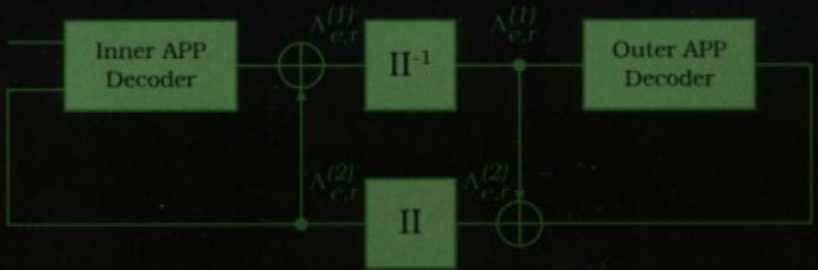
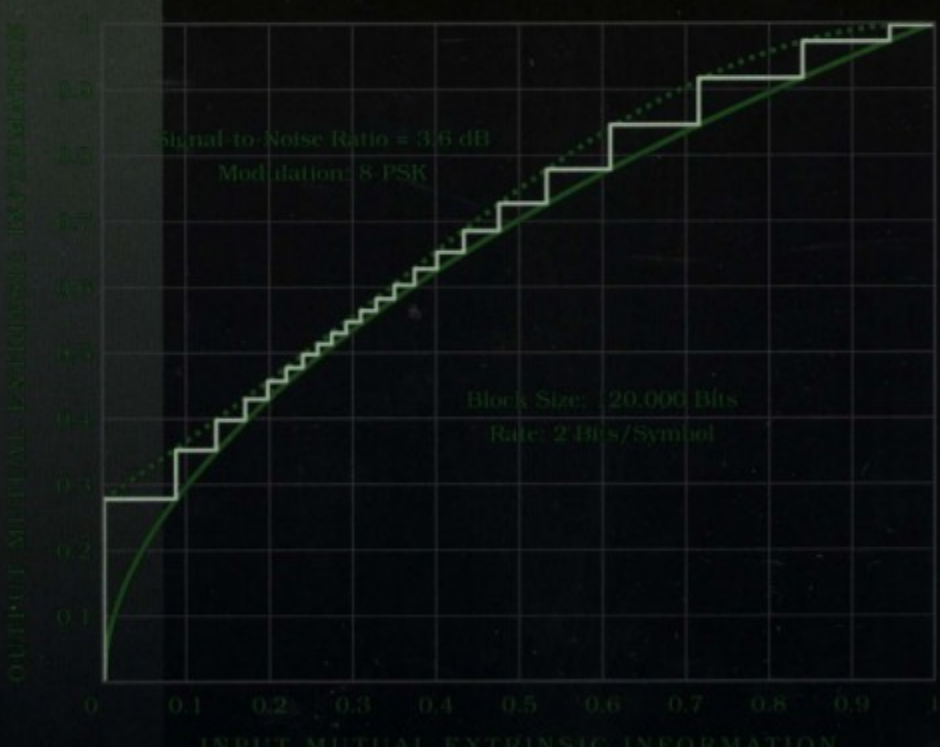
Trellis and Turbo Coding

CHRISTIAN B. SCHLEGEL
LANCE C. PÉREZ





IEEE SERIES ON DIGITAL & MOBILE COMMUNICATION

John B. Anderson, *Series Editor*

TRELLIS AND TURBO CODING

IEEE Press Series on Digital & Mobile Communication

The IEEE Press Digital & Mobile Communication Series is written for research and development engineers and graduate students in communication engineering. The burgeoning wireless and personal communication fields receive special emphasis. Books are of two types—graduate texts and the latest monographs about theory and practice.

John B. Anderson, *Series Editor*
Ericsson Professor of Digital Communication
Lund University, Sweden

Advisory Board

Books in the IEEE Press Series on Digital & Mobile Communication

John B. Anderson, *Digital Transmission Engineering*
Rolf Johannesson and Kamil Sh. Zigangirov, *Fundamentals of Convolutional Coding*
Raj Pandya, *Mobile and Personal Communication Systems and Services*
Lajos Hanzo, Peter J. Cherriman, and Jürgen Streit, *Wireless Video Communications: Second to Third Generation Systems and Beyond*
Lajos Hanzo, F. Clare A. Somerville and Jason P. Woodard, *Voice Compression and Communications: Principles and Applications for Fixed and Wireless Channels*
Mansoor Shafi, Shigeaki Ogose and Takeshi Hattori (Editors), *Wireless Communications in the 21st Century*
Raj Pandya, *Introduction to WLLs: Application and Deployment for Fixed and Broadband Services*

TRELLIS AND TURBO CODING

Christian B. Schlegel

Lance C. Pérez

John B. Anderson, *Series Editor*

IEEE PRESS

A JOHN WILEY & SONS, INC., PUBLICATION

IEEE Press
445 Hoes Lane
Piscataway, NJ 08855

Published simultaneously in Canada.

Library of Congress Cataloging-in-Publication Data

Schlegel, Christian.
Trellis and Turbo coding / Christian B. Schlegel, Lance C. Pérez.
p. cm.
Includes bibliographical references and index.
ISBN 0-471-22755-2 (cloth)
1. Error-correcting codes (Information theory) 2. Trellis-coded modulation. 3. Digital communications. 4. Coding theory. I. Perez, Lance. II. Title.

TK5102.96.S33 2004
621.382–dc22
2003063355

Printed in the United States of America.

10 9 8 7 6 5 4 3 2 1

To our children to whom belongs the new world:

Christina, Vincent, Isabelle,
Patrick, and Lindsey

CONTENTS

PREFACE

In 1997 I completed my book *Trellis Coding*, which contained a single chapter on turbo coding. Since then, turbo coding and its associated iterative decoding method, the turbo principle, has swept through the communications society like a wildfire and established itself as the error coding method of choice. Its ease of implementation and its phenomenal performance, which in many cases pushes right up to Shannon's theoretical limits, lets one speculate that turbo coding is *the* right way to encode and decode digital data. Any potentially new method would have to measure itself against turbo coding and thus have virtually no chance of providing further gains. In recent years the implementation of turbo codes has also made great strides, and practically any decoding speeds can now be achieved by suitable parallelization of the decoding algorithms, leaving data storage and memory access as the limiting functions. Storage, however, is always necessary according to the requirements of Shannon's theory to process large blocks of data in order to approach a channel's capacity limit.

The turbo principle has found application even outside the narrow field of error control coding, in multiple access channel communications, in signalling over channels with intersymbol or interchannel interference, and, more recently, in source coding. In all these applications the principle has established itself quickly as a very powerful processing method, leading to the design of receivers which are far superior to conventional methods.

Since the publication of *Trellis Coding*, it has become evident that turbo coding had to be given central treatment, and Lance Pérez and I have embarked on writing up the material on turbo coding to fit together with the already existing material on trellis coding. Some of the less central parts of the old book have been discarded, the remainder has been updated and prepared to match the new chapters on turbo coding. *Trellis and Turbo Coding* is the result and reflects the fact that these two methods closely operate together. Most turbo codes use small trellis codes as their constituent building blocks. What we have learned from the last decade of intensive research is that we only want to use trellis codes with small state-space complexities, and if stronger codes are needed, to wire up trellis codes into concatenated turbo systems.

The first part of the book follows largely the original material in *Trellis Coding*, enhanced with many updates. It not only covers independent trellis coding methods, such as trellis coded modulation and its use in voiceband modems, but also lays the foundation for the component decoding methodology for turbo coded systems. The second part is completely new and deals with all major forms

of turbo coding, viz. parallel and serial concatenated systems, low-density parity check coding, and iterative decoding of product codes. These methods and the material on trellis coding comprise the majority of channel coding strategies in use today, and likely for a long time to come.

Since turbo coding has achieved theoretical limits with manageable effort, any possible future advances are necessarily very limited, and we will undoubtedly see a migration of research and development from designing codes to building codecs. In that sense we believe that the topic of error control coding has reached a level of maturity and completion which will elevate it from a research domain to a classic theory. We hope that this book covers this theory in a comprehensive and accessible manner, and that the reader will experience the same awe and fascination that we felt researching the material for it.

ACKNOWLEDGMENTS

I am grateful to my co-author Lance Pérez for helping me conquer the vast material on turbo codes which has emerged during the last decade, to Sheryl Howard for co-authoring the chapter on factor graphs, and to Christopher Winstead for co-authoring the chapter on low-density parity check codes. I also wish to acknowledge my colleagues Robert Hang, Paul Goud, Sheryl Howard, and Christopher Winstead at the High-Capacity Digital Communications Laboratory at the University of Alberta for their careful reviewing of the chapters of this book, as well as Barbara Krzymien for her cover design drafts.

CHRISTIAN B. SCHLEGEL

Edmonton, Alberta, Canada
September, 2003

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. Real-world data is typically in analog form; this is the only way we can perceive it with our senses. This analog information needs to be encoded into a digital representation—for example, into a string of ones and zeros. The conversion from analog to digital and back are processes that have become ubiquitous, as, for example, in the digital encoding of speech.

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, even 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 called *error control coding*.

The digitization of data is convenient for a number of other reasons too. The design of signal processing algorithms for digital data seems much easier than designing analog signal processing algorithms. The abundance of such digital algorithms, including error control and correction techniques, combined with their ease of implementation in *very large-scale integrated* (VLSI) circuits, has led to many successful applications of error control coding in practice.

Error control coding was first applied in deep-space communications where we are confronted with low-power communications channels with virtually unlimited

Trellis and Turbo Coding, by Christian B. Schlegel and Lance C. Pérez
ISBN 0-471-22755-2

bandwidth. On these data links, convolutional codes (Chapter 4) are used with sequential and Viterbi decoding (Chapter 7), and the future will see the application of turbo coding (Chapter 10). The next successful application of error control coding was to storage devices, most notably the compact disk player, which employs powerful Reed–Solomon codes [19] 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 right up toward the theoretical limit of the channel. Nowadays coding is routinely applied to satellite communications [39, 46], teletext broadcasting, computer storage devices, logic circuits, semiconductor memory systems, magnetic recording systems, audio–video systems, and modern mobile communications systems like the pan-European TDMA digital telephony standard GSM [33], IS 95 [44], CDMA2000, and IMT2000, all new digital cellular standards using spread-spectrum techniques.

1.2 THE RISE OF DIGITAL COMMUNICATIONS

Aside from the technological advantages the digital processing has over analog, there are also very good theoretical reasons to limit attention to the processing of digital signals. Modern digital communication theory started in 1928 with Nyquist's seminal work on telegraph transmission theory [34]. 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] 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 that are used to represent a band-limited 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 were completely sufficient to characterize any band-limited signal in the presence of noise—that is, on any communications system.

[1] Since 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 band-limited to $[-1/2T, 1/2T]$:

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

With these results, technology has moved toward completely digital 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 COMMUNICATIONS SYSTEMS

Figure 1.1 shows the basic configuration of a point-to-point digital communications link. The data to be transmitted over this link can 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 [20]. Usually the digital data is source-encoded to remove unnecessary redundancy from it; that is, the source data is compressed [12]. This source encoding has the effect that the digital data which enter the encoder has statistics which resemble that of a random symbol source with maximum entropy; that is, 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

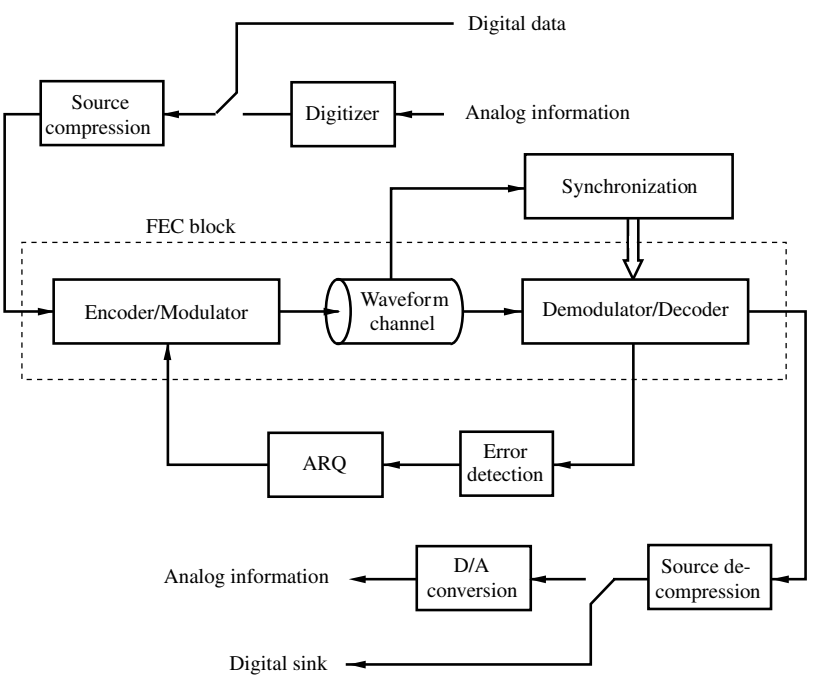


Figure 1.1 System diagram of a complete point-to-point communication system for digital data. The forward error control (FEC) block is the topic of this book.

channel. The modulator converts the discrete channel symbols into waveforms which are transmitted through the waveform channel. The demodulator reconverts the waveforms 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.

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 3 and 12). The books by Meyr and Ascheid [30] and by Meyr, Moeneclaey, and Fechtel [31] treat synchronization issues in detail. Since synchronization is a relatively slow estimation process, and data detection is a fast process, we usually have these two operations separated in real receiver implementations as indicated in Figure 1.1. However, we would like to note at this point that novel, iterative receiver designs are increasingly integrating these auxiliary functions (see e.g., Chapter 12).

Another important feature in some communication systems is *automatic repeat request* (ARQ). In ARQ the receiver also performs error detection and, through a return channel, requests retransmission of erroneous data blocks, or data blocks that cannot be reconstructed with sufficient confidence [23]. 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 the same 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 [22, 23] for data communications.

This book deals only with FEC, the dashed block in Figure 1.1. The reason why we can do this relatively easily is due to different functionalities of the various blocks just discussed. Each of them is mostly a separate entity with its own optimization strategy, and data are simply passed between the different blocks without much mutual interaction. A notable point in Figure 1.1 is that the Encoder/Modulator and the Demodulator/Decoder are combined operations. This is the basic novelty that gave birth to trellis-coded modulation. This joint encoder/modulator design was first proposed by Wozencraft [58] and Massey [28] and was then realized with stunning results by Ungerboeck [47–49].

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 goodness 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 clearly more detrimental than an error in the least significant bit. Researchers have also looked at schemes with unequal error protection (e.g., [16]). However, those methods usually are a variation of the basic theme of obtaining a minimum error rate.

1.4 ERROR CONTROL CODING

The modern approach to error control in digital communications started with the ground-breaking work of Shannon [42], Hamming [17], and Golay [14]. While Shannon put down a theory that 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 that is irretrievably lost; but by clever design, errors could be corrected, or avoided altogether. This new thinking was revolutionary. But 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 curious shortcoming of Shannon's theory. While it clearly states fundamental limits on communication efficiency, its methodology provides no insight on how to actually achieve these limits, since it is based largely 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 [25].

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

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

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

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

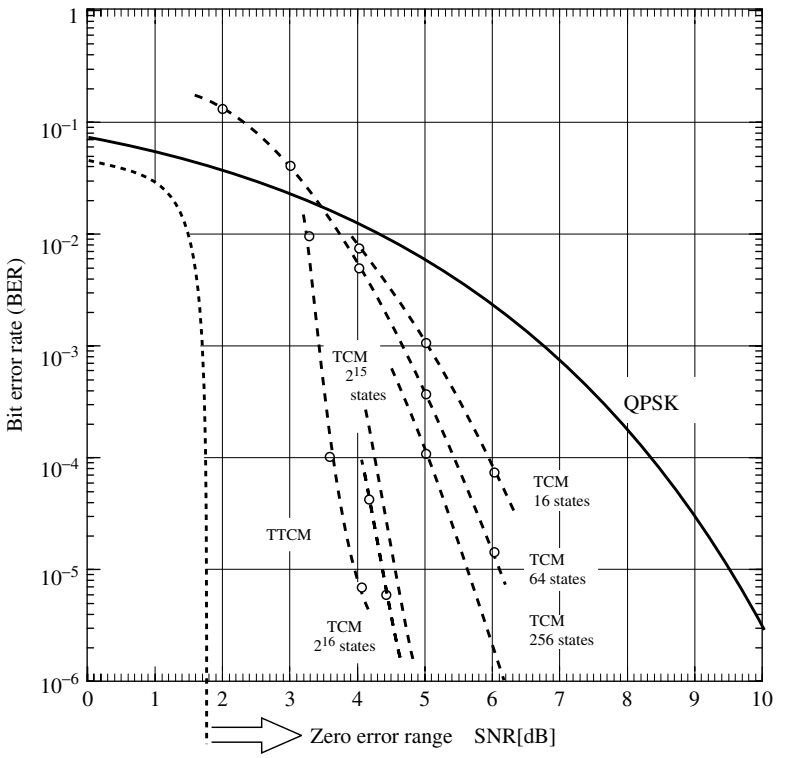


Figure 1.2 Bit error probability of quadrature phase-shift keying (QPSK) and selected 8-PSK trellis-coded modulation (TCM) and trellis-turbo-coded modulation (TTCM), and block-turbo-coded (BTC) systems as a function of the normalized signal-to-noise ratio.

be increased; that is, the S/N had to be improved. Figure 1.2 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 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 (Section 1.3), which is the maximum that QPSK can achieve, as long as SNR $>$ 1.5 (1.76 dB), shattering conventional wisdom. The limit on SNR is calculated using (1.1); see Section 1.5.

