectors u_r , i.e., they calculate $\Pr(u_r | \mathbf{y}^{(i)})$; $i = 1, 2$. From these output a posteriori probabilities, extrinsic probabilities $\Pr_e^{(i)}(u_r)$ are calculated by removing the a priori input probabilities $\Pr_a^{(i)}(u_r)$ by division, i.e.,

$$\Pr_e^{(i)}(u_r) = \frac{1}{\alpha} \frac{\Pr(u_r | \mathbf{y}^{(i)})}{\Pr_a^{(i)}(u_r)} \quad \forall u_r, \quad \alpha = \sum_u \frac{\Pr(u | \mathbf{y}^{(i)})}{\Pr_a^{(i)}(u)}. \quad (8.39)$$

This is the DIV operation in Figure 8.28. The normalization factor α is used to make the extrinsic probabilities sum up to unity. The operation in (8.39) avoids accumulation of a priori probabilities analogously to the subtraction of the LLR in the binary turbo decoder.

In the first half iteration no a priori information exists, and hence no information is available during time intervals corresponding to punctured symbols. In this case, initial a

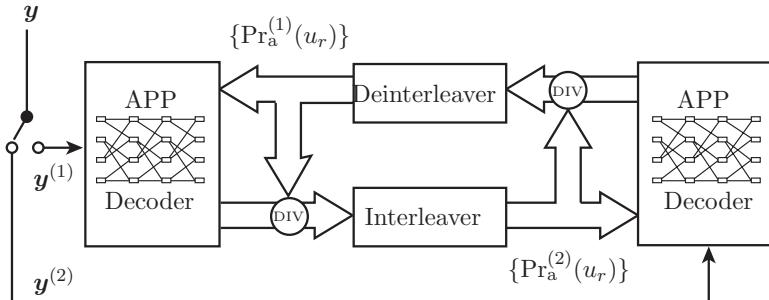


Figure 8.28: Block diagram of a turbo decoder with two constituent decoders.

priori probabilities are calculated as

$$\Pr_a^{(1)}(u_r) = \sum_{v_r^{(0)}} p(y_r | \underbrace{u_r, v_r^{(0)}}_{x_r}), \quad (8.40)$$

that is, the unknown parity is simply averaged out.

Figure 8.29 shows simulated performance curves, taken from [58, 59, 60] for a TTCM system using 8-state constituent 8-PSK trellis codes. The channel capacity limit for this channel is at $E_b/N_0 = 1.9$ dB for unconstrained input signals and increases to $E_b/N_0 = 2.9$ dB if the channel symbols are restricted to 8-PSK. Thus the performance of this systems comes close to the Shannon limit even for the small block length of $N = 5,000$ symbols.

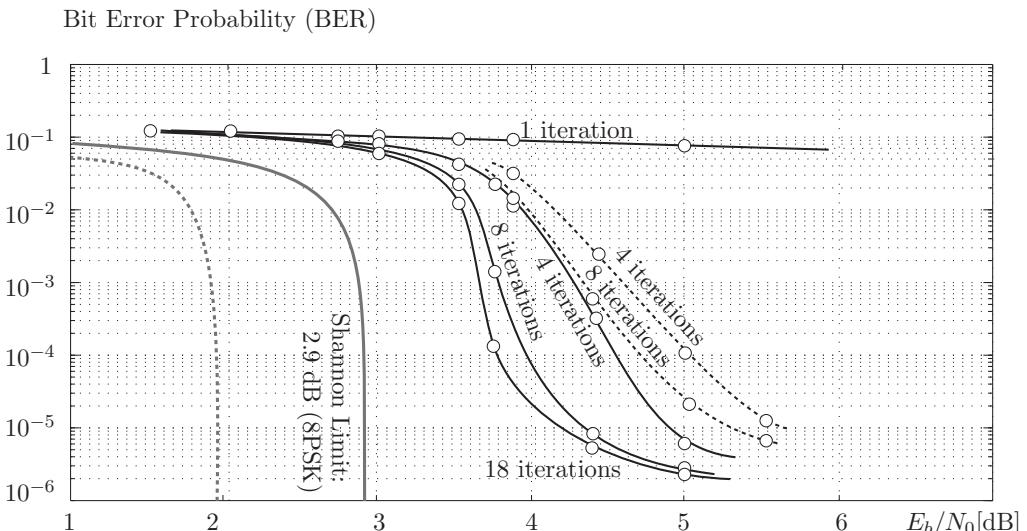


Figure 8.29: Simulations of a turbo-trellis-coded modulation system using two 8-PSK component trellis codes. Solid lines for $L = 5,000$ symbols, dashed lines for $L = 1024$ symbols.

Other methods of parallel concatenation also exist; [60], for example, discusses a 16-QAM TTCM system where 2 input bits are encoded by two rate $R = 1/2$ trellis encoders. The encoders are separated by the usual interleavers, which are realized independently for each of the two input bit streams. The resulting 4 output bits are mapped into a 16-QAM

symbol, whereby the two bits from each encoder choose the signal points along either the quadrature or the inphase axis. The advantage is that no puncturing is required, the disadvantage, however, is the large signal set expansion required.

8.15 Serial Concatenation

Serial concatenation for trellis-coded modulation has a number of advantages. The most obvious one is that the outer code can be a regular binary error control code, and only the inner code needs to drive a modulator. A block diagram of this structure is shown in Figure 8.30. A sequence of binary information symbols u_r are encoded in a regular binary encoder, whose outputs v_r are sequences of symbols of m -tuples of binary coded bits which are interleaved either symbol-wise or bit-wise and then fed into a trellis encoder which accepts m input bits and generates the output symbol x_r . Again, the example shown in Figure 8.30 uses a rate $R = 2/3$ 8-PSK trellis encoder as the modulation encoder.

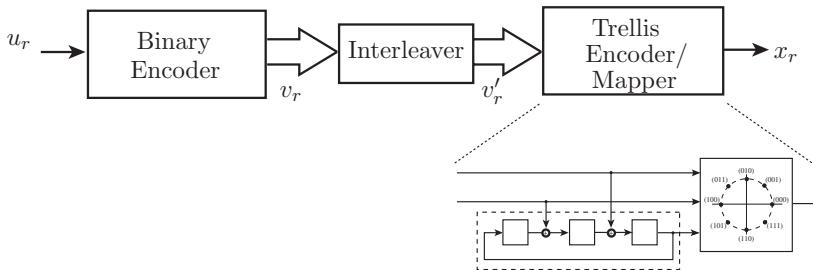


Figure 8.30: Block diagram of a serial turbo encoder for turbo-coded modulation.

The decoder, too, follows the same principle as for binary serially concatenated codes, and it is shown in Figure 8.31. The first APP decoder for the modulation code operates as usual with two inputs, the channel values \mathbf{y} and the set $\{\Pr_a(x_r)\}$ of a priori probabilities of the transmitted symbols x_r . The output of the modulation APP decoder are a posteriori probabilities of the same symbols x_r , denoted by $\{\Pr_e(x_r)\}$, and normalized according to (8.39).

These a posteriori symbols probabilities are now deinterleaved to line them up with the output symbols $v_r = (v_r^{(n-1)}, \dots, v_r(0))$ of the binary code. Now two strategies are possible; the symbol probabilities can be set equal to the probability of a block v_r of binary code symbols, i.e., $\Pr_e(x_r) = \Pr_e(v_r)$, and the APP binary code decoder operates with symbol probabilities in calculating the branch metrics of (7.31). This is most convenient

if the bits v on a branch of the binary code map directly into a symbol x . A more general approach is to use a binary APP code decoder which operates with binary LLR values. In order to obtain these, we need to marginalize the symbol probabilities, i.e.,

$$\Pr(v_r^{(m)} = v) = \sum_{\substack{x(v) \in \mathcal{X} \\ (v^{(m)}) = v}} \Pr_e(x(v)), \quad (8.41)$$

where \mathcal{X} is the constellation alphabet. From (8.41) the *a priori* LLRs for each bit to be used by the binary APP decoder are calculated in the usual manner:

$$\Lambda_a(v_r^{(m)}) = \log \left(\frac{\Pr(v_r^{(m)} = 1)}{\Pr(v_r^{(m)} = 0)} \right). \quad (8.42)$$

On the output side of the binary APP decoder, the LLR values have to be combined into symbol a priori values for the next iteration of the modulation code APP decoder. First, binary bit probabilities are generated from the extrinsic LLR values as

$$\Pr(v_r^{(m)} = 1) = \frac{\exp(\Lambda_e(v_r^{(m)}))}{1 + \exp(\Lambda_e(v_r^{(m)}))}, \quad \Pr(v_r^{(m)} = 0) = 1 - \Pr(v_r^{(m)} = 1). \quad (8.43)$$

The probability of a symbol is then simply the product of the probabilities of its constituent binary bits, i.e.,

$$\{\Pr_a(x_r(v_r))\} = \prod_{i=1}^n \Pr(v_r^{(m)}). \quad (8.44)$$

These symbol probabilities function as a priori information for the next iteration of the outer APP decoder.

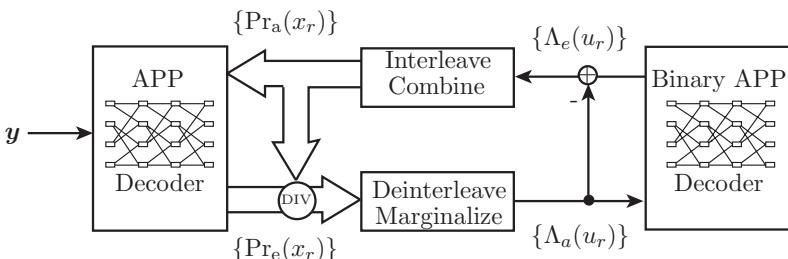


Figure 8.31: Block diagram of a turbo decoder with two constituent decoders.

8.16 EXIT Analysis of Serial TTCM

Turbo-coded modulation systems can also be analyzed following the EXIT method presented earlier. The only notable difference is that we are dealing with symbols rather than bits, which needs additional explanation. In the binary case, probabilities could conveniently be expressed in a single real number using the log-likelihood ratio. For multi-level signals, however, a single number does not suffice. Instead vectors of probabilities have to be used, both in the a priori and a posteriori case.

The coded modulation APP decoder receives reliability information on two inputs, viz. the received channel symbols \mathbf{y} and a-priori symbol probabilities of the transmitted symbols. Under the usual assumption of long random interleavers, the a priori probabilities $\Pr_a(x_r)$ are assumed to be independent. Likewise the a priori LLR values $\Lambda_a(u_r)$ used by the binary APP decoder are assumed to belong to independent bits u_r .

Unlike the binary decoder, the APP decoder for the inner modulation code at each time instant r will generate an entire list of extrinsic output probabilities $\mathbf{p}_{e,r} = [\Pr_e(x_r = \mathbf{s}^{(1)}), \dots, \Pr_e(x_r = \mathbf{s}^{(M)})]$ for the M symbols of the TTCM constellation. Likewise, the input probabilities will be delivered to the decoder as an M -valued vector $\mathbf{p}_{a,r} = [\Pr_a(x_r = \mathbf{s}^{(1)}), \dots, \Pr_a(x_r = \mathbf{s}^{(M)})]$ of input a priori symbol probabilities at time interval r . Both of these M -vectors are random vectors in their own right with their own probability distribution, which we shall denote by $p_a(\mathbf{p}_a)$ and $p_e(\mathbf{p}_e)$. Due to the interleaver assumption, \mathbf{p}_e and \mathbf{p}_a are assumed of the index r .

One way to visualize this is by thinking of a hypothetical discrete-input real-vector output channel, whose inputs are drawn from the signalling alphabet $\mathcal{S} = \{\mathbf{s}^{(1)}, \dots, \mathbf{s}^{(M)}\}$ and whose outputs are the (conditional) probability vectors $\mathbf{p}_{e|\mathbf{s}^{(i)}}$, or $\mathbf{p}_{a|\mathbf{s}^{(i)}}$. This hypothetical channel is not so different from a regular multi-input channel as discussed in Chapter 2, only that it is not an additive noise channel.

The “quality” of this hypothetical channel can be measured by the mutual information between the input $X = \mathbf{s}^{(i)}$ and output $\mathbf{p}_{e|\mathbf{s}^{(i)}}$, denoted by $I(X, \mathbf{p}_e)$, and likewise for the a priori symbols we define the mutual information $I(X, \mathbf{p}_a)$ between the actual signals $\mathbf{s}^{(i)}$ and the probability distribution $\mathbf{p}_{a|\mathbf{s}^{(i)}}$. The information measures $I(X, \mathbf{p}_a)$ and $I(X, \mathbf{p}_e)$ will be used to quantify the reliability of the distributions \mathbf{p}_a and \mathbf{p}_e , i.e., their level of uncertainty.

While the a priori distributions $\mathbf{p}_{a,r}$ in the actual decoding process are generated by an APP decoder, for the purpose of measuring the information transfer behavior of the coded modulation APP decoder they need to be synthesized. There are many ways to do this, and an intuitively correct way of generating them is by passing the transmitted symbols \mathbf{x} through a test channel of a given capacity, typically an additive white Gaussian noise channel. This then results in the information transfer point of view illustrated in Figure 8.32. Both inputs, the channel symbols \mathbf{y} as well as the symbol a priori probabilities,

are fundamentally identical in nature, since both physical and test channel are additive white Gaussian channels. The APP decoder is simply making use of all available a priori probability information to generate exact a posteriori probability information. The fact that the test channel output is first converted into probability vectors is quite irrelevant to the decoding performance.

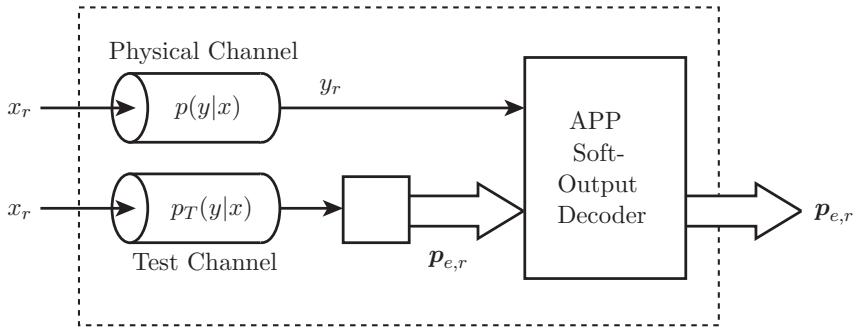


Figure 8.32: The modulation APP decoder as a transformer of probability distributions of the transmitted symbols x_r .

The decoder operates as a memoryless symbol probability transformer—encapsulated in the dashed box—in the iterative decoding systems. The assumed independence between successive symbols is approximated by large interleavers.

Unfortunately due to the non-linear operation of the APP decoder, no analytical methods exist to find the multi-dimensional distribution \mathbf{p}_e , and also the distribution \mathbf{p}_a is difficult to obtain analytically. It is therefore typical to evaluate these information measures using Monte-Carlo simulations of the individual codes of interest.

Plots of $I(X, \mathbf{p}_a)$ versus $I(X, \mathbf{p}_e)$, i.e., (EXIT) charts, can be used to study the convergence behavior of large iterative decoding systems, such as determining the onset of the turbo cliff, just as in the case of binary coded systems. Examples of EXIT charts are presented in subsequent sections for a number of specific coded modulation systems.

8.17 Differential-Coded Modulation

An elegant method of combining coding and modulation is to use a differential encoder as the modulation code. Differential modulation is accomplished by generating the transmitted symbols as the (complex) product of the information symbol v_r with a previous transmitted symbol, i.e., $x_r = x_{r-1}v_r$, $v_r, x_r \in \mathcal{X}$. The process is initiated at the beginning

of the transmission with a known symbol $x_0 = s^{(i)}$. In the case of N -PSK the multiplication is traditional complex multiplication, but other forms are also used. In Section 8.18 we discuss systems where \mathcal{X} is a set of unitary space-time codes.

Basic differential decoding is accomplished by extracting an estimate for v_r from the two most recently received channel symbols y_r and y_{r-1} as

$$\hat{v}_r = \max_v \Re [vy_r^*y_{r-1}] . \quad (8.45)$$

However, we are more interested in using the differential code in conjunction with an outer code in a serially concatenated coding system. Figure 8.33 shows the structure of this serially concatenated coded modulation system. Encoding is performed on a symbol-by-symbol level: The outer code, a binary error control code, outputs m -tuples of bits. The stream of these binary m -tuples are passed through the interleaver and are mapped into symbols from an $M = 2^m$ -ary symbol alphabet \mathcal{X} . They are then differentially modulated according to $x_r = x_{r-1}v_r$, where D in Figure 8.33 is a single symbol delay. The mapping from the binary symbols into modulation symbols v_r is typically chosen as the natural mapping discussed in Chapter 3. The interleaver can operate on a binary bit level, this is advantageous especially if the binary code used is a very simple code with little error correction capability. Both symbol- and bit-interleavers are considered in the sequel.

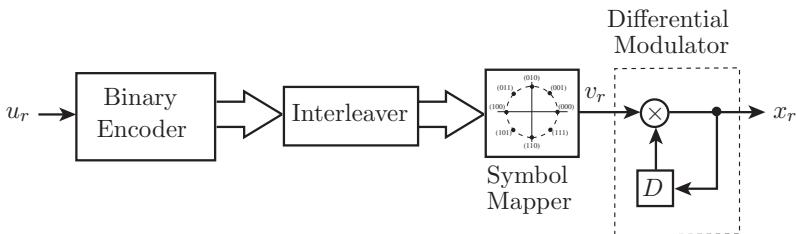


Figure 8.33: Block diagram of a serial encoder for differential turbo coded modulation.

Since the differential modulator is an infinite impulse response filter, it fits the requirements of a suitable “inner code” in a serially concatenated coding system. A symbol-wise turbo decoder [19, 20] for this is identical to that in Figure 8.31. Such systems have been studied for the concatenation of convolutional codes with differential BPSK reported [41, 54], leading to very simple encoders and decoders with very good error performance results, rivaling those of more complex Turbo codes.

Figure 8.34 shows the performance of a serially concatenated differential 8-PSK modulator with a [3,2,2] outer parity code [42, 43]. The bits of the outer parity code are

bit interleaved, since symbol interleaving would not generate independent inputs within the parity-check codewords, which are only of length 3 each. The block size used was 5,000 symbols, corresponding to 10,000 information. As conjectured, the rapid drop-off indicates the threshold behavior typical of turbo-coded systems. Note, however, that this code suffers from a high d_{free} error floor.

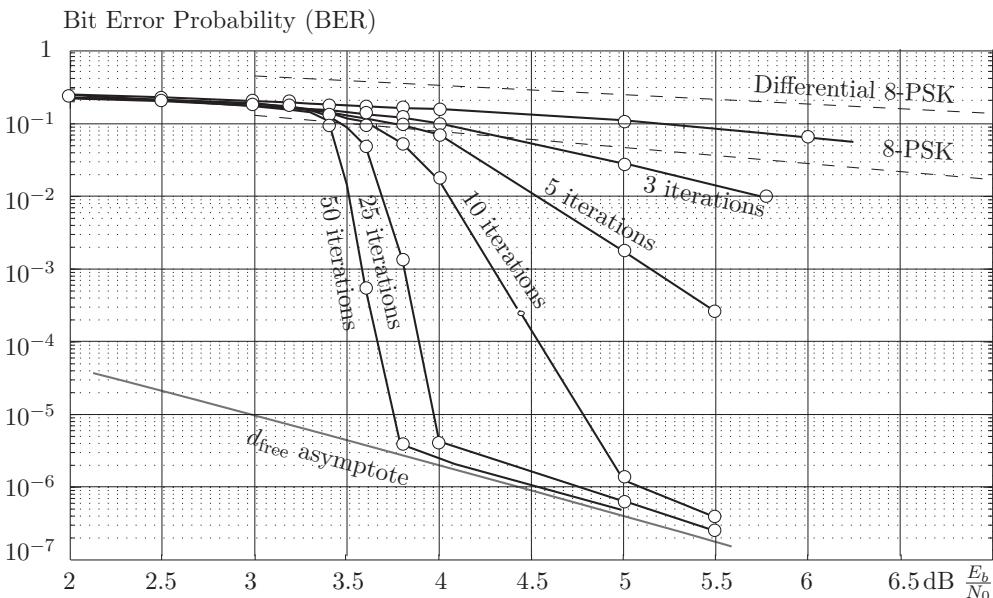


Figure 8.34: Performance of the serially concatenated differential 8-PSK system with an outer parity-check code, compared to regular and differential 8-PSK modulation.

Using the analysis techniques discussed in Section 8.16, we can explain the error performance behavior exhibited in Figure 8.34. First, using EXIT analysis, we find the EXIT chart for this system using the differential 8-PSK modulator as the inner code, shown in Figure 8.35. The input mutual information $I_a = I(X, p_a)$ is plotted on the horizontal axis and the output extrinsic mutual information $I(X, p_e)$ on the vertical axis. Also plotted is the EXIT curve of the [3,2,2] parity-check code used as the outer code. The axis extends from 0 to 1 bit of mutual information; that is, the measure has been normalized per coded

bit. This EXIT chart shows that a narrow convergence channel opens at $E_b/N_0 = 3.4$ dB, exactly where the turbo cliff occurs in the simulations.

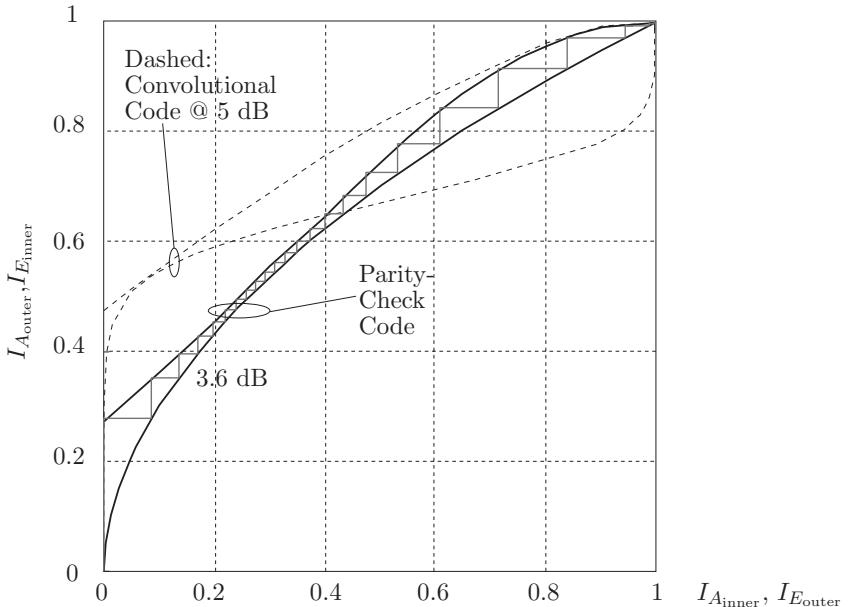


Figure 8.35: Extrinsic information transfer (EXIT) chart for serial concatenation of a $[3,2,2]$ parity-check code with a differential 8-PSK code. The trajectory is drawn for $E_b/N_0 = 3.6$ dB. A 16-state convolutional outer code shown in dashed would not converge until $E_b/N_0 > 5$ dB.

The $[3,2,2]$ parity-check code is very well matched to the differential modulator, since their respective EXIT curves are nearly parallel. Furthermore, the APP decoder for the $[3,2,2]$ parity-check code is extremely simple and can be implemented, for example, in a single ROM lookup table. From the parity-check constraint, it is quite straightforward to calculate the output extrinsic bit log-likelihood ratio as

$$\lambda_E(v_1) = \lambda_A(v_2) + \log\left(\frac{1 + \exp(\lambda_A(v_3) - \lambda_A(v_2))}{1 + \exp(\lambda_A(v_3) + \lambda_A(v_2))}\right), \quad (8.46)$$

where v_1, v_2, v_3 are the bits that make up a single parity-check codeword. It is important that the interleaver between the codes uses bit-interleaving, since the parity-check code requires bit-independent a priori probabilities.

The error floor is explained and quantified via a distance spectrum analysis [43]. Since the inner code has a fully connected trellis, the shortest detour consists of two symbols. Furthermore, the symbols on the merging path into a given state are all identical, since the state equals the most recently transmitted symbol. The minimum distance of the differential 8-PSK code is therefore easily evaluated as $d_{\text{free}}^2 = 0.586$, which is very small and provides no improvement over regular 8-PSK.

Including the outer code in our analysis, we ask if such short paths are permissible by the constraints of the outer code. Table 8.2 lists all bit weights that are mapped into length 2 detour path from the all-zero path, using the natural labeling of 8-PSK symbols.

Symbol 1	Symbol 2	d^2	Weight
001	111	0.586	even
111	001	0.586	even
010	110	2	odd
110	010	2	odd
011	101	3.414	even
101	011	3.414	even
100	100	4	even

Table 8.2: Bit weights that can be mapped into pairs of diverging and merging symbols of the differential 8-PSK trellis.

Note that since the parity-check outer code enforces an overall even weight of the codeword, all pairs with even weight are possible, given that the interleaver maps the bits accordingly. Furthermore, even the small $d^2 = 0.586$ is permissible by the outer code. Extensive experiments with random interleavers have shown that the minimum distance of the complete code equals $d_{\text{min}}^2 = 1.172$ with very high probability, corresponding to two length 2, minimum distance error events.

This can be dramatically improved by choosing a non-natural alternate mapping which maps the triples (000), (111), (001), (100), (010), (011), (110), (101) onto the 8-PSK symbols in a counterclockwise fashion. The weights of length 2 detours are given in Table 8.3.

The only even-weight detour has a large distance, and all of the short-distance detours have odd weight and are thus not allowed by the outer code. Searches with specific interleavers have revealed that the minimum distance of the code with this mapping is $d_{\text{min}}^2 = 2.344$. The performance of this mapping is the one shown in Figure 8.34.

Symbol 1	Symbol 2	d^2	Weight
111	101	0.586	odd
101	111	0.586	odd
001	110	2	odd
110	001	2	odd
100	011	3.414	odd
011	100	3.414	odd
010	010	4	even

Table 8.3: Bit weights that can be mapped into pairs of diverging and merging symbols of the differential 8-PSK trellis.

8.18 Concatenated Space–Time Coding

As a final example we discuss the application of serial concatenation to space–time coding for multiple-input multiple-output channels—see Chapter 2. Under certain conditions, the channel capacity of such a MIMO channel can be increased by a factor as large as the minimum of the number of transmit and receive ports. The information theoretic concepts regarding multiple antenna transmission shall not be discussed here, but there are a sequence of papers that the reader may want to explore [73, 35, 72]. We view the multiple antenna channel as a special case of a modulator, where the modulation symbols are $N_t \times N_c$ matrices with entries which are complex signal points from complex constellations, and N_t is the number of transmit antennas and N_c is length of the space–time modulation symbol, i.e., the number of columns making up the transmit matrix.

Typically each of the $N_t N_r$ subchannels h_{ij} from antenna i to antenna j is subject to independent signal fading, as discussed Chapter 2. At time r , each transmit antenna $i = 1, \dots, t$ sends a complex symbol c_{ir} , which is modulated onto a pulse waveform and transmitted over the channel. Modulation and transmission are identical to the single-antenna channel. Taken as a vector, $\mathbf{c}_r = c_{1r}, \dots, c_{Nr}$ is referred to as a *space–time symbol*. At each receive antenna $j = 1, \dots, N_r$, the signal is passed through a filter matched to the pulse waveform and sampled synchronously. These samples are modeled as $y_{jr} = \sqrt{E_s/N_t} \sum_{i=1}^{N_t} h_{ji} c_{ir} + n_{jr}$, where n_{jr} is a complex Gaussian noise sample. The parameter E_s is the average (received) signal power per space–time symbol, and $E_s/(N_t N_0)$ is therefore the symbol signal-to-noise ratio at each receive antenna.

The transmitted space–time symbols are customarily collected into $N_t \times N_c$ space–time codeword (STC) matrices, $\mathbf{C} = [\mathbf{c}_1^t, \dots, \mathbf{c}_{N_c}^t]$, where rows correspond to different transmit antennas and columns correspond to different times. Considering a sequence of L such

STC transmissions $\mathbf{C}_1, \mathbf{C}_2, \dots, \mathbf{C}_L$, the channel can be written as

$$\mathbf{Y}_r = \sqrt{\frac{E_s}{N_t}} \mathbf{H}_r \mathbf{C}_r + \mathbf{N}_r, \quad (8.47)$$

where \mathbf{Y}_r is the r th received STC, \mathbf{H}_r is the $N_r \times N_t$ channel matrix of transmission path gains, and \mathbf{N}_r is an $N_r \times n$ matrix of complex noise samples.

While we will not study fading channels in any detail (see references [56, 45], for example) multiple-antenna communication usually is studied in the context of signal fading, where each h_{ij} is modeled as an independent Rayleigh fading process by selecting the elements of \mathbf{H}_r as zero-mean unit variance complex Gaussian random variables with i.i.d. real and imaginary parts. One usually distinguishes between *fast* fading, in which the \mathbf{H}_t evolve according to a process whose dominant frequency is much faster than $1/L$, but slower than $1/N_c$, and *quasi-static* fading, in which the \mathbf{H}_r are constant for groups of L code matrices corresponding to the transmission of a complete frame.

The first example of a serially concatenated system using space-time codes, discussed by Schlegel and Grant [63] uses a differential STC presented by Hughes [44] which is based on unitary matrices with a group structure. A similar serially concatenated system is also presented in [2]. In Hughes' code, each STC word takes the form $\mathbf{C} = \mathbf{D}\mathbf{G}$, where \mathbf{D} is an $N_t \times N_c$ matrix and \mathbf{G} belongs to a group of unitary $N_c \times N_c$ matrices ($\mathbf{G}\mathbf{G}^* = \mathbf{I}$), i.e., the product of any two matrices $\mathbf{G}_1\mathbf{G}_2$ results in another matrix \mathbf{G}_3 of the group. The N_c columns of \mathbf{C} are transmitted as N_c consecutive space-time symbols.

A basic example of such a code with $t = n = 2$ is the *Quaternion Code* with symbols from the QPSK constellation. There are 8 STC words, given by

$$\mathcal{Q} = \left\{ \pm \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \pm \begin{pmatrix} j & 0 \\ 0 & -j \end{pmatrix}, \pm \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}, \pm \begin{pmatrix} 0 & j \\ j & 0 \end{pmatrix} \right\}, \quad (8.48)$$

$$\mathbf{D} = \begin{pmatrix} 1 & -1 \\ 1 & 1 \end{pmatrix}, \quad (8.49)$$

This code is isomorphic to Hamilton's Quaternion Group, hence its name.

The novelty of this code is that it can be differentially encoded and decoded analogous to differential PSK. At the start of transmission, the transmitter sends the STC word $\mathbf{C}_0 = \mathbf{D}$. Thereafter, messages are differentially encoded: To send $\mathbf{G}_r \in \mathcal{Q}$ during symbol time r , the transmitter sends

$$\mathbf{C}_r = \mathbf{C}_{r-1}\mathbf{G}_r. \quad (8.50)$$

The group property of the space-time code guarantees that \mathbf{C}_r is a codeword if \mathbf{C}_{r-1} is a codeword. Like differential PSK, a differential receiver for \mathbf{C}_r exists based on the two most recent blocks. It computes [44]

$$\hat{\mathbf{G}} = \max_{\mathbf{G} \in \mathcal{Q}} \Re \operatorname{tr} \mathbf{G} \mathbf{Y}_r^* \mathbf{Y}_{r-1}. \quad (8.51)$$

While such differential codes can be used without channel knowledge as well as provide rapid training on time-varying channels, our purpose is to use them as inner codes in concatenated systems. The differential nature of these codes can furthermore be exploited in rapid channel estimation algorithms, and [63] contains a more detailed discussion of the use of these codes in conjunction with unknown channels.

Figure 8.36 shows the EXIT chart for the system discussed using the quaternion differential STC as the inner code ($I_a = I(X, \mathbf{p}_a)$ on the horizontal axis and $I(X, \mathbf{p}_e)$ on the vertical axis) and 4, 16 and 64 state maximal free distance rate 2/3 convolutional codes (Chapter 4) as the outer code (their input $I(X, \mathbf{p}_a)$ is plotted on the vertical axis). The [3,2,2] outer parity-check code from the last section is also used. The axis extend from 0 to 3 bits of mutual information, where the maximum of 3 bits corresponds to complete knowledge of $\mathbf{s}^{(i)}$ derived from \mathbf{p}_e . In the case of bit interleaving, input and output bit-wise mutual information values are multiplied by three to normalize them to symbols. The differential STC extrinsic information transfer curves are shown for various signal-to-noise ratios, ranging from -1 dB to -1.6 dB in steps of 0.1 dB, and have been evaluated for independent Rayleigh fading MIMO channels.⁶

From the figure, we see that we expect the turbo threshold to occur near -1.2 dB for the 64 state outer code and around -1.3 dB for the 16- and 4-state codes. We also expect that the 16- and 64-state outer codes to result in faster convergence, since the convergence channel between the inner and outer curves are more open for these codes. The parity-check code has the lowest turbo cliff at -1.35 dB since its extrinsic information exchange curve is best matched to the shallow curve of the inner decoder.