Also shown in Figure 1.2 is the performance of several trellis-coded modulation (TCM) and trellis-turbo-coded modulation (TTCM) schemes using 8-ary phase-shift keying (8-PSK) (Chapter 3), 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 1.2 achieve these gains without requiring more bandwidth than the uncoded QPSK system.

As we will discuss in Chapter 3, 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 [52]. The TTCM system is based on turbo coding principles (Chapter 10, 12) 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 year[3] of the transistor, probably the 20th 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 [19].

Figure 1.3 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/s [46] against QPSK and the theoretical performance established via simulations (Figure 1.2). 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 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 refs. [18, 22, and 35–37]). Contrary to TCM, these codes do not preserve bandwidth and the gains in power efficiency in Figure 1.4 are partly obtained by a power bandwidth tradeoff; that is, 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 [50, 51]. 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.

For very low target error probabilities, tandem coding methods, called *concatenated coding*, have become very popular [4, 8, 21, 22]. In *classic concatenation*, as illustrated in Figure 1.5, the FEC codec is broken up into an inner and an outer code. The inner code is most often a trellis code that performs the channel error control. The outer code is typically a high-rate Reed–Solomon (RS) block code,

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

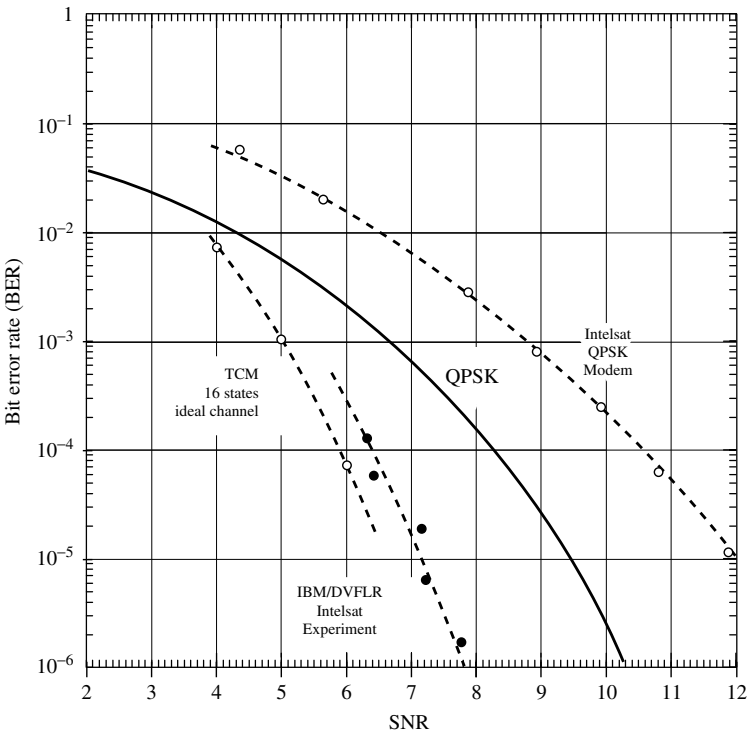


Figure 1.3 Measured bit error probability (QPSK) and a 16-state 8-PSK (TCM) modem over a 64-kbit/s satellite channel [46].

whose 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.

With the discovery of turbo codes [4], new concatenated structures have appeared, shown in Figure 1.6. *Serial concatenation* is very similar to classic concatenation; however, the interleaver Π and deinterleaver Π^{-1} are fundamental. Furthermore, the outer code is usually a very weak error control code, such as a simple parity check code.

The original turbo codes used a parallel arrangement of two codes, called *parallel concatenation*, where both encoders have identical roles. Both arrangements, parallel and serial concatenation, are decoded with the same iterative decoding procedure that alternately invokes soft-output decoders for the two component codes. The structure and workings of these decoders is explored in detail in Chapters 6, 10, 11, and 12.

The field of error control and error correction coding naturally breaks into two disciplines, named somewhat inappropriately block coding and trellis coding. Block coding, which is mostly approached as applied mathematics, has

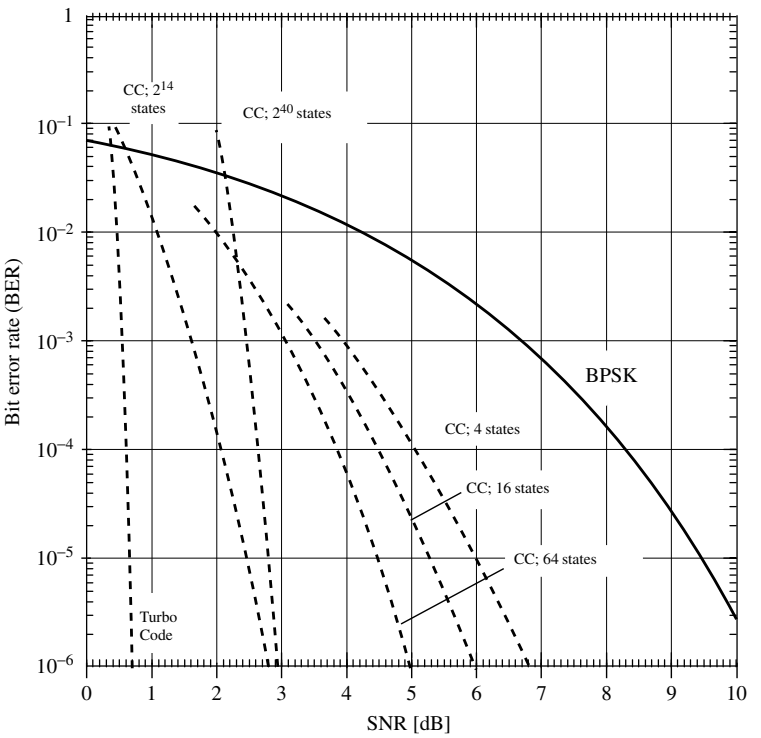


Figure 1.4 Bit error probability of selected rate $R = 1/2$ convolutional and turbo codes as a function of the normalized signal-to-noise ratio. The large-state space code is decoded sequentially, while the performance of all the other convolutional codes is for maximum-likelihood decoding. Simulation results are taken from refs. 4, 18, and 36. The Shannon limit is at SNR = 0 dB.

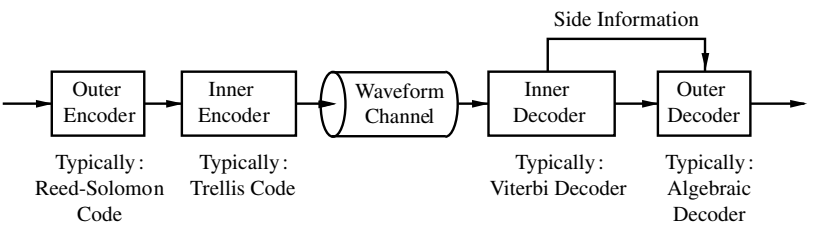


Figure 1.5 Classic concatenated FEC coding using inner and outer codecs.

produced the bulk of publications in error control, whereas 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 when the demodulator no longer makes any (hard) decisions on the transmitted symbols, but passes the received signal

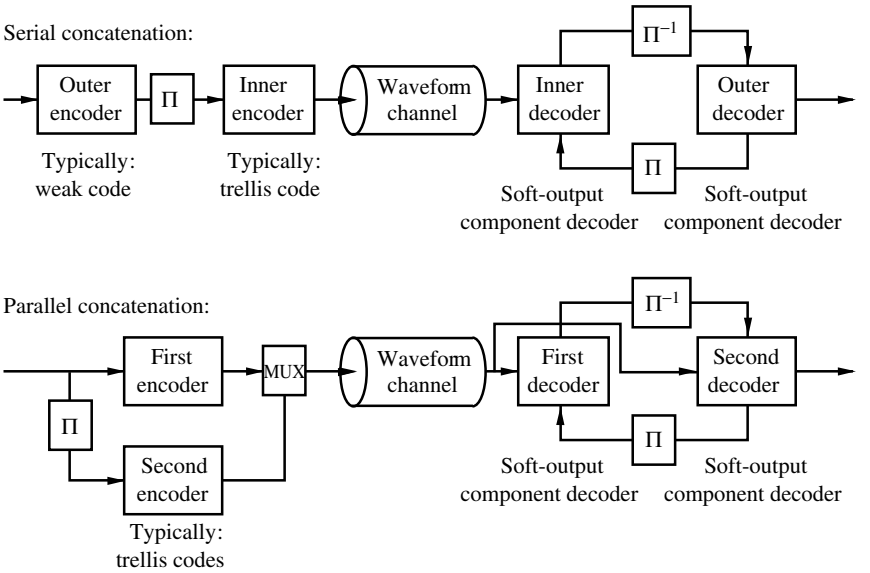


Figure 1.6 Modern concatenated FEC coding using two component codes.

values directly on to the decoder. 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., ref. 15), improving the channel behavior as in concatenated coding. Ironically, many block codes can be decoded very successfully using decoding methods developed for trellis codes (Chapter 7), smearing the boundaries between these two branches.

1.5 BANDWIDTH, POWER, AND COMPLEXITY

Nyquist showed in 1928 [34] 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, there are approximately $2W$ independent signal dimensions per second.[4] 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).

[4] A more precise analysis reveals that there are about $2.4W$ independent signal dimensions per second, see ref. 59.

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

$$\eta = \frac{\text{Transmission rate}}{\text{Channel bandwidth } W} \quad [\text{bits/s/Hz}]. \tag{1.2}$$

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

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

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 become more arbitrary for modulation schemes with infinite spectral occupancy. One commonly used definition is the 99% bandwidth definition; that is, 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

$$S = \frac{kE_b}{T} = RE_b, \tag{1.4}$$

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

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

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:

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

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., $\eta_{\max} \to 0$), we obtain

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

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 distortion in the sense that original information can no longer be reconstructed perfectly [12]. If the source is binary, this compression results in a reconstruction bit error rate (BER) 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

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

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 nonzero bit error rates, given by

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

This is the bound plotted in Figure 1.2 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 [59]

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

or as

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

if we normalize to complex dimensions, in which case R is the rate per complex dimension.

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

$$\frac{E_b}{N_0} \geq \frac{2^{2C_d} - 1}{2C_d}; \qquad \frac{E_b}{N_0} \geq \frac{2^{C_c} - 1}{C_c}. \tag{1.12}$$

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

treating 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 tradeoff between them, regardless of the signal constellation used for transmission.

Figure 1.7 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 (Chapter 12)

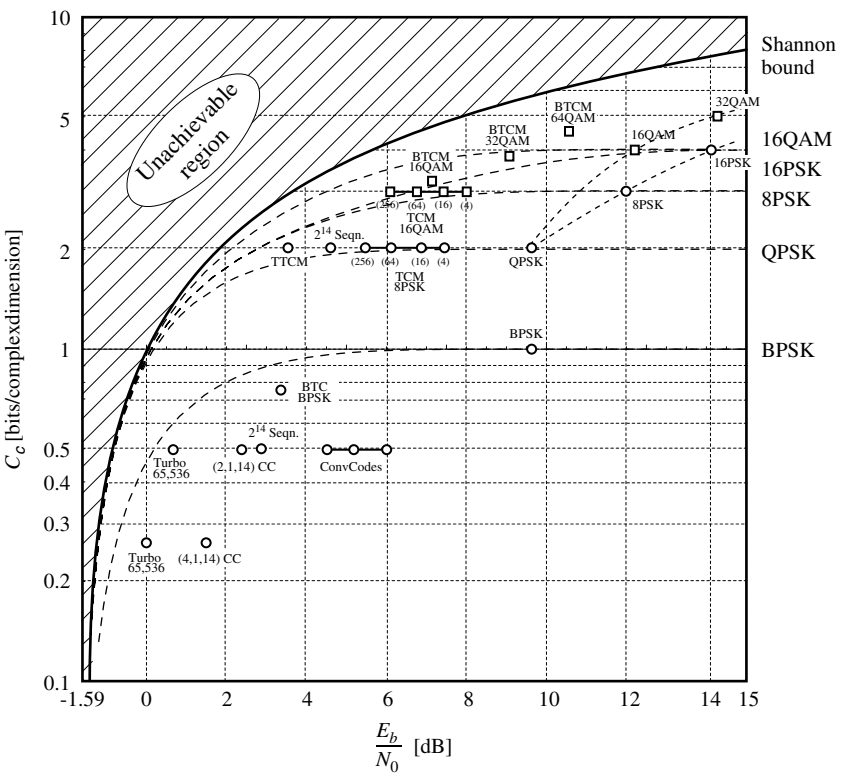


Figure 1.7 Theoretical limits on spectral and power efficiency for different signal constellations and spectral efficiencies achieved by various coded and uncoded transmission methods.

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 $\eta \leq 1$, and we present such methods in Chapter 10 which can be made to approach the Shannon bound arbitrarily closely.

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—for example, for digital TV. The performance points in Figure 1.7 are drawn for $P_b = 10^{-5}$.

Other methods that 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 [40] that causes smaller amplitude variations than Nyquist signaling, and is therefore useful for systems with amplifier nonlinearities, 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 undergone an evolution similar to TCM. The reader is referred to the excellent book by Anderson, Aulin and Sundberg [1], the standard reference for CPM.

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. [43] proved the following *lower bound* on the codeword error probability P_B:

$$P_B > 2^{-N\left(E_{\mathrm{sp}}(R)+o(N)\right)}; \quad E_{\mathrm{sp}}(R) = \max_{\boldsymbol{q}} \max_{\rho>1} \left(E_0(\boldsymbol{q}, \rho) - \rho R\right). \tag{1.13}$$

The exponent $E_0(\boldsymbol{q}, \rho)$ is called the Gallager exponent and depends on the symbol probability distribution $\boldsymbol{q}$ and the optimization parameter ρ; it is discussed in more detail in Chapter 6, as well as in Gallager's book on information theory [11]. 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 1.8 for BPSK modulation [41], together with selected turbo coding schemes and classic concatenated methods. The performance of various length codes follows in parallel to the bound, and we see that the codesize of limiting return is at $N \approx 10^4 - 10^5$, beyond which only small gains are possible. This is the reason why most practical applications of

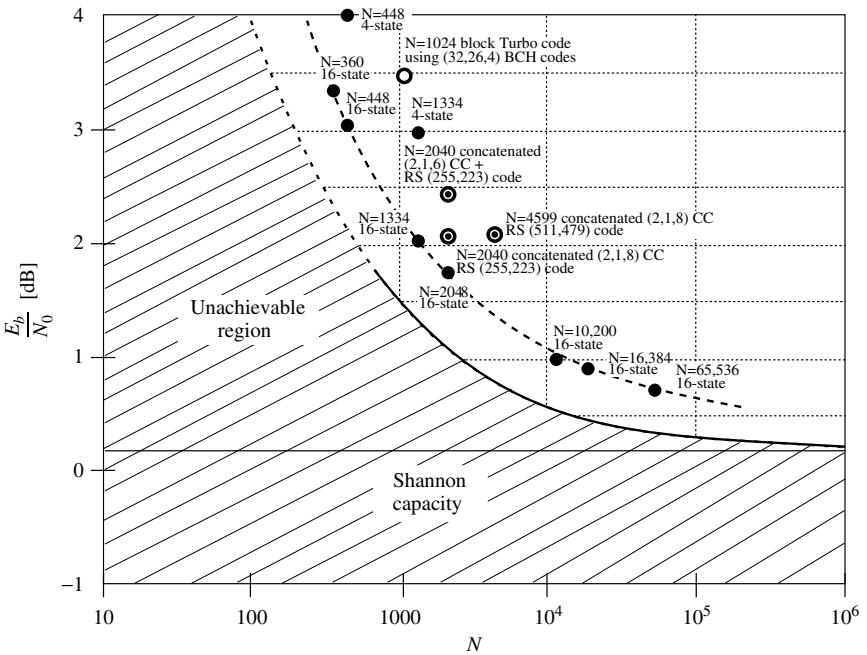


Figure 1.8 Block error rate P_B and sphere-packing lower bound for rate $R = 1/2$ coded example coding systems using BPSK. Turbo codes and selected classic concatenated coding schemes are compared.

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. Implementations of near-capacity error control systems therefore have to process blocks of 10,000 symbols or more, requiring appropriate storage and memory management.

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- 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 7) 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 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 a necessity by the required code complexity.

More recently, decoding circuits based on analog processing have emerged [24, 32, 54], using subthreshold CMOS and bipolar multiplier circuits, variations of the well-known *Gilbert cell* [29]. 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 minute currents 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 to some extent 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 mismatches, body effects, and so on.