Figure 8.37 compares the performance of the concatenated space-time system using the outer codes discussed. The interleaver length was 50,000 symbols, and up to 25 decoding iterations were performed. As predicted by the EXIT analysis, the turbo cliffs occur at the thresholds of -1.35 dB, -1.3 dB and -1.2 dB respectively, and the performance sharply follows the turbo cliff at exactly the predicted values. At this code rate (1 bit per channel use), capacity is at -3.1 dB [73]. Thus the concatenated coding system can achieve virtually error free performance at slightly more than 1.5 dB from the *unconstrained* MIMO channel capacity. The more shallow drop-off of the error curve of the 4-state outer convolutional code is due to the fact that symbol interleaving rather than bit interleaving is used, which underlines the importance of using bit interleaving for weaker outer codes.

It is quite astonishing to see such performance being achieved by the cooperation of two component codes which are individually quite useless. Note that the [3,2,2] parity check code by itself cannot correct any errors and that the quaternion space-time code in its differential encoder realization is catastrophic (see Chapter 4), which also by itself will not impress with good performance. It is difficult to imagine that coding systems

⁶The apparent violation of Shannon's absolute limit of $E_b/N_0 > -1.59$ dB stems from the fact that he signal-to-noise ratio in MIMO systems is measured per receive antenna.

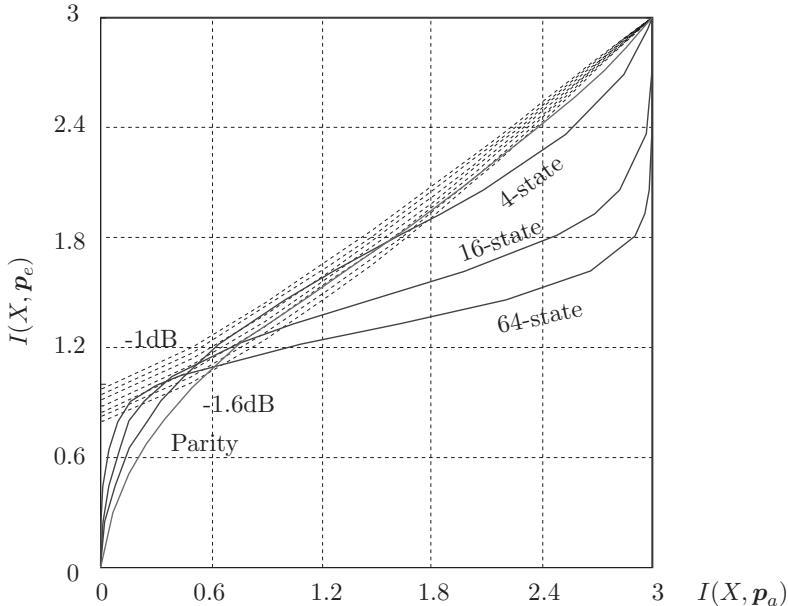


Figure 8.36: Extrinsic information transfer (EXIT) chart for the serial concatenation of 4-, 16-, and 64-state convolutional codes and a [3,2,2] parity-check code with the differential space-time quaternion code, the latter's transfer curves are dashed.

with lower complexity can be found that perform this well. The combination of such simple component codes into a serial coding system with such strong performance and manageable computational decoding complexity is the real contribution of the invention of turbo coding.

Other combinations of codes have also been used successfully. Hochwald and ten Brink [40] have concatenated complete turbo codes with a memoryless signal mapper for 4×4 and 8×8 antenna arrays. Their decoder also operates according to the serial decoding principle from Figure 8.31, but due to the large complexity of the first APP, which has to generate LLRs for individual bits from the high-level modulated input signal, an approximate decoder was used. Nonetheless, the results achieved even for high-level modulation alphabets are excellent, as shown in Figure 8.38.

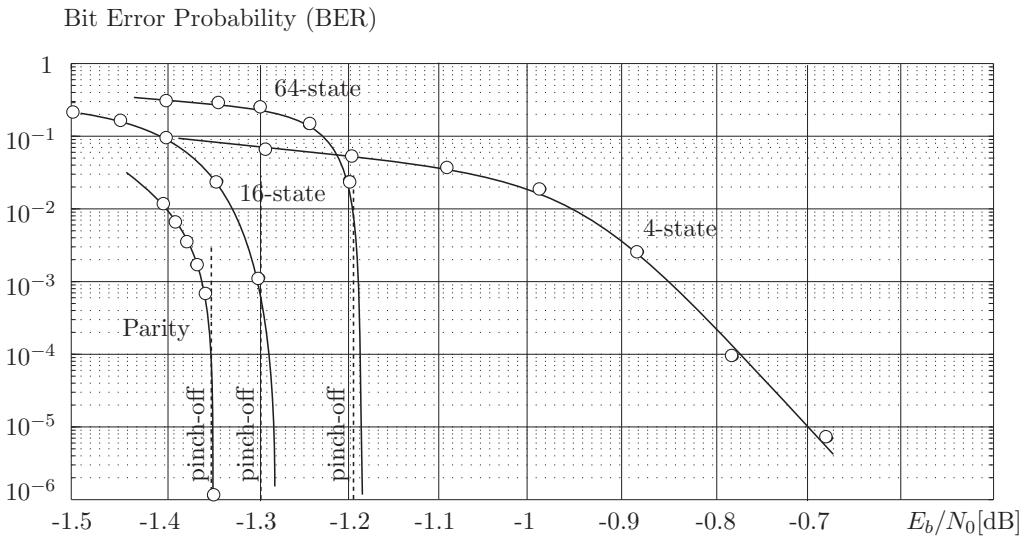


Figure 8.37: Performance of the serially concatenated system with several outer codes for a fast fading channel. (Note the entire scale is only 1.4 dB.)

These simulations use parallel concatenated turbo codes as outer codes, each having memory-2 component encoders given by $h_1(D) = 1 + D^2$ and feedback polynomial $h_0(D) = 1 + D + D^2$. The block size $N = 18,432$. As can be calculated from the modulation alphabet and the number of transmission antennas, the spectral efficiencies are 4, 8, and 12 bits/channel use for QPSK, 16QAM, and 64QAM for the 4×4 system, and twice that for the 8×8 MIMO example.

A general trend becomes evident: It is more and more difficult to approach capacity with larger modulation alphabets. The 64QAM, 8-antenna system is almost 6 dB away from its capacity potential while the smaller constellation and fewer transmit antenna systems come very close to channel capacity. Note also that a much more complex coding machinery is being used than in the previous example of serial concatenation.

8.19 Bit-Interleaved Coded and Generalized Modulation

If we compare the encoding methods of Figure 8.33 with that of Figure 8.30, we note a subtle difference. The latter encoder assigns bits to symbols *before* the interleaving process, while the encoder in Figure 8.33 interleaves the binary bit stream before assigning (mapping) bits to symbols. This process is a form of bit-interleaved modulation, studied

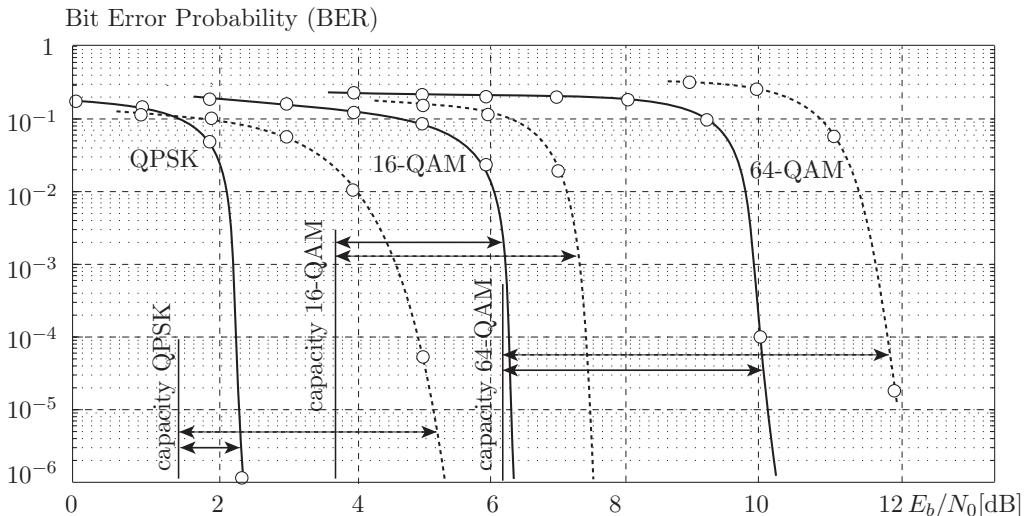


Figure 8.38: Performance of serially concatenated space-time coded systems with outer turbo codes for high spectral efficiencies, examined for 4×4 (solid lines) and 8×8 (dashed lines) MIMO arrangements.

in [22] following the introduction of the concept by Zehavi [81]. The effect of this is that each encoded binary digit passes through its “own” independent channel if the interleaver is large enough, such that the individual bits can be assumed to be affected by independent channel fading levels.

Now on the additive white Gaussian noise channel which forms the primary basis for the discussions in this book, the difference between the two methods is somewhat academic, since the goal of the encoder is to (i) create a coding system with a convergence threshold as close to channel capacity as possible and (ii) avoid codeword pairs whose *Euclidean distance* is small, which then would cause an error floor in the BER versus signal-to-noise ratio relationship.

However, on certain other channel models, such as ideally interleaved flat fading channels, the Euclidean distance is no longer the main performance criteria [26, 27, 62]. Instead, one finds that on such channels the *Hamming distance*, i.e., the number of positions where two codewords differ, is the primary performance parameter. The minimum Hamming distance of the code determines the *diversity* of the code and translates into the slope of the BER versus signal-to-noise ratio behavior. Now, if one interleaves symbols, the symbol Hamming distance is the relevant parameter, and if individual bits are interleaved, the bit Hamming distance becomes the primary code parameter. It is quite easy to see that the

bit Hamming distance of a code can be substantially larger than its symbol Hamming distance, giving bit-interleaving a significant edge on such channels. Reference [22] presents cutoff rates for symbol and bit-interleaved coded modulation showing quantitative rate advantages of bit interleaving for a variety of constellations from QPSK to 256 QAM.

While the original motivation for bit interleaving [which was the challenge to make trellis-coded modulation (as in Chapter 3) more robust on interleaved flat fading channels] is perhaps no longer critical since modern codes typically have a large Hamming distance in either symbols or bits, bit interleaving may have other attractive properties. One of these is that separate data streams can naturally be combined as illustrated in Figure 8.39.

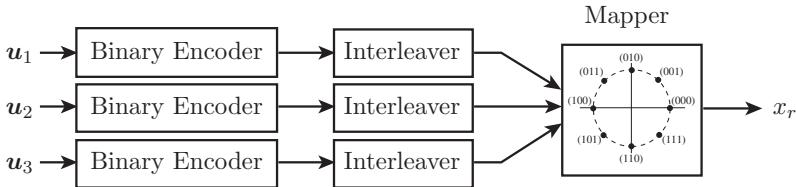


Figure 8.39: Block diagram of a bit-interleaved modulator with separated binary streams.

The advantage of the view espoused in Figure 8.39 is that the different binary data streams do not necessarily need to come from the same source. In multiple antenna systems, for example, each data stream u_i could modulate an individual antenna, with coding and much of the receiver processing done separately and independently of the other data streams. This principle is applied to such a MIMO system in [68]. Furthermore, the different binary encoders could be in separate locations, in which case the “mapper” would be the channel itself; in this case we have a multiple-access channel—see [64] for a more general discussion on multiple access communications.

As discussed in Chapter 3, coded modulation consists of mapping one or several binary data streams onto signals from a finite set of such signals according to a mapping rule. One convenient way of creating the ubiquitous PAM and QAM constellation is to first select one of 2^B amplitude levels for each of B binary data streams. In the case of 8-PAM with $B = 3$, the discrete equi-spaced amplitudes shown in Figure 8.40 are generated as the superposition of three binary constellations, where Bit u_1 has 4 times the power of Bit u_0 , and Bit u_2 has 16 times its power. In effect, this is nothing more than set-partitioning in reverse, where Bit u_0 has the least intraset distance and u_2 has the largest.

In general, any properly labeled 2^B -PAM modulation can be written as the superposition of B binary antipodal amplitudes, i.e.,

$$x = \sum_{j=0}^{B-1} 2^j u_j \quad (8.52)$$

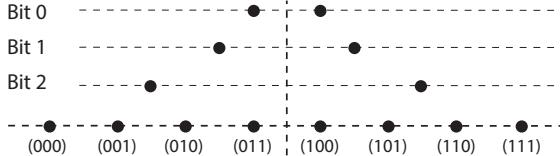


Figure 8.40: Pulse-amplitude modulation (PAM) as superposition of binary antipodal modulation with a geometric power distribution.

where $u_j \in \{-1, 1\}$. If an entire sequence \vec{v} of 2^B -ary PAM symbols is considered, it may be viewed as superposition of B binary data streams with powers $P_0, 4P_0, 16P_0, \dots, 4^{B-1}P_0$ on binary data streams which make up the PAM symbol sequence.

In Figure 8.40 all the binary data streams modulate the same basis waveforms, and therefore the signal points are all co-linear in the geometric signal space representation. However, in multiple user transmission systems, the different coordinated data streams may not necessarily modulate the same waveforms—in fact, this may be rather hard to achieve in practice. The concept of bit-interleaving extended to this case is illustrated in Figure 8.41 below and has been dubbed *generalized modulation* in [65, 67].

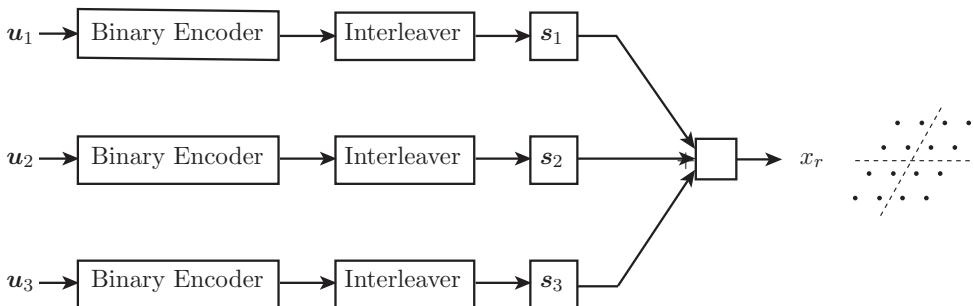


Figure 8.41: Block diagram of a generalized modulation transmission system using separated and physically unconnected binary data streams.

This viewpoint is quite productive in that it suggests a capacity-achieving demodulation method for large constellations based on cancellation. Consider the case where the highest-power uncanceled data stream considers all lower power data streams as noise. Its

maximum rate is then given by the mutual information

$$C_j = I(u_j; y|u_{j+1}, \dots, u_{B-1}) \quad (8.53)$$

where y is the output of the channel. As long as the rate on the j th binary data stream is $R_j < C_j$, it can be correctly decoded using a binary Shannon-capacity achieving code. By virtue of (8.52), knowledge of u_{j+1}, \dots, u_{B-1} implies that these data streams can be canceled from the received signal, and C_j is the capacity of the j th binary data stream. This thought model leads to a successive decoding and canceling method which can achieve the mutual information rate

$$C_{\text{symmetric}} = I(v; y) = \sum_{j=0}^{B-1} C_j \quad (8.54)$$

by the chain rule of mutual information. $C_{\text{symmetric}}$ is of course not equal to the capacity of the additive white Gaussian noise channel $y = v + n$, since the input distribution of v is uniform, rather than Gaussian distributed as required to achieve the channel capacity. In fact, $C_{\text{symmetric}}$ loses the so-called shaping gain of 1.52 dB w.r.t. the capacity of the Gaussian channel [27], unless the basis waveforms used in the modulation process are not equal, i.e., $s_1 \neq s_2 \neq s_3$.

In [67] the authors consider the generalized PAM modulation method which operates with random signals, rather than the more typical orthogonal bases and uses repetition codes as binary error control codes. A two-stage iterative demodulator/decoder, which operates in parallel analogously to an LDPC decoder with an iterative cancelation operation instead of the check node processor, achieves a signal-to-noise ratio (SNR) improvement on each of the binary data streams, such that when these are then treated by external binary error control codes a cumulative data rate per dimension

$$R_d \geq 0.995C_{\text{GMAC}} - 0.12 \quad [\text{bits/dimension}] \quad (8.55)$$

can be achieved, where C_{GMAC} is the Shannon capacity of the Gaussian multiple access channel. That is, this system can achieve a fraction of 0.995 of channel's information theoretic capacity. The result does not depend on size of the system and holds for arbitrarily many data streams u_i .

We conclude this chapter with a potential application of generalized modulation to a multiple access system using user-specific signature sequences (see [64]). Figure 8.42 shows the block diagram of such a multiple user transmitter—receiver chain. Each of a number K of different users transmits a coded binary data stream, where each bit is modulated with a signature waveform, typically a spread-spectrum signal s_k , as illustrated in the top part of Figure 8.42. Without going into details, the resulting channel model is analogous to complex vector channel equation (2.50) of the MIMO channel discussed in Section 2.8.2.

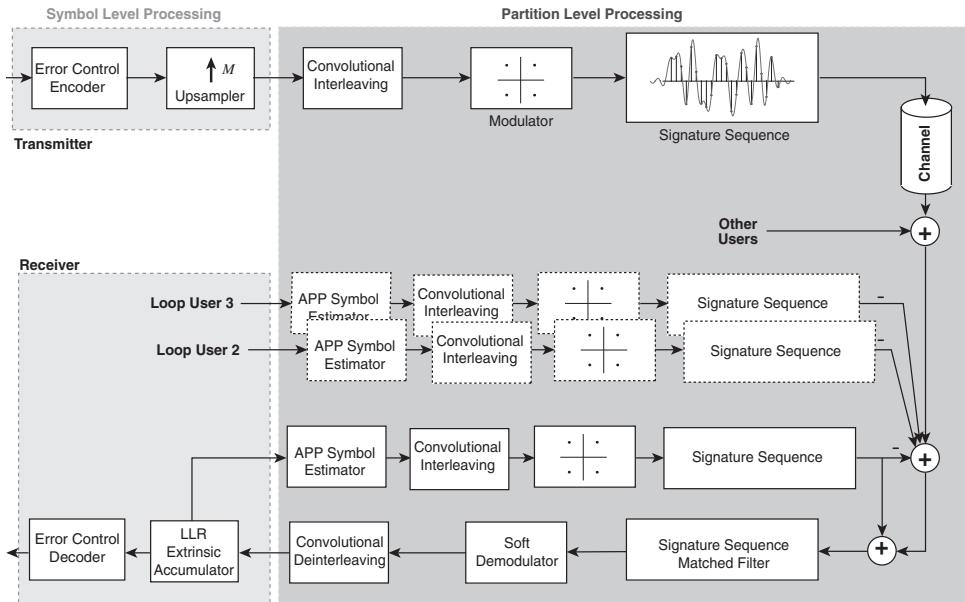


Figure 8.42: Application of generalized modulation to a multiple user system.

The error control coding system is basically external to the system. However, the encoders are followed by a repetition code of rate $1/M$, shown in the figure as the upsampling process, which acts as the error control part in the serially concatenated system which consists of the repetition code and the multiple access channel. The partitioned level processing block constitutes the generalized modulator and demodulator, using the signature sequences of the different users s_k . These signature waveforms are not, and cannot easily be made to be orthogonal. Therefore, the compound constellation at the receiver takes on the form of a generalized modulated signal as discussed above.

Decoding is performed via iterative cancellation and remodulation after symbol LLR combining. The partition-level processing block takes on the form of an LDPC decoder, where “check nodes” are replaced by cancellation nodes. Details of the processing structure and a performance analysis is beyond the scope of this text, but can be found in [65, 67, 66], and an application of the principle to multiple-antenna system is discussed in [68]. This version of generalized modulation is of considerable theoretical interest since it can be shown that the Gaussian multiple access channel capacity can be achieved with this structure, in conjunction with spatial coupling [79]—discussed further in Chapter 10.

Bibliography

- [1] D. Arnold and G. Meyerhans, *The Realization of the Turbo-Coding System*, Swiss Federal Institute of Technology, Zurich, Switzerland, July 1995.
- [2] I. Bahceci and T.M. Duman, “Combined turbo coding and unitary space–time modulation,” *IEEE Trans. Commun.*, vol. 50, no. 8, pp. 1244–1249, Aug. 2002.
- [3] A.S. Barbulescu and S.S. Pietrobon, “Interleaver design for turbo codes,” *Electron. Lett.*, vol. 30, no. 25, p. 2107, Dec. 1994.
- [4] A. S. Barbulescu and S. S. Pietrobon, “Terminating the trellis of turbo-codes in the same state,” *Electron. Lett.*, vol. 31, no. 1, pp. 22–23, Jan. 1995.
- [5] A. S. Barbulescu and S. S. Pietrobon, “Rate compatible turbo codes,” *Electron. Lett.*, vol. 31, no. 7, pp. 535–536, March 1995.
- [6] V. Bhargava and S. Wicker, *Reed–Solomon Codes and Their Applications*, Prentice Hall, Englewood Cliffs, NJ, 1988.
- [7] G. Battail, C. Berrou and A. Glavieux, “Pseudo-random recursive convolutional coding for near-capacity performance,” Proceedings of the Communication Theory Mini-Conference, Globecom ’93, Houston, Texas, pp. 23–27, Dec. 1993.
- [8] G. Battail, “On random-like codes,” unpublished manuscript.
- [9] S. Benedetto and G. Montorsi, “Average performance of parallel concatenated block codes,” *Electron. Lett.*, vol. 31, no. 3, pp. 156–158, Feb. 1995.
- [10] S. Benedetto and G. Montorsi, “Performance evaluation of TURBO-codes,” *Electron. Lett.*, vol. 31, no. 3, pg. 163–165, Feb. 1995.
- [11] S. Benedetto and G. Montorsi, “Unveiling turbo codes: some results on parallel concatenated coding schemes,” *IEEE Trans. Inform. Theory*, vol. IT-42, no. 2, pp. 409–428, March 1996.
- [12] S. Benedetto and G. Montorsi, “Design of parallel concatenated convolutional codes,” *IEEE Trans. Commun.*, vol. COM-44, no. 5, pp. 591–600, May 1996.

- [13] S. Benedetto and G. Montorsi, "Serial concatenation of block and convolutional codes," *IEE Electron. Lett.*, vol. 32, no. 13, pp. 1186–1187, June 1996.
- [14] S. Benedetto, R. Garello, and G. Montorsi, "A search for good convolutional codes to be used in the construction of Turbo codes," *IEEE Trans. Commun.*, vol. COM-46, no. 9, pp. 1101–1105, Sep. 1998.
- [15] S. Benedetto and G. Montorsi, "Iterative decoding of serially concatenated convolutional codes," *IEE Electron. Lett.*, vol. 32, no. 10, pp. 887–888, May 1996.
- [16] S. Benedetto, D. Divsalar, G. Montorsi, and F. Pollara, "Serial concatenation of interleaved codes: Performance analysis, design, and iterative decoding," *TDA Progress Report 42 -126*, pp. 1–26, Aug. 1996.
- [17] S. Benedetto, G. Montorsi, D. Divsalar, and F. Pollara, "A soft-input soft-output maximum a posteriori (map) module to decode parallel and serial concatenated codes," *TDA Progress Report 42-127*, pp. 1–20, Nov. 1996.
- [18] S. Benedetto, D. Divsalar, G. Montorsi, and F. Pollara, "Serial concatenation of interleaved codes: Performance analysis, design, and iterative decoding," *IEEE Trans. Inform. Theory*, Vol. 44, no. 3, pp. 909–26, May 1998.
- [19] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon limit error-correcting coding and decoding: Turbo-codes," Proceedings 1993 IEEE International Conference on Communication, Geneva, Switzerland, pp. 1064–1070, 1993.
- [20] C. Berrou and A. Glavieux, "Near optimum error correcting coding and decoding: Turbo-codes," *IEEE Trans. Commun.*, vol. 44, no. 10, pp. 1261–1271, Oct. 1996.
- [21] C. Berrou and A. Glavieux, "Turbo-codes: General principles and applications," Proceedings of the 6th Tirrenia International Workshop on Digital Communications, Tirrenia, Italy, pp. 215–226, Sep. 1993.
- [22] G. Caire, G. Taricco, and E. Biglieri, "Bit-interleaved coded modulation," vol. 44, no. 3, May 1998.
- [23] G. Caire, G. Taricco, and E. Biglieri, "On the convergence of the iterated decoding algorithm," Proceedings of the 1995 IEEE International Symposium on Information Theory, Whistler, British Columbia, Canada, p. 472, Sep. 1995.
- [24] M. Cedervall and R. Johannesson, "A fast algorithm for computing distance spectrum of convolutional codes," *IEEE Trans. Inform. Theory*, IT-35, pp. 1146–1159, Nov. 1989.
- [25] S.Y. Chung, G.D. Forney, T.J. Richardson, and R. Urbanke, "On the design of low-density parity-check codes within 0.0045 dB of the Shannon limit," *IEEE Comm. Lett.*, vol. 16, 1999.

- [26] D. Divsalar and M.K. Simon, "The design of trellis coded MPSK for fading channels: Performance criteria," *IEEE Trans. Commun.*, vol. 36, pp. 1004–1012, Sept. 1988.
- [27] D. Divsalar and M.K. Simon, "The design of trellis coded MPSK for fading channels: Set partitioning for optimum code design," *IEEE Trans. Commun.*, vol. 36, pp. 1013–1022, Sept. 1988.
- [13] D. Divsalar, H. Jin and R. J. McEliece, "Coding theorems for ‘turbo-like’ codes," *Proceedings of the 1998 Allerton Conference*, pp. 928–936, Oct. 1998.
- [28] D. Divsalar and F. Pollara, "Turbo codes for PCS applications," Proceedings of the 1995 IEEE International Conference on Communications, Seattle, Washington, June 1995.
- [29] D. Divsalar and F. Pollara, "Turbo codes for deep-space communications," *JPL TDA Progress Report 42-120*, Feb. 1995.
- [30] D. Divsalar, S. Dolinar, F. Pollara and R.J. McEliece, "Transfer function bounds on the performance of turbo codes," *JPL TDA Progress Report 42-122*, pp. 44–55, Aug. 1995.
- [31] S. Dolinar, "A New Code for Galileo," TDA Progress Report PR 42-93: January–March 1988, pp. 83–96, May 1988.
- [32] S. Dolinar and D. Divsalar, "Weight distribution for turbo codes using random and nonrandom permutations," *JPL Progress report 42-122*, pp. 56–65, Aug. 1995.
- [33] D. Divsalar, S. Dolinar, and F. Pollara, "Iterative turbo decoder analysis based on density evolution," TMO Progress Report 42-144, Feb. 2001.
- [34] European Telecommunications Standards institute (ETSI) EN 302 307 V1.2.1, European Standard (Telecommunications series), 2009-08.
- [35] G.J. Foschini, "Layered space-time architecture for wireless communication in a fading environment when using multi-element antennas," *Bell Labs Techn. J.*, vol. 1, no. 2, pp. 41–59, Aug. 1996.
- [36] G. D. Forney, *Concatenated Codes*, MIT Press, Cambridge, MA, 1966.
- [37] R. Garello, P. Pierleni and S. Benedetto, "Computing the free distance of turbo codes and serially concatenated codes with interleavers: Algorithms and applications," *IEEE J. Select. Areas Commun.*, vol. 19, no. 5, pp. 800–812, May 1995.
- [38] J. Hagenauer, E. Offer and L. Papke, "Iterative decoding of binary block and convolutional codes," *IEEE Trans. Inform. Theory*, vol. 42, no. 2, pp. 429–445, March 1996.
- [39] C. Heegard and S. Wicker, *Turbo Coding*, Kluwer Academic Publishers, Boston, 1999.

- [40] B.M. Hochwald and S. ten Brink, "Achieving near-capacity on a multiple-antenna channel," *IEEE Trans. Commun.*, vol. 51, no. 3, pp. 399, March 2003.
- [41] P. Höher and J. Lodge, "Turbo DPSK: iterative differential PSK demodulation and channel decoding," *IEEE Trans. Commun.*, vol. 47, no. 6, pp. 837–843, June 1999.
- [42] S. Howard, C. Schlegel, L. Pérez, and F. Jiang, "Differential turbo coded modulation over unsynchronized channels," Wireless and Optical Communications Conference, (WOC 2002) July 17–19, 2002, Banff, Alberta, Canada
- [43] S. Howard, C. Schlegel, "Differential turbo coded modulation with APP channel estimation," *IEEE Trans. Commun.*, vol. 54, no. 8, pp. 1397–1406, Aug. 2006.
- [44] B. Hughes, "Differential space-time modulation," *IEEE Trans. Inform. Theory*, vol. 46, no. 7, pp. 2567–2578, Nov. 2000.
- [45] W.C. Jakes, *Microwave Mobile Communications*, John Wiley & Sons, 1994.
- [46] F. Jiang, *Design and Analysis of Serially Concatenated Convolutional Codes*, Master's Thesis, The University of Nebraska, Lincoln, Dec. 2001.
- [47] O. Joerssen and H. Meyr, "Terminating the trellis of turbo-codes," *Electron. Lett.*, vol. 30, no. 16, pp. 1285–1286, Aug. 1994.
- [48] P. Jung and M. Naßhan, "Performance evaluation of turbo codes for short frame transmission systems," *Electron. Lett.*, vol. 30, no. 2, pp. 111–113, Jan. 1994.
- [49] 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. 285–288, Feb. 1994.
- [50] P. Jung, "Novel low complexity decoder for turbo-codes," *Electron. Lett.*, vol. 31, no. 2, pp. 86–87, Jan. 1995.
- [51] J.W. Layland and L.L. Rauch, "Case studies of technology in the DSN: Galileo, voyager, mariner," in *Evolution of Technology in the Deep Space Network: A History of the Advanced Systems Program*, TDA Progress Report PR 42-130, April–June 1997, Aug. 1997.
- [52] R.J. McEliece, E.R. Rodemich and J.-F Cheng, "The turbo decision algorithm," Proceedings of the 33rd Annual Allerton Conference on Communication, Control and Computing, Monticello, IL, Oct. 1995.
- [53] D. Arnold and G. Meyerhans, *The Realization of the the Turbo-Coding System*, Swiss Federal Institute of Technology, Zurich, Switzerland, July 1995.
- [54] M. Peleg, I. Sason, S. Shamai, and A. Elia, "On interleaved, differentially encoded convolutional codes," *IEEE Trans. Inform. Theory*, vol. 45, no. 7, pp. 2572–2582, Nov. 1999.

- [55] L. C. Pérez, J. Seghers and D. J. Costello, Jr., "A distance spectrum interpretation of turbo codes," *IEEE Trans. Inform. Theory*, vol. 42, no. 6, pp. 1698–1709, Nov. 1996.
- [56] J. Proakis, *Digital Communications*, 1st edition Prentice Hall, Upper Saddle River, NJ, 2007.
- [57] P. Robertson, "Illuminating the structure of parallel concatenated recursive systematic (TURBO) codes," Proc. GLOBECOM '94, vol. 3, pp. 1298–1303, San Francisco, California, Nov. 1994.
- [58] R. Robertson and T. Wörz, "Coded modulation scheme employing turbo codes," *Electron. Lett.*, vol. 31, no. 18, pp. 1546–1547, Aug. 1995.
- [59] R. Robertson and T. Wörz, "A novel bandwidth efficient coding scheme employing turbo codes," *Proc. IEEE Int. Conf. on Commun. ICC'96*, vol. 2, pp. 962–967, 1996.
- [60] R. Robertson and T. Wörz, "Extensions of turbo trellis coded modulation to high bandwidth efficiencies," *Proc. IEEE Int. Conf. Commun.*, vol. 3, pp. 1251–1255, 1997.
- [61] R. Robertson and T. Wörz, "Bandwidth-efficient turbo trellis-coded modulation using punctured component codes," *IEEE J. Select. Areas Commun.*, vol. 16, no. 2, pp. 206–218, Feb. 1998.
- [62] C. Schlegel and D.J. Costello, Jr., "Bandwidth efficient coding on fading channels: Performance analysis and code construction," *IEEE J. Select. Areas Commun.*, vol. 7, pp. 1356–1368, Dec. 1989.
- [63] C. Schlegel and A. Grant, "Concatenated space–time coding," *IEEE Trans. Inform. Theory.*, Vol. 49, No. 9, pp. 2298–2306, Sep. 2003.
- [64] C. Schlegel and A. Grant, *Coordinated Multi-User Communications*, Coordinated Multiuser Communications, Springer, 2006.
- [65] C. Schlegel, Z. Shi, and M. Burnashev, "Asymptotically optimal power allocation and code selection for iterative joint detection of coded random CDMA," *IEEE Trans. Inform. Theory*, vol. 52, no. 9, pp. 4286–4295, Sept. 2006.
- [66] C. Schlegel and D. Truhachev, "Multiple access demodulation in the lifted signal graph with spatial coupling," *IEEE Trans. Inform. Theory*, vol. 59, no. 4, pp. 2459–2470, 2013.
- [67] C. Schlegel, M. Burnashev, and D. Truhachev, "Generalized superposition modulation and iterative demodulation: A capacity investigation," *J. Electr. Comput. Eng.*, vol. 2010, Article ID 153540, Aug. 2010.
- [68] C. Schlegel, D. Truhachev, and Z. Bagley, "Transmitter layering for multi-user MIMO systems," *EURASIP J. Wireless Commun. Networking*, vol. 2008, Article ID 372078, Jan. 2008.