Several such decoders have already been successfully fabricated, tested, or extensively simulated by a few research groups around the world. Figure 1.9 shows the potential power advantage that analog decoding enjoys over digital decoding. The figure shows a number of various digital implementation performance points, extracted largely from ref. 57, whose logarithmic power measure lies on a fairly straight line as a function of the required SNR for a bit error performance of 10^{-3} for rate $R = 1/2$ binary codes. The required SNR is an indirect measure of the decoder complexity, which increases as this value approaches the Shannon limit of 0.2 dB. The analog performance points are drawn for two codes, a small size-eight Hamming code, and a medium-sized product code [55, 56].

Interesting, such analog decoder also have a size advantage as shown in Figure 1.10, which is mainly due to the fact a single "analog" transistor performs the operation of many "digital" transistors. This surprising result occurs in spite of the large parallelism of the decoder circuit and the relatively large transistor size that was used for these early implementations. The parallelism also helps with the speed of these decoders, which, even though the analog processing core will run quite slowly at a few hundred kilohertz, are capable of high data throughputs. For example, the $[16, 11]^2$ product decoder running at a core speed of 500 kHz generates a data throughput of over 60 Mbits/s. Even higher rates are possible with larger codes.

Undoubtedly the issues of chip size, speed, and power consumption are becoming central research questions now that the structure of capacity achieving codes is believed to be well understood. More fundamentally, such questions as what is the minimum amount of energy required to *process* a bit of information given a certain computational platform are likely to become more central as size and speed requirements of future digital communications systems continue to increase. We believe that such issues will be at the core of future research and advances.

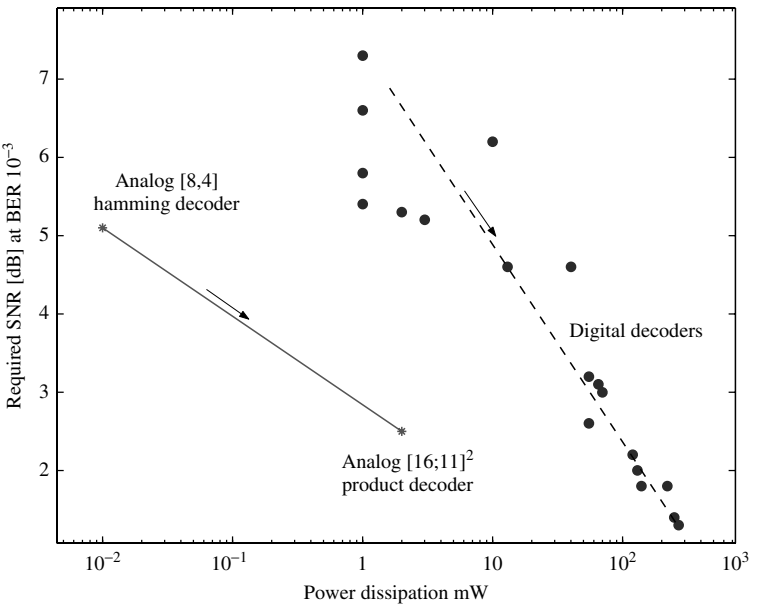


Figure 1.9 Comparison of power consumptions of analog and digital decoders as a function of the code capability.

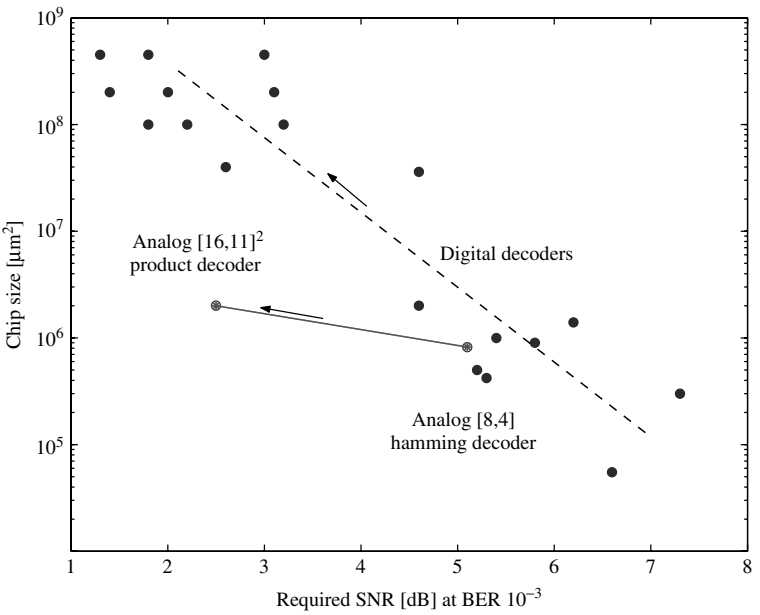


Figure 1.10 Comparison of chip size requirements of analog versus digital decoders as a function of the code capability, which is now plotted on the horizontal axis.

1.6 A BRIEF HISTORY — THE DRIVE TOWARD CAPACITY

Trellis coding celebrated its first success in the application of convolutional codes to deep space probes in the 1960s and 1970s. For a long time afterwards, error control coding was considered an intellectual curiosity with deep-space communications as its only viable application. This is the power-limited case and is 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 spectral density ratio of $E_b/N_0 = 9.6$ dB, along with a spectral efficiency of 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.

One of the earliest attempts to close this signal energy gap was the use of a rate 6/32 biorthogonal (Reed-Muller) block code [25]. 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 $P_b = 10^{-5}$ 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 fivefold 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 Bose–Chaudhuri–Hocquenghem (BCH) codes [2, 26], which enabled the efficient hard-decision decoding of an entire class of block codes. For example, the (255, 123) BCH code has an $R_d \approx 0.5$ bits/symbol and achieves $P_b = 10^{-5}$ with $E_b/N_0 = 5.7$ dB using algebraic decoding.

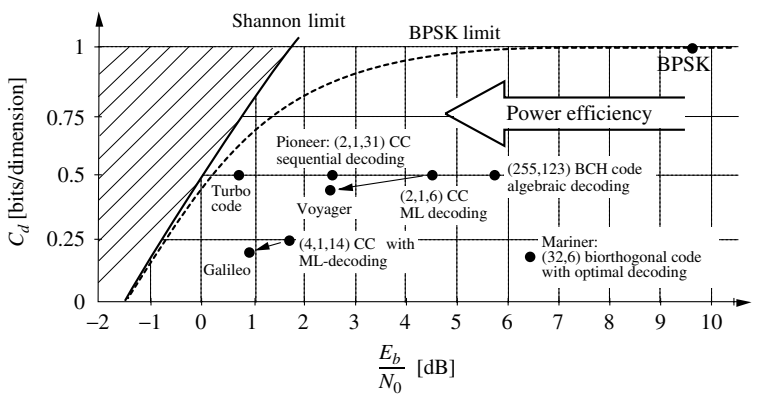


Figure 1.11 Milestones in the drive toward channel capacity achieved by the space systems that evolved over the past 50 years as an answer to the Shannon capacity challenge.

The next step was taken with the introduction of sequential decoding (see Chapter 7) that could make use of soft-decision decoding. Sequential decoding allowed the decoding of long constraint-length convolutional codes and was first used on the Pioneer 9 mission [9]. The Pioneer 10 and 11 missions in 1972 and 1973 both used a long constraint-length (2, 1, 31), nonsystematic convolutional code (Chapter 4) [27]. 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 toward capacity (see Chapter 6). 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 [6] found application in the Galileo mission, where a (4, 1, 14) convolutional code is used. This yields a spectral efficiency of $R_d = 0.25$ bits/symbol and achieves $P_b = 10^{-5}$ at $E_b/N_0 = 1.75$ dB. The performance of this system is also 2.5 dB away from the capacity limit at its rate. 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 [25] 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 gap to capacity 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., ref. 45). 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.

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 prevalent. This attitude was thoroughly overturned by the spectacular success of error control coding on voice-band data transmission systems. Here it was not the power efficiency that was the issue, but rather the spectral efficiency—that is, given a standard telephone channel with an essentially fixed bandwidth and SNR, which was the maximum practical rate of reliable transmission.

The first commercially available voice-band 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. Ungerboeck's invention of trellis-coded modulation in the late 1970s 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 [10]. The latest chapter in

voice-band data modems is the establishment of the CCITT V.34 modem standard [7, 13]. 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 voice-band telephone channel have been reached (according to ref. 53). This needs to be compared to estimates of the channel capacity for a voice-band telephone channel, which are somewhere around 30,000 bits/s. The application of TCM is one of the fastest migrations of an experimental laboratory system to an international standard (V.32–V.34) [5, 7, 13]. Hence, what once was believed to be impossible, achieving capacity on band-limited channels, has become commonplace reality within about a decade and a half after the invention of TCM [47]. The trellis codes used in these advanced modems are discussed in detail in Chapter 3.

In many ways the telephone voice-band channel was an ideal playground for the application of error control coding. Its limited bandwidth of about 3 kHz (400–3400 Hz) implies relatively 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 voice-band 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 is being considered for satellite communications (both geostationary and low-earth orbiting satellites), for land-mobile and satellite-mobile services, for cellular communications networks, for personal communications services (PCS), and for high-frequency (HF) tropospheric long-range communications, among others. To us it seems clear that the age of widespread application of error control coding has only just begun.

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," *Proceedings, 1993 IEEE International Conference on Communication,* Geneva, Switzerland, pp. 1064–1070, 1993.
4. C. Berrou and A. Glavieux, "Near optimal error-correcting coding and decoding: Turbo codes," *IEEE Trans. Commun.,* vol. COM-44, no. 10, pp. 1261–1271, Oct. 1996.
5. Uyless Black, *The V Series Recommendations, Protocols for Data Communications Over the Telephone Network*, McGraw-Hill, New York, 1991.
6. 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.

7. G. D. Forney, L. Brown, M. V. Eyuboglu, and J. L. Moran III, "The V.34 high-speed modem standard," *IEEE Commun. Mag.*, pp. 28–33, Dec. 1996.

8. G. D. Forney, *Concatenated Codes*, MIT Press, Cambridge, MA, 1966.

9. 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.

10. G. D. Forney, "Coded modulation for bandlimited channels," *IEEE Information Theory Society Newsletter*, Dec. 1990.

11. R. G. Gallager, *Information Theory and Reliable Communications*, John Wiley & Sons, New York 1968.

12. R. M. Gray, *Source Coding Theory*, fourth printing, Kluwer Academic Publishers, Boston, 1997.

13. CCITT Recommendations V.34.

14. M. J. E. Golay, "Notes on digital coding," *Proceedings, IEEE*, vol. 37, p. 657, 1949.

15. J. Hagenauer and P. Höher, "A Viterbi algorithm with soft-decision outputs and its applications," *Proceedings, IEEE Globecom'89*, 1989.

16. J. Hagenauer, "Rate compatible punctured convolutional codes (RCPC-codes) and their application," *IEEE Trans. Commun.,* vol. COM-36, pp. 389–400, April 1988.

17. R. W. Hamming, "Error detecting and error correcting codes," *Bell Syst. Tech. J.,* vol. 29, pp. 147–160, 1950.

18. 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.

19. H. Imai et al., *Essentials of Error-Control Coding Techniques*, Academic Press, New York, 1990.

20. N. S. Jayant and P. Noll, *Digital Coding of Waveforms*, Prentice-Hall, Englewood Cliffs, NJ, 1984.

21. 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.

22. S. Lin and D. J. Costello, Jr., *Error Control Coding: Fundamentals and Applications,* Prentice-Hall, Englewood Cliffs, NJ, 1983.

23. 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.

24. 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.

25. F. J. MacWilliams and N. J. A. Sloane, *The Theory of Error Correcting Codes*, North-Holland, New York, 1988.

26. J. L. Massey, "Shift register synthesis and BCH decoding," *IEEE Trans. Inform. Theory,* vol. IT-15, pp. 122–127, 1969.

27. 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.

28. J. L. Massey, "Coding and modulation in digital communications," *Proc. Int. Zürich Sem. Digital Commun.*, Zürich, Switzerland, March 1974, pp. E2(1)–E2(4).

29. C. Mead, *Analog VLSI and Neural Systems*, Addison-Wesley, Reading, MA, 1989.

30. H. Meyr and G. Ascheid, *Synchronization in Digital Communications*, vol. 1, John Wiley & Sons, New York, 1990.

31. H. Meyr, M. Moeneclaey, and S. A. Fechtel, *Digital Communication Receivers*, John Wiley & Sons, New York, 1998.

32. M. Moerz, T. Gabara, R. Yan, and J. Hagenauer, "An analog .25 μm BiCMOS tailbiting MAP decoder," *IEEE Proceedings, International Solid-State Circuits Conference*, pp. 356–357, San Francisco, Feb. 2000.

33. M. Mouly and M.-B. Pautet, *The GSM System for Mobile Communications*, sold by the authors, ISBN 2-9507190-0-7, 1993.

34. H. Nyquist, "Certain topics in telegraph transmission theory," *AIEE Trans.*, pp. 617–644, 1928.

35. J. P. Odenwalder, "Optimal decoding of convolutional codes," Ph.D. thesis, University of California, Los Angeles, 1970.

36. 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.

37. J. G. Proakis, *Digital Communications,* McGraw-Hill, New York, 1989.

38. S. Ramseier and C. Schlegel, "On the bandwidth/power tradeoff of trellis coded modulation schemes," *Proceedings, IEEE Globecom'93*, 1993.

39. 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.

40. C. Schlegel, "Coded overlapped quadrature modulation," *Proceedings, Global Conference on Communication*, GLOBECOM'91, Phoenix, Arizona, Dec. 1991.

41. C. Schlegel and L. C. Perez, "On error bounds and turbo codes," *IEEE Commun. Lett.*, vol. 3, no. 7, July 1999.

42. C. E. Shannon, "A mathematical theory of communications," *Bell Syst. Tech. J.,* vol. 27, pp. 379–423, July 1948.

43. 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.

44. 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.

45. S. ten Brink, "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.

46. G. Ungerboeck, J. Hagenauer, and T. Abdel-Nabi, "Coded 8-PSK experimental modem for the INTELSAT SCPC system," *Proceedings, ICDSC, 7th*, pp. 299–304, 1986.

47. G. Ungerboeck, "Channel coding with multilevel/phase signals," *IEEE Trans. Inform. Theory,* vol. IT-28, no. 1, pp. 55–67, Jan. 1982.

48. G. Ungerboeck, "Trellis-coded modulation with redundant signal sets part I: Introduction," *IEEE Commun. Mag.,* vol. 25, no. 2, pp. 5–11, Feb. 1987.

49. 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.

50. A. J. Viterbi, "Spread spectrum communications—myths and realities," *IEEE Commun. Mag.,* vol. 17, pp. 11–18, May 1979.

51. A. J. Viterbi, "When not to spread spectrum—a sequel," *IEEE Commun. Mag.,* vol. 23, pp. 12–17, April 1985.

52. 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, September 1995.

53. R. Wilson, "Outer limits," *Electronic News*, May 1996.

54. 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 VSLI Conference ARVLSI*, Salt Lake City, March 2000.

55. C. Winstead, J. Dai, S. Yu, C. Myers, R. Harrison, and C. Schlegel, "CMOS analog MAP decoder for (8,4) Hamming code," *IEEE J. Solid State Circuits*, January 2004.

56. C. Winstead and C. Schlegel, "Importance Sampling for SPICE-level verification of analog decoders," *International Symposium on Information Theory (ISIT'03)*, Yokohama, June 2003.

57. 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.

58. 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.

59. J. M. Wozencraft and I. M. Jacobs, *Principles of Communication Engineering*, John Wiley & Sons, New York, 1965, reprinted by Waveland Press, 1993.

CHAPTER 2

Communication Theory Basics

2.1 THE PROBABILISTIC VIEWPOINT

Communications, the transmission of information from a sender to a receiver (destination), is fundamentally a random experiment. The sender selects one of a number of possible messages which is presented (transmitted) to the receiver. The receiver has no knowledge about which message is chosen by the sender; however, this message is chosen from a set of possible messages known to both the sender and the receiver. In transition the signal carrying the message is distorted, mainly by channel noise. This noise in a communications link is a fundamental property of the link. Without noise, the concept of communications would not exist, since it would be possible to move arbitrary amounts of information virtually instantaneously. For example, if we could transmit a chosen voltage signal without noise, we could transmit an exact real number (the voltage level), whose infinite expansion into binary digits would give us an unlimited amount of data to transport. Noise then, in a sense, prevents information from being omnipresent, and it makes possible the modern concepts of communications. Since noise is a random process, it should not be surprising that communication theory draws heavily from probability theory.

Figure 2.1 shows a simplified system block diagram of such a sender/receiver communications 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, in particular in the cases with which this book is concerned. The transmission medium is called the channel and may be a telephone wire line, a radio link, an underwater acoustic link, or any other suitable arrangement, as elaborated in Chapter 1. One of the major impairments in all communications systems is thermal noise, which is generated by the random motion of particles inside the receiver's signal sensing elements and has the property that it adds linearly to the received signal, hence the channel model in Figure 2.1.

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

Trellis and Turbo Coding, by Christian B. Schlegel and Lance C. Pérez
ISBN 0-471-22755-2

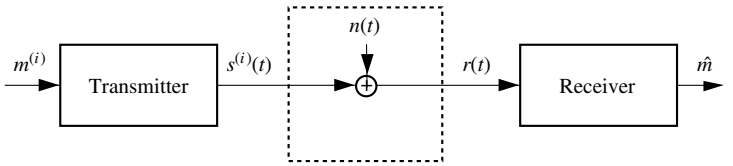


Figure 2.1 Block diagram of a sender/receiver communication system used on a channel with additive noise.

Appendix 2.A. Noise is itself a random quantity, and our communication system becomes a joint random experiment. The output of the transmitter and the output of the noise source are both random quantities. 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 highest achievable reliability. In the context of a discrete set of messages, one speaks of message detection, rather than estimation, since the receiver is not trying to reproduce closely a given signal, but makes a decision for one of a finite number of messages. As a measure of the reliability of this process, we use the almost universally accepted 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, that is, $P_e = \Pr(\hat{m} \neq m^{(i)})$. The solution to this problem is essentially completely known for many cases, and we subsequently present an overview of the optimum receiver principles in additive white Gaussian noise.

2.2 VECTOR COMMUNICATION CHANNELS

A very popular way to generate the signals at the transmitter is to synthesize them as a linear combination of N *basis waveforms* $\phi_j(t)$; that is, the transmitter selects

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

as the transmitted signal for the ith message. Oftentimes the basis waveforms are chosen to be orthonormal; that is, they fulfill the condition

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

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

$$\boldsymbol{s}^{(i)} = \left(s_1^{(i)}, s_2^{(i)}, \ldots, s_N^{(i)}\right). \tag{2.3}$$

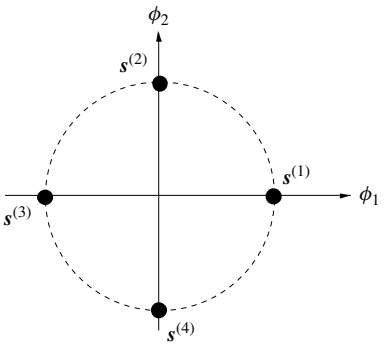


Figure 2.2 Illustration of four signals using two orthogonal basis functions.

We can now visualize the signals geometrically by viewing the signal vectors $\boldsymbol{s}^{(i)}$ 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

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

and $f_1 = k/T$ is an integer multiple of $1/T$. 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 the information on the actual waveform, 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 $\boldsymbol{s}^{(i)}$ 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 $\boldsymbol{s}^{(i)}$.

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

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

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

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

$$\begin{aligned} d_{ij}^2 &= \sum_{l=1}^{N} \left(s_l^{(i)} - s_l^{(j)} \right)^2 \\ &= \sum_{l=1}^{N} \left(\int_{-\infty}^{\infty} (s_l^{(i)} - s_l^{(j)}) \phi_l(t) \right)^2 \\ &= \int_{-\infty}^{\infty} \left(s^{(i)}(t) - s^{(j)}(t) \right)^2 dt \end{aligned} \tag{2.6}$$

is in fact the energy of the difference signal $(s^{(i)}(t) - s^{(j)}(t))$. (It is easier to derive (2.6) in the reverse direction, that is, from bottom to top).

It can be shown [17, 27] 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 $\boldsymbol{r} = \boldsymbol{s}^{(i)} + \boldsymbol{n}$ at the output of the bank of correlators. The statistics of the noise vector $\boldsymbol{n}$ 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

$$\begin{aligned} E[n_l n_j] &= \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} E[n_w(\alpha) n_w(\beta)] \phi_l(\alpha) \phi_j(\beta)\, d\alpha d\beta \\ &= \frac{N_0}{2} \int_{-\infty}^{\infty} \phi_l(\alpha) \phi_j(\alpha)\, d\alpha = \begin{cases} \frac{N_0}{2}, & l = j, \\ 0, & \text{otherwise.} \end{cases} \end{aligned} \tag{2.7}$$

We are pleased to see that, courtesy of the orthonormal basis waveforms, the components of the random noise vector $\boldsymbol{n}$ 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 details see, for example, ref. 2). From the above we conclude that the components of $\boldsymbol{n}$ 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. First, we need not concern ourselves with the actual choices of signal waveforms when discussing receiver algorithms, and, second, 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 third, 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 [27] 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 here.

2.3 OPTIMUM RECEIVERS

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

$$P[m^{(j)}|\boldsymbol{r}], \tag{2.8}$$

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

$$P_c = \int_{\boldsymbol{r}} P[\text{correct}|\boldsymbol{r}]p(\boldsymbol{r})\,d\boldsymbol{r}; \tag{2.9}$$

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

Using Bayes' rule we obtain

$$P[m^{(j)}|\boldsymbol{r}] = \frac{P[m^{(j)}]p(\boldsymbol{r}|m^{(j)})}{p(\boldsymbol{r})}, \tag{2.10}$$

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

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

$$p_n(\boldsymbol{r} - \boldsymbol{s}^{(j)}) = \frac{1}{(\pi N_0)^{N/2}} \exp\left(-\frac{|\boldsymbol{r} - \boldsymbol{s}^{(j)}|^2}{N_0}\right). \tag{2.11}$$

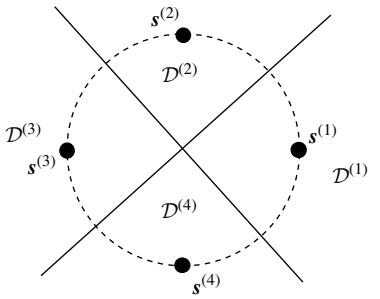


Figure 2.3 Decision regions for ML detection of equiprobable QPSK signals.

Maximizing (2.8) is now seen to be equivalent to minimizing the Euclidean distance

$$|\boldsymbol{r} - \boldsymbol{s}^{(j)}|^2 \tag{2.12}$$

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

The decision rule (2.12) implies *decision regions* $\mathcal{D}^{(j)}$ for each signal point which consist of all the points in Euclidean N-space which are closer to $\boldsymbol{s}^{(j)}$ than any other signal point. These regions are also known as *Voronoi regions*. The decision regions are illustrated for QPSK in Figure 2.3.

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

$$P_e(\boldsymbol{s}^{(i)}) = \int_{\boldsymbol{r} \notin \mathcal{D}^{(i)}} p_n(\boldsymbol{r} - \boldsymbol{s}^{(i)})\, d\boldsymbol{r}. \tag{2.13}$$

Equation (2.13) 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 $\boldsymbol{s}^{(i)}$ is decoded as signal $\boldsymbol{s}^{(j)}$ assuming that there are only these two signals. In order to calculate the two-signal error probability, we may disregard all signals except $\boldsymbol{s}^{(i)}$, $\boldsymbol{s}^{(j)}$. Take, for example, $\boldsymbol{s}^{(i)} = \boldsymbol{s}^{(1)}$ and $\boldsymbol{s}^{(j)} = \boldsymbol{s}^{(2)}$ 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 *pairwise error probability*, is given by

$$P_{\boldsymbol{s}^{(i)} \to \boldsymbol{s}^{(j)}} = Q\left(\sqrt{\frac{d_{ij}^2}{2N_0}}\right), \tag{2.14}$$

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

$$Q(\alpha) = \frac{1}{\sqrt{2\pi}} \int_{\alpha}^{\infty} \exp\left(-\frac{\beta^2}{2}\right) d\beta, \tag{2.15}$$

is a nonelementary integral, referred to as the (Gaussian) Q-function. For a more detailed discussion, see ref. 27. In any case, Equation (2.14) states that the probability of error between two signals decreases exponentially with their squared Euclidean distance d_{ij}, due to the overbound [3]

$$Q\left(\sqrt{d_{ij}^2/2N_0}\right) \leq (1/2) \exp\left(-d_{ij}^2/4N_0\right). \tag{2.16}$$

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

$$u(t) = s^{(i)}(t) \star \phi_l(-t) = \int_{-\infty}^{\infty} s^{(i)}(\alpha)\phi_l(\alpha - t)\, d\alpha. \tag{2.17}$$

If the output of the filter $\phi_l(-t)$ is sampled at time $t = 0$, Equation (2.17) and (2.5) are identical, that is,

$$u(t = 0) = s_l^{(i)} = \int_{-\infty}^{\infty} s^{(i)}(\alpha)\phi_l(\alpha)\, d\alpha. \tag{2.18}$$

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 $|\boldsymbol{r} - \boldsymbol{s}^{(i)}|^2$, or equivalently maximizes

$$2 \cdot \boldsymbol{r} \cdot \boldsymbol{s}^{(i)} - \left|\boldsymbol{s}^{(i)}\right|^2, \tag{2.19}$$

where we have neglected the term $|\boldsymbol{r}|^2$, which is common to all the hypotheses. The correlation $\boldsymbol{r} \cdot \boldsymbol{s}^{(i)}$ is the central part of (2.19) and can be implemented in two different ways: as the basis function matched filter receiver, performing the summation after correlation, that is,

$$\boldsymbol{r} \cdot \boldsymbol{s}^{(i)} = \sum_{j=1}^{N} r_j s_j^{(i)} = \sum_{j=1}^{N} s_j^{(i)} \int_{-\infty}^{\infty} r(t)\phi_j(t)\, dt, \tag{2.20}$$

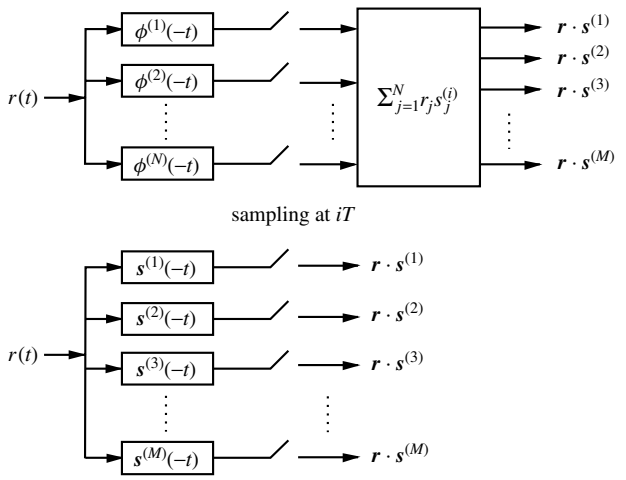


Figure 2.4 The basis function matched filter receiver shown on top and the signal matched filter receiver shown in the bottom. M is the number of messages.

or as the signal matched filter receiver performing

$$\boldsymbol{r} \cdot \boldsymbol{s}^{(i)} = \int_{-\infty}^{\infty} r(t) \sum_{j=1}^{N} s_j^{(i)} \phi_j(t)\, dt = \int_{-\infty}^{\infty} r(t) s^{(i)}(t)\, dt. \tag{2.21}$$

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. One notable exception are spread-spectrum systems [28].

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 a *sufficient statistics*.

2.5 MESSAGE SEQUENCES

In practice, our signals $s^{(i)}(t)$ will most often consist of a sequence of identical, time-displaced waveforms, called pulses, described by

$$s^{(i)}(t) = \sum_{r=-l}^{l} a_r^{(i)} p(t - rT), \tag{2.22}$$

where $p(t)$ is some pulse waveform, the $a_r^{(i)}$ are the discrete symbol values from some finite signal alphabet (e.g., binary signaling: $a_r^{(i)} \in \{-1, 1\}$), and $2l + 1$ is

the length of the sequence in numbers of symbols. The parameter T is the timing delay between successive pulses, also called the *symbol period*. The output of the filter matched to the signal $s^{(i)}(t)$ is given by

$$\begin{aligned} y(t) &= \int_{-\infty}^{\infty} r(\alpha) \sum_{r=-l}^{l} a_r^{(i)} p(\alpha - rT - t)\, d\alpha \\ &= \sum_{r=-l}^{l} a_r^{(i)} \int_{-\infty}^{\infty} r(\alpha) p(\alpha - rT - t)\, d\alpha, \end{aligned} \tag{2.23}$$

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

$$y(t=0) = \sum_{r=-l}^{l} a_r^{(i)} \int_{-\infty}^{\infty} r(\alpha) p(\alpha - rT)\, d\alpha = \sum_{r=-l}^{l} a_r^{(i)} y_r, \tag{2.24}$$

where $y_r = \int_{-\infty}^{\infty} r(\alpha) p(\alpha - rT)$ is the output of a filter matched to the pulse $p(t)$ sampled at time $t = rT$. The signal matched filter can therefore be implemented by a pulse matched filter, whose output $y(t)$ is sampled at multiples of the symbol time T.

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

$$\int_{-\infty}^{\infty} p(t - rT) p(t - hT)\, dt = \delta_{rh}. \tag{2.25}$$

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

$$z(t) = \sum_{r=-l}^{l} a_r^{(i)} p(t - rT) \star p(-t) = \sum_{r=-l}^{l} a_r^{(i)} g(t - rT), \tag{2.26}$$

where $g(t - rT) = \int_{-\infty}^{\infty} p(\alpha - rT) p(\alpha - t)\, d\alpha$ is the composite pulse/pulse-matched filter waveform. If (2.25) holds, $z(t)$ is completely separable and the rth 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. If successive symbols $a_r^{(i)}$ are chosen independently, the system may then be viewed as independently using a one-dimensional signal constellation system $2l+1$ times. This results in tremendous savings in complexity and is the state of the art signaling.

Condition (2.25) ensures that symbols transmitted at different times do not interfere with each other; that is, we have *intersymbol interference*-free signaling. Condition (2.25) is known as the Nyquist criterion, which requires that the

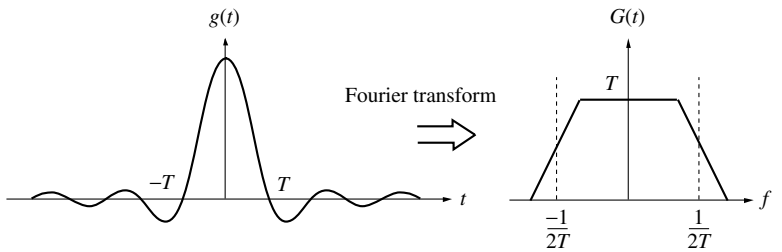


Figure 2.5 Example of a Nyquist pulse with trapezoid frequency spectrum.

composite waveform $g(t)$ pass through zero at all multiples of the sampling time T, except $t = 0$, that is,

$$\int_{-\infty}^{\infty} p(t - rT)p(t - hT)\,dt = \delta_{rh} \Rightarrow g(rT) = \begin{cases} 1, & \text{if } r = 0, \\ 0, & \text{otherwise.} \end{cases} \tag{2.27}$$

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

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

$$\begin{aligned} g(rT) &= \int_{-\infty}^{\infty} G(f)e^{2\pi jfrT}\,df \\ &= \sum_{n=-\infty}^{\infty} \int_{(2n-1)/2T}^{(2n+1)/2T} G(f)e^{2\pi jfrT}\,df \\ &= \int_{-1/2T}^{1/2T} \sum_{n=-\infty}^{\infty} G\left(f + \frac{n}{T}\right) e^{2\pi jfrT}\,df. \end{aligned} \tag{2.28}$$

If the folded spectrum

$$\sum_{n=-\infty}^{\infty} G\left(f + \frac{n}{T}\right); \qquad -\frac{1}{2T} \le f \le \frac{1}{2T}, \tag{2.29}$$

equals a constant, T for normalization reasons, the integral in (2.28) evaluates to $g(rT) = \sin(\pi r)/(\pi r) = \delta_{0r}$; that is, the sample values of $g(t)$ are zero at all nonzero multiples of T, as required. Equation (2.29) 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 spectrum is given by

$$G(f) = \begin{cases} T, & |f| \le \frac{1-\beta}{2T}, \\ \frac{T}{2}\left(1 - \sin\left(\frac{\pi(2T|f|-1)}{2\beta}\right)\right), & \frac{1-\beta}{2T} \le |f| \le \frac{1+\beta}{2T}, \\ 0, & \frac{1+\beta}{2T} \le |f|, \end{cases} \tag{2.30}$$

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 very bandwidth-efficient signals.

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

$$g(t, \beta) = \frac{\sin(\pi t/T)}{\pi t/T} \frac{\cos(\beta \pi t/T)}{1 - (2\beta t/T)^2}. \tag{2.31}$$

Pulses $g(t, \beta)$ and their spectra are shown for several values of the roll-off factor in Figure 2.6.

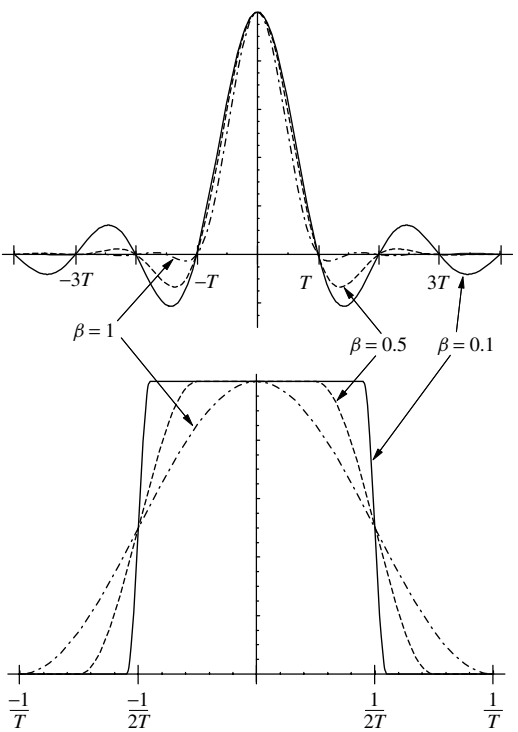


Figure 2.6 Spectral raised cosine Nyquist pulses and their spectrum.