- [69] J. Seghers, *On the Free Distance of TURBO Codes and Related Product Codes*, Final Report, Diploma Project SS 1995, Number 6613, Swiss Federal Institute of Technology, Zurich, Switzerland, Aug. 1995.
- [70] J. Statman, G. Zimmerman, F. Pollara and O. Collins, “A long constraint length VLSI Viterbi decoder for the DSN,” *TDA Progress Report 42-95*, July–Sept. 1998.
- [71] Y. V. Svirid, “Weight distributions and bounds for turbo codes,” *Eur. Trans. Telecommun.*, vol. 6, vo. 5, pp. 543–556, Sept.–Oct. 1995.
- [72] V. Tarokh, A.F. Naguib, N. Seshadri, and A.R. Calderbank, “Space–time codes for high data rate wireless communications: Performance criterion and code construction,” *IEEE Trans. Commun. Theory*, pp. 744–765, March 1998.
- [73] I.E. Telatar, “Capacity of multi-antenna Gaussian channels,” *Eur. Trans. Telecommun.*, vol. 10, pp. 585–595, Nov. 1999.
- [74] S. ten Brink, “Convergence behavior of iteratively decoded parallel concatenated codes,” *IEEE Trans. Commun.*, vol. 49, no. 10, pp. 1727–1737, Oct. 2001.
- [75] S. ten Brink, “A rate one-half code for approaching the Shannon limit by 0.1 dB,” *IEE Electro. Lett.*, vol. 36, no. 15, pp. 1293–1294, July 2000.
- [76] S. ten Brink, “Convergence of iterative decoding,” *IEE Electron. Lett.*, vol. 35, no. 15, pp. 1117–1119, June 1999.
- [77] S. ten Brink, “Design of serially concatenated codes based on iterative decoding convergence,” *Proceedings 2nd International Symposium on Turbo Codes & Related Topics*, Brest, France, Sept. 2000.
- [78] S. ten Brink and G. Kramer, “Design of repeat-accumulate codes for iterative detection and decoding,” *IEEE Trans. Signal Proc.*, vol. 51, no. 11, pp. 2764–2772, Nov. 2003.
- [79] D. Truhachev and C. Schlegel, “Coupling data transmission for capacity-achieving multiple-access communications,” arXiv:1209.5785.
- [80] N. Wiberg, H.-A. Loeliger, and R. Kötter, “Codes and iterative decoding on general graphs,” *Proceedings of the 1995 IEEE International Symposium on Information Theory*, p. 468, June, 1995.
- [81] E. Zehavi, “8-PSK trellis codes for a Rayleigh channel,” *IEEE Trans. Commun.*, vol. 40, pp. 873–884, May 1992.

Chapter 9

Turbo Coding: Applications

9.1 Interleavers

When it comes to applying turbo codes, the low error floor often caused by random interleaving is a problem, and much of the design of the codes goes into designing the interleavers, especially since there is a limited choice of component codes that are practical.

An interleaver Π is a mapping of indices given by a permutation. All the possible permutations of N elements comprise a *permutation group*; that is, the permutations have an algebraic group structure, with an inverse element—the reverse permutation—an identity element, and combining two permutations always produces a third. This is the closure of the group. Permutations can be decomposed into cycles, for example, the interleaver $\Pi_{16} = \{15, 10, 1, 12, 2, 0, 13, 9, 5, 3, 8, 11, 7, 4, 14, 6\}$ from Section 8.2 can written in the disjoint cycle decomposition with two cycles

$$\Pi_{16} = \{(0, 15, 6, 13, 4, 2, 1, 10, 8, 5, 0), (3, 12, 7, 9, 3)\}, \quad (9.1)$$

which simply means that the symbols map as follows: $1 \rightarrow 15 \rightarrow 6 \rightarrow 13 \rightarrow 4 \rightarrow 2 \rightarrow 1 \rightarrow 10 \rightarrow 8 \rightarrow 5 \rightarrow 0 \rightarrow 15$ into a cycle of length 8, and a second cycle of length 4.

The interleaver can also be expressed by its index mapping function

$$d_I(i) = j, \quad 0 \leq i, j < N, \quad (9.2)$$

which we will use most to describe the interleaves.

Let us start with the most common interleaver, the block interleaver of size $N = m \times n$, which functions by writing the incoming information symbols row by row into a rectangular array. Interleaving is accomplished by reading out the information column by column. This interleaver has the indexing function

$$d_{I_B}(i) = ni + \left\lfloor \frac{i}{m} \right\rfloor \mod N, \quad (9.3)$$

where $\lfloor x \rfloor$ is the largest integer $< x$. This floor function makes the interleaver difficult to analyze, and a ‘linearized’ interleaver is introduced in [28] with indexing function

$$d_{I_N}(i) = ki + u \mod N, \quad (9.4)$$

where k and u are fixed integers, and k is relatively prime to N , i.e., the greatest common divisor of k and N , $\gcd(k, N) = 1$. The index k is called the *angular coefficient* of the linear interleaver. Coefficients k of approximately \sqrt{N} seem to work best, resulting in good scattering of the index points [28]. The interleaving function $j = d_{I_N}(i)$ has only one solution and thus describes a proper permutation with a one-to-one mapping of elements. This can be seen as follows, assuming that two indices map into the same interleaved index, that is,

$$\begin{aligned} ki_1 + u &= ki_2 + qN, \\ k(i_1 - i_2) &= qN. \end{aligned} \quad (9.5)$$

But, since the $\gcd(k, N) = 1$, and $|i_1 - i_2| < N$, we conclude that $i_1 = i_2$.

Furthermore, if the input sequence has weight 2, and the two 1’s are separated by t positions, say at position i and $i + t$, then the output symbols are located at

$$ki + u, k(i + t) + u \longrightarrow \tau = kt \text{ positions apart.} \quad (9.6)$$

For any given τ there exists a unique value of t , and these interleaves can generate large values of d_{\min} , but fail to also generate low multiplicities. The latter, which we call a *thin spectrum*, is paramount for the performance of the codes.

Now, if the input sequence is

$$u(D) = ((1 + D^t) + (1 + D^t)D^\tau) D^q, \quad (9.7)$$

the corresponding output sequence is

$$u'(D) = \left((1 + D^\tau) + (1 + D^\tau)D^{t'} \right) D^{q'}, \quad (9.8)$$

However, since

$$(1 + D^t) + (1 + D^t)D^\tau = (1 + D^\tau) + (1 + D^\tau)D^t, \quad (9.9)$$

choosing $\tau = z_{\min}$ we have a situation where the interleaver fails to break up pairs of z_{\min} sequences, which reappear after interleaving, causing low-weight parity sequences in both encoders. Thus, block interleavers may achieve good values of d_{free} , but fail to generate that “thin” spectrum.

The next step towards improving performance of turbo codes in the error floor region was achieved by introducing S -random interleavers introduced in [8], which explicitly avoid

mapping close indices at the input to close indices at the output. Generation of spread interleavers as defined on Page 370 is straightforward. They are generated analogously to random interleavers, selecting each index $d_I(i)$ randomly and then testing if $|d_I(i) - d_I(l)| \geq T$ for all $i - S < l < i$, i.e., by checking all the previously chosen positions. If an index choice fails to fulfill this condition anywhere, it is discarded and new index is chosen. The process continues until all N indices have been chosen. Evidently, the search time for an appropriate interleaver increases with S and T , and the algorithm may not complete successfully if these parameters are chose too aggressively. However, according to [8], values of $S, T < \sqrt{N}/2$ usually complete within reasonable time. Spread interleavers successfully map short weight 2 input sequences into long weight 2 input sequences and thus avoid the problem of a low free distance as discussed earlier.

In an effort to construct deterministic interleavers which have the statistical properties of random interleavers, Takeshita and Costello [28] designed what they called *quadratic interleavers*, which are defined by the quadratic indexing function

$$d_{I_N}(i) = \frac{ki(i+1)}{2} \mod N, \quad (9.10)$$

where k is an odd constant. Statistical comparisons with “ideal” uniform interleavers were conducted for weight 1 and weight 2 sequences. Weight 1 sequences are indeed uniformly shifted into all N possible positions as expected, while the statistics of weight 2 input sequences do not precisely match those expected by an ideally random interleaver. Nonetheless, quadratic interleavers perform well and close to random interleavers, including in the error floor region—and therefore present no improvement over other, pseudo-randomly chosen permutations, for example those based on *maximum-length shift* register designs, so-called m -sequences, which have good random statistical properties [10]. Their main advantage of these methods is the rapid algorithmic construction, which may have to be performed at the receiver on the fly, as code parameters change in adaptive communication systems. They also avoid large look-up tables otherwise necessary.

Figure 9.1 shows different interleavers for $N = 512$ as 2-dimensional scatter plots, i.e., the horizontal axis is the original index i and the vertical axis is the permuted index $d_I(i)$. The scatter plots give a simple visual representation of a given interleaver. The left plot in Figure 9.1 is that of a linear block interleaver, the middle plot is that of a quadratic, and the right plot shows an S -random interleaver. A truly random interleaver has a scatter plot which is visually indistinguishable from that of a quadratic interleaver. We see the basic effects of the interleavers in yet another way, as it becomes evident that the linear interleaver gives a good spread, and hence a large d_{free} , but its regular pattern allows for large multiplicities, while the quadratic interleaver offers random-like behavior, but has poor spreading, leading its turbo code to suffer from a high error floor. The S -random interleaver, in contrast, combines a good spreading property with the random-like quality of the index points which avoids the accumulation of large multiplicities.

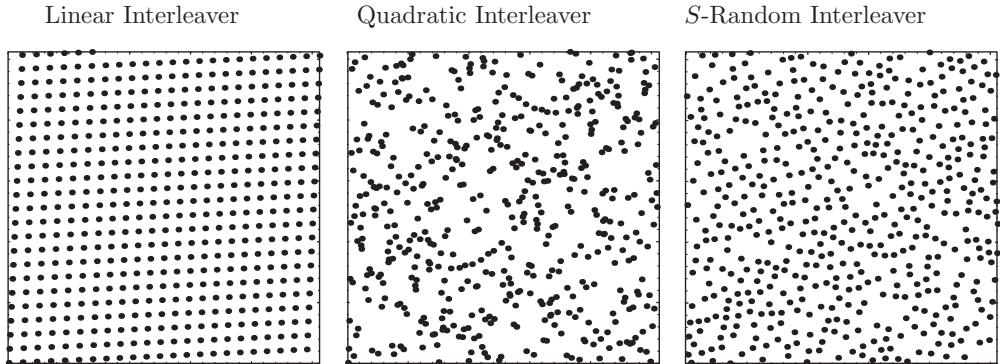


Figure 9.1: Two-dimensional scatter plots of a linear, a quadratic, and an S -random interleaver for a block size of $N = 512$.

As an example of special designed interleavers, we discuss the *Dithered Relatively Prime* (DRP) interleavers invented by Crozier and Guinard [7], which represent a popular and effective interleaving method combining simpler component interleavers. There are

- An input interleaver I_a for a fixed small blocks of size R . It is given as a predefined permutation of R elements.
- An output interleaver which is also fixed and of size W (typically $W = R$).
- A central interleaver which is a simple linear interleaver over the entire block.

This interleaver, which acts as a single permutation, is generated by two local dithering operations and a global permutation operation. This allows the interleaver to be represented as a collection of short vectors, while the central interleaver can be calculated recursively, allowing the overall permutation to be stored and implemented using less memory than otherwise needed.

The arrangement of these interleaves and the combined scrambling effects are shown in Figure 9.2. The notation $r([i]_R)$ and $w([i]_W)$ denote the integer part of i when divided by R and W , respectively. M is the least common multiple (l.c.m) of R and W , and represents the number of pairs of indices that need to be stored to describe the two dither component interleaves. M is therefore a measure of their size, and as can be seen in Figure 9.3, even small values of M are sufficient to have a noticeable impact on the error floor onset. $M = 1$ has a performance similar to a random interleaver at this code size.

The authors [7] used a their own method of generating high-spread in the interleavers [4]. These examples illustrate that with sufficient design efforts, the d_{free} problem

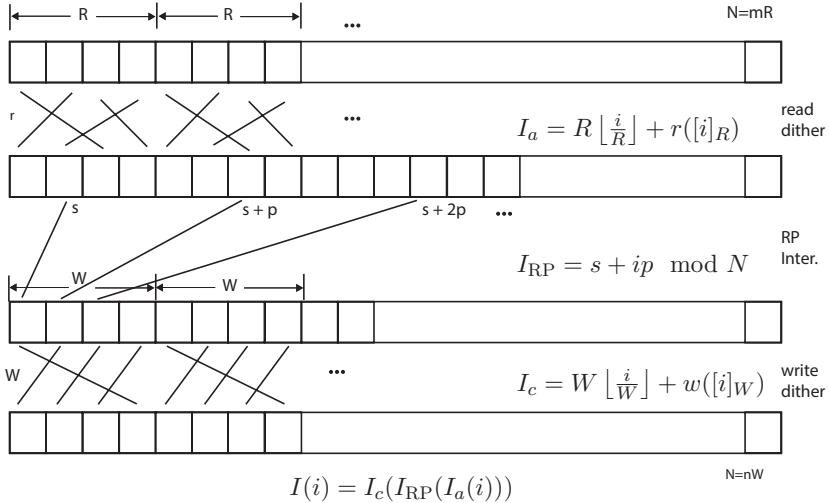


Figure 9.2: Dithered-prime interleaved developed by Crozier and Guinand.

or random turbo codes can largely be controlled and low-weight codewords are eliminated with appropriate hand-tailored designs.

While cycle-reduction designs like those discussed in connection with the interleavers for LDPC codes in Section 6.5.2 are also relevant for turbo coding, the interleavers here are more based on the two principles of randomness and spreading. The influence that the designs have are then evaluated by finding the code's free distance, through the methods of short component sequences are discussed in Chapter 8.

Table 9.1 shows the impressive free distances achievable with DRP interleavers, with values of d_{free} exceeding 50. The error floor behavior of these codes hence is vastly superior to that of the original standard turbo codes. Such procedures de facto eliminate the error floor in turbo codes.

While pseudo codewords—that is, patterns that satisfy the component code constraints without being codewords—can exist in these codes, their impact on the error performance appears to be negligible. This is likely due to the fact that the component codes enforce locally consistent code sequences, thus allowing less room for non-codeword error patterns to be practically relevant. We recall that with LDPC codes, the exact opposite was the case in the error floor, which was dominated by non-codewords.

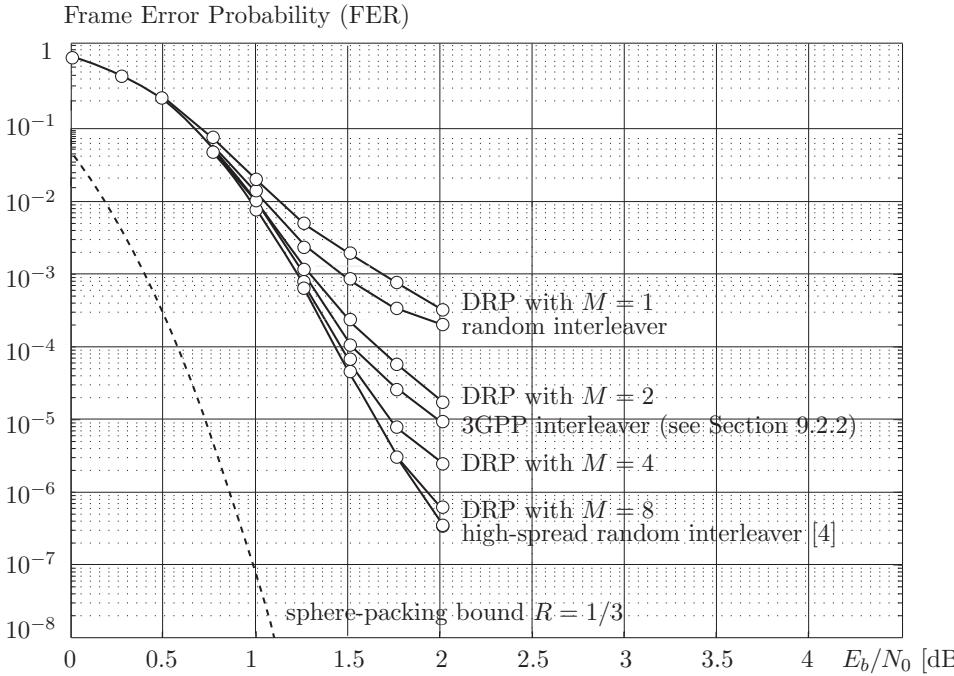


Figure 9.3: Performance of short length-256 turbo codes with different level of DRP interleaves. $M = R = W$ was chosen for simplicity.

N	$M = W = R$	d_{free}
512	1	28
512	2	36
512	4	38
512	8	44
1024	8	50
2048	8	54
4096	16	54
8192	16	58

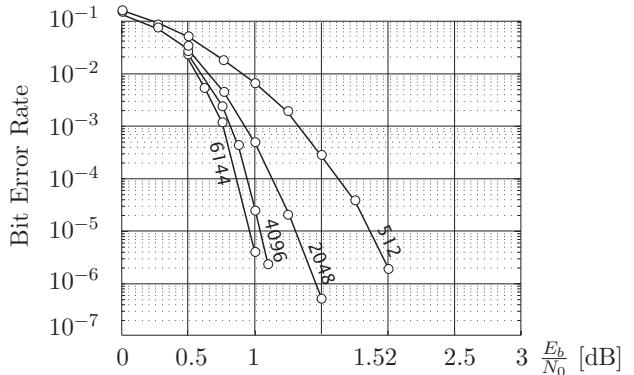


Figure 9.4: Achievable free distances of mid-range rate $R = 1/3$ turbo codes with DRP interleaver designs. Source [7]. Right: Bit error performance for the code sizes indicated.

Finally, we wish to discuss a variant for interleaving, the *convolutional interleaver*. Convolutional interleaving can present an advantage over block interleaving for streaming-based applications, where the size of the system is not necessarily fixed or predetermined. Such a system will be discussed in Section 10.2 on convolutional LDPC codes. Apart from the delay associated with the initial zero-padding, convolutional interleavers can also significantly reduce the delays in the overall system, as well as reduce memory requirements by about one-half [23].

As shown in Figure 9.5, a convolutional interleaver is designed by diagonally splitting the $B \times L_N$ rectangular array of a block interleaver into two triangular structures [12]. The lower triangular part is inserted between the channel encoder and the modulator. The upper triangular is the convolutional de-interleaver and is inserted between the channel and demodulator.

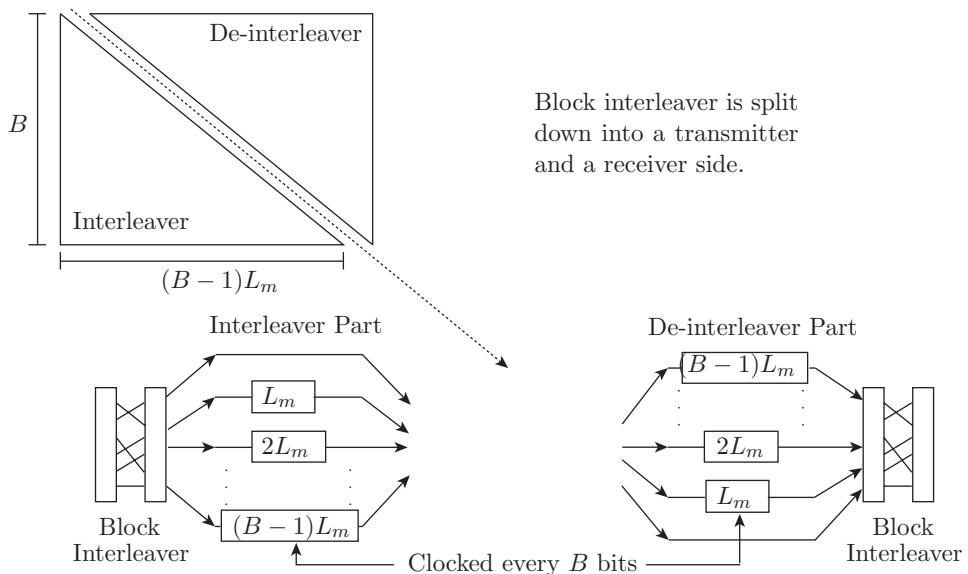


Figure 9.5: (B, L_N) convolutional interleaver with step size L_M .

This structure is referred to as a (B, L_N) convolutional interleaver. B is chosen to be larger than the average length of the burst errors expected in the channel, or some other relevant parameter. It is known as the interleaver depth, and represents the separation between two consecutive symbols, as they appear over the channel. L_N is the maximum delay inserted, where $L_N = (B - 1)L_M$, and L_M is the step size of the convolutional

interleaver. At the transmitter, the coded sequence is fed into a $B \times L_N$ triangular array of shift registers. The i th ($1 \leq i \leq B$) shift register has a length of $(i - 1)L_M$ stages. The registers are clocked once every B symbols and the oldest symbols in the registers are shifted out to the channel. Both the total interleaving delay and required memory are $B(B - 1)L_M$, as can be seen from the block representation in Figure 9.5. This is less than half of that of an equivalent (B, BL_M) block interleaver.

At the receiver, the de-interleaver applies an inverse operation through a structure of shift-registers, as shown in Figure 9.5. In the application of interleavers to turbo-like codes, where randomization is important, the convolutional interleaver is typically preceded by a random block interleaver of size B , which randomizes the inputs via an input buffer and interleaver structure as shown in Figure 9.5. The effects of adding this random ingress and egress interleaves is visible in the scatter diagram of two convolutional interleaver approaches shown in Figure 9.6, where the parameters are $B = 25$ and $L_M = 1$. The regular convolutional interleaver's pattern is very regular and thus ill-suited for iterative coding systems. The interleaver to the right uses 50 different, size- B block interleavers which are prepended to the convolutional interleaver and are cyclically applied. This makes the overall interleaver resemble a random interleaver while maintaining the convolutional delay constraint. The delay of the regular interleaver equals $B(B - 1) = 600$, while that of the randomized interleaver is $B(B + 1)$.

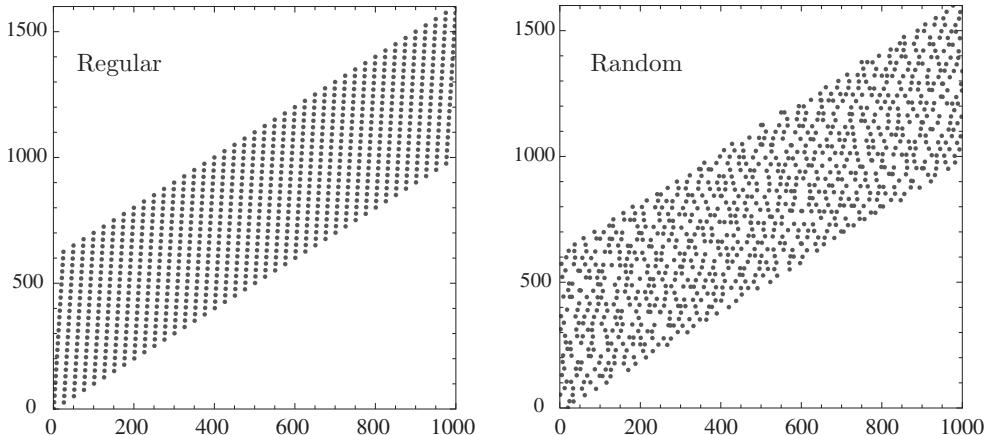


Figure 9.6: Scatter diagram of convolutional interleavers with $B = 25$ and $L_M = 1$.

9.2 Turbo Codes in Telecommunication Standards

Given their stunning performance, it is not surprising that turbo codes have been rapidly adopted for application in many commercial systems. This has led to a number of standards which incorporate turbo codes, ldpc codes, or other versions of graph-based, iteratively decoded error control codes. We use the subsequent sections to highlight specific coding aspects with the help of a few prominent new standards that include these codes. The goal here is not to be exhaustive in our discussion, but to bring a broad flavor of how these new codes have been adapted to a variety of specific communications situations.

9.2.1 The Space Data System Standard

Space, especially deep-space communications, has always been a test ground for new communications methodologies, and has played that role for error control coding systems for well over half a century now. The Consultative Committee for Space Data Systems (CCSDS), which is composed of space agencies and industrial associates worldwide, was among the first to develop a channel coding standard based on parallel concatenated turbo codes. This new standard was to provide an alternative to the traditional concatenated coding system which used a 64-state convolutional code with an outer (255,223) Reed–Solomon code, the traditional concatenated coding workhorse space communications. The new standard based on turbo codes can provide an additional coding gain of 2.5 dB at $P_b = 10^{-5}$ if the lowest rate of $R = 1/6$ is used [24]. The CCSDS Telemetry Channel Coding standard [5] uses a turbo code with two component codes with selectable rates and block-lengths.

The encoder for this code is shown in Figure 9.7. Two rate $R = 1/4$ recursive convolutional encoders are used to generate $R = 1/2, 1/3, 1/4$, and $R = 1/6$ turbo codes. The two switching matrices combine these rates, where a solid circle means every symbol is taken, and an open circle means every other symbol is taken. The feedback polynomial $h_0 = D^4 + D + 1 = 23$ of both encoders is a primitive polynomial of degree 4 and the feed-forward polynomials are $h_1 = 33$, $h_2 = 25$, and $h_3 = 37$. The various rates are achieved by using the connections shown on the right side of Figure 9.7. For example, to achieve $R = 1/4$ the four outputs are the systematic bit, the second and third parity bits from encoder 1 and the first parity bit from encoder 2.

Trellis termination is accomplished by first terminating code number 1, then code number 2, by equating the input bit to the feedback bit for four symbol clock ticks. This causes the encoders to be loaded with the all-zero state.

The interleaver is a block permutation interleaver that has good spreading factors and, unlike the S -random interleavers, has an algorithmic implementation and is scalable. The interleavers are fashioned according to Berrou’s analytical algorithm [6]. The CCSDS standard allows five interleaver lengths $N_1 = 1,784$, $N_2 = 3,568$, $N_3 = 7,136$, $N_4 = 8,920$,

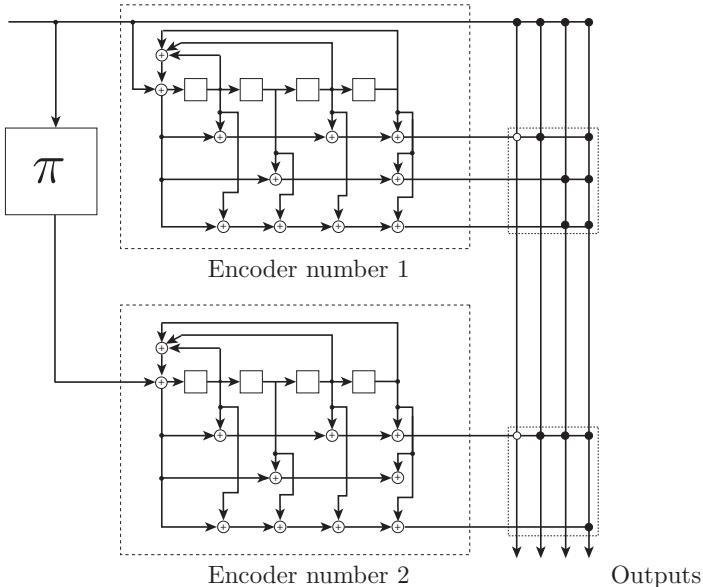


Figure 9.7: The CCSDS turbo encoder. A solid circle means every symbol is taken, and an open circle means every other symbol is taken.

and $N_5 = 16,384$. The first four block lengths are chosen to be compatible with an outer Reed–Solomon code with interleaving depths of 1, 2, 4, and 5, respectively. The largest block length is allowed for those users requiring the most power efficiency and able to tolerate a larger decoder latency. The BER and frame error rate (FER) performance of the CCSDS turbo code with $N = 8,920$ and various rates is shown in Figure 9.8.

9.2.2 3G Wireless Standards

The third-generation (3G) mobile telecommunication standard [19] was developed by the International Telecommunications Union (ITU). The system is more formally known as the International Mobile Telecommunications 2000 (IMT-2000) system, targeting wireless wide-band packet-switched data services of speeds up to 2 Mbit/s. There were two main bodies undertaking work for the IMT-2000 standardization: The Third-Generation Partnership Project (3GPP) and 3GPP2. 3GPP addressed a wide-band CDMA (W-CDMA) air interface, and 3GPP2 addressed the cdma2000 air interface, which evolved from the

Frame/Bit Error Probability (BER)

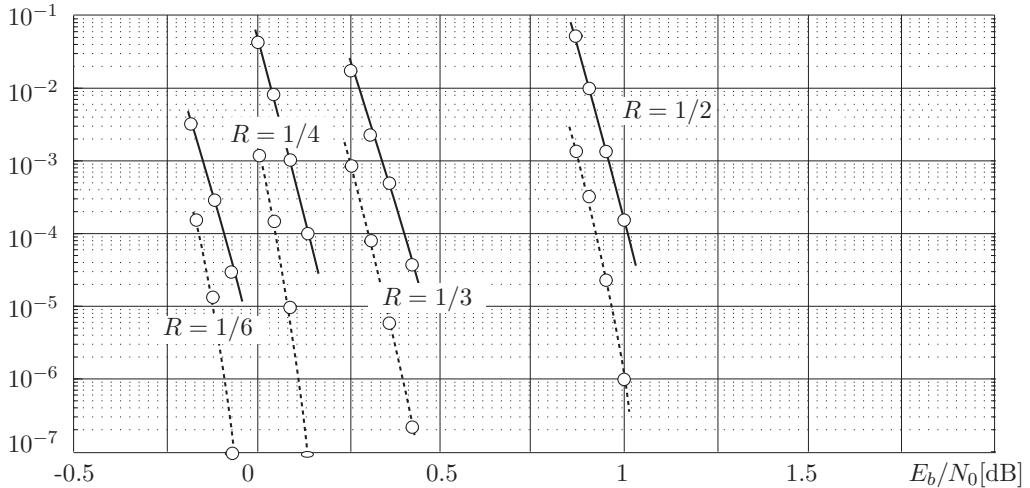


Figure 9.8: Frame error (solid curves) and bit error rate (dashed curves) performance of the CCSDS turbo code family with $N = 8,920$.

IS-95 interim standard. Both specify the use of turbo codes. Besides a constraint-length 9, convolutional code 3GPP also specifies the rate $R = 1/3$ turbo code shown in Figure 9.9 using 8-state constituent component codes.

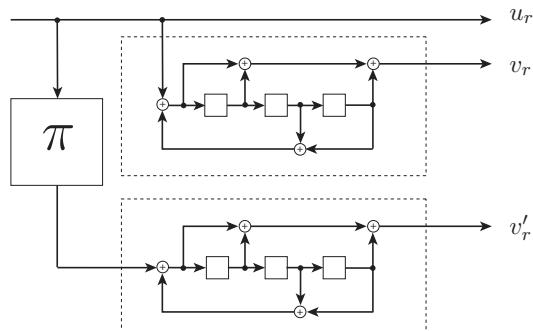


Figure 9.9: The 3GPP turbo encoder uses 8-state component codes.

The cdma2000 standard specifies 3 types of convolutional codes with rates $R = 1/4, 1/3$, and $1/2$, all with constraint-length 9. The convolutional codes are used for signaling channels. The turbo codes use code rates of $R = 1/2, 1/3, 1/4$, or $R = 1/5$, and employ 8-state constituent component codes. The encoder is shown in Figure 9.10 and reveals its similarity to the 3GPP encoder. Different rates are achieved by puncturing.

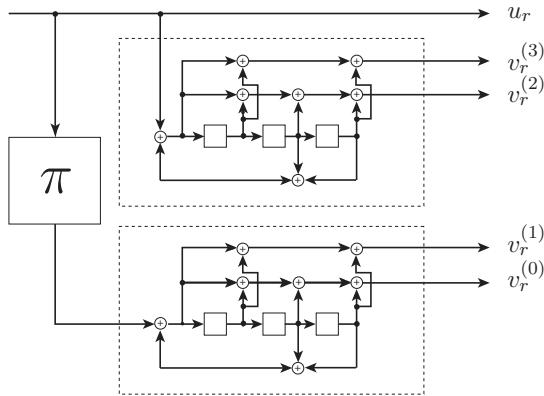


Figure 9.10: The cdma2000 turbo encoder also uses 8-state component codes.

For multimedia communications a wide range of possible block lengths are required, and the 3GPP standard specifies interleavers for block sizes ranging from $40 - 5,114$. Due to this, interleavers with an efficient algorithmic generation method are preferred. Prime interleavers were proposed [25] and later adopted by 3GPP [30]. The interleavers adopted by 3GPP2 have similar structures as those for 3GPP [31].

In general, all these interleavers are based on a block interleaver, called the *mother interleaver*. The information bits are written into a size $m \times n$ array, after which intra-row and inter-row permutations are performed. The final interleaved bits are read out column by column. The different interleavers differ in the way these permutations are performed. Two approaches are used to accomplish the intra-row permutations. In the so-called prime interleavers [25, 63], the rows are basically circularly shifted by prime numbers distinct to each row. In the interleavers for 3GPP2 the circular shifting is accomplished with a group of angular coefficients. The net effect of both strategies is to make the interleaver less regular w.r.t. to the original mother interleaver in order to avoid the high multiplicities of the former. The interleavers are also easy to construct algorithmically, which is a primary requirement for rapid block length changes. The visual scatter plots of examples of these two interleaver types shown in Figure 9.11 give an impression of their randomness.

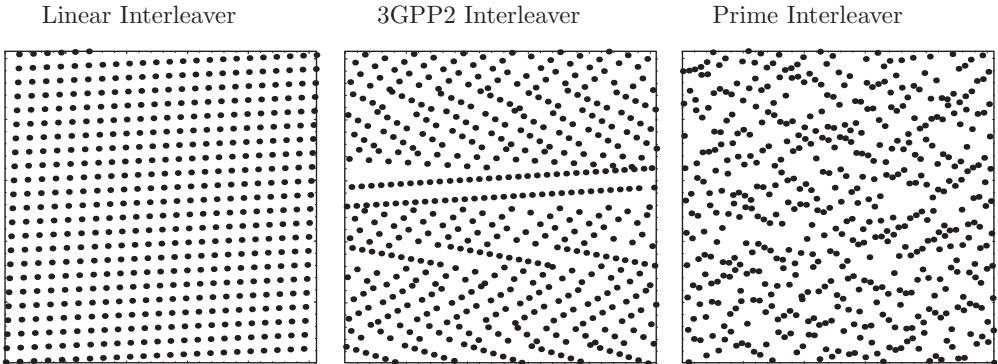


Figure 9.11: Two-dimensional scatter plots of a linear and the two 3GPP interleavers for a block size of $N = 512$.

9.2.3 Digital Video Broadcast Standards

The digital video broadcasting project (DVB) was founded in 1993 to create a framework for digital video services [32]. The DVB *return channel via satellite* RCS prescribes two coding schemes for the traffic and control channels: (i) a concatenated coding method with a by-passable outer Reed–Solomon (RS) code with an also by-passable inner non-systematic convolutional code, and (ii) a duo-binary turbo coding method described here. During a session the system is to stay with one of the coding methods. Both codes have been standardized for the return channel of digital broadcasting by the DVB committee and the European Telecommunications Standards Institute (ETSI), aiming at providing two-way, asymmetric broadband internet access at speeds between 144 kbit/s and 2 Mbit/s to complement ADSL and cable modems [32]. The code used is a variation of the standard turbo code. It uses two input bits (duo-binary) and only a single encoder that is used alternately to generate the direct parity stream and the interleaved parity stream. Its block diagram is shown in Figure 9.12. The input bits are alternately taken from the direct input stream and the interleaved input stream and fed into the recursive duo-binary encoder which produces the parity bits. The rates supported are $R = 1/2, 2/5, 1/2, 2/3, 3/4, 4/5$, and $R = 6/7$ through various puncturing schemes. Bits are interleaved in pairs, and there are N pairs with $N \in \{48, 64, 212, 220, 228, 424, 432, 440, 752, 848, 856, 864\}$ to provide for short frames.

The way the duo-binary encoder works is that it first encodes the direct information sequence and then, in a second pass, the interleaved sequence. This is done by reusing the same encoder. A small complication is introduced in that the codes are tail-biting, and therefore the final state must be found in a first encoding pass, which then determines the

starting state for the actual encoding pass that produces the parity sequence. Decoding is performed in the standard fashion discussed in Chapter 8.

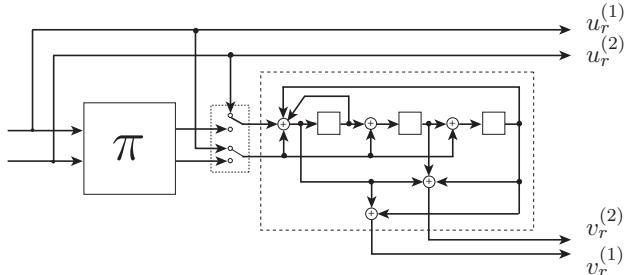


Figure 9.12: Self-concatenated turbo encoder used in the digital video broadcast standard.

In the early 2000's it became clear that advances in satellite technology would allow increased efficiency, and new channel coding schemes, combined with higher-order modulation, were developed [33]. They promise a capacity gain on the order of 30% at a given transponder bandwidth. The new coding standards are realized by a concatenated system with an inner BCH and an outer Repeat-Accumulate (LDPC) code at rates $1/4, 1/3, 2/5, 1/2, 3/5, 2/3, 3/4, 4/5, 5/6, 8/9$, and $9/10$. The FEC coded block can have a length of either $n = 64,800$ or $n = 16,200$ bits.

It's informative to look at the error control structure of this standard as shown in Figure 9.13. This format is very common to many implementations. The outer BCH code is primarily intended to control the error floor, which, according to latest knowledge, is not strictly necessary.

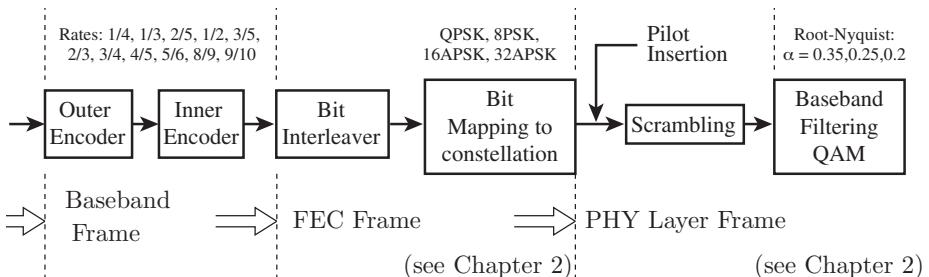


Figure 9.13: Error control coding system for the DVB-S2 standard.

The error control code used in the DVB-S2 standard is a repeat-accumulate code (RA)—see Chapter 6. This code provides an easy encoding mechanism and excellent performance, especially in the error floor region after optimization of the interleaver. As discussed in Chapter 8, RA codes also form a natural link between turbo and LDPC codes and can be regarded as either. Figure 9.14 therefore shows the performance of the rate $R = 1/2$ DVB-Sa code of length $n = 64,800$ [33] in comparison with the original turbo code, which has approximately twice its length, and therefore fares a bit better in the waterfall region.

The standard also provides for a reduced code length of $n = 64,800$ [33] to improve latency. These two base codes are combined into a wide range of code rates (from $1/4$ up to $9/10$); 4 constellation sizes with spectral efficiencies ranging from 2 bit/s/Hz to 5 bit/s/Hz . The transmission spectrum is shaped with Nyquist signals with roll-off factors 0.35, 0.25, and 0.20. Adaptive coding and modulation (ACM) optimizing coding modulation parameters is carried out on a frame-by-frame basis.

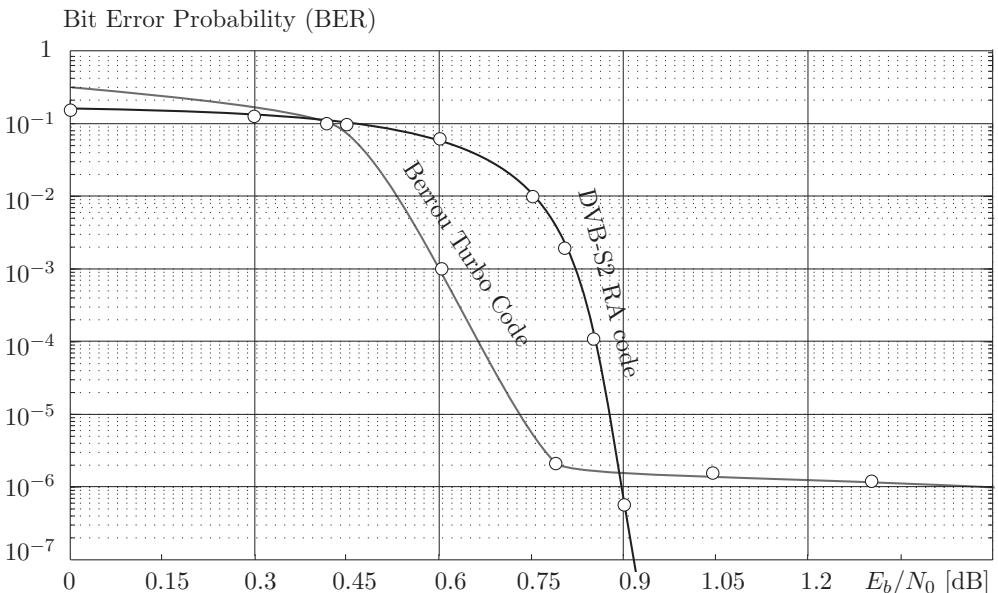


Figure 9.14: Comparison of the DVB-S2 RA code of length 64,800 and the original Berrou turbo code of length $2 \times 65,536$.

9.3 Product Codes and Block Turbo Decoding

In a few applications, such as an option in the IEEE standard 802.16 for broadband wireless access, commercialized under the name “WiMAX,” yet another variation of graph-based codes with iterative decoding is used. These codes are the product codes introduced by Elias in 1954 [9], without, however, presenting an iterative or practical decoding operation.

Product codes also represent a variant of the general turbo coding principle where two relatively weak component codes encode a block of identical, but permuted, information symbols. The difference here is that the interleaver is regular, in fact it is a row/column block interleaver. We have seen that for component convolutional codes such interleavers are not very efficient, however, for block component codes, this restriction does not pose the quite the same problem. Nonetheless, as we will see, the performance of product codes is clearly inferior to that of parallel or serial concatenated turbo codes and LDPC codes. One plausible reason for their limited use is that they are less encumbered by patent claims and that early implementations of decoders were readily available before the widespread use of concatenated turbo decoders. However, we speculate that their popularity is waning and they won’t be likely be used in future applications.

The structure of a product code is shown in Figure 9.15 and comprises a rate $R_2 = k_2/n_2$ row and a rate $R_1 = k_1/n_1$ column code, often taken to be the same code. The overall transmission rate of the product code is $R = R_1R_2$, but often the parity checks on parity checks (dashed box in Figure 9.15) are not transmitted. This increases the rate and produces more favorable values of E_b/N_0 ; however, it also reduces the minimum distance and thus the error floor performance of the code.

The problem of encountering a small value of d_{free} does not typically occur with these codes, since it is quite evident that the minimum distance of the code, including the parity on parities, is the product of the minimum distances for the component codes, and is given by $d_{\text{free}} = d_{\text{free},\text{row}}d_{\text{free},\text{col}}$. Thus large minimum distance codes are easily constructed, and product codes do not suffer from the problem of an error floor as do parallel concatenated turbo codes in the absence of careful interleaver design.

Even though product codes have been well known for a long time (see, e.g., [13]), decoding them to their potential using iterative decoding principles has not been realized until the works of Hagenauer et al. [11] and Pyndiah [20], who both used proper probability message passing algorithms. The iterative decoder which achieves the codes full potential is again derived from the turbo decoding principle, and only minor changes are required. It is shown (again) in Figure 9.16.

Three types of symbols make up a coded block $\mathbf{y} = (\mathbf{y}_k, \mathbf{y}_{p_1}, \mathbf{y}_{p_2})$, where \mathbf{y}_k are the received k_1k_2 information bits, analogous to the systematic information bits in parallel concatenation. \mathbf{y}_{p_1} is the $k_1(n_2 - k_2)$ block of received parity symbols for the row codes, and \mathbf{y}_{p_2} is the $k_2(n_1 - k_1)$ block of parity symbols for the column codes. Each code in each

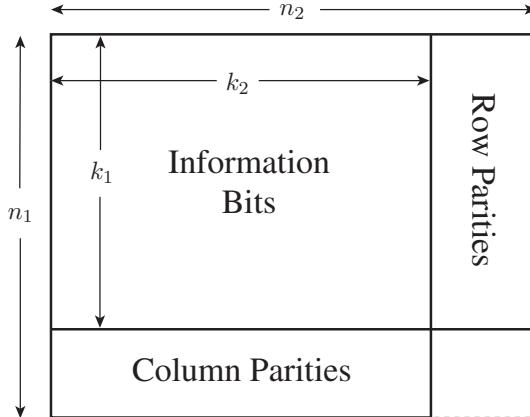


Figure 9.15: Structure of product codes.

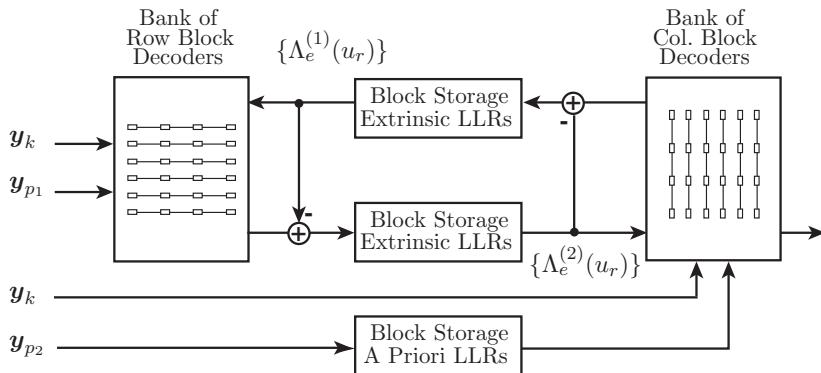


Figure 9.16: Block diagram of a block turbo decoder with banks of constituent soft block component decoders.

column and each row independently decodes the information symbols, using only received signals from its codewords plus extrinsic a priori information on the information bits. The decoders output a posteriori extrinsic information, usually in the form of LLRs, which are stored in memory. Iterations alternate between row decoding and column decoding.

Extrinsic information is generated only for the information bits. Despite short cycles in the factor graphs of these codes, excellent performance is achieved. APP decoders for the component block codes as discussed in Chapter 5 can be used, but those for suboptimal approximate algorithms seem to be more popular due to their complexity advantage.

9.4 Approximate APP Decoding

Trellis-based APP decoding for block codes may in many cases be too complex due to the irregular structure of the trellis of these codes. In such cases, approximate methods have proven to be workable alternatives. On a basic level, any APP decoder needs to compute the bit log-likelihood ratio

$$\Lambda_r = \log \left(\frac{\Pr(b_r = 1|\mathbf{y})}{\Pr(b_r = -1|\mathbf{y})} \right) \quad (9.11)$$

for each information bit b_r or coded binary symbols v_r . Clearly, an exhaustive way of doing this is by calculating the sums

$$\Pr(b_r = 1|\mathbf{y}) = \sum_{\mathbf{x} \in \mathcal{X}_r^{(+)}} \Pr(\mathbf{x}|\mathbf{y}), \quad (9.12)$$

$$\Pr(b_r = -1|\mathbf{y}) = \sum_{\mathbf{x} \in \mathcal{X}_r^{(-)}} \Pr(\mathbf{x}|\mathbf{y}), \quad (9.13)$$

where $\mathcal{X}_r^{(+)}$ is the set of codewords \mathbf{x} corresponding to messages with $b_r = 1$, and $\mathcal{X}_r^{(-)}$ is the set corresponding to messages with $b_r = -1$.

These sums are prohibitively complex since there are too many terms to evaluate, and suitable approximations need to be found. We begin by considering the likelihood ratio (9.11) that the decoder is expected to compute, after we substitute the conditional Gaussian probability density function $\Pr(\mathbf{y}|\mathbf{x}) = \frac{1}{(N_0\pi)^{n/2}} \exp\left(-\frac{|\mathbf{y}-\mathbf{x}|^2}{N_0}\right)$, giving

$$\Lambda_r = \log \left(\frac{\sum_{\mathbf{x} \in \mathcal{X}_r^{(+)}} \exp\left(-\frac{|\mathbf{y}-\mathbf{x}|^2}{N_0}\right)}{\sum_{\mathbf{x} \in \mathcal{X}_r^{(-)}} \exp\left(-\frac{|\mathbf{y}-\mathbf{x}|^2}{N_0}\right)} \right). \quad (9.14)$$

The problem is that there is no short way of calculating this sum of exponentials.

However, if we restrict the sums to only a single term each, $\mathbf{x}_r^{(+)}$ and $\mathbf{x}_r^{(-)}$, we can approximate (9.14) by the much simpler expression

$$\Lambda_r \approx \frac{1}{N_0} \left(|\mathbf{y} - \mathbf{x}_r^{(-)}|^2 - |\mathbf{y} - \mathbf{x}_r^{(+)}|^2 \right). \quad (9.15)$$

We now need to find the two terms $\mathbf{x}_r^{(+)}$ and $\mathbf{x}_r^{(-)}$ which generate the best approximation. This is accomplished by choosing $\mathbf{x}_r^{(+)}$ as the codeword at minimum distance to \mathbf{y} with $b_r = 1$, and $\mathbf{x}_r^{(-)}$ as the codeword at minimum distance to \mathbf{y} with $b_r = -1$.

Furthermore, (9.15) can rewritten as

$$\Lambda_r \approx \frac{2}{N_0} \left(y_r + \underbrace{\sum_{\substack{l=1 \\ (l \neq r)}}^{n_1(n_2)} y_l x_l^{(+)} p_l}_{\Lambda_{e,r} : \text{extrinsic information}} \right), \quad p_l = \begin{cases} 0, & x_l^{(+)} = x_l^{(-)}, \\ 1, & x_l^{(+)} \neq x_l^{(-)}. \end{cases} \quad (9.16)$$

An iterative update equation can now be constructed by equating the term $2y_r/N_0$ above as the soft channel input value and the sum as the symbols extrinsic information, since both represent LLR ratios. This would work well if a true APP decoder, which properly uses a priori information, is used, and the extrinsic update algorithm discussed in Chapter 8 should be used.

However, since the component decoders which are typically used cannot make direct use of a priori information, the following LLR update equation represents a reasonable alternative: For subsequent decoding cycles we let

$$y_r^{(i)} \leftarrow y_r^{(i-1)} + \alpha_{i-1} \Lambda_{e,r}^{(i-1)}, \quad y_r^{(0)} = y_r. \quad (9.17)$$

The central question is, of course, how to find the two codewords $\mathbf{x}_r^{(-)}$ and $\mathbf{x}_r^{(+)}$. Many ad hoc techniques have been explored, and among them we find the *Chase Algorithm* [3]. This algorithm is conceptually simple and exploits the fact that the noise probability decays exponentially with the magnitude of the noise. Succinctly stated, the algorithm restricts the search to a sphere of candidate codewords around the received vector \mathbf{y} and draws the two codewords $\mathbf{x}_r^{(-)}$ and $\mathbf{x}_r^{(+)}$ from this reduced set, which is significantly less complex to search than all of \mathcal{X}_r . The algorithm begins by identifying the least reliable bits of a received sequence \mathbf{y} and enumerates all corresponding binary vectors while fixing the remaining more reliable bits. These vectors form a fixed number of candidates and are decoded into candidate codewords \mathbf{x} , from which the two reduced-size sets $\mathcal{X}_r^{(+)}$ and $\mathcal{X}_r^{(-)}$ are then generated. However, there are a number of difficulties associated with this algorithm:

- The algorithm is ad hoc and it is difficult to evaluate the performance/complexity trade-off without extensive simulations.
- There is a chance that no codeword $\mathbf{x}_r^{(-)}$ (or, alternatively, no codeword $\mathbf{x}_r^{(+)}$) is found in the reduced set. In this case, $\Lambda_r = \beta b_r$ is used an approximate value, where β is an ad hoc adjustment parameter.

- c) A complex ad hoc scaling system $\alpha = [\alpha_0, \dots, \alpha_I]$ is necessary to optimize the performance of the algorithm.

Nonetheless, using the Chase algorithm in this fashion as an approximation of APP decoding produces surprisingly good performance with a manageable computational effort. Figure 9.17 shows the simulated performance of a $[\text{BCH}(64,51,6)]^2$ code with a coding rate of $R = 0.635$. At this rate, the Shannon limit is at $E_b/N_0 = 0.65$ dB, and this limit rises to $E_b/N_0 = 1.1$ dB if the signal constellation is constrained to BPSK.

The simulation results were taken from [20], and the parameters were chosen as $\alpha = [0, 0.2, 0.3, 0.5, 0.7, 0.9]$, $\beta = [0.2, 0.4, 0.6, 0.8, 1, 1]$. Decoding via Chase algorithm used 16 candidate vectors, changing the 4 least reliable bits.

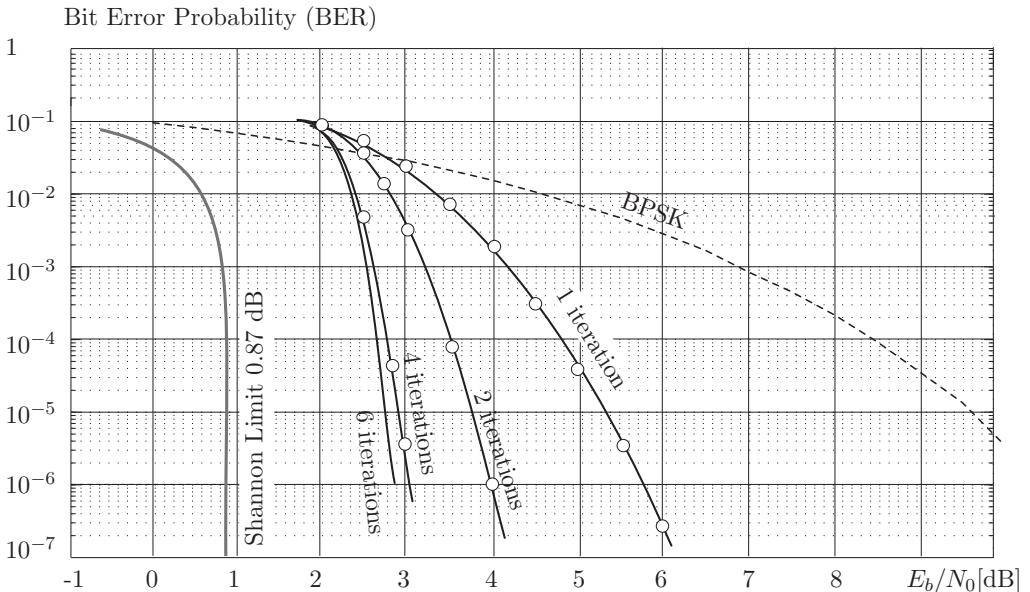


Figure 9.17: Simulations of a $[\text{BCH}(64,51,6)]^2$ product code using iterative decoding with Chase-algorithm-based approximations to the APP algorithm.

Advanced Hardware Architectures, Inc., Pullman, WA, was an early developer who built a series of VLSI decoders based on the concept of product codes. Their codes use

extended Hamming codes as components and use both 2- and 3-dimensional arrays. The extension of the decoding algorithm to 3 dimensions is straightforward. Table 9.1 shows the code combinations and rates in their product line. All codes used are Hamming codes. Partly due to the ready availability of VLSI decoders, product codes with iterative decoding have enjoyed some, however limited, success in standards and applications.

Code Combinations	N	K	R
(16,11) × (16,11)	256	121	0.4727
(32,26) × (16,11)	512	286	0.5586
(16,11) × (16,11) × (4,3)	1024	363	0.3545
(32,26) × (32,26)	1024	676	0.6602
(64,57) × (8,4) × (4,3)	2048	684	0.3340
(32,26) × (16,11) × (4,3)	2048	858	0.4189
(32,26) × (64,57)	2048	1482	0.7236
(32,26) × (16,11) × (8,3)	4096	1144	0.2793
(16,11) × (16,11) × (16,11)	4096	1331	0.3250
(32,26) × (32,26) × (4,3)	4096	2028	0.4951
(64,57) × (64,57)	4096	3249	0.7932

Table 9.1: Table of code combinations offered by AHA corporation.

Figure 9.18 shows simulation results of a [Hamming(64,57,4)]² code with a code rate of $R = 0.793$ for a block size of $n = 4096$. The Shannon bound at that rate is at $E_b/N_0 = 1.1$ dB, whereas the capacity limit using BPSK is $E_b/N_0 = 2$ dB. The competing system is a concatenated RS/convolutional code system with an overall rate of $R = 0.790$.

9.5 Product Codes with High-Order Modulations

We now apply product codes to larger constellations. This is the reason why the entire treatment of product codes and block turbo decoding has been placed in this chapter on turbo-coded modulation. The product encoder generates an array of output binary symbols, and in order to use larger constellations, these binary digits from the FEC encoder are *Gray mapped* onto the larger constellations.

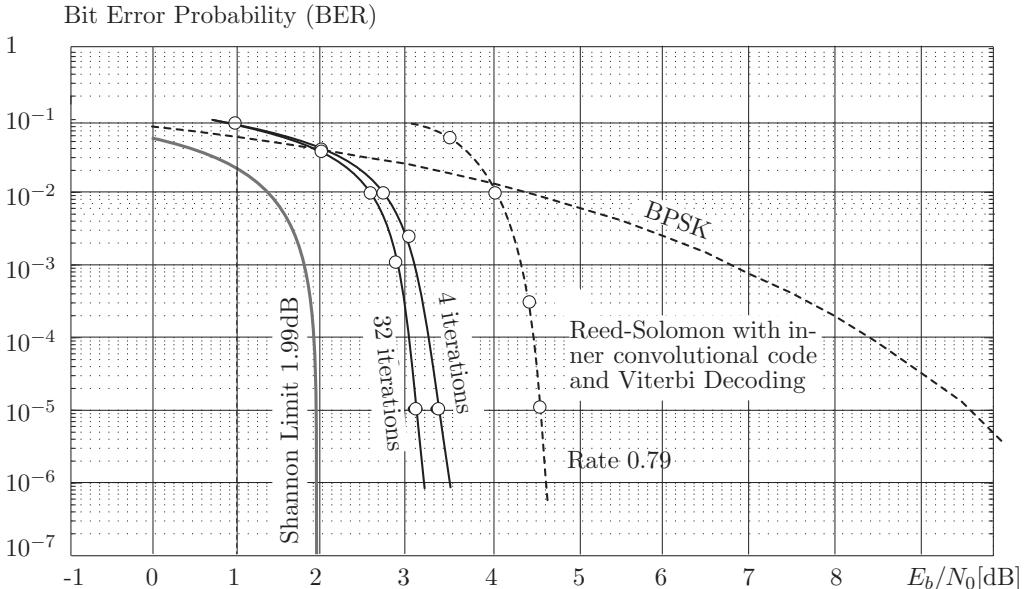


Figure 9.18: Simulations of a $[EHC(64,57,4)]^2$ product code using iterative decoding with AHA’s approximation to the APP algorithm. This code has been standardized for the IEEE 802.16 MAN standard. The competing coding system is a traditional concatenated code with an inner convolutional code and an outer RS code.

The only significant change w.r.t. binary modulation occurs at the decoder, where the LLR calculation has to be revisited. Since the information symbol is “buried” in the larger constellation, such as 8-PSK or 16-QAM, its a priori channel probability is calculated via the marginalization

$$\Pr(b_r = b | y_r) = \sum_{\substack{x_r \in \mathcal{X} \\ (b_r=b)}} \Pr(x_r = x | y_r), \quad (9.18)$$

where b_r is the binary symbol processed by the BTC decoder, and x_r is the modulated symbol from the larger signal constellation used.

From this the bit channel likelihood ratio which is needed at the decoder input is

calculated as

$$\Lambda_r = \log \left(\frac{\sum_{\substack{x_r \in \mathcal{X} \\ (b_r=1)}} \Pr(x_r = x | y_r)}{\sum_{\substack{x_r \in \mathcal{X} \\ (b_r=0)}} \Pr(x_r = x | y_r)} \right). \quad (9.19)$$

Note that this marginalization operation has to be performed only once before iterative decoding begins.

Figure 9.5 shows the performance of such a product-coded modulation system using a large 64-QAM constellation. The code used is a $[\text{Hamming}(64,57,4)]^2$ product code, modulated onto a 64QAM constellation using Gray mapping. This system achieves a data rate of $R = 4.76$ bits/symbol with a block size of $N = 683$ symbols. The Shannon limit at this rate is at $E_b/N_0 = 8.49$ dB. The competing system is a concatenated RS-Trellis Code system with an overall rate of $R = 4.69$ bits/symbol, also shown in the figure. As noted before in the situation with space-time codes, the gap between code performance and the Shannon bound increases with increased spectral efficiency. This is due to the larger modulation alphabets which represent “weak” codes that cannot be decoded efficiently.

9.6 The IEEE 802.16 Standard

IEEE 802.16 is a standard for Wireless Metropolitan Area Networks (MAN), in which product codes with iterative decoding have been standardized as optional FEC for both the up-, and the down-link. The code standardized in 802.16 is a two-dimensional code with one of two Hamming component codes. These are either the [64,57,4], or the [32,26,4] extended Hamming code. The block information sizes of 3249, and 676, respectively, do not match all the required packet sizes, and the code is matched to the different packets by shortening. The longer of the codes achieves a performance of only about 1.2 dB from the Shannon limit with up to 32 iterations, and about 1.5 dB from the Shannon limit at only 4 iterations (see Figure 9.18).

The codes are shortened in a straightforward way by deleting rows and columns until the required block size is achieved. Deleting rows or column reduces the number of bit that are transmitted. At the decoder these bits are then simply assumed to be zero and are known. Decoding otherwise continues in the iterative fashion discussed in this chapter. The shortening affects the rate of the code, but has no other major effect; in particular, the minimum distance and error performance are maintained.

Turbo product codes allow for relatively high data rates. They are mainly intended for satellite communications, wireless internet access, wireless LAN, microwave systems,

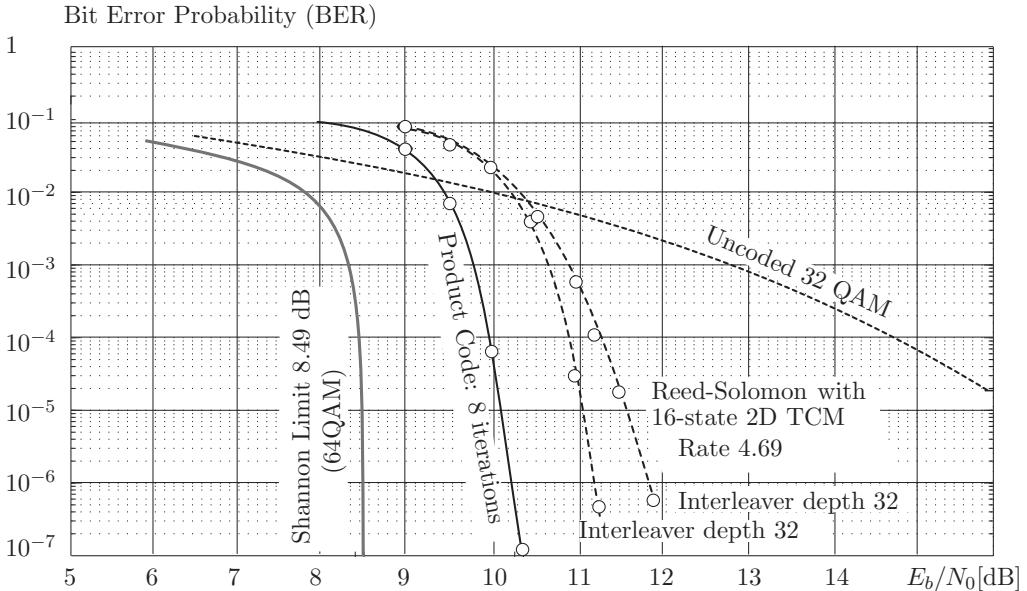


Figure 9.19: Simulations of a $[EHC(64,57,4)]^2$ product code mapped into 64-QAM constellation symbols to increase spectral efficiency. 32 QAM is used as comparison since it provides roughly the same spectral efficiency.

and mobile communications applications. They were also investigated for future wire-line applications such as for ADSL applications but have not been adopted there.