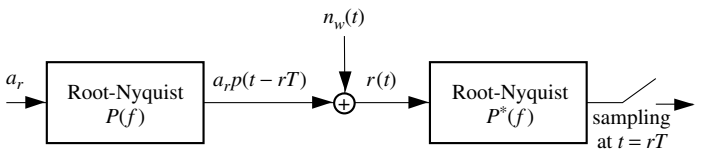


Figure 2.7 An optimal communication system using root-Nyquist signaling.

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

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 spillover. 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 nonflat 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., refs. 17 and 22).

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

$$s_0(t) = x(t)\sqrt{2}\cos(2\pi f_0 t), \tag{2.32}$$

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

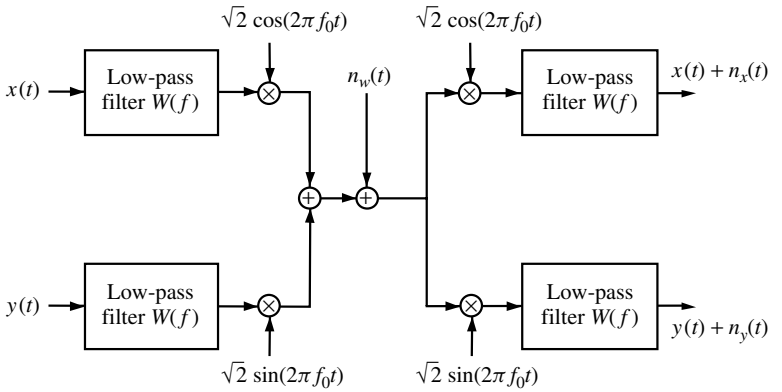


Figure 2.8 Quadrature DSB-SC modulation/demodulation stages.

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, ref. 27) that this arrangement is optimal; that is, no information or optimality is lost by using the product demodulator for DSB-SC modulated signals.

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 two-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

$$\begin{aligned} s_0(t) &= x(t)\sqrt{2}\cos(2\pi f_0 t) - y(t)\sqrt{2}\sin(2\pi f_0 t) \\ &= \mathrm{Re}[(x(t) + jy(t))\sqrt{2}e^{2\pi j f_0 t}], \end{aligned} \tag{2.33}$$

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

$$s(t) = \sum_{r=-l}^{l} (a_r + jb_r)p(t - rT) = \sum_{r=-l}^{l} c_r p(t - rT), \tag{2.34}$$

where c_r is a complex (two-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

$$n_x(t) = \sqrt{2} \int_{-\infty}^{\infty} n_w(\alpha) \cos(2\pi f_0 \alpha) w(t - \alpha)\, d\alpha, \tag{2.35}$$

$$n_y(t) = \sqrt{2} \int_{-\infty}^{\infty} n_w(\alpha) \sin(2\pi f_0 \alpha) w(t - \alpha)\, d\alpha, \tag{2.36}$$

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

$$\begin{aligned} \mathrm{E}[n_x(t)n_x(t+\tau)] &= 2 \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \cos(2\pi f_0 \alpha) \cos(2\pi f_0 \beta) w(t - \alpha) w(t + \tau - \beta) \\ &\qquad \times \mathrm{E}[n_w(\alpha) n_w(\beta)]\, d\alpha d\beta \\ &= \frac{N_0}{2} \int_{-\infty}^{\infty} (1 + \cos(4\pi f_0 \alpha))\, w(t - \alpha) w(t + \tau - \alpha)\, d\alpha \\ &= \frac{N_0}{2} \int_{-\infty}^{\infty} w(t - \alpha) w(t + \tau - \alpha)\, d\alpha \\ &= \frac{N_0}{2} \int_{-W}^{W} e^{-2\pi j f \tau} df = N_0 W \frac{\sin(2\pi W \tau)}{2\pi W \tau}, \end{aligned} \tag{2.37}$$

where we have used Parseval's relationships (ref. 27, pp. 237–238) in the third step; that is, the power of $n_x(t)$ equals $E[n^2(t)] = WN_0$. Equation (2.37) is the correlation function of white noise after passage through a low-pass filter of bandwidth W; that is, 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

$$\begin{aligned} E\left[n_x(t)n_y(t+\tau)\right] &= 2 \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \cos(2\pi f_0 \alpha) \sin(2\pi f_0 \beta) w(t - \alpha) w(t + \tau - \beta) \\ &\qquad \times E[n_w(\alpha) n_w(\beta)]\, d\alpha d\beta \\ &= \frac{N_0}{2} \int_{-\infty}^{\infty} \sin(4\pi f_0 \alpha) w(t - \alpha) w(t + \tau - \alpha) d\alpha = 0. \end{aligned} \tag{2.38}$$

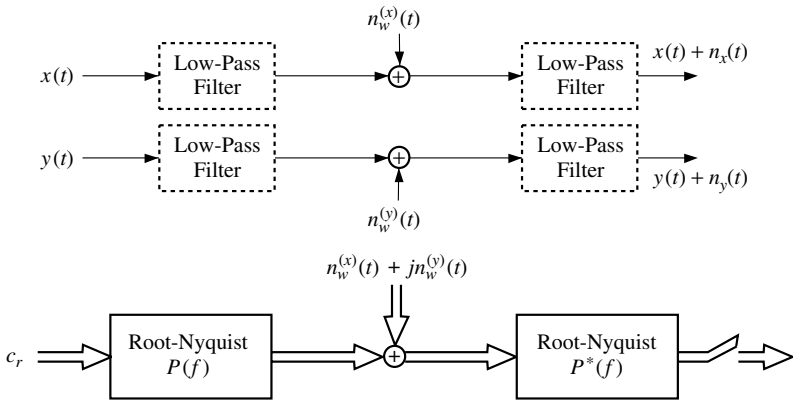


Figure 2.9 Modeling of a DSB-SC systems in additive white Gaussian noise by two independent AWGN-channels. These two channels can be represented as one complex channel model, shown in the lower part of the figure, and referred to as the equivalent complex baseband model for DSB-SC modulation.

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.

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

$$E[n(t)n^*(t)] = 2N_0 W \frac{\sin(2\pi W\tau)}{2\pi W\tau}, \tag{2.39}$$

which is a shorthand version of (2.37) and (2.38). 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 ($n_x(t) = n_w^{(x)}(t), n_y(t) = n_w^{(x)}(t)$) with correlation function

$$E[n(t)n^*(t)] = N_0\delta(t). \tag{2.40}$$

If Nyquist pulses are used with independent complex data symbols c_r, each sample is in fact an independent, two-dimensional signal constellation, hence the ubiquity of two-dimensional signal constellations. Figure 2.10 shows some of the more popular two-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

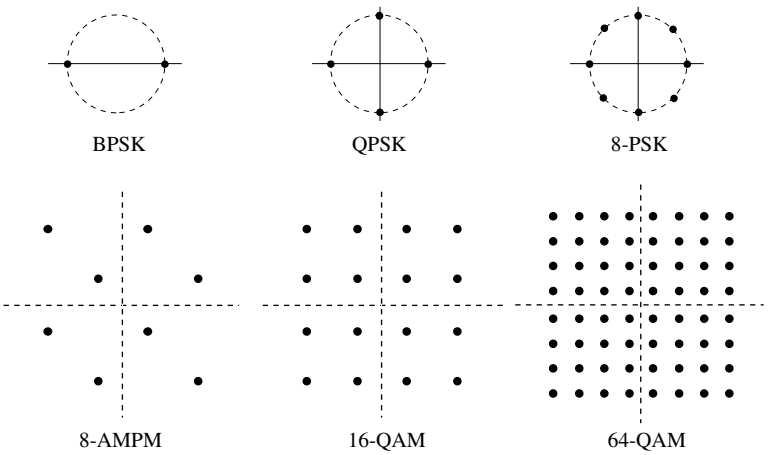


Figure 2.10 Popular two-dimensional signal constellations .

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.

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

$$\eta = \frac{\text{Bit rate}}{\text{Channel bandwidth } W} \text{ [bits/s/Hz]}. \tag{2.41}$$

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 1.7. The bandwidth efficiency of QPSK is twice that of BPSK for the same value 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 1.7 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 (2.41), we must first find the power spectrum of the complex pulse train

$$s(t-\delta) = \sum_{r=-l}^{l} c_r\, p(t - rT - \delta), \tag{2.42}$$

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 (2.42) is that, if we let $l \to \infty$, $s(t-\delta)$ can be made stationary. Starting with the autocorrelation function

$$\begin{aligned} R_s(t, t+\tau) &= E[s^*(t)s(t+\tau)] = E[s^*(t-\delta)s(t+\tau-\delta)] \\ &= \sum_{p=-\infty}^{\infty} \sum_{q=-\infty}^{\infty} E[c_p^* c_q] E_\delta[p(t-pT-\delta)p(t+\tau-qT-\delta)], \end{aligned} \tag{2.43}$$

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

$$R_{cc}(r) = E[c_q^* c_{q+r}] \tag{2.44}$$

depends only on r. This lets us rewrite (2.43) as

$$\begin{aligned} R_s(t, t+\tau) &= \sum_{r=-\infty}^{\infty} R_{cc}(r) \sum_{q=-\infty}^{\infty} E_\delta[p(t-qT-\delta)p(t+\tau-(q+r)T-\delta)] \\ &= \sum_{r=-\infty}^{\infty} R_{cc}(r) \sum_{q=-\infty}^{\infty} \frac{1}{T} \int_0^T p(t-qT-\delta) \\ &\qquad \times p(t+\tau-(q+r)T-\delta)\, d\delta \\ &= \sum_{r=-\infty}^{\infty} R_{cc}(r) \frac{1}{T} \int_0^T p(\delta) p(\delta+\tau-rT)\, d\delta, \end{aligned} \tag{2.45}$$

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 $R_s(\tau)$, that is,

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

where

$$C(f) = \sum_{r=-\infty}^{\infty} R_{cc}(r) e^{-j2\pi frT} \tag{2.46}$$

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 34).

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 (ref. 17, pp. 548 ff.) or correlative coding [19], $R_{cc}(r)$ 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 mean[1] $E[c_q] = 0$. Constellations for which this is achieved by the uniform probability distribution for the symbols are called *symmetric constellations* (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 ref. 3).

2.8 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 shown in Figure 2.11.

Each transmit antenna is operated by a 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 intersymbol interference, given that the symbol rate is slower than the channel dispersion. The composite channel can be described by the complex vector equation

$$\boldsymbol{y}_r = \boldsymbol{H}\boldsymbol{c}_r + \boldsymbol{n}_r, \tag{2.47}$$

where $\boldsymbol{y} = (y_{1r}, \ldots, y_{N_r r})$ is the N_r-dimensional received complex vector at time r, $\boldsymbol{c}_r = (c_{1r}, \ldots, c_{N_t r})$ is the N_t-dimensional vector of complex transmit symbols, $\boldsymbol{H}$ is the $N_r \times N_t$ matrix of complex path gains, and $\boldsymbol{n}_r$ is an

[1] If the mean of the symbols c_r is not equal to zero, discrete frequency components appear at multiples of $1/T$ (see ref. 17, Section 3.4).

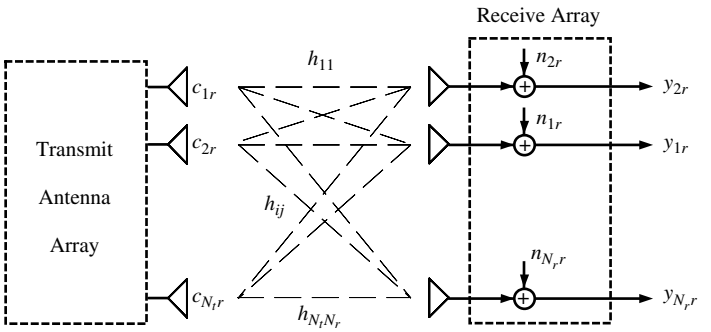


Figure 2.11 Multiple-input multiple-output (MIMO) channel for increased capacity wireless systems.

N_r-dimensional noise vector. Note that each receive antenna is demodulated independently; hence there are N_r independent noise sources.

The Shannon capacity of this channel can be calculated by applying the multidimensional capacity formula [5, 7] and is given by

$$C = \log_2 \left[\det \left(\boldsymbol{I}_{N_r} + \frac{E_s}{N_0 N_t} \boldsymbol{H} \boldsymbol{H}^+ \right) \right] \quad \text{[bits/channel use]}, \tag{2.48}$$

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 (2.48) to a single (real) dimension leads back to Equation (1.11).

We now apply the singular-value decomposition (SVD) to the channel matrix $\boldsymbol{H}$ to obtain a decomposition of the channel equation. The SVD [11] of this matrix is given by

$$\boldsymbol{H} = \boldsymbol{U} \boldsymbol{D} \boldsymbol{V}^+ \tag{2.49}$$

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

Simple manipulations now lead to a decomposed equivalent channel form:

$$\begin{aligned} \boldsymbol{y}_r &= \boldsymbol{H} \boldsymbol{c}_r + \boldsymbol{n}_r \\ &= \boldsymbol{U} \boldsymbol{D} \boldsymbol{V}^+ \boldsymbol{c}_r + \boldsymbol{n}_r \\ \tilde{\boldsymbol{y}}_r = \boldsymbol{U}^+ \boldsymbol{y}_r &= \boldsymbol{D} \tilde{\boldsymbol{c}}_r + \tilde{\boldsymbol{n}}_r . \end{aligned} \tag{2.50}$$

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

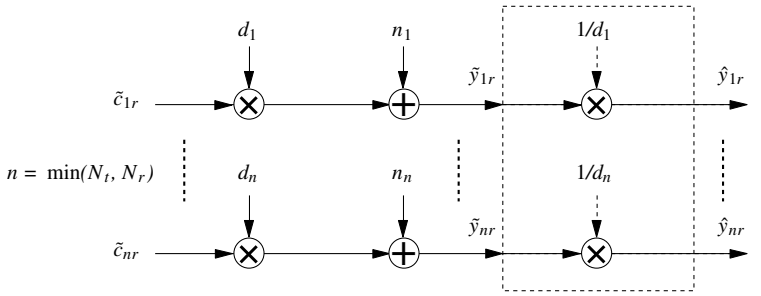


Figure 2.12 Decomposition of the MIMO channel into parallel Gaussian channels via the SVD .

Equation (2.50) is now simply an aggregate of $\min(N_r, N_t)$ parallel channels as illustrated in Figure 2.12. Each channel has power d_j^2, corresponding to the jth singular value of $\boldsymbol{H}$. 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 2.12 has no effect on capacity and turns the channels into n parallel channels affected by additive noise sources with variances $N_0/d_1^2, \ldots, N_0/d_n^2$.

Then the capacity of this set of parallel channels is given by the following [ref. 9, Section 7.5, pp. 343 ff.]

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

$$C = \sum_{j=1}^{n} \log\left(1 + \frac{d_j^2 E_n}{N_0}\right) = \sum_{j=1}^{n} \log\left(\frac{d_j^2 \mu}{N_0}\right) \tag{2.51}$$

and is achieved by the energy allocation

$$\begin{aligned} \frac{N_0}{d_n^2} + E_n &= \mu, \qquad N_0 < \mu d n^2, \\ E_n &= 0, \qquad N_0 \geq \mu d n^2. \end{aligned} \tag{2.52}$$

This theorem is known as the *Waterfilling Capacity Theorem* (Figure 2.13). It says that the available energy should be distributed such that low-noise channels receive more energy than high-noise channels. Its statement can be visualized

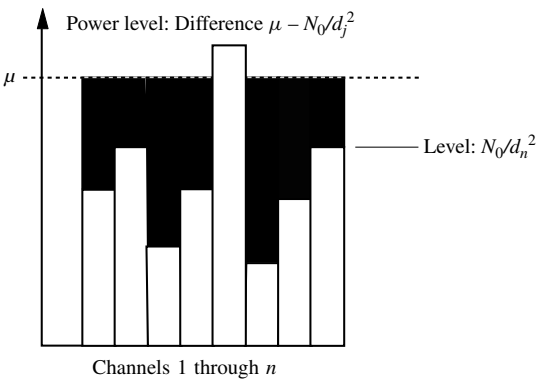


Figure 2.13 Illustration of the Waterfilling Capacity theorem.

in the following figure, where the power levels are shown in black, by thinking of the total available power as liquid that is poured into connected containers whose base level is at the height of the noise power—think of the noise as sediment.