9.7 Decoding of Polar Codes

We introduced Polar Codes in Section 4.16 as a relative of the Reed–Muller codes, both based on a Hadamard construction. However, Polar Codes are more general, and have been shown to be capacity-achieving in the limit of a large block length N [1]. Unlike Reed–Muller codes, Polar Code decoding is carried out on the code’s graph, which we construct below. As indicated in Section 4.16, decoding proceeds bit-by-bit and is based on successive cancellation, which is added to the otherwise familiar message passing mechanisms for graph-based codes we discussed in this book, primarily those from Chapter 6.

We begin our discussion with the code construction graph from Figure 4.28, which we

have extended to a size $N = 8$ example in Figure 9.20. This code is redrawn into the graphical diagram of Figure 9.21, where the order of the stages has been reversed, and we have reverted to the standard notation from Chapter 6 for variable nodes as round circles and check nodes as square boxes. With a little work it can easily be verified that the two graphs represent the same code, but we will find Figure 9.21 more amenable to our discussion of the decoding procedure, and the graph now resembles a Tanner graph for an LDPC code at least in form.

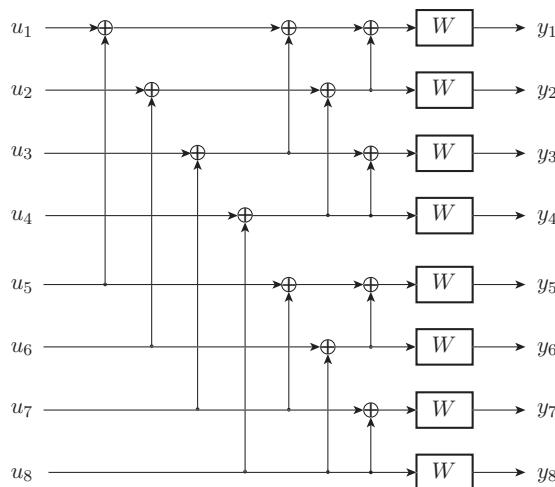


Figure 9.20: Size $N = 2^n = 8$ example polar code.

In principle we can now apply a message passing algorithm on the graph of Figure 9.21 with any arbitrary schedule to find the likelihood values of u_1, \dots, u_8 after injecting the channel log-likelihood ratios (LLRs) $\lambda_1, \dots, \lambda_8$. However, as can readily be appreciated, the graph in Figure 9.21 has cycles, and regular message passing as discussed in Chapter 6 may not lead to optimal performance.

In his seminal paper, Arikan [1] introduced an algorithm based on successive cancellation, which we will discuss now. This algorithm, in essence, forms the basis for all variants of Polar Code decoders. It borrows the message passing methodology from LDPC decoding, with the addition of a cancellation step, which prevents the decoder from having to go through multiple iterations. However, since the code graph has $\mathcal{O}(N \log N)$ edges, even this two-step cancellation algorithm has an order complexity of $\mathcal{O}(N \log N)$, or $\mathcal{O}(\log N)$ per bit, which grows with N , unlike the order decoding complexity of LDPC decoders.

We begin our discussion by noting that the entire code structure in Figure 9.21 consists

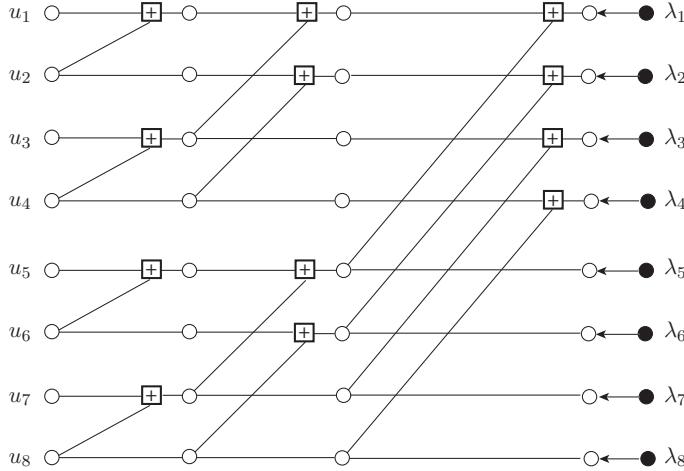


Figure 9.21: Reverse flow diagram of the size $N = 8$ polar code of Figure 9.20.

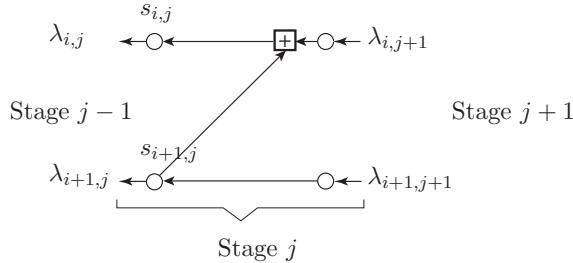


Figure 9.22: “Z”-butterfly which makes up the core structure of the decoding graph in Figure 9.21. The figure shows a butterfly at an arbitrary stage j .

of concatenated and interlocking “Z”-butterflies. As shown in Figure 9.22, such a structure has two binary inputs $s_{i,j}$ and $s_{i+1,j}$, and two output bits connected with a single parity check node. The messages through this butterfly are computed as follows: From the right we receive LLR bit values from an up-stream stage $j + 1$, which are used to compute the LLR values of the down-stream bits. Using the notation in Figure 9.22, we compute

$$\lambda_{i,j} = 2 \tanh^{-1} \left(\tanh\left(\frac{\lambda_{i,j+1}}{2}\right) \tanh\left(\frac{\lambda_{i+1,j+1}}{2}\right) \right), \quad (9.20)$$

$$\lambda_{i+1,j} = (1 - 2\hat{s}_{i,j})\lambda_{i,j+1} + \lambda_{i+1,j+1} \quad (9.21)$$

Equation (9.20) is the basic check node update which computes the LLR of node $s_{i,j}$, while (9.21) is the cancelation step, which depends on the knowledge of $s_{i,j}$, or its decision $\hat{s}_{i,j}$. Depending on its value, the LLR $\lambda_{i,j+1}$ is either added to, or subtracted from, $\lambda_{i+1,j+1}$. It is at this point where the frozen bits of the polar code come into play.

Decoding now progresses from left to right with LLR propagation, and from right to left with hard bit decision propagation. As soon as both input LLRs of a given butterfly are available, the top output LLR of the previous stage can be computed (see Figure 9.22) and as soon as the top left bit is known, the bottom left LLR can also be computed. Butterflies can therefore be activated only when the required inputs are available, and the algorithm is intrinsically sequential and potentially slow. However, substantial speed-up can be obtained by activating non-interacting butterflies simultaneously. This is illustrated in Figure 9.23 with our example code.

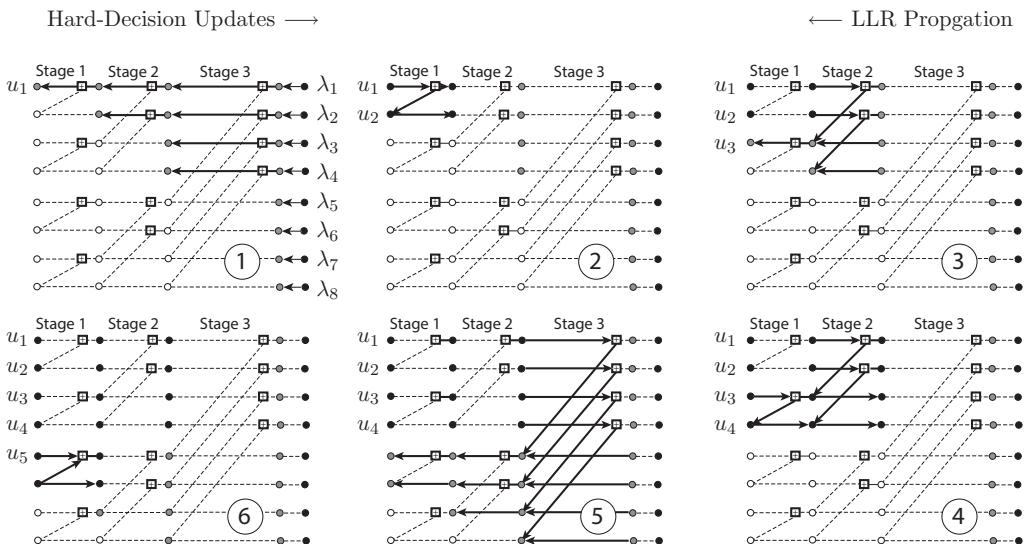


Figure 9.23: Butterfly activation schedule for the example code of size $N = 8$. Solid lines mean activation at that decoding step, grey circles mean input LLR computed, and black circles are decided bits.

In a first phase, 7 butterflies can be activated, 4 in Stage 3, 2 in Stage 2, and 1 in Stage 1. These are marked by solid lines in Step 1, Figure 9.23. We have indicated the availability of LLR information at an intermediate node by shading it in grey. The butterflies have to be activated sequentially through the stages. At the end of the first

phase, bit u_1 has its final LLR and can be decoded; however, it had better be a frozen bit, since its LLR will be very unreliable after seven parity-check gates have “diluted” the LLR. In fact, this is a bad channel, and we may assume u_1 known and darken the circle to indicate this in the figure.

The value of u_1 can now be back-propagated to Stage 2, and the LLR from Stage 2 can be propagated to bit u_2 , which now also has all its LLR information. A decision on u_2 can therefore be made. It will also likely be a frozen bit and is already known. The next forward propagation is now enabled and shown in Step 3.

At this stage the decoding pattern becomes evident. As soon as an information bit u_i has all the required information, a decision can be made, and the bit becomes known, or frozen. Snapshots 1, 3, and 5 in Figure 9.23 illustrate the forward propagation of the likelihood values, while snapshots 2, 4, and 6 show the back propagation of known (frozen) bits. If a bit is already frozen by the code constraint, we technically don’t need to wait for the LLRs but can back-propagate its value directly. This is the basis for some of the accelerated decoding methods in the literature [15, 17, 18, 29, 21].

It is evident that this basic cancelation algorithm entails significant computational redundancy, which can be exploited by more careful designs. Yazdi and Kschischang [29] introduced such a simplified decoder by putting the butterflies into 3 classes: zero-rate, where both bits are known since frozen; half-rate, where one of the bits is frozen, and full-rate, where both bits need to be computed. They report an acceleration of up to twenty-fold, depending on code rate and size.

A fairly rich literature has sprung up regarding decoding methods, as well as implementation designs for polar codes [2, 22, 14]. In general, they are all based on variants of the successive cancelation methodology, even though more general message passing methods using LDPC-type algorithms have been studied. However, given the nested butterfly structure of the codes, general message passing is not only complex, but its performance will strongly depend on the update schedule that is used.

9.8 Polar Code Performance and Outlook

Comparisons between Polar Code implementations and the performance of the more established turbo and LDPC codes abound, but exact comparisons are difficult due to the many parameters that can be chosen. It is however fair to say that Polar Codes present an interesting alternative code family with excellent performance, which is not based on sparse graphs.

The figure below, adapted from [16], presents a performance comparison favoring polar codes. Taken with a grain of salt, we can conclude that comparable levels of performance can be achieved with all three types of coding, at least for half-rate codes. The codes reported are the following: Polar a) is the standard successive cancelation algorithm and

shows the poorest performance. Polar b) uses an enhanced SC decoder which allows up to L candidate path to be explored in parallel at each level [27]. Polar c) is a CRC-enhanced polar code, which uses a 24-bit CRC code in conjunction with the decoding algorithm [17], and Polar d) uses a CRC enabled decoder with an adaptive list size that progressively grows [29]. These two latter coding systems are examples of concatenated codes which are sometimes used to improve the performance of shorter codes, like the size $N = 1024$ polar codes discussed here. Strictly speaking, these codes cannot be directly compared to the turbo codes or the other decoding algorithms, however, the reader get the general performance picture: With enough effort and engineering designs, additional performance gains are typically possible with short and medium-sized codes. All polar codes discussed here were constructed using a method presented by Tal and Vardy [26]; see [16].

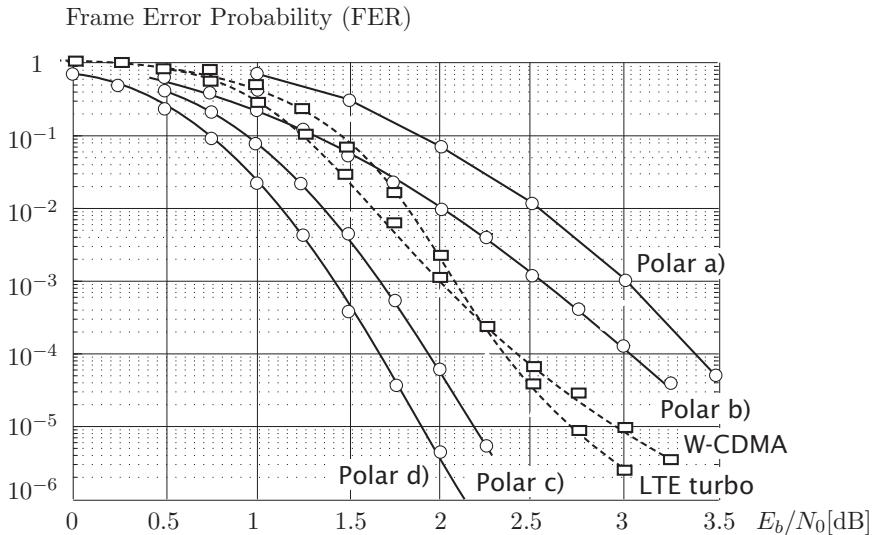


Figure 9.24: Frame error probability of rate $R = 0.5$ and block length $N = 1024$ polar codes with various decoding algorithms, compared to standardized turbo codes of equivalent rate and size.

Bibliography

- [1] E. Arikan, “Channel polarization: A method for constructing capacity-achieving codes for symmetric binary-input memoryless channels,” *IEEE Trans. Inform. Theory*, vol. 55, no. 7, July 2009, pp. 3051–3073.
- [2] A. Balatsoukas-Stimming, A.J. Raymond, W.J. Gross, and A. Burg, “Hardware architecture for list successive cancellation decoding of polar codes,” *IEEE Trans. Circuits Systems II*, vol. 61, no. 8, pp. 609–613, August 2014.
- [3] D. Chase, “A class of algorithms for decoding block codes with channel measurement information,” *IEEE Trans. Info. Theory*, pp. 170–182, Jan. 1972.
- [4] S. Crozier, “New high-spread high-distance interleavers for turbo codes,” *20th Biennial Symposium on Communications*, Kingston, Ontario, Canada, pp. 3–7, May 28–31, 2000.
- [5] *Telemetry Channel Coding*, Consultative Committee on Space Data Systems, CCSDS 101.0-B-5, June 2001.
- [6] *Recommendations for Space Data System Standards*, Consultative Committee on Space Data Systems, CCSDS 101.0-B-6, Oct. 2002.
- [7] S. Crozier and P. Guinand, “High-performance low-memory interleaver banks for turbo-codes,” US Patent, No. 6,857,087, Feb. 15, 2005.
- [8] S. Dolinar and D. Divsalar, “Weight distributions for turbo codes using random and nonrandom permutations,” *JPL TDA Prog. Rep. 42-122*, pp. 56–65, Aug. 1995.
- [9] P. Elias, “Error-free Coding,” *IEEE Trans. Inform. Theory*, vol. IT-4, pp. 29–37, Sept. 1954.
- [10] S.W Golomb, *Shift Register Sequences*, Holden-Day, San Francisco, 1967.
- [11] J. Hagenauer, E. Offer, and L. Papke, “Iterative decoding of binary block and convolutional codes,” *IEEE Trans. Inform. Theory*, vol. 42, no. 2, pp. 429–445, Mar. 1996.
- [12] S. A. Hanna, “Convolutional interleaving for digital radio communications,” *IEEE 2nd International Conference on Universal Personal Communications*, vol. 1, pp. 443–447, Oct. 1993.

- [13] S. Lin and D.J. Costello, Jr., *Error Control Coding: Fundamentals and Applications*, 2nd edition, Prentice Hall, Upper Saddle River, NJ, 2004.
- [14] C. Leroux, A.J. Raymond, G. Sarkis, I. Tal, A. Vardy, and W.J. Gross, “Hardware implementation of successive-cancellation decoders for polar codes,” *J. Signal Proc. Systems*, vol. 69, no. 3, pp. 305–315, Dec. 2012.
- [15] C. Leroux, A.J. Raymond, G. Sarkis, and W.J. Gross, “A semi-parallel successive-cancellation decoder for polar codes,” *IEEE Trans. Signal Proc.*, vol. 61, no. 2, pp. 289–299, Jan. 2013.
- [16] K. Niu, K. Chen, J. Lin, and Q.T. Zhang, “Polar codes: Primary concepts and practical decoding algorithms,” *IEEE Commun. Mag.*, pp. 192–203, Jul. 2014.
- [17] K. Niu and K. Chen, “CRC-aided decoding of polar codes,” *IEEE Commun. Lett.*, vol. 16, no. 10, pp. 1668–1671, Oct. 2012.
- [18] K. Niu and K. Chen, “Stack decoding of polar codes,” *Electron. Lett.*, vol. 48, no. 12, pp. 695–697, Jun. 2012.
- [19] R. Prasad, W. Mohr, and W. Konhauser, *Third Generation Mobile Communication System*, Artech House, Norwood, MA, 2000.
- [20] R.M. Pyndiah, “Near-optimum decoding of product codes: block turbo codes,” *IEEE Trans. Commun.*, vol. 46, no. 8, pp. 1003–1010, Aug. 1998.
- [21] A.J. Raymond and W.J. Gross, “A scalable successive-cancellation decoder for polar codes,” *IEEE Trans. Signal Proc.*, vol. 62, No. 20, pp. 5339–5347, 2014.
- [22] G. Sarkis, P. Giard, A. Vardy, C. Thibeault, and W.J. Gross, “Fast polar decoders: Algorithm and implementation,” *IEEE J. Select. Areas Commun.*, vol. 32, no. 5, pp. 946–957, May 2014.
- [23] B. Sklar, *Digital Communications Fundamentals and Applications*, Prentice Hall, 2nd edition, Englewood Cliffs, NJ, 2001.
- [24] M.R. Soleymani, Y. Gao, and U. Uilaipornswai, *Turbo Coding for Satellite and Wireless Communications*, Kluwer Academic Publishers, Dordrecht, 2002.
- [25] A. Shibutani, H. Suda, and F. Adachi, “Complexity reduction of turbo coding,” *Proc. IEEE Vehicular Technology Conference VTC’99 Fall*, Sept. 1999.
- [26] I. Tal and A. Vardy, “How to construct polar codes,” *IEEE Trans. Inform. Theory*, vol. 59, no. 10, Oct. 2013.
- [27] I. Tal and A. Vardy, “List decoding of polar codes,” *IEEE Int. Symp. Inform. Theory (ISIT)*, 2011, pp. 1–5..
- [28] O.Y. Takeshita and D.J. Costello, Jr., “New deterministic interleaver designs for turbo codes,” *IEEE Trans. Inform. Theory*, vol. 46, no. 6, pp. 1988–2006, Sept. 2000.

- [29] A.A. Yazdi and F.R. Kschischang, “A simplified successive-cancellation decoder for polar codes,” *IEEE Commun. Lett.*, vol. 15, no. 12, Dec. 2011.
- [30] *Multiplexing and Channel Coding (FDD)*, TSG RAN TS25.212 v5.4.0, March 2003.
- [31] *Physical Layer Standard for CDMA2000*, Spread Spectrum Systems Release C, C.S0002-C Version 1.0, May 2003.
- [32] ETSI EN 301 790 V1.5.1 (2009-05) *Digital Video Broadcasting (DVB); Interaction channel for satellite distribution systems*, 2009.
- [33] ETSI EN 302 307 V1.2.1 (2009-08) Digital Video Broadcasting (DVB); Second generation (DVB-S2), 2012.

Chapter 10

Convolutional LDPC Codes and Spatial Coupling

10.1 Capacity: The Ultimate Limit

As we argued in Chapters 6–9, error control coding based on sparse graphical codes has revolutionized the field of error control coding. These systems now routinely are designed for performance within a fraction of the capacity limit of the application they are designed for. However, the inability to design codes which can provably achieve the channel capacity has haunted theoreticians since the early days of coding theory. This, up to recently, open-ended journey towards the Shannon capacity has been eloquently summarized in [4]. It is therefore quite common to find the terminology “near-capacity,” or “capacity-approaching” coding throughout the literature as a testimony to this shortcoming. While not of much relevance to the practitioner, it nonetheless leaves the theory of coding incomplete.

In a final development, this issue has now begun to be settled as well. The invention of polar codes [1] has, for the first time, produced a class of codes which can provably achieve the capacity of the channel, binary-input symmetric-output memoryless channels to be precise. Polar codes were discussed in Chapter 4 as an ingenious variation of Reed–Muller codes. While polar codes have been hailed as a theoretical leap forward, their applications to real-world systems has to date not yet progressed significantly beyond test implementations. However, much progress has been achieved in accelerating the intrinsically slow decoding algorithm [2, 16, 18, 19], despite the sequential nature of the algorithm. Even though the algorithm is message-passing by nature (Chapter 4), it tends to be more complex than the algorithms for the competing LDPC code families. A main advantage of polar codes is their theoretical approachability; that is, their performance theory is concise and neat.

Nonetheless, as argued throughout this book, iterative, belief-propagation-based decoding of sparse graphical codes has virtually completely replaced all other coding methods and has literally invaded the communications standards. Communications over wireless local-area networks (WiFi), WiMAX, digital video broadcasting, twisted-pair cable modems (IEEE 802.3an), networking over power and phone lines, and coaxial cable all use LDPC coding. Clearly, the inability of the theoreticians to prove that these codes are formally capacity-achieving has not dampened the enthusiasm with which they were received by the technical community. This last “obstacle”, if we can call it such, has now also found a solution in the idea of spatial coupling. The work in [11, 12] has demonstrated that spatially coupled ensembles of such sparse codes have an improved decoding threshold which approaches that of maximum a posteriori decoding (MAP) of the uncoupled ensemble. This effect has been called *threshold saturation*, and it essentially states that if we wish to decode, say an LDPC code, to its MAP threshold, we need to spatially couple it with other copies of that code and then apply essentially the same message passing decoding algorithm.

The idea of spatial coupling has quickly spread to other applications such as source coding and multiple access communications. The main process is analogous in all these cases, however, and therefore we can restrict ourselves to the case of coupling graph-based error control codes. The results in Section 10.4, however, are general in nature. For a lucid introduction to spatially coupled coding see [3]. While it has taken researchers a while to clarify the mechanisms of spatial coupling, its effects have been known for a while. Arguably the seminal starting point of this method was the construction of terminated convolutional low-density parity-check codes (LDPCCs) by Feldström and Zigangirov [7], which has itself given rise to a series of developments [6, 23] on variations of convolutional code structures. We will start our discussion of spatial coupling with the exposition of LDPCCs in Section 10.2, largely following the original development in [7], as reinterpreted in [3]. And while LDPCCs have their own rightful place in the pantheon of error control codes, one of their main contribution is to usher in the concept of spatial coupling, which we will then discuss in more generality in Section 10.3. Spatial coupling will be developed from the concept of the protograph already encountered in Chapter 6.

While the concept of threshold saturation, i.e., the effect that the convergence threshold of a coupled ensemble under message passing approaches the MAP threshold of the individual member, is a key concept, and also observed for other applications, such as joint decoding [20], its mathematical treatment can be complex and messy. We therefore concentrate on a more general convergence analysis of spatial coupling, which takes its intuition from the field of statistical physics [22]. We introduce the concept of Lyapunov stability, which allows us to obtain a theoretically compact and well-founded viewpoint. This approach is then used to illustrate how we can show that coupled ensembles can reach the capacity of the transmission channel, without having to first show threshold saturation. This will be done in Section 10.4.

10.2 Low-Density Parity-Check Convolutional Codes

10.2.1 New LDPC Codes from Old

The low-density parity-check codes discussed in Chapter 6 are linear block codes of large size, and their sparse graphical structure allowed them to be decoded effectively via message passing. Feldström and Zigangirov [7] argued that codes with similar properties should also exist if one takes a convolutional code point of view. Convolutional codes, of course, have no predefined size and can be operated in a continuous manner. Similarly, such infinite sparse structures can be constructed by “unwrapping” a block LDPC code in the following manner.

We take the parity-check matrix of the (3,6) LDPC from Figure 6.2 and cut diagonally as shown in Figure 10.1. This produces a lower part, which we call \mathbf{H}_0 , and an upper part \mathbf{H}_1 .

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

Figure 10.1: Unwrapping of Gallager-type low-density parity-check matrix \mathbf{H} .

These diagonal partitions are then pasted together as two triangular matrices \mathbf{H}_0 on top of \mathbf{H}_1 , and this structure is repeated diagonally as shown in Figure 10.2. The resulting parity-check matrix has infinite extensions in both directions and is given by \mathbf{H}_{conv} shown in the figure. Since the original matrix \mathbf{H} is sparse, the parity-check matrix for the convolutional LDPC code is also sparse and can therefore, in principle, be decoded in exactly the same manner as a conventional LDPC code. We will see in Section 10.2.2 that a much more efficient pipelined decoder architecture can be designed.

$$\mathbf{H}_{\text{conv}} = \begin{bmatrix} \ddots & & & & & \\ & \ddots & & & & \\ & & \mathbf{H}_0 & & & \\ & & \mathbf{H}_1 & \mathbf{H}_0 & & \\ & & \mathbf{H}_0 & \mathbf{H}_1 & \mathbf{H}_0 & \\ & & \mathbf{H}_1 & \mathbf{H}_1 & \mathbf{H}_0 & \\ & & & \mathbf{H}_1 & \mathbf{H}_0 & \mathbf{H}_1 \\ & & & & \mathbf{H}_0 & \mathbf{H}_0 \\ & & & & & \ddots \end{bmatrix}$$

Figure 10.2: Construction of the parity-check matrix of a convolutional LDPC code.

This new code can also be interpreted as a convolutional code (Chapter 4). In fact, it is a *time-varying* convolutional code with a constraint length $\mu = N$, where N is the length of the originating LDPC code. Recall that the constraint length is the number of future output symbols (bits) which are affected by the current input symbol (bit).

The construction in Figure 10.2 is not unique. We could equally well have applied the cut and paste operation to sub-matrices of \mathbf{H} , or, equivalently, to an original code of different dimensions. This is illustrated in Figure 10.3 for an original code size of 6×12 , also using \mathbf{H} and generating the same \mathbf{H}_{conv} , but this time it is interpreted as a band-diagonal matrix of width 3, instead of 2. Its convolutional parity-check matrix is also shown in the figure, labeled differently, although it is the same code. It can be appreciated that there are a large number of ways of building such convolutional LDPC codes from given block LDPC codes, and Pusane et al. [15] have constructed extensive lists of codes and shown through simulations that the convolutional LDPC codes thus obtained exhibit a substantial performance gain over their block-oriented counterparts, an important observation in the context of the concept of spatial coupling discussed in the next section.

Of course, the code cannot be operated indefinitely in practice. Therefore, it has a starting point and generally also a termination point. This may happen after a possibly

large number L of sections, analogously to the operation of a convolutional code. Technically, \mathbf{H}_{conv} becomes a terminated band-diagonal matrix of size $L(N - K) \times LN$. It turns out, however, that this termination on one or both ends of the code has a curious side effect. Since the “local” rate, that is, the ratio of information bits to coded bits, is smaller around the termination, we observe a locally improved bit error rate. This effect is closely linked to the convergence effects of spatial coupling.

$$\mathbf{H}_{\text{conv}} = \begin{bmatrix} \ddots & \mathbf{H}_1 & & & \\ & \mathbf{H}_2 & \mathbf{H}_4 & & \\ & \mathbf{H}_3 & \mathbf{H}_5 & \mathbf{H}_1 & \\ & & \mathbf{H}_6 & \mathbf{H}_2 & \mathbf{H}_4 \\ & & & \mathbf{H}_3 & \mathbf{H}_5 & \mathbf{H}_1 \\ & & & & \mathbf{H}_6 & \mathbf{H}_2 & \mathbf{H}_4 \\ & & & & & \mathbf{H}_3 & \mathbf{H}_5 \\ & & & & & & \ddots \end{bmatrix}$$

Figure 10.3: Alternate construction of the parity-check matrix in Figure 10.2.

Constructing the Tanner graph of a convolutional LDPC code will reveal that the connections, while being “locally” random, reflect the band-diagonal structure of the parity-check matrix. It is therefore instructive to draw the graph in “sections” as shown in Figure 10.4 below, which shows the first three sections of \mathbf{H}_{conv} from Figure 10.2. It can clearly be seen that the first bits are checked by more low-weight check nodes than the rest of the code. This is equivalent to assuming that the symbols (bits) to the left of the starting point are “known,” and set to logic zero in this case. This gives these bits a higher degree of noise resistance.

Clearly, for this effect to be fully noticeable, the base code size has to be much larger than $N = 24$ as in the example. In fact, the sections of the convolutional LDPC are typically derived from a block LDPC code, and we have seen in Chapter 6 that sizes of $N \approx 10^3$ and larger are typical. This makes figures of the type of Figure 10.4 unwieldy and

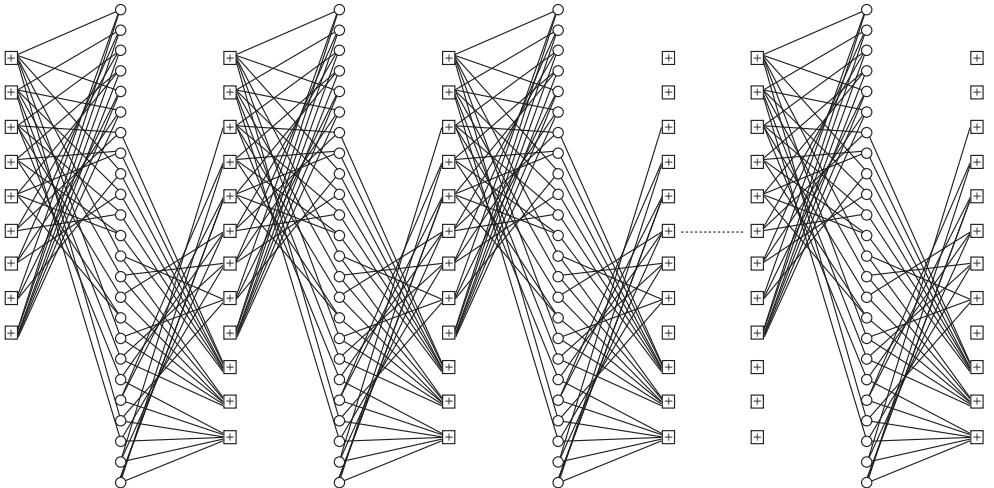


Figure 10.4: Tanner graph of the convolutional LDPC from Figure 10.2.

confusing. We therefore resort back to the representation of codes via their protographs, as discussed in Section 6.5.1.

This protograph representation of the code from Figure 10.4 is shown in Figure 10.5. We recall that each edge of the protograph represents $N/2$ edges, connecting $N/2$ variable nodes with $N/2$ check nodes, each represented in the protograph by a single variable node and check node, respectively. As discussed in Chapter 6 these $N/2$ pairs of nodes are connected by a permuted arrangement of the edges, where each protograph edge represents a different permutation. Note that the size of the graph copies represented by the protograph, here $N/2$, depends on the code and its representation.

While the representation in Figure 10.5 is compact and easy to conceptualize, we need to keep in mind that it is somewhat more general than the original convolutional LDPC code construction. In order to regenerate the original codes, we need to make sure that the outermost parity-check blocks in \mathbf{H}_{conv} have triangular structure, which imposes restrictions on some of the permutations associated with the edges of the protograph. However, from a theoretical point of view, there are little differences, and in the next section we will focus on the analysis of the structure in Figure 10.5.

Each section in Figure 10.4 represents an N -size section of the convolutional LDPC code, and we can now clearly identify that the nodes close to the endpoints of the structure have better protections. The outermost parity nodes have degree of only two, the next closer nodes have degree four, and the inner nodes have full regular degree of six.

Also note that there is a small rate loss that is associated with this termination. That

is, the code's design rate is computed from the base matrix dimensions $N - K \times N$ as

$$R = 1 - \frac{(L+1)(N-K)}{LN} \xrightarrow{L \rightarrow \infty} \frac{K}{N}, \quad (10.1)$$

and quickly approaches the design rate of the base code.

Density evolution studies to find the decoding threshold for convolutional LDPCs have been studied in [14] and summarized in [3] for various lengths L of the structure in Figure 10.2. This has been done for regular ($J/2J$) convolutional base codes. These thresholds are shown in Figure 10.6 below and show an interesting trend: As $L \rightarrow \infty$, regular codes with a sufficiently large degree profile, i.e., large J , achieve a threshold that coincides with the capacity limit of the binary-input AWGN channel at $E_b/N_0 = 0.19$ dB at rate $R = 0.5$. We will prove this capacity-achieving property of regular coupled LDPC codes in Section 10.4.

Note also that for small L , the rate loss experienced by the convolutional LDPC code is given by (10.1), and that it requires codes of length several tens of sections to make marked progress in the threshold and to overtake the threshold of the block version of the same code.

One may wonder if the effort to attain these gains is worth the increased complexity of the system. While it is true that the computational decoding effort per bit is unchanged with respect to that of the base block code, the entire code structure is now much larger. When before we would work with block sizes of 1,000–10,000, now these are the constraint lengths of the codes, and the actual codelengths will be on the order of 100,000 and longer. This has to be compared with the gain of a fraction of a dB in terms of performance. Such gains are also achievable with irregular block LDPC codes, although these require careful control of the error floor. Perhaps in the application for streaming data or largely varying block sizes which have to be changed on demand, these codes could be an appropriate engineering choice; see, e.g., [21]. However, the main excitement of convolutional LDPCs

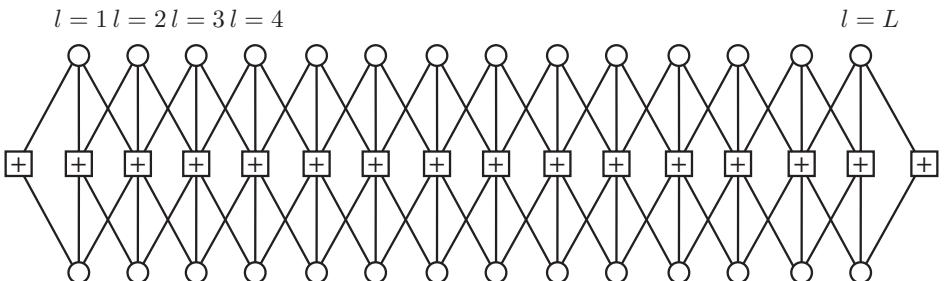


Figure 10.5: Protograph representation of the convolutional LDPC from Figure 10.2.

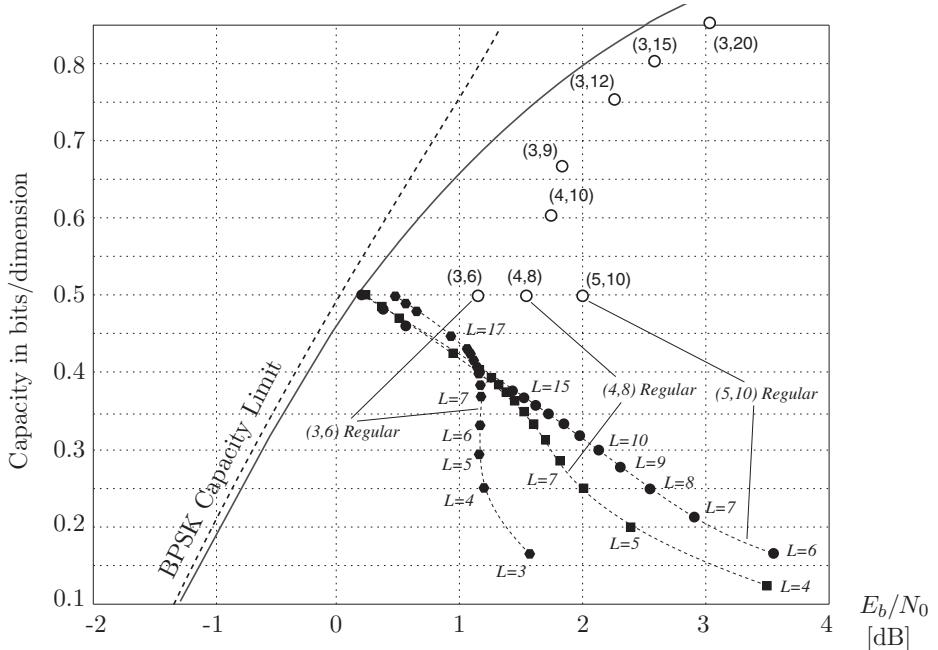


Figure 10.6: Decoding thresholds for convolutional LDPC codes for various lengths L and degree profiles under message-passing decoding. The open circles are the thresholds for the regular (J,2J) block LDPCs as discussed in Chapter 6.

is in their relationship to spatial coupling, which in turn allows us to finally prove the achievability of Shannon's capacity with a message-passing code ensemble.

10.2.2 Decoding Convolutional LDPC Codes

In principle, decoding of convolutional LDPC codes proceeds in a manner identical to that for their block oriented versions. That is, as discussed in Chapter 6, Page 257, check node update rules (6.6) and variable node update rules (6.7) alternate for a number I of iterations until convergence has occurred.

However, since \mathbf{H}_{conv} can be very large, a direct application of the update rules results in significant wastage of storage elements and excessive decoding delays. Given the band-diagonal nature of H_{conv} , implementing the iterations as successive operations that progress through the data set sequentially is much more efficient [7, 21] and can be realized with a number I of distinct and separate processor, each playing the role of an "iteration."

Due to the diagonal nature of the parity-check matrix, processors have to use overlapping data as illustrated in Figure 10.3.

The decoder operates with banks of first-in, first-out register banks of size $2N$, i.e., spanning two of the original block sizes. Each register bank has $d_v + 1$ registers, where the registers are attached to the “sockets” of the variable nodes. As a block of N symbols enters the decoder, the LLR values are computed according to (6.5) and shifted into the first N registers of the decoder. The second N registers are the LLR values from the previous received data block, which are shifted out of the first N positions. The first register bank will continue to contain these LLRs, while the other d_v register banks will hold the messages returned by the check nodes by the variable nodes.

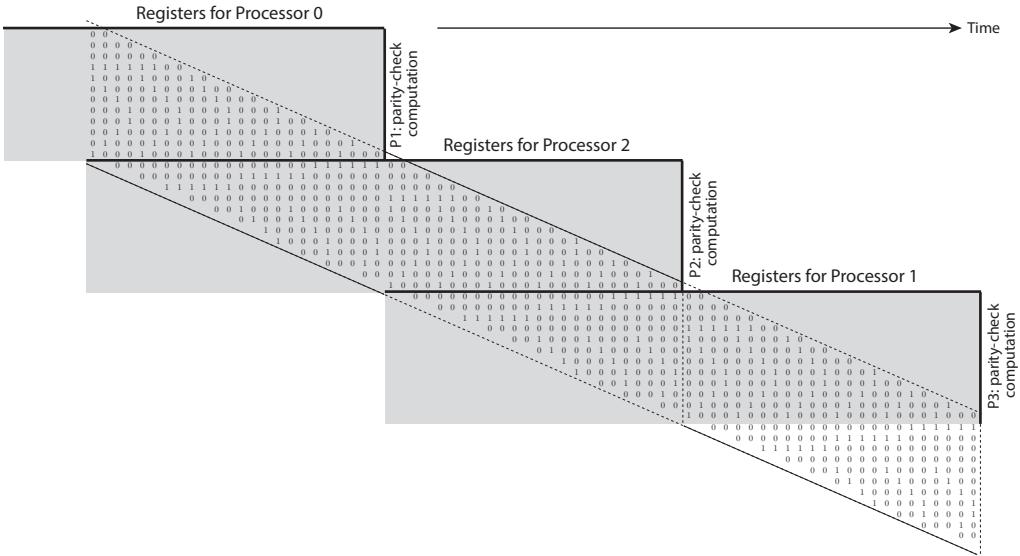


Figure 10.7: Illustration of the pipelined processing for the LDPCC decoder from Figure 10.2.

As in the standard LDPC decoding algorithms, the LLR values are passed to the check nodes which compute their respective output messages following (6.6). This is the “horizontal” pass of the processor. Viewing Processor 1 in Figure 10.7, we see that all messages for this operation are available in its register banks. The check nodes will now overwrite these values, and the registers then contain the return messages from the check nodes. Concurrently, Processors 2–I are all updating their registers with the check-to-variable

messages, using their own register banks which contain iterated messages as we shown below.

In the step the variable nodes update the messages using (6.7). However, it has only sufficient data to compute the second N variable-to-check node messages, which are updated in the second half of the register banks. At the same time, Processors $2-I$ are also updating their registers using in an identical manner. All registers are then shifted by N positions, where the second N values in each $2N$ register bank are shifted into the first N positions of the registers of the subsequent processor. In this manner the d_v register banks contain further updated check-to-variable node messages for each additional processor.

Note that a processor does not exactly correspond to an iteration, since half of the message used in the check node updates are from one iteration, while the other half is from a previous iteration. That is, Processor i uses variable-to-check node message corresponding to iteration i and iteration $i + 1$. Further limitations are the fact that the number of processors is not easily changed, especially during operation, and therefore dynamic buffering methods, such as [17], which are often used in the application of block LDPC codes to reduce the average number of iterations cannot easily be incorporated.

Nonetheless, a 180 nm implementation with 10 processors and $N = 128$ for a regular (3,6) code achieves a throughput of 175 Mbit/s [21] with a core size of 10 mm². Given the rapid progress of miniaturization, we extrapolate that a 22 nm implementation could be built with a footprint of 1 mm² and a speed of 1 Gbit/s.

While we have spent our effort in discussion the decoding and performance of convolutional LDPCs, we need to at least touch on the encoding operation. As we have seen in Chapter 6, direct encoding of block LDPC codes is not immediately obvious, and several simplifying approaches have been utilized. However, the direct and straight-forward encoding method of codes is appealing and accounts at least in part for the popularity of such codes as the repeat-accumulate codes, or generally protograph based codes as in [5]. Feldström and Zigangirov present a simple convolutional encoder with time-varying taps for their codes, which is possible given the specific code construction method they used.

Convolutional LDPC codes, or coupling codes can, of course, also be accomplished with other base codes, such as the protograph based codes studied and designed at the Jet Propulsion Laboratory (JPL) [5]. The advantages of coupling other than regular block LDPC codes is also pointed out in [10], who spatially couple repeat-accumulate codes, also achieving better thresholds for finite codelengths, as well as simpler encoding structures.

10.3 Spatial Coupling: A General View

The convolutional LPDC codes from Section 10.2 are just one special realization of the generic concept of spatial coupling. In fact, the representation in Figure 10.5 is more general, and we wish to use this general form for our further discussion. Figure 10.8

represents this concept again in a more generalized view. Each photograph, to the left, represents a complete LDPC block code of length N . That is, for a $(3,6)$ code, for example, the bold lines would represent the $3N$ connections between N variable nodes and $N/2$ check nodes.

What happens in the coupling process now is that a fixed portions of these connections are routed to the neighbor codes. The number of neighbors involved on each side of this bipartite graph is called the coupling window W . In our example $W = 5$. While coupling can, in general, involve arbitrary numbers of edges, we will concentrate on uniform edge spreading. That is, each line in the coupled graph represents $1/w$ of the edges of the original code. Edges are permuted randomly, observing however that the original node degrees of the uncoupled codes are preserved. Other than this, in general, no special structures like the one encountered in Section 10.2 are assumed. Note that the “dangling” check nodes on either side of the coupled structure implies that $1/5$ of the variable nodes in the first and last code are fixed logic “zero.” It is advantageous to think of the missing connections into the check nodes as “known” symbols. While not critical in a practical setup, this will help us in the analysis of the coupled system.

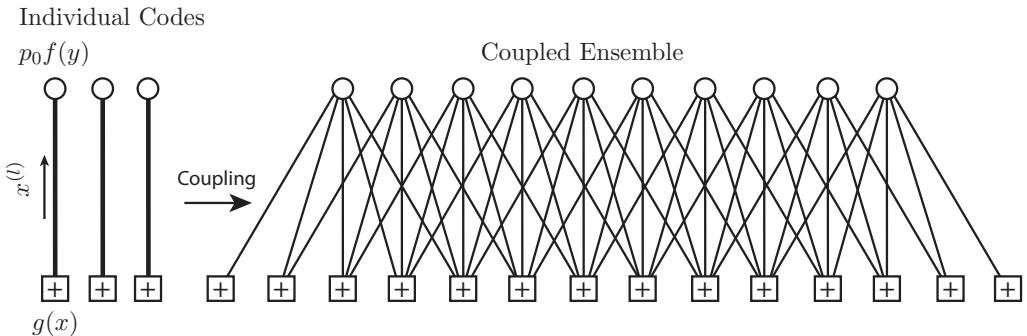


Figure 10.8: Protograph representation of a coupled set of LDPC codes with coupling window width $w = 5$; that is, each original code is coupled to the 4 nearest neighbors.

While there clearly exist many analogous ways of implementing the connectivity with neighbor codes, such as the method discussed in Section 10.2, we wish to discuss the dynamical behavior of such a coupled ensemble here, and not to dwell on potential implementation strategies and issues.

To start off the discussion, we recall the iteration equation (6.35) which describes the statistical behavior of a regular block LDPC code on an AWGN channel from Chapter 6,

reproduced here as

$$x^{(l)} = \phi\left(\frac{4E_s}{N_0} + (d_v - 1)\phi^{-1}\left(1 - \left(1 - x^{(l-1)}\right)\right)^{d_c-1}\right) \quad (6.35)$$

where the function $\phi(\cdot)$ is defined in (6.32) and plays the role of a bit error estimate, but only in the sense that if $x^{(l)} \rightarrow 0$, then the average BER of a large random code ensemble will also go to zero, and we say the code “converges.” Recall that the $x^{(l)}$ are related to the message traveling from the check nodes to the variable nodes, but this is not relevant to the analysis.

While the single code convergence formula (6.35) for the AWGN channel is perhaps the most relevant in practice, it is often easier to discuss basic concepts with a simpler channel model, and we therefore also consider the convergence equation for a block LDPC code on a binary erasure channel (BEC), given in (6.10), reproduced here as

$$x^{(l)} = f(x^{(l-1)}, p_0) = p_0\left(1 - \left[1 - x^{(l-1)}\right]^{d_c-1}\right)^{d_v-1}, \quad (6.10)$$

where $x^{(l)}$ here is the probability of bit erasure at iteration l in a (d_v, d_c) regular LDPC block code of infinitely large size $N \rightarrow \infty$.

If we abbreviate the statistical check node function in (6.10) by

$$g(x) = 1 - (1 - x)^{d_c-1}, \quad (10.2)$$

and abbreviate the statistical variable node function by

$$f(y, p_0) = p_0 y^{d_v-1}, \quad (10.3)$$

then dynamic behavior of a regular LDPC code (6.10) is succinctly summarized by the iteration equation

$$x = f(g(x), p_0) = p_0 f(g(x)). \quad (10.4)$$

The condition of convergence (to zero error) discussed in Chapter 6 is equivalent to the condition that (10.4) has only one fixed point at $x = 0$, and the supremum of p_0 such that only one solution to (10.4) exists is the initial erasure probability that the code can correct. In other words, convergence to $x^{(\infty)} = 0$ requires

$$x - \epsilon f(g(x)) > 0, \quad \forall p_0 \geq x > 0. \quad (10.5)$$

This function is illustrated for a number of erasure probabilities p_0 in Figure 10.9, and from numerical experimentation we find that the largest erasure probability that can be corrected by the code is $p_0 = 0.4294$.

In the uncoupled case, each LPDC code will converge with a dynamic behavior governed by the convergence equation (10.4), which involves only a single parameter. Once we couple the system, we now have to deal with individual convergence behaviors for each individual system, and, in general, we need to consider a vector convergence equation of the form

$$\mathbf{x} = \mathbf{F}(\mathbf{x}, p_0) \quad (10.6)$$

where $\mathbf{x} = [x_1, \dots, x_L]$ is the vector of the statistical variable describing all the $L + 1$ coupled dynamical systems.

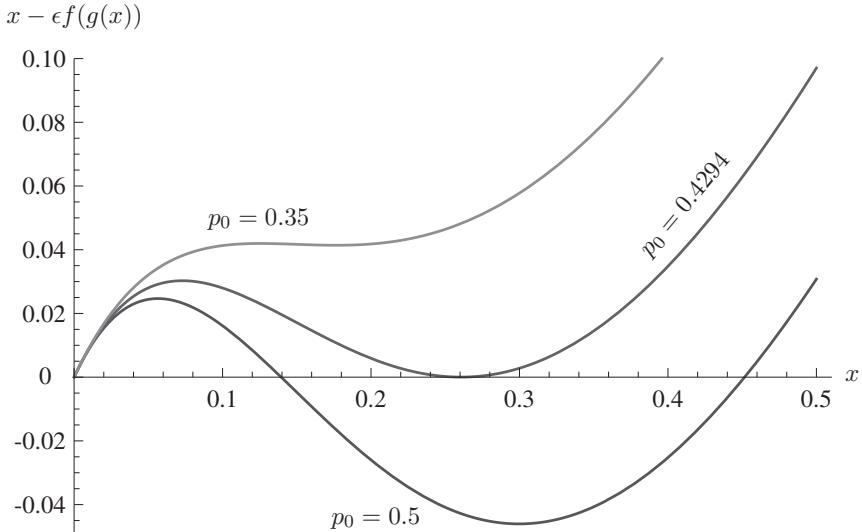


Figure 10.9: Statistical convergence functions for the (3,6) LDPC block code ensemble.

There are, as mentioned, a large number of ways to couple systems, but here we concentrate on coupling using a uniform window of size w , in which case the variables are updated as

$$x_i^{(l+1)} = \frac{1}{w} \sum_{k=0}^{w-1} f \left(\frac{1}{w} \sum_{j=0}^{w-1} g \left(x_{i+j-k}^{(l)}, p_0 \right) \right), \quad (10.7)$$

As we will see, the key impact of spatial coupling, as well as the complication in the analysis, stems from the fact that the coupled systems have certain boundary conditions. Specifically, dynamic variables that lie outside the window, i.e., values $x_l, l \notin [0, L]$, refer

to component systems that do not exist. If we inspect Figure 10.8 more closely, we see that this effect can be modeled by assuming $x_l = 0, \forall l \notin [0, L]$; that is, the corresponding variables are known. On the side we note that this can also refer to symbols that are a priori known, such a pilot signal, frame markers, preambles, and other fixed signals we may wish to couple into the code structure.

With this model, Equation (10.7) is now completely general, and, together with the boundary condition on the elements x_l not included in \mathbf{x} , defines a given spatially coupled code ensemble and describes its averaged dynamic performance.

For short-hand we define the vector functions

$$\mathbf{g}(\mathbf{x}, \varepsilon) \text{ and } \mathbf{f}(\mathbf{x}), \quad (10.8)$$

which defines the operation where the functions $f(\cdot)$ and $g(\cdot)$ are applied component-wise to all entries of the vector \mathbf{x} . We also define the matrix \mathbf{A} as the band-diagonal matrix with entries $1/w$ for all $j \geq i$ and $j - i \leq w$, and zeros elsewhere. With this, (10.7) can succinctly be written as the vector update equation

$$\mathbf{x}^{(l+1)} = \mathbf{A}^T \mathbf{f} \left(\mathbf{A} \mathbf{g} \left(\mathbf{x}^{(l)}, \varepsilon \right) \right) = \mathbf{F}(\mathbf{x}^{(l)}; \varepsilon). \quad (10.9)$$

Note that if $w = 0$, \mathbf{A} is the identity matrix and (10.9) reduces to the case of individual block LDPC codes discussed in Chapter 6.

In this context, one is typically interested in the largest ε such that for any possible initial vector $\mathbf{x}^{(0)}$, $\lim_{l \rightarrow \infty} \mathbf{x}^{(l)} = \mathbf{0}$. This parameter is typically a signal-to-noise ratio, channel raw error rate, or a channel erasure rate as we have seen before.

Introduce the set \mathcal{X}^0 as any set such that $\mathbf{x} \in \mathcal{X}^0$. Since the entries of \mathbf{x} are strictly non-negative—since they represent an error rate, or an absolute error—the following conditions are satisfied:

$$\mathbf{F}(\mathbf{x}; \varepsilon) \leq \mathbf{x} \text{ and } \mathbf{F}(\mathbf{x}; \varepsilon) \in \mathcal{X}^0. \quad (10.10)$$

Then for any l and $\mathbf{x}^{(l)} \in \mathcal{X}^0$ we have $\mathbf{x}^{(l+1)} \leq \mathbf{x}^{(l)}$ and $\mathbf{x}^{(l+1)} \in \mathcal{X}^0$, i.e., $\{\mathbf{x}^{(l)}\}$ is a monotonically non-increasing sequence. If $\mathbf{x}^{(0)} \in \mathcal{X}^0$ then $\mathbf{x}^{(l)} \in \mathcal{X}^0$ for any $l \geq 0$.

If, in particular, the initial $\mathbf{x}^{(0)}$ is such that $\mathbf{x}^{(1)} = \mathbf{F}(\mathbf{x}^{(0)}; \varepsilon) < \mathbf{x}^{(0)}$, then

$$\mathbf{x}^{(2)} = \mathbf{F}(\mathbf{x}^{(1)}; \varepsilon) < \mathbf{F}(\mathbf{x}^{(0)}; \varepsilon) = \mathbf{x}^{(1)},$$

and therefore $\mathbf{x}^{(0)} > \mathbf{x}^{(1)} > \mathbf{x}^{(2)} > \dots$, i.e., the sequence $\{\mathbf{x}^{(i)}, i = 0, 1, 2, \dots\}$ is strictly monotonically decreasing and it converges to a limiting $\mathbf{x}^{(\infty)} \geq \mathbf{0}$.

The limiting $\mathbf{x}^{(\infty)}$ should be a *stable fixed point*. If $\mathbf{x}^{(\infty)}$ is an *unstable fixed point*, then the system passes through that $\mathbf{x}^{(\infty)}$ to another fixed point.

The system (10.9) shows some very interesting convergence behavior. As discussed above, if $w = 0$, then all the component systems converge according to the scalar convergence equations (6.35) and (6.10) for the two example cases of the AWGN channel and

the BEC channel discussed here. However, if $w \neq 0$, the boundary conditions, by acting in an “anchoring” manner, cause the dynamical variable x_l close to the edges to converge faster than the interior points. This “perturbation” at the boundaries now propagates to the interior of the system in a wavelike manner.

Figure 10.10 illustrates this dynamic process for an $L = 100$ coupled system with a window size of $w = 8$. The plots show the individual values of x_l at iteration number 10, 30, and 50. As can be guessed, convergence progresses inward from the two boundaries at a constant speed until all $x_l = 0$.

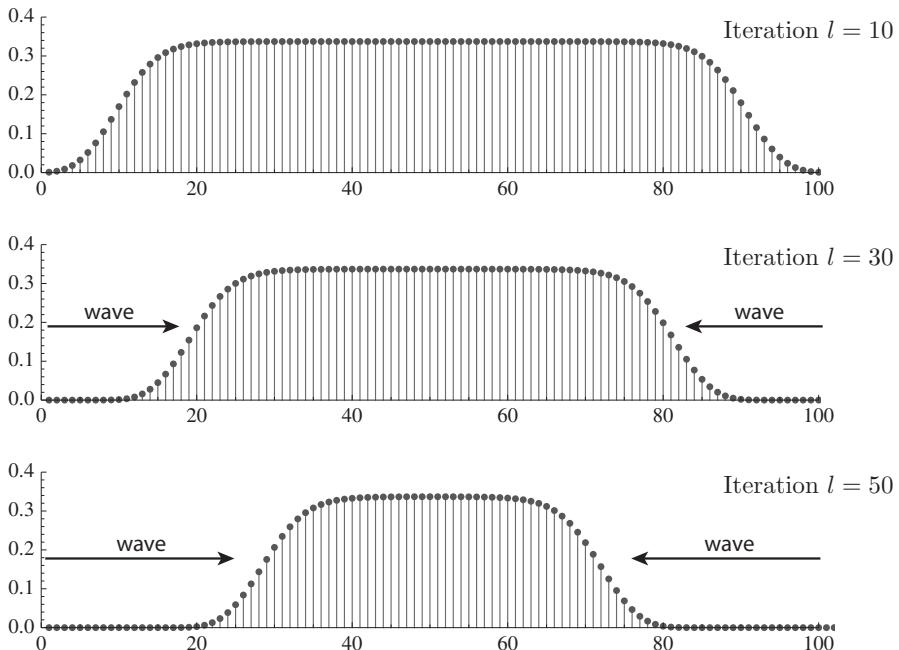


Figure 10.10: Illustration of the convergence for a coupled (3,6) LDPC code ensemble with a coupling window of $w = 8$.

Quite evidently, the boundary condition makes a significant difference in the convergence of the coupled system. Without the boundary, as occurs for example if the coupled chain is closed back on itself in a ring of coupled systems, convergence of the system would be no different than the convergence of a single LDPC code ensemble. The intricacies of

the effect of the boundary condition will be explored in Section 10.4.

A way to investigate the system (10.9) was pioneered in [22] and then developed in [25, 26]. It is based on using the following function $U(x) : x \rightarrow \mathbb{R}^1$, called the *potential function*¹:

$$U(x, \varepsilon) = \int_0^x g'(z, \varepsilon) [z - f(g(z, \varepsilon))] dz. \quad (10.11)$$

The motivation for using the function $U(\mathbf{x}, \varepsilon)$ in [22] was based on a continuous-time approximation for the system (10.9), given by

$$\frac{d\mathbf{x}(t)}{dt} = \mathbf{F}(\mathbf{x}(t); \varepsilon) - \mathbf{x}(t), \quad t > 0, \quad t \rightarrow \infty, \quad (10.12)$$

and, in turn, on the close relation of an analog of the function $U(x)$ to the Bethe free energy of the system. This will be further developed shortly in Section 10.4.

For now it suffices to state the main result from this development, which is the following

Theorem 10.1 *For large window sizes, i.e., $w \rightarrow \infty$, an iterative system of sparse graphs coupled as in (10.9) converges to $\mathbf{x}^{(l)} \rightarrow \mathbf{0}$, if $U(x) > 0$; that is, the coupled system threshold is given as*

$$\varepsilon_c^* = \sup\{\varepsilon : U(x, \varepsilon) > x, \forall x \leq \varepsilon\} \quad (10.13)$$

The mechanics of this theorem are illustrated in Figure 10.11, where $U(x, \varepsilon)$ is plotted for the coupled (3,6) LDPC code ensemble from Figure 10.9 for the BEC channel. As per theorem, a large coupled ensemble will converge for erasure rates up to $p_0 = 0.48815$, very close to capacity at $C_{\text{BEC}} = 0.5$.

It is not too difficult to argue that $\varepsilon_c^* \geq \varepsilon^*$ for any w , that is, coupling always improves performance. In fact, Theorem 10.1 allows us to prove the optimality of certain message-passing systems in coupled sparse graphs. An example is the following:

Theorem 10.2 *For large window sizes, i.e., $w \rightarrow \infty$, an iterative system of sparse coupled regular (d_v, d_c) LDPC codes achieves the capacity of the BEC channel, i.e.,*

$$\varepsilon_c^* \rightarrow \alpha \quad \text{for } \alpha = \frac{d_v}{d_c} \quad \text{and} \quad d_v \rightarrow \infty \quad (10.14)$$

Equivalently, the rate of the code $R = 1 - d_v/d_c$ approaches the channel capacity as in

$$R \rightarrow 1 - \varepsilon = C_{\text{BEC}} \quad \text{for } \varepsilon \rightarrow \varepsilon_c^*. \quad (10.15)$$

¹This terminology stems from the fact that the integral in (10.29)—discussed later in this chapter—does not depend on the curve \mathcal{C} from $\mathbf{0}$ to \mathbf{x} along which this integral is computed.

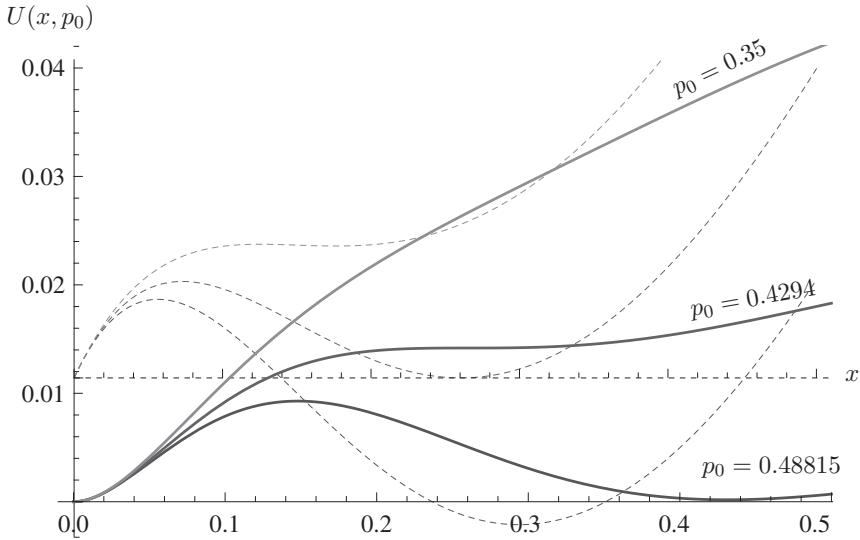


Figure 10.11: Potential functions for the coupled (3,6) LDPC code ensemble on the BEC channel, compared to the convergence functions for the uncoupled codes from Figure 10.9.

Proof: For the (d_v, d_c) -LDPC codes the function $U(x)$ from (10.11) takes the form

$$U(x, \varepsilon) = \frac{1}{d_c} - \frac{(1-x)^{d_c}}{d_c} - x(1-x)^{d_c-1} - \frac{\varepsilon}{d_v} \left[1 - (1-x)^{d_c-1} \right]^{d_v}, \quad (10.16)$$

with $f(x) = x^{d_v-1}$ and $g(x) = 1 - (1-x)^{d_c-1}$. We need to find the maximum $\varepsilon^* = \varepsilon^*(d_v, d_c)$ such that $U(x, \varepsilon^*) \geq 0$, $x \in [0, 1]$. Consider the case $d_v \rightarrow \infty$ and $d_v/d_c \rightarrow \alpha$, $\alpha < 1$, as in the conditions of the theorem. We will show that $\varepsilon^*(d_v, d_c) = \alpha$. Set $\varepsilon = \alpha$ and $d_v = d_c\alpha$. Then

$$\begin{aligned} d_c U(x, \alpha) &= 1 - (1-x)^{d_c} - d_c x(1-x)^{d_c-1} - \left[1 - (1-x)^{d_c-1} \right]^{d_c\alpha}, \\ U'_x(x, \alpha) &= (d_c - 1)(1-x)^{d_c-2} \left\{ x - \alpha \left[1 - (1-x)^{d_c-1} \right]^{d_c\alpha-1} \right\}. \end{aligned}$$

Since $U(1, \alpha) = 0$ and $U'_x(x, \alpha) > 0$, $x \geq \alpha$, it makes sense to consider only $x < \alpha$.

We may also exclude small x from further consideration as follows. We have

$$x - \alpha \left[1 - (1-x)^{d_c-1} \right]^{d_c\alpha-1} \geq x - \alpha[(d_c - 1)x]^{d_c\alpha-1} \geq 0,$$

$$x \leq x_0 = \frac{1}{(d_c - 1)} \left[\frac{1}{\alpha(d_c - 1)} \right]^{1/(d_c\alpha-2)},$$

where $x_0 \geq 1/(2d_c)$. Therefore $U'_x(x, \alpha) \leq 0$, if $x \leq 1/(2d_c)$. Since $U(0, \alpha) = 0$, all that remains is to consider the case $x = b/d_c$, $1/2 < b < \alpha d_c$. Since $(1-x)^{d_c} \leq e^{-d_c x}$ and $(1-x)^{d_c} \geq e^{-d_c x/(1-x)}$, we have for $1/2 < b < \alpha d_c$

$$d_c U(b/d_c, \alpha) \geq 1 - e^{-b} - b e^{-b} - \left[1 - e^{-b/(1-\alpha)}\right]^{d_c \alpha} \geq o(1), \quad d_c \rightarrow \infty.$$

Therefore

$$\inf_{0 \leq x \leq 1} U(x, \alpha) = o(1/d_c), \quad d_c \rightarrow \infty,$$

which proves the theorem. Q.E.D

We note that the proof of Theorem 10.2 was quite elementary, benefiting from the more general results of Theorem 10.1. This is in some contrast to the machinery invoked in [12], which proceed in a different way. First, it is shown that the threshold of a coupled LDPC ensemble has an improved convergence threshold ε_c^* and that this threshold approaches the convergence threshold of maximum-likelihood decoding of the underlying block LDPC code ensemble. This is called *threshold saturation*. It then remains to show under what conditions the original code ensemble can achieve capacity under maximum likelihood decoding. Here, we can show directly that the regular code ensemble when coupled achieves capacity.

The phenomenon of threshold saturation and—therefore, capacity-achieving performance if the underlying code ensemble permits this—appears to be quite general [11] and has also been shown for other communications scenarios, such as the isotropic multiple access channel used with uniformly random signal sets [20]. Coupling clearly is a powerful methodology and represents the missing link that has plagued coding theory since its inception in the 1940’s. Finally, the theoretician can rest assured that message-passing error control code decoding is theoretically optimal, both in performance and in order complexity of decoding. The practitioner of coding has long been satisfied with the powerful capabilities of message passing decoding, as evidenced by the numerous standards that involve LDPC codes or turbo codes. If and what impact spatial coupling will have on the application of new coding methods remains an open point at this time, but given that the basic block versions of these codes already approach capacity to within a fraction of a dB in many cases, we speculate that practical uses of spatial coupling may be reserved for very special cases.