Proof: Consider

$$I(\boldsymbol{x};\hat{\boldsymbol{y}}) \overset{(1)}{\leq} \sum_{j=1}^{n} I(x_j;\hat{y}_j) \qquad \text{(1) Independent } x_n \tag{2.53}$$

$$\overset{(2)}{\leq} \underbrace{\sum_{n=1} \frac{1}{2}\log\left(1+\frac{E_n}{\sigma_n^2}\right)}_{f(\boldsymbol{E})} \qquad \text{(2) Gaussian } x_n \tag{2.54}$$

where $\sigma_n^2 = N_0/dn^2$.
Since equality can be achieved in both inequalities above, the next step is to find the maximizing energy distribution $\boldsymbol{E} = [E_1, \ldots, E_N]$. We identify the following sufficient and necessary conditions via constrained maximization using a Lagrange multiplier:

$$\begin{aligned} \frac{\partial f(\boldsymbol{E})}{\partial E_n} &= \lambda;\ E_n \geq 0; \\ \frac{1}{2(\sigma_n^2 + E_n)} &= \lambda, \\ \sigma_n^2 + E_n &= \frac{1}{2\lambda} = \mu. \end{aligned} \tag{2.55}$$

A more detailed version of this proof is given in ref. 9. Q.E.D.

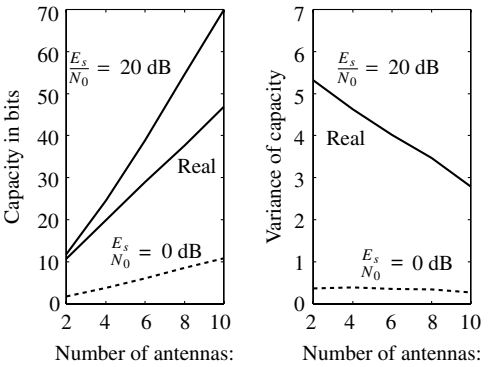


Figure 2.14 Capacities achievable with MIMO channels in suitable environments.

The capacity of MIMO wireless channels has been studied in a number of recent papers [7, 8, 15, 25] and analyzed as a function of the transmission environment [2, 10, 26]. Under favorable conditions it is possible that this channel increases the Shannon capacity by a factor of n.

Figure 2.14 gives some numerical values of the capacity potential of MIMO channels. The numbers were calculated assuming Rayleigh fading for each of the paths, and the capacity formula (2.48) was averaged over the fading distributions. The maximal capacity is achieved for a situation where all component channels h_{ij} are statistically independent, a common assumption. The second curve labeled "real" assumes a more realistic situation using a model with 15 scattering objects uniformly distributed inside a ring of 50 m with a distance of 2 km separating the transmit and receive arrays. The carrier frequency is 2.4 GHz and the symbol time is 1 MHz.

As concluded in ref. 2, if not enough signal power is available, the MIMO channel cannot realize its capacity potential, shown by the dashed curve in the figure, which is virtually identical to the capacity of a single antenna system. The increase with numbers of antennas is only due to the additional energy collected by the larger number of receive antennas. This due to the fact that the capacity formula is of the form $C = n \log(1 + \text{SNR})$, which, for small values of the SNR, can be approximated by $C \approx n\,\text{SNR}$, in which case the number of channels and the SNR play the same role; that is, doubling the SNR has the same effect as doubling the number of channels. In order to exploit the capacity of the MIMO channel, sufficiently large signal power must be available.

The right-hand plot in Figure 2.14 shows the variance of the capacity plotted on the left-hand side as the scattering situation is varied. Due to the effects of the law of large numbers (e.g., a 10×10 MIMO channel consists of 100 largely independent component channels), the variance of the channel capacity diminishes rapidly. Measurements made on antenna arrays [12, 13, 26] have shown that the channel capacity varies by only a few percent among different indoor locations and antenna orientations.

Communications over MIMO channels typically takes two approaches, space-time coding [1, 14, 23, 24] or space-time layering [7, 8]. 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 nonlinear processing techniques. We will discuss applications of turbo coding principle to space-time coding for MIMO channels in Chapter 12.

APPENDIX 2.A

The random waveform $n(t)$ that 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, that is,

$$n(t) = \sum_{i=-\infty}^{\infty} n_i w(t - iT_s - \delta), \tag{2.56}$$

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 (2.56) stationary. A sample function of $n(t)$ is illustrated in Figure 2.15.

A stationary Gaussian process is completely determined by its mean and correlation function (see, e.g., refs. 6 and 20). The mean of the noise process is assumed to be zero, and the correlation function can be calculated as

$$R(\tau) = \mathrm{E}\left[\sum_{i=-\infty}^{\infty}\sum_{j=-\infty}^{\infty} n_i n_j w(t - iT_s - \delta)w(t + \tau - jT_s - \delta)\right]$$

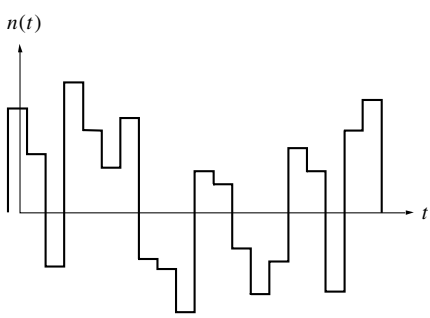


Figure 2.15 Sample function of the discrete approximation to the additive channel noise $n(t)$.

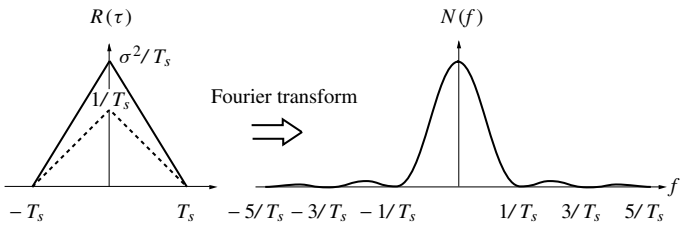


Figure 2.16 Correlation function $R(\tau)$ and power spectral density $N(f)$ of the discrete approximation to the noise function $n(t)$.

$$= \sigma^2 \mathrm{E}_\delta \left[\sum_{i=-\infty}^{\infty} w(t - iT_s - \delta) w(t + \tau - jT_s - \delta) \right]$$
$$= \sigma^2 \Delta_{T_s}(\tau), \tag{2.57}$$

where $\sigma^2 = E\left[n_i^2\right]$ 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 2.16.

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

$$N(f) = \int_{-\infty}^{\infty} R(\tau) e^{-2\pi j \tau f} \, d\tau, \tag{2.58}$$

where f is the frequency variable. The power density spectrum $N(f)$ is also shown in Figure 2.16. Note that as $T_s \to 0$ in the limit, $N(f) \to \sigma^2$. In the limit approximation, (2.56) 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 [27].

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 it is irrelevant for the filter output how we model the noise in its stop-bands.

As $n(t) \to n_w(t)$, the correlation function $R(\tau)$ will degenerate into a pulse of width zero and infinite height as $T_s \to 0$, that is, $R(\tau) \to \sigma^2 \delta(\tau)$, 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 $\delta(t)$, that is,

$$\int_{-\infty}^{\infty} \delta(t - \alpha) f(t) \, dt = f(\alpha), \tag{2.59}$$

where $f(t)$ is an arbitrary function, which is continuous at $t = \alpha$. Property (2.59) is easily proven by using (2.57) and carrying out the limit operation in the integral (2.59). In fact, the relation

$$\int_a^b \delta(\tau - \alpha) f(\tau)\, d\tau = \lim_{T_s \to 0} \int_a^b \Delta_{T_s}(\tau - \alpha) f(\tau)\, d\tau = f(\alpha), \tag{2.60}$$

for any α in the interval (a, b), can be used as a proper definition of the impulse function. The limit in (2.60) could be taken inside the integral

$$\delta(\tau - \alpha) = \lim_{T_s \to 0} \Delta_{T_s}(\tau - \alpha). \tag{2.61}$$

However, any function that is zero everywhere except at one point equals zero when integrated in the Riemann sense; hence, Equation (2.61) is a symbolic equality only, to be understood in the sense of (2.60). An introductory discussion of distributions can be found, for example, in ref. 16, pp. 269–282.

BIBLIOGRAPHY

1. S. Alamouti, "A simple transmit diversity technique for wireless communications," *IEEE J. Select. Areas Commun.*, vol. 16, no. 8, pp. 1451–1458, Oct. 1998.
2. Z. Bagley and C. Schlegel, "Classification of correlated flat fading MIMO channels (multiple antenna channels)," *Proceedings, Third Canadian Workshop on Information Theory*, Vancouver, Canada, June 2001.
3. 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.
4. R. E. Blahut, *Digital Transmission of Information*, Addison-Wesley, New York, 1990.
5. T. M. Cover and J. A. Thomas, *Elements of Information Theory*, John Wiley & Sons, New York, 1991.
6. W. Feller, *An Introduction to Probability Theory and Its Applications*, vol. 1 and 2, revised printing of the 3rd edition, John Wiley & Sons, New York, 1970.
7. 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.
8. 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, March 1998.
9. R. G. Gallager, *Information Theory and Reliable Communication*, John Wiley & Sons, New York, 1968.
10. D. Gesbert, H. Bölcskei, D. A. Gore, and A. J. Paulraj, "Outdoor MIMO wireless channels: Models and performance prediction," *IEEE Trans. Commun.*, to appear.
11. G. H. Golub and C. F. Van Loan, *Matrix Computations*, 3rd edition, John Hopkins Press, Baltimore, 1996.

12. P. Goud, Jr., C. Schlegel, R. Hang, W. Krzymien, Z. Bagley, S. Messerly, P. Watkins, W. Rajamani, "MIMO channel measurements for an indoor office environment," *IEEE Wireless '03*, July 7–9, Calgary, AB, Canada.

13. P. Goud, Jr., C. Schlegel, R. Hang, W. Krzymien, Z. Bagley, S. Messerly, M. Nham, W. Rajamani, "Indoor MIMO channel measurements using dual polarized patch antennas," *Proceedings IEEE PACRIM '03*, Aug. 28–30, Victoria, BC, Canada.

14. B. L. Hughes, "Differential space-time modulation," *IEEE Trans. Inform. Theory*, vol. 46, no. 7, pp. 2567–2578, Nov. 2000.

15. 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.

16. A. Papoulis, *The Fourier Integral and Its Applications*, McGraw-Hill, New York, 1962.

17. J. G. Proakis, *Digital Communications*, 3rd edition, McGraw-Hill, New York, 1995.

18. S. Ramseier and C. Schlegel, "On the Bandwidth/power tradeoff of trellis coded modulation schemes," *Proceedings, IEEE Globecom'93*, 1993.

19. S. Ramseier, "Bandwidth-efficient correlative trellis coded modulation schemes," *IEEE Trans. Commun.,* vol. COM-42,, no. 2/3/4, pp. 1595–1605, 1994.

20. K. S. Shanmugan and A. M. Breipohl, *Random Signals: Detection, Estimation and Data Analysis*, John Wiley & Sons, New York, 1988.

21. C. E. Shannon, "A mathematical theory of communications," *Bell Syst. Tech. J.*, vol. 27, pp. 379–423, July 1948.

22. B. Sklar, *Digital Communications: Fundamentals and Applications*, Prentice-Hall, Englewood Cliffs, NJ, 1988.

23. 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.

24. 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.

25. E. Telatar, "Capacity of multi-antenna Gaussian channels," *Eur. Trans. Telecommun.*, vol. 10, no. 6, Nov.–Dec. 1999.

26. A. L. Swindlehurst, G. German, J. Wallace, and M. Jensen, "Experimental measurements of capacity for MIMO indoor wireless channels," *Proceedings Third IEEE Signal Processing Workshop*, Taoyuan, Taiwan, March 20–25, 2001.

27. J. M. Wozencraft and I. M. Jacobs, *Principles of Communication Engineering*, John Wiley & Sons, New York, 1965, reprinted by Waveland Press, 1993.

28. R. L. Peterson, R. E. Ziemer, and D. E. Borth, *Introduction to Spread-Spectrum Communications*, Prentice-Hall, Englewood Cliffs, NJ, 1995.

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 invariably involves an increase in the complexity of a communication system. It has become apparent over the past few decades that while there are strict limits on the power and bandwidth resources, the complexity of systems could steadily be 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 Ungerboeck [28–31], subsequently termed *trellis-coded modulation* (TCM). We start this chapter with an illustrative example of this intriguing 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 $\boldsymbol{s}^{(2)}$ and signal $\boldsymbol{s}^{(1)}$ (compare Figure 2.2). This will happen with probability

$$P_{\boldsymbol{s}^{(1)} \to \boldsymbol{s}^{(2)}} = Q\left(\sqrt{\frac{d^2 E_s}{2N_0}}\right), \tag{3.1}$$

where $d^2 = 2$ for a unit energy signal constellation, and E_s is the average energy per symbol. Instead of using such a system by transmitting 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 s_r at time r is defined by the contents of the delay cells, that is, $s_r = (s_r^{(2)}, s_r^{(1)}, s_r^{(0)})$. The second part is called a *signal mapper*, and its function is a memoryless mapping of the three bits $v_r = (u_r^{(2)}, u_r^{(1)}, v_r^{(0)})$ into one of the eight symbols of an 8-PSK signal set. The FSM accepts two input bits $u_r = (u_r^{(2)}, u_r^{(1)})$ at each symbol time r, and it transitions from a state s_r to one of four possible successor states s_{r+1}. In this fashion the encoder generates the

Trellis and Turbo Coding, by Christian B. Schlegel and Lance C. Pérez
ISBN 0-471-22755-2

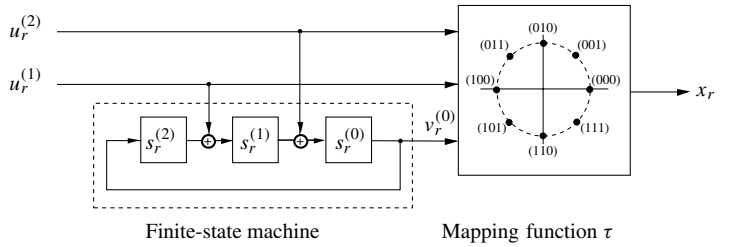


Figure 3.1 Trellis encoder with an 8-state finite-state machine (FSM) driving a 3-bit to 8-PSK signal mapper.

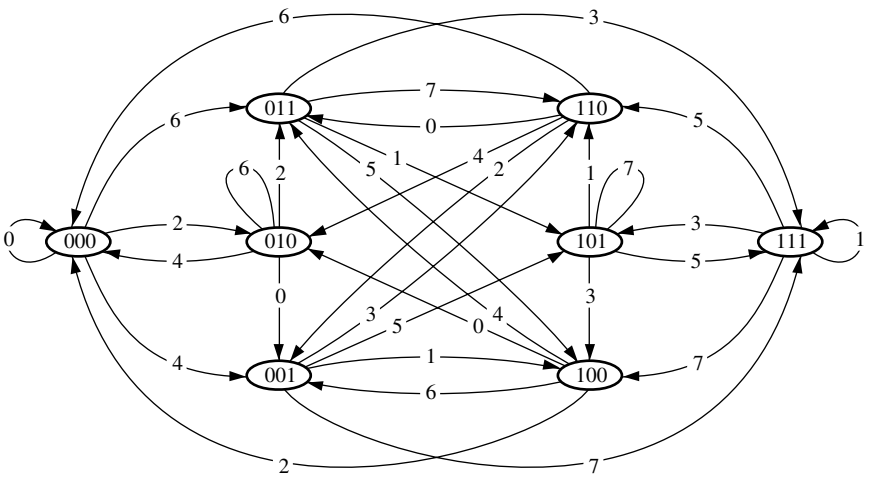


Figure 3.2 State transition diagram of the encoder from Figure 3.1. The labels on the branches are the encoder output signals $x(v)$, in decimal notation.

(possibly) infinite-length sequence of symbols $\boldsymbol{x} = (\ldots, x_{-1}, x_0, x_1, x_2, \ldots)$. Assuming that the encoder is operated in a continuous fashion, there are four choices at each time r, which allows us to transmit two information bits/symbol, the same as with QPSK.

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 $u = (u^{(2)}, u^{(1)})$ which cause the transition, as well as by either the output triple $v = (u^{(2)}, u^{(1)}, v^{(0)})$ or the output signal $x(v)$. (In Figure 3.2 we have used $x(v)$, represented in decimal notation, that is, $x_{\text{dec}}(v) = u^{(2)}2^2 + u^{(1)}2^1 + v^{(0)}2^0$).

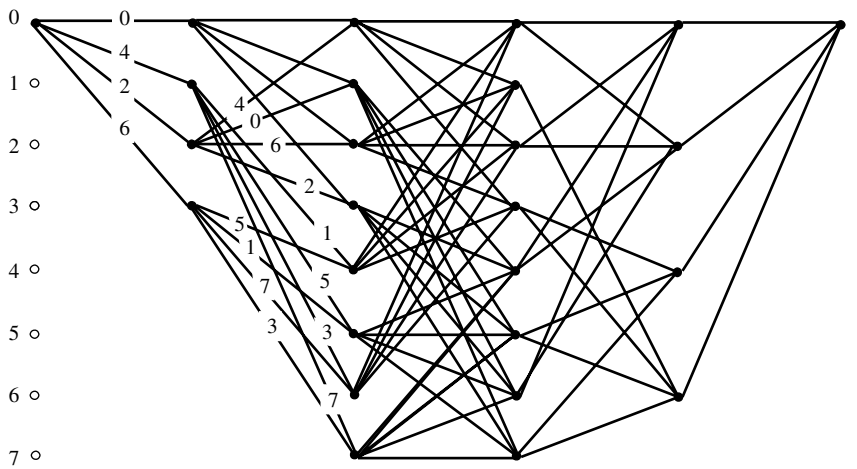


Figure 3.3 Trellis diagram of the encoder from Figure 3.1. The code is terminated to length $L = 5$.

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 infinite, 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.

Now each path through the trellis corresponds to a unique message and is associated with a unique signal sequence (compare Section 2.5). The term *trellis-coded modulation* originates from the fact that these encoded sequences consist of modulated symbols, rather than binary digits.

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 that can be in any given sequence, and these restrictions can be exploited by a smart decoder. In fact, what counts is the distance between signal sequences $\boldsymbol{x}$, 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 $\boldsymbol{x}^{(e)}$ (erroneous) and $\boldsymbol{x}^{(c)}$ (correct). These two sequences differ in the three symbols shown. An optimal decoder will make an

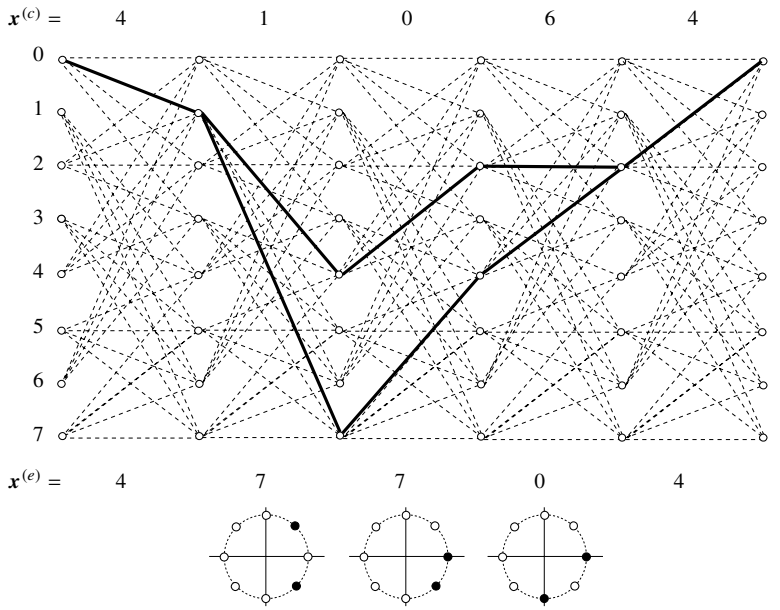


Figure 3.4 Section of the trellis of the encoder in Figure 3.1. The two solid lines depict two possible paths with their associated signal sequences through this trellis. The numbers on top are the signals transmitted if the encoder follows the upper path, and the numbers at the bottom are those on the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

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 $\boldsymbol{x}^{(e)}$ and $\boldsymbol{x}^{(c)}$, which, incidentally, is much larger than the QPSK distance of $d^2 = 2$. Going through all possible sequence pairs $\boldsymbol{x}^{(e)}$ and $\boldsymbol{x}^{(c)}$, one finds that those highlighted in Figure 3.4 have the smallest squared Euclidean distance; 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 doing 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; that is, 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 an entire later chapter (Chapter 6) 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's

share of the work is performed by the decoder, about which we will have to say more later.

3.2 GROUP-TRELLIS CODES

While there exist an infinite number of ways to implement the FSM which introduces the symbol correlation, we will mainly concentrate on the subclass of FSMs that generate a group trellis (to be defined below). This restriction sounds quite severe, but we do this for two reasons. First, by far the largest part of the published literature over the past two decades deals with trellis codes of this kind, and, second, we will show in Chapter 6 that there exist group-trellis codes that can achieve the capacity of an additive white Gaussian noise channel using a time-varying mapping function. It is worth pointing out that there are important examples of trellis codes which do not use a group trellis, such as the nonlinear rotationally invariant codes (to be discussed in Section 3.7).

Let us then start by considering a group trellis. The group[1] property of the class of trellises considered here relates to their topology in the sense that such a trellis "looks" the same for every possible path through the trellis. It is symmetrical in that every path has the same number of neighbor paths of a given length. This concept is most easily captured by assigning different elements from a group $\mathcal{S}$ to different states and assigning elements (not necessarily different) from a group $\mathcal{B}$ to the branches. A trellis is then a group trellis if the paths through the trellis, described by sequences of pairs $(b_0, s_0), (b_1, s_1), \ldots, (b_L, s_L)$, also form a group, where the group operation is the element-wise group operation in the component groups $\mathcal{S}$ and $\mathcal{B}$, denoted by $\oplus$. Our definition follows.

Definition 3.1 *The trellis generated by a finite-state machine is a group trellis, if, for any L, there exists a labeling of paths* $p = (b_0, s_0), \ldots, (b_L, s_L)$, *where* $b_r \in \mathcal{B}$ *and* $s_r \in \mathcal{S}$ *such that* $e = p^{(i)} \oplus p^{(j)} = (b_0^{(i)} \oplus b_0^{(j)}, s_0^{(i)} \oplus s_0^{(j)}), \ldots, (b_L^{(i)} \oplus b_L^{(j)}, s_L^{(i)} \oplus s_L^{(j)})$ *is also a path in this trellis. The* $\oplus$*-operation is taken as the appropriate group operation in* $\mathcal{B}$ *or* $\mathcal{S}$, *respectively.*

Note that this definition could have been made simpler by only considering the sequences of states which also uniquely determines a path through the trellis. The problem with such a definition is that the mapper function can no longer be

[1] A group $\mathcal{G}$ is a set of elements with the following properties:

(i) Closure under a binary group operation (usually called addition) and denoted by $\oplus$; that is, if $g_i, g_j \in \mathcal{G}$, then $g_i \oplus g_j = g_l \in \mathcal{G}$.

(ii) There exists unit element 0 such that $g \oplus 0 = g$.

(iii) Every element g possesses an additive inverse $(-g) \in \mathcal{G}$, such that $g \oplus (-g) = 0$.

(iv) (For Abelian groups) The addition operation commutes: $g_i \oplus g_j = g_j \oplus g_i$.

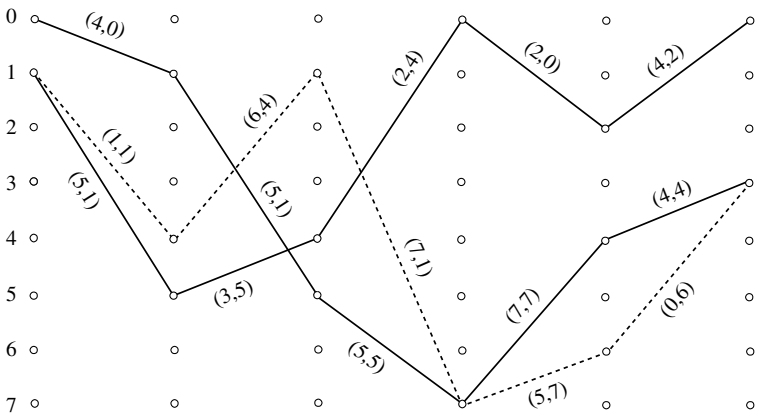


Figure 3.5 Additive path labeling in a group trellis.

made memoryless, since it would depend on pairs of states. However, the labeling $p^{(i)} = (s_1, s_0), (s_2, s_1), (s_3, s_2), \ldots$ is valid and works with a memoryless mapper.

As an example, consider the trellis generated by the FSM of the previous section as illustrated again in Figure 3.5. The state label is the decimal representation of the binary content of the three delay cells—for example, $5 \equiv (101)$—and addition is the bitwise XOR operation (GF(2) addition). The branch labels b are likewise the decimal representation of the two binary input digits, that is, $b = u$. The addition of the two solid paths in Figure 3.5 results in the dotted path which is also a valid path.

All we need is some FSM to generate a group trellis, and that can conveniently be done by a structure like the one in Figure 3.6. In general, a group-trellis encoder can be built as shown in Figure 3.6a for an FSM with 2^{ν} states and two input bits/symbol, where the mapping function is an arbitrary memoryless function $x_r = \tau(u_r, s_r)$ of the current state of the FSM and the new input digits $u_r = (u_r^{(k)}, \ldots, u_r^{(1)})$. Figure 3.6b shows how trellis codes were originally constructed by only using the output bits $(u_r, s_r) = v_r = (v_r^{(n)}, v_r^{(n-1)}, \ldots, v_r^{(0)})$ in the mapping function $x_r = \tau(v_r)$. This is clearly a subclass of the class of general encoders. The reason why the construction of trellis codes was approached in this fashion was the FSM implementation in Fig. 3.6b is in effect a convolutional encoder.[2] Convolutional codes have enjoyed tremendous success in coding for noisy channels. Their popularity has made them the subject of numerous papers and books, and we will discuss convolutional codes in more detail in Chapter 4. For our purpose, however, they mainly serve as generators of the group trellis; that is, if a convolutional code is used to generate the trellis, it will always be a group trellis.

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

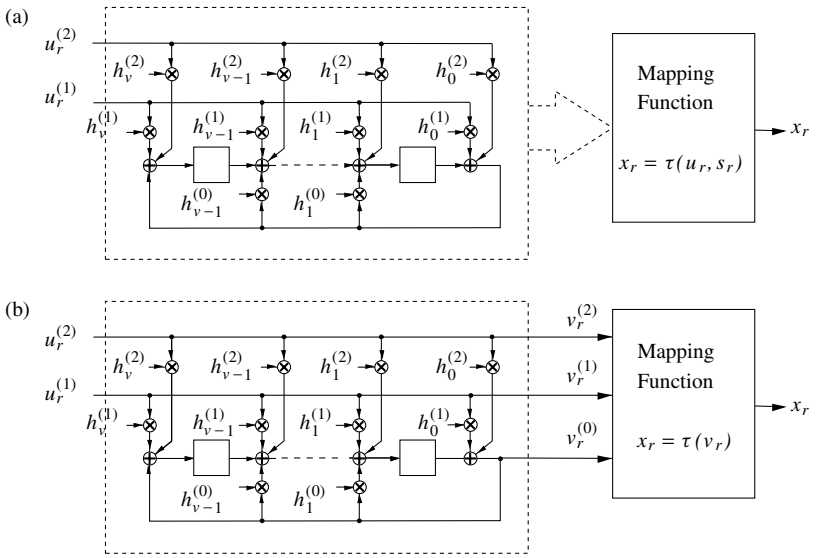


Figure 3.6 General structure of a group-trellis encoder and the structure originally used to construct trellis codes.

3.3 THE MAPPING FUNCTION

So far we have not said much about the mapping function $x_r = \tau(u_r, s_r)$ 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 $x^{(i)} = \tau(b^{(i)}, s^{(i)})$ is generated by the mapping function τ, given as input the branch label $(b^{(i)}, s^{(i)})$. We now wish

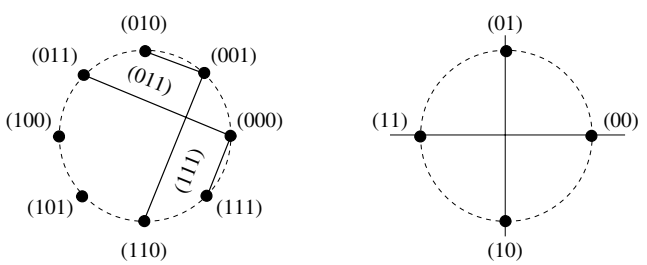


Figure 3.7 8-PSK and QPSK signal sets with natural and Gray mapping.

to explore how the distance between two branch signals $d^2 = \left\|x^{(1)} - x^{(2)}\right\|^2$ is related to the branch labels. If the squared Euclidean distance between $x^{(1)}$ and $x^{(2)}$, given by

$$d^2 = \left\|\tau(b^{(1)}, s^{(1)}) - \tau(b^{(2)}, s^{(2)})\right\|^2, \tag{3.2}$$

depends only on the difference between the branch labels, that is,

$$\begin{aligned} d^2 &= \left\|\tau(b^{(1)}, s^{(1)}) - \tau(b^{(2)}, s^{(2)})\right\|^2 \\ &= \left\|\tau(\underbrace{b^{(1)} \oplus (-b^{(2)})}_{e_b}, \underbrace{s^{(1)} \oplus (-s^{(2)})}_{e_s}) - \tau(0,0)\right\|^2 \\ &= \|\tau(e_b, e_s) - \tau(0,0)\|^2, \end{aligned} \tag{3.3}$$

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 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 counterclockwise fashion. Note that this mapping is not regular, since the binary difference $v = (011)$ can produce the two distances $d^2 = 0.56$ and $d^2 = 3.14$. We will see later (Section 3.8) that no regular mapping exists for an 8-PSK signal set.

If the labels of a group trellis are used to drive a regular mapper, we obtain a *regular* trellis code, which has the property that the set of distances from a given path $p^{(i)}$ to all other paths does not depend on $p^{(i)}$ by virtue of the additivity of the branch labels that are mapped into a regular signal set. In fact, such a code is a generalized group code,[3] also known as a *geometrically uniform* code (Section 3.8). This makes code searches and error performance analysis much easier since we now can concentrate on one arbitrary correct path and know that the distances of possible error paths to it are independent of the actual path chosen. In practice, one usually chooses the path whose labels are the all 0 pairs—that is, the all-zero sequence $(0, 0), \ldots, (0, 0)$.

This simplification is not possible for 8-PSK due to the nonregularity of the mapping. One has an intuitive feeling though that the nonregularity of the 8-PSK signal set is not severe. Let us look again at Equation (3.3), and instead of

[3]A code $\mathcal{C}$ is a group code if there exists a group operation $\oplus$, such that the *codewords* (not only the branch labels) in $\mathcal{C}$ form a group. A regular code is a generalized group code in the following sense: Even though the additivity among codewords (sequences) $\boldsymbol{x}^{(i)}$ does not hold, the weaker, generalized relation $d^2(\boldsymbol{x}(\boldsymbol{v}^{(i)}), \boldsymbol{x}(\boldsymbol{v}^{(j)})) = d^2(\boldsymbol{x}(\boldsymbol{v}^{(i)} \oplus -\boldsymbol{v}^{(j)}), \boldsymbol{x}(\boldsymbol{0}))$ holds, concerning only squared Euclidean distances.

looking at individual distances between pairs, we look at the set of distances as we vary over signal pairs on the branches $(b^{(1)}, s^{(1)})$ and $b^{(1)} \oplus e_b, s^{(2)}$ by going through all branch labels $b^{(1)}$; that is, we look at the set

$$\{d^2\} = \left\{ \left\| \tau(b^{(1)}, s^{(1)}) - \tau(b^{(1)} \oplus e_b, s^{(2)}) \right\|^2 \right\}. \tag{3.4}$$

Considering the 8-PSK constellation with natural mapping, the branch labels $v = (b, s) = (v^{(2)}, v^{(1)}, v^{(0)})$, and the squared Euclidean distance becomes a function of the binary difference e_v between two 3-bit branch labels. Evaluating the sets of distances for all label differences e_v, we find the following:

e_v	$\{d^2\}$
000	{0,0,0,0}
001	{0.56,0.56,0.56,0.56}
010	{2,2,2,2}
011	{0.56,0.56,3.14,3.14}
100	{4,4,4,4}
101	{3.14,3.14,3.14,3.14}
110	{2}
111	{0.56,0.56,3.14,3.14}

We note that these sets[4] of distances do not explicitly depend on the actual states $s^{(1)}$ and $s^{(2)}$, but depend only on the label difference e_v. As we will see in Chapter 6, this is a useful generalization of the notion of regularity, and signal sets with this property are called *quasi-regular*.

Definition 3.2 *A signal set is called quasi-regular if all sets of distances* $\{d^2\} = \{\|\tau(u, s_i) - \tau(u \oplus e_b, s_i \oplus e_s)\|^2\}$ *between the signals* $\tau(u, s_i)$ *and* $\tau(u \oplus e_b, s_i \oplus e_s)$ *from the states* s_i *and* $s_i \oplus e_s$ *are independent of* s_i *as we pass over all pairs of branches* (u, s_i) *and* $(u \oplus e_b, s_i \oplus e_s)$ *by varying* u*, are independent of* s_i*, and depend only on the label difference* (e_b, e_s)*.*

This definition may not sound very useful at the moment, but we will see in Chapter 6 that, since we have to average over all inputs u in order to obtain an average error performance, the independence of the sets of distances from the correct state s_i, together with the additivity of the group trellis, will allow us to substantially reduce the complexity of distance search algorithms and error performance calculations.

Having defined quasi-regular signal sets, we would like to know which signal sets are quasi-regular. This is, in general, not a simple task since we might have

[4]Strictly speaking, $\{0.56, 0.56, 3.14, 3.14\} = \{2 \times 0.56, 2 \times 3.14\}$ is called a family since we also keep multiple occurrences of each entry, but for simplification we will refer to it as a set.

to relabel signal points. However, popular classes of signal sets are quasi-regular, for example:

Theorem 3.1 *An MPSK signal set with natural mapping is quasi-regular for* $M = 2^n$.