10.4 Spatial Coupling: Convergence Analysis

10.4.1 Problem Setup

In this section we present a convergence analysis based on the well-established Lyapunov theory [13]; for a modern version see [8]. To recap the convergence problem from the

preceding section, we work with a set $\mathcal{X}^0 = \mathcal{X}^0(\varepsilon) \subseteq \mathcal{X}$ such that for $\mathbf{x} \in \mathcal{X}^0$, the following conditions are satisfied

$$\mathbf{F}(\mathbf{x}; \varepsilon) \leq \mathbf{x} \text{ and } \mathbf{F}(\mathbf{x}; \varepsilon) \in \mathcal{X}^0. \quad (10.17)$$

That is, by virtue of our iteration problem, we are only considering the convergence case where \mathbf{x} is non-increasing. Note that, as can be appreciated from Figure 10.9 for example, if we start with a suitably small x and a large value of p_0 , the iterations may actually diverge to a stable point. This behavior is not of interest insofar that it does not represent any reasonable application scenario.

Now then, for any l and $\mathbf{x}^{(l)} \in \mathcal{X}^0$ we have $\mathbf{x}^{(l+1)} \leq \mathbf{x}^{(l)}$ and $\mathbf{x}^{(l+1)} \in \mathcal{X}^0$, i.e., $\{\mathbf{x}^{(l)}\}$ is a monotonically non-increasing sequence, and with $\mathbf{x}^{(0)} \in \mathcal{X}^0$ all $\mathbf{x}^{(l)} \in \mathcal{X}^0$ for any $l \geq 0$.

If, in particular, we let the initial point be $\mathbf{x}^{(0)}$, and $\mathbf{x}^{(1)} = \mathbf{F}(\mathbf{x}^{(0)}; \varepsilon) < \mathbf{x}^{(0)}$, then

$$\mathbf{x}^{(2)} = \mathbf{F}(\mathbf{x}^{(1)}; \varepsilon) < \mathbf{F}(\mathbf{x}^{(0)}; \varepsilon) = \mathbf{x}^{(1)},$$

and therefore $\mathbf{x}^{(0)} > \mathbf{x}^{(1)} > \mathbf{x}^{(2)} > \dots$, i.e., the sequence $\{\mathbf{x}^{(i)}, i = 0, 1, 2, \dots\}$ is strictly monotonically decreasing and it converges to the limiting $\mathbf{x}^{(\infty)}(\mathbf{x}^{(0)}, \varepsilon) \geq \mathbf{0}$. That is, unless we happen to start at a fixed point, the convergence function (10.17) can be shown to be a strict inequality. For simplicity, we now consider the case $\mathbf{x}^{(0)} = \mathbf{1}$ for which those conditions are satisfied from above. One can always normalize the convergence equations such that this assumption is met. Of interest to us is the question: Under what conditions on $\mathbf{F}(\mathbf{x}; \varepsilon)$ do we have $\mathbf{x}^{(\infty)}(\varepsilon) = \mathbf{0}$?

The limiting $\mathbf{x}^{(\infty)}$ should be a *stable fixed point*. If $\mathbf{x}^{(\infty)}$ is an *unstable fixed point*, then the system passes through that $\mathbf{x}^{(\infty)}$ to another fixed point.

10.4.2 Lyapunov Approach

Essentially, Lyapunov built a theory whereby the often exceedingly complicated study of the behavior of individual trajectories in dynamical systems is reduced to the study of the system properties in certain regions of interest. This is done by using *Lyapunov functions* in these regions. In other words, rather than studying the behavior of the entire trajectory of a complex dynamical system, we study its behavior in a certain neighborhood. Knowing this, we can deduce the behavior of all trajectories in this neighborhood.

Denote by \mathcal{X}^0 the smallest set containing $\mathbf{x}^{(l)}$ for all $l \geq 0$, and by \mathcal{X}^1 an open set such that $\mathcal{X}^0 \subseteq \mathcal{X}^1$, but $\mathbf{0} \notin \mathcal{X}^1$. Denote by $\mathcal{E}_0 = \mathcal{E} \setminus \{0\}$, and let $\mathcal{E}_2 \subseteq \mathcal{E}_0$ be a subset of \mathcal{E}_0 .

Definition 10.1 *The solution $\mathbf{x}^{(l)} \equiv \mathbf{0}$ to (10.9) is globally asymptotically stable if $\lim_{l \rightarrow \infty} \mathbf{x}^{(l)} = \mathbf{0}$ for all $\mathbf{x}^{(0)} \in \mathcal{X}^0$.*

This type of problem was approached by Lyapunov using a certain type of functions in the region \mathcal{X}^0 (or a region containing it). These functions are now known as Lyapunov functions, which are given by the following

Definition 10.2 A continuous function $V(\mathbf{x}) : \mathcal{X} \rightarrow \mathbb{R}^1$ is called a Lyapunov function for the system (10.9) with $\mathbf{x} \in \mathcal{X}^0$, $\varepsilon \in \mathcal{E}_2$, if it satisfies the following conditions:

$$V(\mathbf{0}) = 0; \quad (10.18)$$

$$V(\mathbf{x}) > 0, \quad \mathbf{x} \in \mathcal{X}^1; \quad (10.19)$$

$$V(\mathbf{f}(\mathbf{g}(\mathbf{x}); \varepsilon)) - V(\mathbf{x}) < 0, \quad \mathbf{x} \in \mathcal{X}^1, \quad \varepsilon \in \mathcal{E}_2. \quad (10.20)$$

The following result is an adaptation of Lyapunov's direct method (1892) [8, Theorem 13.2]. It gives sufficient conditions for global asymptotic stability of the system (10.9).

Theorem 10.3 Given that $V(\mathbf{x})$ is a Lyapunov function for the system (10.9) for $\varepsilon \in \mathcal{E}_2$, the solution $\mathbf{x}^{(l)} \equiv \mathbf{0}$ is globally asymptotically stable.

That is, in the case stipulated by Theorem 10.3, the coupled system will converge to the all-zero solution. The condition (10.19) is a bit stronger than the first of the assumptions in (10.10). It is important also that the condition (10.20) holds in an open neighborhood of a limiting point of the system (10.9).

Proof: Lypunov's Theorem 10.3 is actually quite easy to prove as follows. Suppose trajectory $\mathbf{x}^{(l)}$ does not converge to zero. Given that (10.20) is the difference of $V(\mathbf{x})$ as the process advances through iterations, $V(\mathbf{x}^{(l)})$ converges to some value, say, $\mathbf{x}^{(\infty)}$. Since, per assumption, $\mathbf{x}^{(l)}$ does not converge to $\mathbf{0}$, there exists some δ such that for all l , $\delta \leq V(\mathbf{x}^{(l)}) \leq V(\mathbf{x}^{(0)})$. Working only with continuous functions $V(\mathbf{x})$, the supremum of $\dot{V}(\mathbf{x}) = -a < 0$ is attained somewhere in \mathcal{X}^1 . Hence, for all l we have

$$\begin{aligned} V(\mathbf{x}^{(L)}) &= V(\mathbf{x}^{(0)}) + \sum_{l=1}^L (V(\mathbf{x}^{(l)}) - V(\mathbf{x}^{(l-1)})) \\ &\leq V(\mathbf{x}^{(0)}) - aL, \end{aligned} \quad (10.21)$$

which, for $L > \lceil V(\mathbf{x}^{(0)})/a \rceil$ implies $V(\mathbf{x}^{(L)}) < 0 < \delta$, and this is a contradiction. Consequently, every trajectory $\mathbf{x}^{(l)}$ converges to $\mathbf{0}$ and the theorem is proven. q.e.d.

The next question is how to find an appropriate Lyapunov function. We start by representing the system (10.9) in the form

$$\mathbf{x}^{(l)} - \mathbf{x}^{(l+1)} = \mathbf{q}(\mathbf{x}^{(l)}), \quad l = 0, 1, 2, \dots. \quad (10.22)$$

where $\mathbf{q}(\mathbf{x}) = \mathbf{x} - \mathbf{F}(\mathbf{x})$. In order to avoid dealing with the trajectory $\{\mathbf{x}^{(l)}\}$, we construct a special type of Lyapunov function for (10.22). For that purpose we first construct such a function for the following continuous-time analog of the system (10.22)

$$\frac{d\mathbf{x}(t)}{dt} = -\mathbf{q}(\mathbf{x}(t)), \quad t > 0, \quad t \rightarrow \infty, \quad (10.23)$$

where $\mathbf{q}(\mathbf{x}(t)) = \mathbf{x}(t) - \mathbf{F}(\mathbf{x}(t))$.

We apply the variable gradient method for constructing Lyapunov functions to the system (10.23) [8, Chapter 3.4]. It will be a Lyapunov function for the system (10.22) as well.

Let $V : \mathcal{X} \rightarrow \mathbb{R}^1$ be a continuously differentiable function and let

$$\mathbf{h}(\mathbf{x}) = \left(\frac{\partial V}{\partial \mathbf{x}} \right)^\top.$$

i.e., $\mathbf{h}(\mathbf{x})$ is the gradient of $V(\mathbf{x})$, defined as

$$\frac{\partial V}{\partial \mathbf{x}} = \left[\frac{\partial V}{\partial x_1}, \frac{\partial V}{\partial x_2}, \dots, \frac{\partial V}{\partial x_d} \right],$$

The derivative of $V(\mathbf{x})$ along the trajectories of (10.23) is now simply given by

$$\frac{dV(\mathbf{x})}{dt} = -\frac{\partial V}{\partial \mathbf{x}} \mathbf{q}(\mathbf{x}) = -\mathbf{h}^\top(\mathbf{x}) \mathbf{q}(\mathbf{x}). \quad (10.24)$$

Next, we need to assure that condition (10.19) is fulfilled, that is, that we construct $\mathbf{h}(\mathbf{x})$ such that $\mathbf{h}(\mathbf{x})$ is the gradient for a positive function. We also need to ensure condition (10.20), that is

$$\frac{dV(\mathbf{x})}{dt} = -\mathbf{h}^\top(\mathbf{x}) \mathbf{q}(\mathbf{x}) < 0, \quad \mathbf{x} \in \mathcal{X}, \quad \mathbf{x} \neq \mathbf{0}. \quad (10.25)$$

The function $V(\mathbf{x})$ can then be computed from the line integral

$$V(\mathbf{x}) = \int_0^{\mathbf{x}} \mathbf{h}^\top(s) ds. \quad (10.26)$$

Recall that the line integral of a gradient vector $\mathbf{h} : \mathbb{R}^d \rightarrow \mathbb{R}^d$ is path independent, and hence, integration in (10.26) can be taken along any path joining the origin to $\mathbf{x} \in \mathcal{X}$.

It is further known [8, Proposition 3.1] that $\mathbf{h}(\mathbf{x})$ is the gradient of a real-valued function $V : \mathbb{R}^d \rightarrow \mathbb{R}^1$ if and only if the Jacobian matrix $\partial \mathbf{h} / \partial \mathbf{x}$ is symmetric, i.e., iff

$$\frac{\partial h_i}{\partial x_j} = \frac{\partial h_j}{\partial x_i}, \quad i, j = 1, \dots, d. \quad (10.27)$$

Referring to (10.25), we look for $\mathbf{h}(\mathbf{x})$ of the form $\mathbf{h}(\mathbf{x}) = \mathbf{B}(\mathbf{x})\mathbf{q}(\mathbf{x})$, where $\mathbf{B}(\mathbf{x})$ is an $n \times n$ -positive-definite matrix. Then due to (10.25) we need ($\mathbf{x} \in \mathcal{X}^1, \mathbf{x} \neq \mathbf{0}$)

$$\mathbf{h}^\top(\mathbf{x})\mathbf{q}(\mathbf{x}) = \mathbf{q}^\top(\mathbf{x})\mathbf{B}^\top(\mathbf{x})\mathbf{q}(\mathbf{x}) > 0, \quad (10.28)$$

and from (10.26) we have

$$V_{\mathbf{B}}(\mathbf{x}) = \int_0^{\mathbf{x}} [\mathbf{B}(s)(s - \mathbf{f}(\mathbf{g}(s)))]^\top ds. \quad (10.29)$$

For any positive-definite $n \times n$ -matrix $\mathbf{B}(\mathbf{x})$ the function $V_{\mathbf{B}}(\mathbf{x})$ from (10.29) is a Lyapunov function for the system (10.23), if condition (10.28) is satisfied, and $V_{\mathbf{B}}(\mathbf{x}) > \mathbf{0}$. Then

$$\lim_{t \rightarrow \infty} \mathbf{x}(t) = \mathbf{0}. \quad (10.30)$$

We now show that the function $V_{\mathbf{B}}(\mathbf{x})$ from (10.29) is a Lyapunov function for the system (10.9) as well. Condition (10.28) takes the form

$$[\mathbf{x} - \mathbf{f}(\mathbf{g}(\mathbf{x}))]^\top \mathbf{B}^\top(\mathbf{x}) [\mathbf{x} - \mathbf{f}(\mathbf{g}(\mathbf{x}))] > 0, \quad (10.31)$$

for $\mathbf{x} \in \mathcal{X}^1, \mathbf{x} \neq \mathbf{0}$. Specifically, if we choose $\mathbf{B}(\mathbf{x}) = \mathbf{g}'(\mathbf{x})$, then

$$V_{\mathbf{B}}(\mathbf{x}) = U(\mathbf{x}), \quad (10.32)$$

which will lead us directly to the scalar potential function (10.11). As a result, we get:

For any positive-definite $d \times d$ -matrix $\mathbf{B}(\mathbf{x})$ the function $V_{\mathbf{B}}(\mathbf{x})$ from (10.29) is Lyapunov function for the system (10.9), if the condition (10.31) and

$$V_{\mathbf{B}}(\mathbf{x}) > 0, \quad \mathbf{x} \in \mathcal{X}^1, \quad \mathbf{x} \neq \mathbf{0} \quad (10.33)$$

are satisfied. In this case

$$\lim_{l \rightarrow \infty} \mathbf{x}^{(l)} = \mathbf{0}. \quad (10.34)$$

The key approach now is the following: Given a suitable Lyapunov function, we can obtain a region of the parameter space $\epsilon \in \mathcal{E}_2$ for which our coupled system converges. The actual convergence region may be larger than \mathcal{E}_2 and also may depend on $V(\mathbf{x})$. Furthermore, as it turns out, \mathcal{E}_2 critically depends on the initial conditions of the coupled system.

For example, it can quite easily be verified from (10.9) that if a coupled system is connected back from the end to the beginning in a manner analogous to the tail-biting constraints discussed in Chapter 4, then the point $\mathbf{x}_u = (\dots, x^\infty, x^\infty, x^\infty, \dots)$ is a stable point of the recursion (10.9), irrespective of the window size w , and x^∞ is the fixed point

of the uncoupled system. Consequently, (10.5) applies and coupling has *no effect* in this situation, that is, the system behavior is no different to that of the uncoupled system. The same is true if the coupled system extends indefinitely in both directions.

As alluded to above, the boundary condition of a coupled system has a profound impact on its convergence behavior. As in the vast majority of papers published on the subject, we study the two-sided boundary condition $x_i = 0, i < 0; i > L$, which can also be thought of as *anchoring* of the spatially coupled system. One-sided anchoring works as well, in which case there is simply only a forward convergence propagation taking place, rather than a bidirectional convergence as in Figure 10.10.

Formally we therefore introduce the following:

Definition 10.3 *The coupled-system threshold with $x_i = 0, i < 0; i > L$ is defined as*

$$\varepsilon_c^* = \sup \{ \varepsilon \in \mathcal{E}_2 | \mathbf{x}^\infty(\mathbf{1}; \varepsilon) = \mathbf{0} \}. \quad (10.35)$$

Evidently $\varepsilon_c^* \geq \varepsilon_s^*$ in general, with equality if $w = 0$, or, for example, if the L identical systems are arranged in a circle such that no boundary exists as discussed above.

We will also use:

Definition 10.4 *For a positive-definite matrix \mathbf{B} the coupled-system threshold $\varepsilon_c(\mathbf{B})$ is defined as*

$$\varepsilon_c(\mathbf{B}) = \sup \left\{ \varepsilon \in \mathcal{E}_2 | \min_{\mathbf{x} \in \mathcal{X}_0} V_{\mathbf{B}}(\mathbf{x}) \geq 0 \right\}. \quad (10.36)$$

For any positive-definite matrix \mathbf{B} we have

$$\varepsilon_c^*(\mathbf{B}) \leq \varepsilon_c^*, \quad (10.37)$$

which expresses the fact that the Lyapunov function provides only a sufficient condition for convergence, not a necessary condition. As we have calculated in Section 10.3, if our threshold $\varepsilon_c^*(\mathbf{B})$ coincides with the capacity limit of the channel, then evidently the capacity theorem provides the necessary condition, in which case the Lyapunov function chosen furnishes the largest convergence region \mathcal{E}_2 .

Since we have already identified our Lyapunov candidate function in (10.32), we now simply need to identify the region of validity \mathcal{E}_2 for this function. However, we run into the following difficulty with (10.20), which requires a more detailed look at that condition. Considering regions \mathcal{X}_0 that are large enough to be of interest for global convergence in applied dynamical systems, that is, those that include actual starting points of the decoders, we encounter fixed points $\mathbf{x}^\infty \neq \mathbf{0}$, i.e., points for which $\mathbf{x}^\infty - \mathbf{f}(\mathbf{g}(\mathbf{x}^\infty, \varepsilon)) = \mathbf{0}$.

For these points, condition (10.20) needs to be reexamined. Specifically, we must show that in a region \mathcal{X}_0 of interest, any such fixed point \mathbf{x}^∞ cannot be stable, and that the Lyapunov condition (10.20) can be relaxed to a non-strict inequality at these points. In practice, this means that any such \mathbf{x} will move away from $\mathbf{x} = \mathbf{f}(\mathbf{g}(\mathbf{x}, \varepsilon))$ due to any disturbance, and re-enter the region convergence. In the sequel we will demonstrate what these points are and how progress away from the non-stable fixed points. This is indeed the convergence mechanism that we observe in experiments.

For a vector $\mathbf{x} = (x_{-L}, \dots, x_i, x_{i+1}, \dots)$ introduce the shift operator \mathbf{S} such that $\mathbf{S}\mathbf{x} = (0, x_{-L}, \dots, x_{i-1}, x_i, \dots)$. We note that if \mathbf{x}^∞ is a fixed point, then $\mathbf{S}\mathbf{x}^\infty$ is also a fixed point. This is trivially true for \mathbf{x}_u defined above. The validity of this can be verified by realizing that the equations (10.9) are shift invariant.

This leaves us with fixed points \mathbf{x}^∞ for which $x_i = 0, i < 0; i > L$. To shorten the discussion, we assume now without any loss of generality that $L \rightarrow \infty$, that is, we eliminate the right border. Due to the monotonicity of $\mathbf{x}^{(l)}$ in l , we have $x_i^\infty < x_{i+1}^\infty$; that is, \mathbf{x}^∞ is an increasing sequence in l . An example of this situation is shown in Figure 10.10 by considering only the left-hand side.

Now if w is sufficiently large, any intermediate point $\mathbf{x}_\lambda^\infty = \lambda\mathbf{x}^\infty + (1 - \lambda)\mathbf{S}\mathbf{x}_\lambda^\infty$ is also a fixed point if \mathbf{x}^∞ is a fixed point, i.e., $\mathbf{x}_\lambda^\infty - \mathbf{f}(\mathbf{g}(\mathbf{x}_\lambda^\infty, \varepsilon)) = \mathbf{0}$. Therefore, there exists a continuous “ridge” $\mathbf{x}_\rho = \mathbf{x}^\infty + \rho\mathbf{S}\mathbf{x}^\infty$ of such unstable fixed points. It is along this ridge that the system converges to $\mathbf{x}^\infty = \mathbf{0}$ as we now show.

The Lyapunov conditions guarantee convergence of $\mathbf{x}^{(l)}$ towards $\mathbf{0}$, but the ridge \mathbf{x}_ρ “intercepts” $\mathbf{x}^{(l)}$, primarily since convergence along the ridge is slower. The argument relies on the following logic: We will show that the non-zero ridge points, which are fixed points of the convergence equation, are nevertheless unstable because $V_{\mathbf{B}}(\mathbf{x}_\rho)$ is a saddle point of $V_{\mathbf{B}}(\mathbf{x})$ and not a local minimum. Therefore no \mathbf{x}_ρ can be stable. Furthermore, convergence proceeds along the ridge as what can be interpreted as a convergence wave; see Figure 10.10.

Theorem 10.4 *There exists a function $w_0(f, g)$ such that for any positive-definite matrix \mathbf{B} , $w \geq w_0(f, g)$, $L \geq 2w + 1$, and $\varepsilon < \varepsilon_c^*(\mathbf{B})$; the only stable fixed point of the system (10.9) is $\mathbf{x}_0 = \mathbf{0}$.*

Proof. We have $\mathbf{x}^{(l+1)} < \mathbf{x}^{(l)}$ and $V_{\mathbf{B}}(\mathbf{x}^{(l+1)}) < V_{\mathbf{B}}(\mathbf{x}^{(l)})$ for all $l \geq 0$, and the sequence $\{\mathbf{x}^{(l)}\}$ converges to a fixed point $\mathbf{x}_0 \in \mathbf{x}_\rho$, which is a (local) extremum of the function $V_{\mathbf{B}}(\mathbf{x})$. Since $\mathbf{x}_0 - \varepsilon\mathbf{f}(\mathbf{g}(\mathbf{x}_0)) = \mathbf{0}$, we have $V'_{\mathbf{B}}(\mathbf{x}_0) = \mathbf{0}$ and therefore need to consider second derivatives (the Hessian). Using a Taylor expansion we write

$$\begin{aligned} V_{\mathbf{B}}(\mathbf{x}) &= V_{\mathbf{B}}(\mathbf{x}_0) \\ &+ \frac{1}{2}(\mathbf{x} - \mathbf{x}_0)^T V''_{\mathbf{B}}(\mathbf{x}_0)(\mathbf{x} - \mathbf{x}_0) + O(\|\mathbf{x} - \mathbf{x}_0\|^3), \end{aligned}$$

where

$$V''_{\mathbf{B}}(\mathbf{x}_0) = \mathbf{B}(\mathbf{x}_0) [\mathbf{I} - \mathbf{f}'(\mathbf{g}(\mathbf{x}_0, \varepsilon))].$$

In general, if the $L + 1 \times L + 1$ matrix $\mathbf{I} - \varepsilon \mathbf{f}'(\mathbf{g}(\mathbf{x}_0))$ is not positive definite—that is, if it has a negative eigenvalue—then \mathbf{x}_0 cannot be a local minimum of the function $V_{\mathbf{B}}(\mathbf{x})$, and \mathbf{x}_0 is a saddle point. For the specific system (10.9) we have

$$\mathbf{f}'(\mathbf{g}(\mathbf{x}_0)) = \mathbf{A}^T f'(\mathbf{A}g(\mathbf{x}, \varepsilon)) \mathbf{A}g'(\mathbf{x}, \varepsilon) = \mathbf{C}(\mathbf{x}_0)$$

Typically, if ε is sufficiently small then there exists only the zero fixed point $\mathbf{x}_0 = \mathbf{0}$, that is, condition (10.19) holds as a strict inequality. This is somewhat trivially true for $\varepsilon < \varepsilon^*$; i.e., the parameter is less than the threshold of the uncoupled system.

As ε grows, it reaches some $\varepsilon_1 \geq \varepsilon^* > 0$ where non-zero fixed point(s) $\mathbf{x}_0 \neq \mathbf{0}$ appear. Since the first derivative of $V_{\mathbf{B}}(\mathbf{x})$ is zero only in the direction of the ridge \mathbf{x}_ρ , and strictly negative everywhere else as per (10.20), we need to examine the sign of the second derivative only in the direction of \mathbf{x}_ρ , that is, letting $\mathbf{z}_0 = \mathbf{x}_0 - \mathbf{S}\mathbf{x}_0$, we study for $\varepsilon > \varepsilon_1$ when and if

$$\mathbf{z}_0^T V''_{\mathbf{B}}(\mathbf{x}_0) \mathbf{z}_0 \leq 0, \quad (10.38)$$

which is equivalent to the condition

$$\mathbf{z}_0^T \mathbf{C}(\mathbf{x}_0) \mathbf{z}_0 > 1. \quad (10.39)$$

The matrix \mathbf{C} is non-negative, i.e., all its elements are non-negative. Therefore its spectral radius $\rho(\mathbf{C})$ equals its maximal eigenvalue. Moreover, if \mathbf{C} has a positive eigenvector, then the corresponding eigenvalue is $\rho(\mathbf{C})$ [9, Chapter 8].

If $w = 1$, then the fixed point $\mathbf{x}_0 = (x_0, \dots, x_0)$ and matrix \mathbf{C} is diagonal with equal diagonal elements and the scenario reduces to the uncoupled case discussed above. If $w > 1$, then the fixed point \mathbf{x}_0 consists of nondecreasing components. For the form (10.9) of the dynamical system, the i th row C_i of \mathbf{C} is given as

$$\begin{aligned} C_i &= \varepsilon a_i D_i, \quad c_i = f'_g(y_i) g'(y_i), \\ y_i &= \frac{1}{w^2} \sum_{k=0}^{w-1} \sum_{j=0}^{w-1} x_{0,i+j-k}, \quad D_i = (D_{i,0}, \dots, D_{i,n}), \\ D_{i,j} &= \frac{w - |i - j|}{w^2}, \quad |i - j| \leq w, \\ D_{i,j} &= 0, \quad |i - j| \geq w. \end{aligned} \quad (10.40)$$

Diagonal elements of \mathbf{C} are $\{\varepsilon c_i/w, i = 1, \dots, n\}$. For the matrix \mathbf{D} of rows $\{D_i\}$ and the matrix \mathbf{C} of rows $\{C_i\}$ we have

Lemma 10.5 *For any $w \geq 1$ and $L \geq 2w + 1$ the matrix \mathbf{D} has the maximal eigenvalue $\rho(\mathbf{D}) = 1$. Matrix \mathbf{C} has the maximal eigenvalue $\rho(\mathbf{C}) = \varepsilon \max_i c_i$.*

Therefore, if $\varepsilon > 1 / \max_i c_i$, then $\rho(\mathbf{C}) > 1$, and $\mathbf{I}_n - \varepsilon \mathbf{f}'(\mathbf{g}(\mathbf{x}_0))$ has a negative eigenvalue as required to render \mathbf{x}_0 unstable.

Remember that we still have to verify constraint (10.33), i.e., $V_{\mathbf{B}}(\mathbf{x}) > 0$, $\mathbf{x} \neq \mathbf{0}$, which sets the upper bound for ε . For ε satisfying both constraints, the only fixed point of the system (10.9) is $\mathbf{x}_0 = \mathbf{0}$.

It remains to clarify the condition $\varepsilon > 1 / \max_i c_i$. We limit ourselves here to the following result.

Lemma 10.6 *There exists a function $w_0(f, g)$ such that for any $w \geq w_0(f, g)$, $L \geq 2w + 1$, and $\varepsilon > \varepsilon_{\mathbf{s}}^*$ the matrix $\mathbf{I}_n - \varepsilon \mathbf{f}'(\mathbf{g}(\mathbf{x}_0))$ has a negative eigenvalue.*

From Lemma 10.6, constraint (10.33), and Definition 10.3, Theorem 10.4 follows. \square

We have presented this proof methodology based on Lyapunov systems theory since we believe it represents a general methodology to treat coupled systems based on well-known and accepted dynamic systems theory. We note that this approach presented here in general form, but pioneered in [22, 25, 26], reduces the proof of convergence theorem to showing the conditions for which (10.1) holds. An example of this application to the case of iterative demodulation on Gaussian multiple access channels is discussed in [24].

Bibliography

- [1] E. Arikan, “Channel polarization: A method for constructing capacity-achieving codes for symmetric binary-input memoryless channels,” *IEEE Trans. Inform. Theory*, pp. 879–883, June 2010.
- [2] A. Balatsoukas-Stimming, A.J. Raymond, W.J. Gross, and A. Burg, “Hardware architecture for list successive cancellation decoding of polar codes,” *IEEE Trans. Circuits Systems II*, vol. 61, no. 8, pp. 609–613, Aug. 2014.
- [3] D.J. Costello, Jr., L. Dolecek, T.E. Fuja, J. Kliewer, D.G.M. Mitchell, and R. Smarandache, “Spatially coupled sparse codes on graphs: Theory and practice,” *IEEE Commun. Mag.*, pp. 168–176, July 2014.
- [4] D.J. Costello, Jr. and G.D. Forney, “Channel coding: The road to channel capacity,” *Proceedings of the IEEE*, vol. 95, no. 6, pp. 1150–1177, June 2007.
- [5] D. Divsalar, S. Dolinar, C.R. Jones, and K. Andrews, “Capacity-approaching protograph codes,” *IEEE JSAC*, vol. 27, no. 6, pp. 876–888, Aug. 2009.
- [6] K. Engdahl and K.S. Zigangirov, “On the theory of low density convolutional codes I,” *Probl. Peredachi Inf.*, vol. 35, no. 4, pp. 295–310, 1999.
- [7] A.J. Felström and K.S. Zigangirov, “Time-varying periodic convolutional codes with low-density parity-check matrix,” *IEEE Trans. Inform. Theory*, vol. 45, no. 5, pp. 2181–2190, Sept. 1999.
- [8] W.M. Haddad and V.S. Chellaboina, *Nonlinear Dynamical Systems and Control: A Lyapunov-Based Approach*. Princeton University Press, Princeton, NJ, 2008.
- [9] R.A. Horn and C.R. Johnson, *Matrix Analysis*. Cambridge University Press, New York, 1985.
- [10] S. Johnson and G. Lechner, “Spatially coupled repeat-accumulate codes,” *IEEE Comm. Letters*, vol. 17, no. 2, pp. 373–376, Feb. 2013.
- [11] S. Kudekar, T. Richardson, and R. Urbanke, “Spatially coupled ensembles universally achieve capacity under belief propagation,” *IEEE Trans. Inform. Theory*, vol. 59, no. 12, pp. 7761–7813, Dec. 2013.

- [12] S. Kudekar, T. Richardson, and R. Urbanke, “Threshold saturation via spatial coupling: Why convolutional LDPC ensembles perform so well over the BEC,” *IEEE Trans. Inform. Theory*, vol. 57, no. 2, pp. 803–834, Feb. 2011.
- [13] A.M. Lyapunov, *The General Problem of the Stability of Motion*, Kharkov, Mathematical Society, Kharkov, Russia, 1892.
- [14] M. Lentmaier, A. Sridharan, D. Costello, Jr., and K. Zigangirov, “Iterative decoding threshold analysis for LDPC convolutional codes,” *IEEE Trans. Inform. Theory*, vol. 56, no. 10, pp. 5274–5289, Oct. 2010.
- [15] A.E. Pusane, R. Smarandache, P.O. Vontobel, and D.J. Costello, Jr., “Deriving good LDPC convolutional codes from LDPC block codes,” *IEEE Trans. Inform. Theory*, vol. 57, no. 2, pp. 835–57, Feb. 2011.
- [16] A.J. Raymond and W.J. Gross, “A scalable successive-cancellation decoder for polar codes,” *IEEE Trans. Signal Proc.*, vol. no. 20, pp. 5339–5347, 2014.
- [17] M. Rovini and A. Martinez, “On the addition of an input buffer to an iterative decoder for LDPC Codes,” *Proc. IEEE 65th VTC2007-Spring*, pp. 1995–1999, 2007.
- [18] G. Sarkis and W.J. Gross, “Implementation of Polar Decoders,” *Advanced Hardware Design for Error Correcting Codes*, P. Coussy and C. Chavet, Springer, New York, 2014.
- [19] G. Sarkis, P. Giard, A. Vardy, C. Thibeault, and W.J. Gross, “Fast polar decoders: Algorithm and implementation,” *IEEE J. Selected Areas Commun.*, vol. 32, no. 5, pp. 946–957, May 2014.
- [20] C. Schlegel and D. Truhachev, “Multiple access eemodulation in the lifted signal graph with spatial coupling,” *IEEE Trans. Inform. Theory*, Vol. 59, No. 4, pp. 2459–2470, 2013.
- [21] R. Swamy, S. Bates, T.L. Brandon, B.R. Cockburn, D.G. Elliott, J.C. Koob, and Z. Chen, “Design and test of a 175-Mb/s, rate-1/2 (128,3,6) low-density parity-check convolutional code encoder and decoder,” *IEEE J. Solid-State Circuits*, vol. 42, no. 10, pp. 2245–2256, Oct. 2007.
- [22] K. Takeuchi, T. Tanaka, and T. Kawabata, “A phenomenological study on threshold improvement via spatial coupling,” Arxiv preprint *arXiv:1102.3056*, 2011.
- [23] R.M. Tanner, D. Sridhara, A. Sridhara, T.E. Fuja, and D.J. Costello, Jr., “LDPC block and convolutional codes based on circulant matrices,” *IEEE Trans. Inf. Theory*, vol. 50, no. 12, pp. 2966–2984, Dec. 2004.
- [24] D. Truhachev and C. Schlegel, “Coupling data transmission for capacity-achieving multiple-access communications,” *arXiv:1209.5785*.
- [25] A. Yedla, Y-Y. Jian, P.S. Nguyen, and H.D. Pfister, “A Simple Proof of Threshold Saturation for Coupled Scalar Recursions,” Arxiv preprint *arXiv:1204.5703*, 2012.
- [26] A. Yedla, Y-Y. Jian, P.S. Nguyen, and H.D. Pfister, “A Simple Proof of Threshold Saturation for Coupled Vector Recursions,” Arxiv preprint *arXiv:1208.4080v2*, 2012.

Index

- M*-algorithm, 179, 186
- 10GBase-T Ethernet, 250
- 128-DSQ constellation, 103
- 16-QAM, 76, 79
- 32-cross, 76, 79
- 32-point rotated cross signal, 97
- 3G, 440
- 3G standards, 440
- 3GPP, 440
- 3GPP2, 440
- 4-dimensional trellis, 89
- 6-cycle, 287
- 64-QAM, 76, 79
- 64-bit IEEE 754 double-precision, 329
- absorbing set, 320, 321
- absorption
 - dynamic set, 323
- abstract
 - state, 145
- accumulate-repeat-accumulate code, 284
- ACE, 334
- ACS, 199
- adaptive
 - equalization, 99
- add–compare–select, 199
- additive noise, 28
- additive white Gaussian noise channel
 - binary-input, 249
- advanced hardware architectures, Inc., Pullman, WA, 450
- Aji, 299
- algebra
 - basic, 139
- algorithm
 - M*, 186, 239
 - T*, 188
 - a posteriori probability, 179
 - breadth-first, 183, 239
 - constant-log-APP, 210
 - Euclid, 111
 - exactness sum-product, 308
 - Fano, 185
 - log-App, 207
 - log-max-APP, 209
 - stack, 183
 - Viterbi, 179, 198, 240
- analysis
 - density evolution, 250
- anchoring, 479, 487
- Anderson, 188
- antenna
 - multiple channels, 48
- APP, 179, 373
- APP algorithm, 298
- APP decoding
 - approximate, 448
- approximate
 - APP decoding, 448
- Approximate Cycle EMD, 334
- approximate cycle extrinsic message degree, 332
- approximate triangularization, 296
- ARA, 284

- Arikan, 168, 455
- ARQ, 6
- asymptotic coding, 83
- Aulin, 190
- autocorrelation function, 41
- AWGN, 232
- AWGN channel, 269, 276, 476
 - decoding LDPC, 257
- backward recursion, 202
 - partial, 210
- Bahl, 113, 200
- bandwidth
 - efficiency, 12, 43
- Barnes-Wall, 90
- Barnes-Walls lattice, 159
- base code, 471
- BASE-T channel, 101
- baseband
 - complex equivalent model, 39
- basic
 - algebra, 139
 - encoder equivalence, 138
 - equivalent encoding matrix, 143
 - minimal encoder, 116, 137, 144
- basis
 - orthonormal, 29
 - vector, 80
 - waveforms, 29
- Bayesian network, 298, 299
- Bayes rule, 32
- BCJR, 298
- BEC, 168, 476
- BEC channels
 - decoding LDPC codes, 260
- belief, 304
- belief propagation, 298, 466
- Berlekamp-Massey, 111
- Berrou, 372, 439
 - turbo code, 445
- Bethe free energy, 480
- Bhattacharyya
 - bound, 169
- big Viterbi decoder, 351
- Biglieri, 222
- binary
 - partition chain, 88
- binary entropy, 14
- binary erasure channel, 168, 476
 - error probability evolution, 259
- binary superposition, 420
- binary symmetric channels, 266
- binary-input
 - additive white Gaussian noise channel, 249
- binary-input symmetric-output memoryless channels, 465
- bipartite
 - graph, 250
- bit-interleaved, 418
- bit-interleaved modulation, 420
- block code
 - BCH, 21
 - linear, 251
 - Reed-Muller, 20
 - Reed-Solomon, 10
- block turbo decoding, 446, 447
- Boltzmann's constant, 19
- bound
 - Bhattacharyya, 169
 - Chernoff, 226
 - random coding, 225
 - Shannon, 13
 - square root, 129
 - transfer-function, 220
 - union, 220
- BPSK capacity limit, 280
- branch metric, 198
- BSC channel, 267, 269
- butterfly activation, 457

- cable
 - copper, 53
 - twisted-pair copper, 101
- cabling
 - CAT 6 Ethernet, 55
- Calderbank, 90
- cancelation law, 140
- capacity, 7, 52, 465
 - channel, 231
 - Gaussian multiple access channel, 422
 - multi-dimensional, 49
- case study
 - communications system, 53
- CAT 6 Ethernet cabling, 55
- CCITT, 22
- CCITT V.32, 94
- CCSDS, 383, 439, 441
 - Telemetry Channel, 439
- CD, 111
- cdma2000, 3, 440
- chain
 - binary partition, 88
 - partition, 77
- channel, 28
 - band-limited, 7
 - BASE-T, 101
 - binary erasure, 168
 - capacity, 231
 - Gaussian, 7
 - satellite, 7
 - test, 409
- channels
 - binary symmetric, 266
 - capacity, 480
 - multiple antenna, 48
 - vector communication, 29
- chart
 - EXIT, 400
- Chase Algorithm, 449
- check node left degree, 290
- check nodes, 252
- checkerboard lattice, 81
- Chernoff bound, 226
- CMOS, 1
- Cocke, 113, 200
- code
 - accumulate-repeat, 284
 - accumulate-repeat-accumulate, 284
 - bit-interleaved, 418
 - construction, 71, 281
 - convolutional, 72, 115
 - coset, 85
 - differential space-time, 417
 - ensemble, 253
 - Extended Hamming, 165
 - generator matrix, 113
 - Goppa, 111
 - Hamming, 131
 - irregular LDPC, 250
 - linear, 112
 - linear-trellis, 236
 - low-density parity-check, 21, 249
 - maximum free-distance convolutional, 151
 - minimal trellis convolutional, 133
 - polar, 166
 - product, 451
 - quaternion, 415
 - Reed–Muller, 162
 - Reed–Solomon, 111
 - spatially coupled repeat-accumulate, 474
 - terminated convolutional, 366
 - time-varying convolutional, 468
 - turbo, 249, 351
- code design
 - error floors, 331
- codeword
 - low-weight, 358
- coding
 - asymptotic, 83
 - concatenated space-time, 414

- fundamental gain, 90
- communication
 - deep-space, 10
 - system case study, 53
- commutative
 - ring, 138, 140
- compact disc, 111
- complex
 - envelope, 41
 - equivalent baseband model, 39
- complexity
 - trellis, 120
 - Viterbi algorithm, 200
- concatenated RS-Trellis Code, 453
- concatenation
 - classic, 10
 - parallel, 11, 418
 - serial, 11
- concentration theorem, 276
- concept
 - ring-algebraic, 142
- constellation
 - 128-DSQ, 103
 - dithered square, 54
- constituent decoders, 403
- constituent encoders, 403
- construction
 - code, 281
 - PC-trellis systematic, 127
 - squaring, 154
 - twisted squaring, 157
- consultative committee for space data systems, 439
- contracted PC-trellis, 130
- convergence, 276
 - analysis, 482
 - equation, 272
 - function, irregular codes, 275
- convolution
 - codes, 115
- convolutional
 - interleaver, 437
 - LDPC, 470
 - maximum free-distance codes, 151
- convolutional code, 72
 - cascaded, 383
 - minimal trellis, 133
- convolutional LDPC code, 465, 471
- Conway, 80, 161
- copper
 - twisted-pair cable, 101
- copper cable, 53
- correlation
 - function, 62
 - receiver, 31
- coset code, 85
- Costello, 222, 433
- coupled dynamical system, 477
- coupled-system threshold, 480, 487
- coupling
 - spatial, 465, 466
- coupling window W , 475
- covering problem, 80
- CPM, 16
- CRC
 - code, 459
 - enabled decoder, 459
- criterion
 - stopping, 304
- cross
 - 32-point rotated signal, 97
 - 32, 76, 79
- crosstalk
 - far-end, 59
 - near-end, 59
- Crozier, 434
- cubic lattice, 81
- cut-set
 - lower bound, 132
- cutoff rate, 232

- computational, 234
- CWEF, 386
- cycle reduction, 286
- dangling checks, 323
- DE, 250
- decision
 - depth, 199
 - region, 341
- decoder
 - APP, 373, 395, 406
 - ML, 224
 - tree, 181
- decoding
 - a posteriori probability, 200
 - convolutional LDPC codes, 471
 - LDPC on AWGN channels, 257
 - maximum likelihood, 197
 - polar codes, 453
 - sequential, 21, 183
 - soft-decision, 112
- decoding LDPC codes
 - BEC channels, 260
 - decoding threshold, 261, 471
- decomposition
 - singular-value, 49
- deep-space, 10
- degree
 - node, 253
- degree distribution, 254
- delay-free generator, 134
- demodulator
 - product, 40
- dense packing, 81
- density
 - evolution, 471
 - evolution analysis, 250, 273
 - Gaussian function, 32
 - power spectral, 31
- design rate, 252
- DFT, 47
 - inverse, 47
- diagram
 - trellis, 68
- differential 8-PSK, 411
- differential modulator, 410
- differential-coded modulation, 409
- digital video broadcasting, 466
- dimension
 - signal, 12
- Diracs delta
 - function, 31
- discrete fourier transform, 47
- distance
 - Euclidean, 155
 - free, 74, 214
 - Hamming, 115
 - increment, 216
 - spectrum, 214, 356
 - squared Euclidean, 30
 - vector Euclidean, 241
- distance spectrum, 214
- distributive law
 - generalized, 299
- Dithered Relatively Prime, 434
- dithered square constellation, 54
- diversity, 420
- domain
 - ideal principal, 141
 - integral, 140
- dominant absorption set, 336
- doped repeat scramble, 399
- DRP, 434
- DRP interleaver, 436
- DRS, 399
- DSB-SC, 40
- DSL, 55
- DSQ, 54
- duo-binary turbo, 442
- DVB, 442

- return channel via satellite RCS, 442
- DVB-S2, 250, 383, 444
- dynamic
 - absorption set, 323
- effective free distance, 370
- efficiency
 - bandwidth, 12, 43
- eigenvalue, 221
 - largest eigenvector, 325
- eIRA, 331
- element
 - primitive, 287
- Elias, 446
- EMD, 332
- encoder, 79
 - catastrophic, 136
 - constituent, 354
 - feedback component, 369
 - feedback form, 390
 - generic trellis, 85
 - minimal, 134, 146
 - minimal basic, 116, 137, 144
 - minimal systematic, 116
 - non-systematic feedforward, 116, 118
 - non-systematic form, 116
 - primitive component, 370
 - recursive systematic feedback, 118
 - systematic, 149
 - systematic feedback, 118
- encoders
 - basic equivalence, 138
- encoding
 - equivalent basic matrix, 143
- enhanced SC decoder, 458
- ensemble
 - codes, 253
- envelope
 - complex, 41
- equalization
 - adaptive, 99
- equation
 - convergence, 272
- equivalence
 - basic encoders, 138
 - basic encoding matrix, 143
 - generator matrices, 134
- equivalent
 - complex baseband model, 39
- equivalent FSM, 241
- erasure decoding, 266
- error
 - pair-wise probability, 33
- error exponent, 232
- error floor, 319
- error floor region, 259
- error floors
 - code design, 331
- error probability evolution
 - binary erasure channels, 259
- Ethernet, 22
 - CAT 6 cabling, 55
- ETSI, 442
- Euclid's algorithm, 111
- Euclidean
 - space, 29
 - squared distance, 30
- Euclidean distance, 155
 - vector, 190
- European Telecommunications Standards Institute, 442
- exactness
 - sum-product algorithm, 308
- excision, 341
- EXIT, 353, 417
 - analysis, 376
 - analysis serial TTCM, 408
 - analysis, serially concatenated code, 397
 - chart, 379, 400, 412
- EXIT analysis, 279

- Extended Hamming code, 165
- extended irregular repeat accumulate codes, 331
- extrinsic
 - information, 374
 - information exchange, 353
 - information principle, 255
 - information transfer, 376
 - message degree, 332
- factor
 - invariant theorem, 141
 - roll-off, 43
- factor graph, 298, 303, 309
 - trellises, 305
- fading
 - quasi-static, 415
- Fano algorithm, 185
 - flowchart, 186
- far-end crosstalk, 59
- fast fading, 417
- FDM, 39
- FEC, 5
- feedback
 - recursive systematic encoder, 118
 - systematic encoder, 118
- feedforward
 - non-systematic encoder, 116, 118
- Feldström, 466
- FET, 2
- FEXT, 59
- finite-state machine, 67
- first event error probability, 213
- fixed point solution, 264
- folded spectrum, 37
- form
 - encoder non-systematic, 116
- forward recursion, 201
- forward-backward, 298
- fourier transform, 37
- frame error, 441
- frame error rate, 319
- free distance, 74
 - asymptote, 359
 - effective, 370, 391
 - maximum, 383
 - maximum convolutional codes, 151
 - turbo code, 358
- FSM, 67, 211
 - equivalent, 241
- function
 - autocorrelation, 41
 - conditional weight enumerating, 386
 - correlation, 62
 - Diracs delta, 31
 - Gaussian density, 32
 - input redundancy weight enumerating, 366
 - input-output weight enumerating, 386
 - input-output weight enumerating, 365
- function node, 299
- function-to-variable messages, 302
- fundamental coding gain, 90
- fundamental volume, 81
- future
 - subcode, 121
- Gallager, 226, 251, 321
 - exponent, 16
- Gallager algorithms, 266
- Galois fields, 287
- Gaussian
 - consistent distribution, 271, 326
- Gaussian density function, 32
- Gaussian multiple access channel
 - capacity, 422
- general edge, 309
- generalized distributive law, 299
- generalized modulation, 421
- generator

- delay-free, 134
- minimum-span matrix, 124
- generator matrix, 80
 - equivalence, 134
- generic trellis encoder, 85
- geometrically uniform, 215
- Gershgorin cycle theorem, 222
- Gilbert cell, 18
- girth, 322
- Glavieux, 372
- global asymptotic stability, 483
- Golay, 7
- Goppa code, 111
- Gosset, 90
- Gosset lattice, 158
- gradient, 485
- Grant, 415
- graph
 - bipartite, 250
- graph lifting, 282
- graphical function representation, 299
- Gray mapping, 75, 451
- greatest common divisor, 432
- group
 - Hamilton's quaternion, 415
- GSM, 3
- Guinard, 434
- Hölder's inequality, 230
- Hadamard transform, 170
- Hagenauer, 446
- Hamiltion, 415
- Hamming, 7, 112
 - distance, 420
- Hamming code, 131, 303, 340
 - extended, 294
 - trellis, 121
- Hamming distance, 115
- Hamming weight, 357
 - minimum, 123
- HDTV, 111
- high-definition television, 111
- high-order modulation, 451
- high-spread in the interleaver, 434
- historical notes, 106
- Holder's inequality, 238
- Hughes, 415
- ideal, 141
 - principal, 141
 - principal domain, 141
- IEEE
 - 802.11n, 250
 - 802.16, 452, 453
 - 802.16e, 250
 - 802.3an, 22, 53, 101, 250, 466
 - 802.3an LDPC, 320, 336
 - 803.2 Ethernet, 287
- implementation loss, 9
- importance sampling, 328, 336, 340
- IMT 2000, 3, 440
- indicator function, 226
- inequality
 - Hölder's, 230
 - Jensen's, 228
 - triangle, 240
- information
 - mutual, 167, 169
- inner BCH, 444
- integers modulo p , 140
- integral domain, 140
- interference
 - intersymbol, 53
- interference-free
 - intersymbol signaling, 36
- interleaver, 353, 431
 - 3GPP2, 442
 - block, 437
 - DRP, 436
 - linear, 442

- linearized, 432
- mother, 442
- prime, 442
- probabilistic uniform, 365
- quadratic, 433
- random, 363
- rectangular, 361
- S-random, 432
- spread, 433
- International Telecommunications Union, 440
- intersymbol
 - interference-free signaling, 36
 - intersymbol interference (ISI), 53
- invariance
 - rotational, 92, 93
- invariant
 - factor theorem, 141
- inverse
 - DFT, 47
 - polynomial right, 137
- invertible, 135
- IOWEF, 365, 386
- irregular
 - block LDPC codes, 471
 - LDPC code, 250, 254
- irregular code
 - convergence function, 275
- IRWEF, 366
- IS, 336
- iterative
 - matrix inversion, 221
- iterative decoding, 371
- SCCC, 394
- iterative system
 - sparse graphs, 480
- Jelinek, 113, 184, 188, 200
- Jensen's inequality, 228, 236
- Jet Propulsion Laboratory, 474
- Johannesson, 148
- JPL, 474
- kernel, 122
- kissing number, 81
- Kronecker power, 166
- Lagrange multiplier, 51
- Lang, 161
- lattice, 80
 - Barnes-Walls, 159
 - checkerboard, 81
 - cubic, 81
 - Gosset, 158
 - partition tree, 89
 - Schl  fi, 81
 - trellis, 154
- Laurent series, 117, 134
- law
 - cancelation, 140
- LDPC, 249
 - block code ensemble, 477
 - code graph, 314
 - convolutional, 470
 - decoding, 267
 - decoding convolutional codes, 471
 - decoding on AWGN channels, 257
 - encoding of codes, 291
 - irregular block codes, 471
 - irregular code, 250, 254
 - irregular codes, 269
 - irregular cycle optimized codes, 279
 - regular, 251
 - regular coupled codes, 471
 - repeat-accumulate, 444
 - reversible codes, 293
 - RS-based, 287
 - short-cycle-free codes, 286
 - specialized codes, 294
 - thresholds regular codes, 280
 - triangular codes, 292

- LDPCC, 466
- least mean-squares, 60
- Leech, 90
- lemma
 - matrix inversion, 244
- limit
 - Shannon, 10
- linear
 - block code, 251
 - code, 112
 - linear-trellis code, 236
 - linearized interleaver, 432
 - LLR, 208, 256, 376, 394
 - transformer, 376
 - LLR check node messages, 312
 - LLR clippings, 336
 - LLR update equation, 449
 - LLR variable node messages, 312
- LMS, 60
- locally optimal, 314
- log-likelihood ratio, 208, 256
- Longstaff, 161
- lookup table, 208
- loop
 - phase-locked, 92
- low-density parity-check code, 21, 249
- low-density parity-check convolutional codes, 467
- lower bound
 - cut-set, 132
- LTE, 3
- LTE-A, 3
- Lyapunov candidate function, 487
- Lyapunov functions, 482
- Lyapunov theory, 482
- m-sequence, 433
- machine
 - finite-state, 67
- MacKay, 249
- MAN, 453
- MAP, 32, 256, 466
- mapping
 - Gray, 75
 - natural, 74
- Markov chain, 299
- Massey, 113, 183
- matched filter, 33, 180
- matrix
 - code generator, 113
 - equivalent basic encoding, 143
 - generator, 80
 - generator equivalence, 134
 - minimum-span generator, 124
 - parity-check, 112, 251
- matrix inversion lemma, 244
- maximal
 - stopping set, 265
- maximum
 - a posteriori probability, 256
 - free-distance convolutional codes, 151
- maximum a posteriori, 32
- maximum a posteriori decoding, 466
- maximum likelihood decoding, 197
- maximum likelihood receiver, 32
- maximum-length shift, 433
- McEliece, 120, 299
- mean translation, 342
- mean-shift importance sampling, 343
- message
 - passing, 255
 - sequences, 35
- metric
 - branch, 198
 - partial, 181, 190
- MFD, 383
- MIMO, 50, 52
- MIMO systems, 416
- min-sum approximation, 258
- minimal

- basic encoder, 116, 137, 144
- encoder, 134, 146
- realization, 392
- systematic encoder, 116
- trellis, 120
- minimal trellis
 - convolutional codes, 133
- minimum
 - Hamming weight, 123
- minimum-span
 - generator matrix, 124
- mission
 - Galileo, 21
 - Mariner, 21
 - Pioneer, 21
 - Viking, 21
 - Voyager, 21
- ML receiver, 32
- model
 - complex equivalent baseband, 39
- modulation
 - binary superposition, 420
 - differential-coded, 409
 - generalized, 421
 - trellis-coded, 22, 67, 69
 - turbo-trellis coded, 402, 403
- modulo
 - p integers, 140
- Monte-Carlo simulations, 339
- Moore's law, 2
- MPSK, 92
- MSGM, 125
- multi-dimensional capacity, 49
- multiple antenna channels, 48
- multiplexing
 - orthogonal frequency-division, 46
- multiplicity, 214
- multiplier
 - Lagrange, 51
- mutual information, 167, 169
- natural
 - mapping, 74
- Neal, 249
- near-end crosstalk, 59
- NEXT, 59
- node
 - check, 252
 - degree, 253
 - variable, 252
- noise
 - additive, 28
 - one-sided, 31
 - Thermal, 28
 - white, 63
- non-systematic
 - encoder form, 116
 - feedforward encoder, 116, 118
- notes
 - historical, 106
- NP-complete, 311
- nullspace, 112
- Nyquist, 4, 12
 - pulse, 37
 - sampling theorem, 4
- Nyquist signals, 445
- OFDM, 46
- one-sided noise, 31
- optimum
 - receivers, 31
- orthogonal frequency-division multiplexing, 46
- orthogonality, 31, 36
- orthonormal basis, 29
- Osthoff, 193
- overbiasing, 346
- packing
 - dense, 81
- pair-wise error probability, 33, 214

- parity-check matrix, 473
- PAM, 101, 420
- parallel edges, 254
- parietal syndrome, 113
- parity-check
 - matrix, 112, 251
 - trellis, 113
- Parseval's relationships, 42
- partial
 - metric, 190
 - received sequence, 181
- partition
 - binary chain, 88
 - chain, 77
 - lattice tree, 89
 - set, 75
- past
 - subcode, 121
- PC-trellis
 - contracted, 130
 - systematic construction, 127
- PCM, 6
- Pearl, 298
- performance, 436
- permutation, 253
- permutation group, 431
- permutation matrices, 281
- permutter, 281
- Perron–Frobenious theorem, 221
- phase-locked loop, 92
- photograph, 475
- PLL, 92
- polar code performance, 458
- polar codes, 166, 465
- polarization, 169
- polynomial
 - right inverse, 137
- positive eigenvector, 489
- positive-definite matrix, 487
- potential function, 480
- power
 - Kronecker, 166
- power spectral density, 31
- precoding
 - Tomlinson–Harashima, 54, 58
- prime, 140
- primitive
 - element, 287
 - feedback polynomial, 391
- principal
 - ideal, 141
 - ideal domain, 141
- probabilistic uniform interleaver, 365
- probability
 - first event error, 213
 - matrix, 326
 - pair-wise error, 33, 214
 - propagation, 298
- problem
 - covering, 80
 - sphere packing, 80
- processor, 473
- product
 - code, 446, 451
 - demodulator, 40
- proto-graph, 281, 470, 471
- proto-matrix, 281
- PSTN, 98
- pulse
 - Nyquist, 37
 - spectral raised-cosine, 37
 - time-orthogonal, 180
- Pyndiah, 446
- QAM, 79, 420
 - 16, 76, 79
 - 64, 76, 79
- QPSK, 7, 29
- quadratic encoding complexity, 256
- quadratic interleaver, 433

- quadrature double side-band suppressed carrier, 40
- quadrature signal, 40
- quaternion code, 415
- RA, 289, 400, 445
- random coding
- analysis optimal decoding, 223
 - sequential decoding analysis, 233
- random interleaver, 363
- rank, 252
- rate
- symbol, 35
- ratio
- log-likelihood, 256
- Raviv, 113, 200
- receiver
- correlator, 31
 - maximum-likelihood, 32, 180
 - ML, 32
 - optimum, 31
- rectangular array, 431
- rectangular interleaver, 361
- recursive
- systematic feedback encoder, 118
- Reed–Muller code, 162, 453
- Reed–Solomon code, 111
- Reed-Muller, 465
- regions
- Voronoi, 32
- regular, 215
- coupled LDPC codes, 471
- regular LDPC, 251
- Reiffen, 183
- relationship
- Parseval, 42
- repeat-accumulate, 400
- code, 284, 289, 445
 - LDPC, 444
- return barrier, 193
- Richardson, 347
- ring
- algebraic concepts, 142
 - commutative, 138, 140
- Robertson, 402
- roll-off factor, 43
- ROM, 208
- root-Nyquist signaling, 39
- rotational invariance, 92, 93
- Rouanne, 222
- RS, 10
- RS-based
- LDPC, 287
- rule
- Bayes, 32
 - de l'Hôpital, 231
- S-random interleaver, 432
- SCCC, 383
- iterative decoding, 394
 - weight enumerator, 385
- schedule, 302
- Schläfli lattice, 81
- Schlegel, 327, 415
- scrambler, 138
- self-concatenated
- turbo encoder, 444
- sequences
- message, 35
- sequential decoding, 183, 234
- serial concatenation, 383, 406
- convolutional codes, 383
- serial TTCM
- EXIT analysis, 408
- series
- Laurent, 117, 134
- set
- span, 126
- set partition, 75
- Shannon, 7

- bound, 13
- capacity, 20
- limit, 249
- shaping, 85
- short cycles, 286
- short-cycle-free LDPC codes, 286
- signal
 - 32-point rotated cross, 97
 - dimension, 12
 - quadrature, 40
- signal-to-noise ratio, 7
- pinch-off, 381
- signaling
 - intersymbol interference-free, 36
 - root-Nyquist, 39
- singular values, 49
- singular-value decomposition, 49
- SISO, 200
- size
 - state, 133
- sliding window, 210
- Sloane, 80, 90, 161
- sockets, 253
- soft-decision decoding, 112
- space
 - Euclidean, 29
 - state, 121
- space data system standard, 439
- space-time coding, 414
- span
 - set, 126
- sparse, 251
- sparse graph
 - iterative system, 480
- sparse graphical codes, 465
- spatial coupling, 465, 466, 474, 482
- spectral raised-cosine pulse, 37
- spectrum
 - folded, 37
- sphere
- packing problem, 80
- spread interleaver, 433
- spreading factor, 370
- square
 - twisted construction, 157
- square root
 - bound, 129
- squares
 - least mean, 60
- squaring
 - construction, 154
- stable fixed point, 478, 482
- stack algorithm, 183
- standard, 439
 - CCSDS, 383
 - DVB-S2, 383
- state
 - abstract, 145
 - space, 121
- state size, 133
- statistics
 - sufficient, 34
- stopping
 - criterion, 304
- stopping set, 321
 - maximal, 265
- stopping set S, 265
- subcode
 - future, 121
 - past, 121
- sublattice, 83
- subsets, 77
- subthreshold, 18
- successive cancelation, 171
- successive cancelation algorithm, 458
- sufficient statistics, 34
- sum-product
 - exactness algorithm, 308
 - sumproduct algorithm, 302
- SVD, 49

- symbol
 - rate, 35
- syndrome
 - partial, 113
- system
 - communications case study, 53
- systematic
 - construction PC-trellis, 127
 - encoder, 149
 - feedback encoder, 118
 - minimal encoder, 116
 - recursive feedback encoder, 118
- tail-biting trellis, 129, 131
- Takeshita, 433
- Tanner, 298
- Tanner graph, 252, 322
- TCM, 22, 67
- television
 - high-definition, 111
- test channel, 409
- theorem
 - Gershgorin, 222
 - Gershgorin's cycle, 240
 - invariant factor, 141
 - non-optimality, 197
 - Perron–Frobenious, 221
 - waterfilling, 50
- Thermal noise, 28
- thin spectrum, 432
- third generation, 440
- THP, 54
- threshold, 259, 329
- threshold saturation, 466, 482
- thresholds
 - regular LDPC codes, 280
- time-orthogonal pulse, 180
- time-varying convolutional code, 468
- Tomlinson–Harashima precoding, 54, 58
- transfer-function bound, 220
- transform
 - Hadamard, 170
- transition matrix, 217
- trapping sets of LDPC, 347
- traveling wave tube, 16
- tree decoder, 181
- tree search
 - depth-first, 185
 - exhaustive, 197
- trellis, 115
 - 4-dimensional, 89
 - coded modulation, 69
 - complexity, 120
 - diagram, 68
 - factor graph, 305
 - generic encoder, 85
 - Hamming code, 121
 - lattice, 154
 - minimal, 120
 - minimal convolutional codes, 133
 - parity-check, 113
 - tail-biting, 129, 131
- trellis-based APP decoding, 448
- trellis-coded modulation, 67
- triangle inequality, 240
- truncation length, 199
- TTCM, 402
- turbo cliff, 381
- turbo code, 249
 - BER bound, 358
 - distance spectrum, 356
 - free distance, 358
 - graph, 314
 - iterative decoding, 371
 - original, 352
 - standards, 439
 - weight enumerator, 364
- turbo codes, 351
- turbo encoder
 - self-concatenated, 444

- turbo product code, 453
- turbo-trellis coded modulation, 402
- twisted squaring construction, 157
- twisted-pair cabale modems, 466
- twisted-pair copper cable, 101
- Ungerböck , 67
- uniformity, 222
- union bound, 214
- unstable fixed point, 478, 482
- UTI, 440
- V.32, 96
- V.33, 96
- V.34, 22
- V.90, 100
- V.fast, 98, 99
- variable gradient method, 485
- variable nodes, 252
- variable-to-function messages, 302
- variance scaling, 341
- vector
 - basis, 80
 - communication channels, 29
 - Euclidean distance, 190, 241
- Viterbi, 198, 226
 - algorithm, 179
 - big decoder, 351
- Viterbi algorithm, 198
 - complexity, 200
- voiceband modem, 22
- volume
 - fundamental, 81
- Voronoi, 80
- Voronoi regions, 32
- Wörz, 402
- Wan, 148
- waterfilling theorem, 50
- waveforms
 - basis, 29
- weight enumerator, 364
- white noise, 63
- Wiberg, 298
- WiFi, 466
- WiMAX, 250, 446, 466
- Wolf, 113, 222
- Wozencraft, 183
- Yang, 331
- Zehavi, 222
- zero-state, 212
- Zhang, 75, 327
- Zigangirov, 184, 466

IEEE PRESS SERIES ON DIGITAL AND MOBILE COMMUNICATION

John B. Anderson, Series Editor
University of Lund

1. *Wireless Video Communications: Second to Third Generation and Beyond*
Lajos Hanzo, Peter J. Cherriman, and Jurgen Streit
2. *Wireless Communications in the 21st Century*
Mansoor Sharif, Shigeaki Ogose, and Takeshi Hattori
3. *Introduction to WLLs: Application and Deployment for Fixed and Broadband Services*
Raj Pandya
4. *Trellis and Turbo Coding*
Christian B. Schlegel and Lance C. Perez
5. *Theory of Code Division Multiple Access Communication*
Kamil Sh. Zigangirov
6. *Digital Transmission Engineering, Second Edition*
John B. Anderson
7. *Wireless Broadband: Conflict and Convergence*
Vern Fotheringham and Shamla Chetan
8. *Wireless LAN Radios: System Definition to Transistor Design*
Arya Behzad
9. *Millimeter Wave Communication Systems*
Kao-Cheng Huang and Zhaocheng Wang
10. *Channel Equalization for Wireless Communications: From Concepts to Detailed Mathematics*
Gregory E. Bottomley
11. *Handbook of Position Location: Theory, Practice, and Advances*
Edited by Seyed (Reza) Zekavat and R. Michael Buehrer
12. *Digital Filters: Principle and Applications with MATLAB*
Fred J. Taylor
13. *Resource Allocation in Uplink OFDMA Wireless Systems: Optimal Solutions and Practical Implementations*
Elias E. Yaacoub and Zaher Dawy
14. *Non-Gaussian Statistical Communication Theory*
David Middleton
15. *Frequency Stabilization: Introduction and Applications*
Venceslav F. Kroupa
16. *Mobile Ad Hoc Networking: Cutting Edge Directions, Second Edition*
Stefano Basagni, Marco Conti, Silvia Giordano, and Ivan Stojmenovic
17. *Techniques for Surviving the Mobile Data Explosion*
Dinesh Chandra Verma and Paridhi Verma

18. *Cellular Communications: A Comprehensive and Practical Guide*
Nishith D. Tripathi and Jeffrey H. Reed
19. *Fundamentals of Convolutional Coding*, Second Edition
Rolf Johannesson and Kamil Sh. Zigangirov
20. *Trellis and Turbo Coding*, Second Edition
Christian B. Schlegel and Lance C. Perez

WILEY END USER LICENSE AGREEMENT

Go to www.wiley.com/go/eula to access Wiley's ebook EULA.