Proof: We have already shown that 8-PSK with natural mapping is quasi-regular. That this is the case in general can easily be seen by inspection.

3.4 CONSTRUCTION OF CODES

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 distance, termed d^2_{free}, that is, $d^2_{\text{free}} = \min_{\boldsymbol{x}^{(i)},\boldsymbol{x}^{(j)}} \|\boldsymbol{x}^{(i)} - \boldsymbol{x}^{(j)}\|^2$. A code with a large d^2_{free} is generally expected to perform well, and d^2_{free} has become the major design criterion for trellis codes. (The minimum squared Euclidean distance for the 8-PSK trellis code of Section 3.1 is $d^2_{\text{free}} = 4.586$.)

One heuristic design rule [29], which was used successfully in designing codes with large d^2_{free}, 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. The mapper 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 of their difference. In order to achieve this, we need to make sure that 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 Ungerboeck's [29] 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 $h^{(2)} = h_1^{(2)}, \cdots, h_{\nu-1}^{(2)}$, $h^{(1)} = h_1^{(1)}, \cdots, h_{\nu-1}^{(1)}$ and $h^{(0)} = h_1^{(0)}, \cdots, h_{\nu-1}^{(0)}$ are much harder to find, and one usually resorts to computer search programs or heuristic construction algorithms.

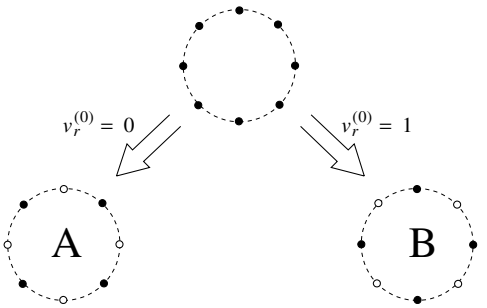


Figure 3.8 8-PSK signal set partitioned into constituent QPSK signal sets.

TABLE 3.1 Connectors, Free Squared Euclidean Distance and Asymptotic Coding Gains of Some Maximum Free Distance 8-PSK Trellis Codes[a]

Number of States	$h^{(0)}$	$h^{(1)}$	$h^{(2)}$	d^2_{free}	$A_{d_{\text{free}}}$	$B_{d_{\text{free}}}$	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

[a]The codes with an * were found by exhaustive computer searches [23, 31], while the other codes were found by various heuristic search and construction methods [23, 24]. The connector polynomials are in octal notation.

Table 3.1 shows the best 8-PSK trellis codes found to date using 8-PSK with natural mapping. The figure gives the connector coefficients, d^2_{free}, the average path multiplicity $A_{d_{\text{free}}}$ of d^2_{free} and the average bit multiplicity $B_{d_{\text{free}}}$ of d^2_{free}. $A_{d_{\text{free}}}$ is the average number of paths at distance d^2_{free}, 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 6. (The connector coefficients are given in octal notation, for example, $h^{(0)} = 23 = 10\,011$, where a 1 means connected and a 0 means disconnected.)

TABLE 3.2 Table of 8-PSK Codes Using a Different Mapping Function [38]

Number of States	$h^{(0)}$	$h^{(1)}$	$h^{(2)}$	d^2_{free}	$A_{d_{\text{free}}}$	$B_{d_{\text{free}}}$
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

From Table 3.1 one can see that an asymptotic coding gain (coding gain for 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 with 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 1000 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 performance, in particular 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 [5] and Zhang [38, 39]. Zhang also used another mapping ((000), (001), (010), (011), (110), (111), (100), (101)) to improve the bit multiplicity. The search criterion employed involved minimizing the bit multiplicities of several spectral lines in the distance spectrum of a code.[5] Table 3.2 gives the best 8-PSK codes found so far with respect to the bit error probability.

If we go to higher-order signal sets such as 16-QAM, 32-cross, 64-QAM, and so on, there are, at some point, not enough states left such that each diverging branch leads to a different state, and we have parallel transitions, that is, two or more branches connecting two states. Naturally we would want to assign signals with large distances to such parallel branches to avoid a high probability of error, since the probability of these 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

[5]The notion of distance spectrum of a trellis code will be introduced and discussed in Chapter 6.

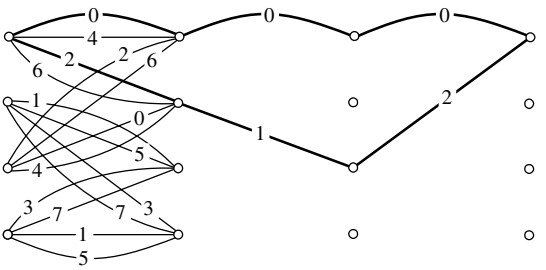


Figure 3.9 Four-state 8-PSK trellis code with parallel transitions.

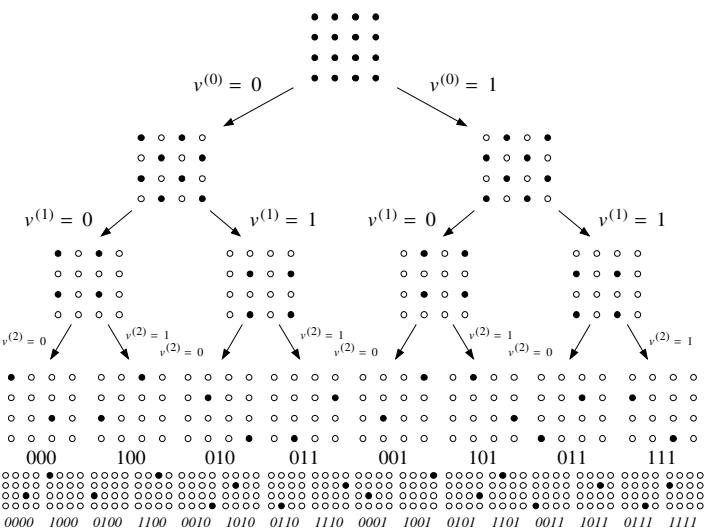


Figure 3.10 Set partitioning of a 16-QAM signal set into subsets with increasing minimum distance. The final partition level used by the encoder in Figure 3.11 is the fourth level—that is, the subsets with two signal points each.

their distance is $d^2 = 4$, which gives the 3-dB asymptotic coding gain of the code over QPSK ($d^2 = 2$).

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

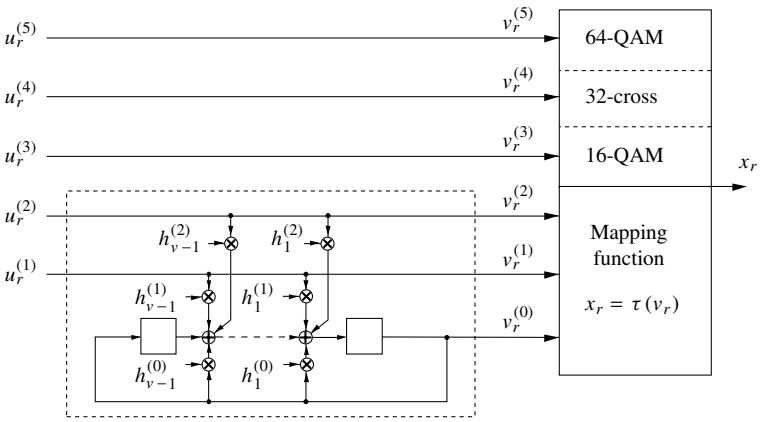


Figure 3.11 Generic encoder for QAM signal constellations.

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, 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 constellations, which are nested versions of each other, such as 16-QAM, 32-cross, 64-QAM, and so on.

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.

As in the case of 8-PSK codes, efforts have been made to improve the distance spectrum of a code—in particular, to minimize the bit error multiplicity of the first few spectral lines. Some of the improved codes using a 16-QAM constellation are listed in Table 3.4 together with the original codes from Table 3.3, which are marked by "Ung". The improved codes, taken from ref. 38, use the signal mapping shown in Figure 3.12 below. Note also that the input line $u_r^{(3)}$ is also fed into the encoder for these codes.

TABLE 3.3 Connectors and Gains of Maximum Free Distance QAM Trellis Codes[a]

Number of States	Connectors $h^{(0)}$	$h^{(1)}$	$h^{(2)}$	d^2_{free}	Asymptotic Coding Gain 16-QAM/ 8-PSK	32-Cross/ 16QAM	64-QAM/ 32-Cross
4	5	2	—	4.0	4.4 dB	3.0 dB	2.8 dB
8	11	2	4	5.0	5.3 dB	4.0 dB	3.8 dB
16	23	4	16	6.0	6.1 dB	4.8 dB	4.6 dB
32	41	6	10	6.0	6.1 dB	4.8 dB	4.6 dB
64	101	16	64	7.0	6.8 dB	5.4 dB	5.2 dB
128	203	14	42	8.0	7.4 dB	6.0 dB	5.8 dB
256	401	56	304	8.0	7.4 dB	6.0 dB	5.8 dB
512	1001	346	510	8.0	7.4 dB	6.0 dB	5.8 dB

[a]The codes in this table were presented by Ungerboeck in ref. 31.

TABLE 3.4 Original 16-QAM Trellis Code and Improved Trellis Codes Using a Nonstandard Mapping

2^ν	$h^{(0)}$	$h^{(1)}$	$h^{(2)}$	$h^{(3)}$	d^2_{free}	$A_{d_{\text{free}}}$	$B_{d_{\text{free}}}$
Ung 8	11	2	4	0	5.0	3.656	18.313
8	13	4	2	6	5.0	3.656	12.344
Ung 16	23	4	16	0	6.0	9.156	53.5
16	25	12	6	14	6.0	9.156	37.594
Ung 32	41	6	10	0	6.0	2.641	16.063
32	47	22	16	34	6.0	2	6
Ung 64	101	16	64	0	7.0	8.422	55.688
64	117	26	74	52	7.0	5.078	21.688
Ung 128	203	14	42	0	8.0	36.16	277.367
128	313	176	154	22	8.0	20.328	100.031
Ung 256	401	56	304	0	8.0	7.613	51.953
256	417	266	40	226	8.0	3.273	16.391

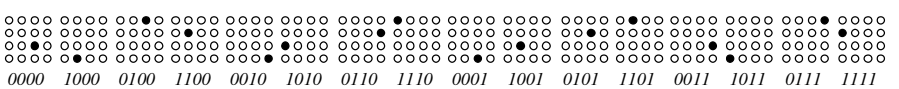


Figure 3.12 Mapping used for the improved 16-QAM codes.

3.5 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 in many situations. This formulation is a particularly convenient tool in the discussion of multidimensional trellis codes. (The two-dimensional complex constellations discussed so far are not considered multidimensional.) We begin by defining a lattice:

Definition 3.3 *An N-dimensional lattice Λ is the set of all points*

$$\Lambda = \{x\} = \{i_1\mathbf{b}_1 + \cdots + i_N\mathbf{b}_N\}, \tag{3.5}$$

where x is an m-dimensional row vector (point) in $\mathbf{R}^m$, $\mathbf{b}_1, \ldots, \mathbf{b}_N$ are N linearly independent basis vectors in $\mathbf{R}^m$, and $i_1, \ldots, i_N$ 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 the sphere packing problem, the covering problem or the quantization problem [4, 14]. 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 ref. 4. 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, then let

$$\mathbf{M} = \begin{bmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \\ \vdots \\ \mathbf{b}_N \end{bmatrix} \tag{3.6}$$

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

$$R = \begin{bmatrix} 1 & -1 & & & & & & \\ 1 & 1 & & & & & & \\ & & 1 & -1 & & & & \\ & & 1 & 1 & & & & \\ & & & & \ddots & \ddots & & \\ & & & & & & 1 & -1 \\ & & & & & & 1 & 1 \end{bmatrix}. \tag{3.7}$$

R 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, namely, 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 that 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; that is, 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 lattice[6] 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, that is, $D_2 = RZ^2$. D_4, the *Schläfli lattice*, is the densest lattice packing in four dimensions. We will return to D_4 in our discussion of multidimensional 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 3.13, 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 constellation. The effect of this shaping is summarized in the shaping gain γ_s.

If we consider the rectangular constellations (see Figure 3.13), the area of the square and the cross is given by $V(Z^2)\ 2^n$, where 2^n is the number of signal

[6]The 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 one-dimensional lattices Z, which D_N is not.

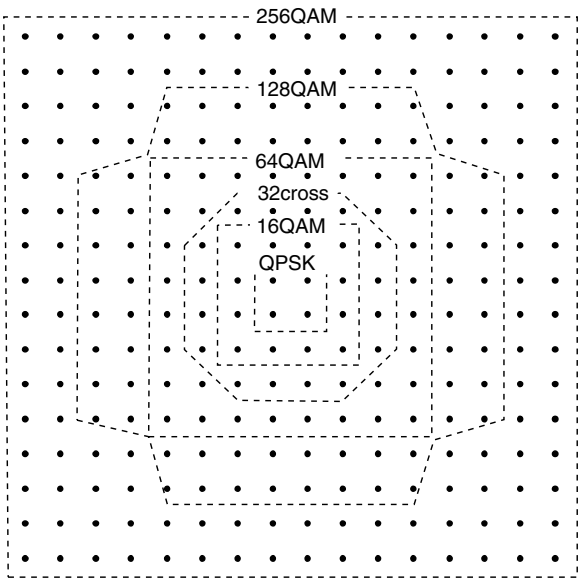


Figure 3.13 The integer lattice Z^2 as a template for the QAM constellations.

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 $E_s = \frac{2}{3}2^n$ for the square and $E_s = \frac{31}{48}2^n$ 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 [8]

$$\gamma_s = \frac{\pi(N+2)}{12}\left(\left(\frac{N}{2}\right)!\right)^{-2/N}, \qquad N \text{ even.} \tag{3.8}$$

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

The fundamental coding gain[7]

$$\gamma(\Lambda) = d_{\min}^2/V(\Lambda)^{2/N} \tag{3.9}$$

[7]The fundamental coding gain is related to the center density δ used in ref. 4 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.

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

$$\gamma(D_N) = \frac{2}{(2V(\Lambda))^{2/N}} = 2^{1-\frac{2}{N}}\gamma(Z^N), \tag{3.10}$$

and $\gamma(Z^N) = 1$ shall be used as reference henceforth. For example, $\gamma(D_4) = 2^{\frac{1}{2}}$; that is, D_4 has a coding gain of $2^{\frac{1}{2}}$ or 1.51 dB over Z^4. The definition (3.9) 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

$$Z^2/RZ^2/2Z^2/2RZ^2 = Z^2/D_2/2Z^2/2D_2. \tag{3.11}$$

Every level of the partition in (3.11) creates a sublattice Λ' of the original lattice Λ as shown in Figure 3.14. The remaining points, if we delete Λ' from Λ, are a shifted version Λ'_s of Λ', called its coset. (There are p cosets in a p-ary partition.) Λ' and Λ'_s together make up Λ, that is, $\Lambda = \Lambda' \bigcup \Lambda'_s$. (For example, $Z^2 = RZ^2 \bigcup (RZ^2 + (1, 0))$.) The partition chain (3.11) generates $2^3 = 8$ cosets of $2RZ^2$ 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 that 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; that is, all points equivalent to $\mathbf{c}$ 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

$$\Lambda = \Lambda' + [\Lambda/\Lambda'], \tag{3.12}$$

where $[\Lambda/\Lambda']$ is just a fancy way of writing the set of all such vectors $\mathbf{c}$. Equation (3.12) is called the coset decomposition of Λ in terms of cosets (translates) of the lattice Λ', and the number of such different cosets is the order of the

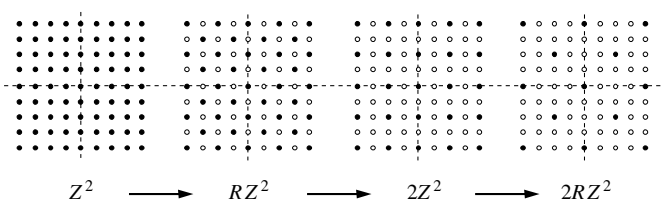


Figure 3.14 Illustration of the binary partition chain (3.11).

partition, denoted by $|\Lambda/\Lambda'|$. As an example, consider the decomposition of Z^2 into $Z^2 = RZ^2 + \{(0, 0), (0, 1)\}$. 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 3.14.

The lattice partitions discussed here are all binary partitions; that is, at each level the lattice is split into two cosets as in (3.11). 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, that is,

$$\Lambda_1/\Lambda_2/\cdots\Lambda_K, \tag{3.13}$$

for which (3.11) may serve as an example of a chain of four 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 $v = (v^{(1)}, \ldots, v^{(K)})$; that is, each coset is given by Λ_K shifted by

$$\mathbf{c}(v) = \sum_{i=1}^{K} v^{(i)}\mathbf{c}^{(i)}, \tag{3.14}$$

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 3.15.) 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 (3.14).

In fact, the partition chain (3.11) 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 3.16. 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

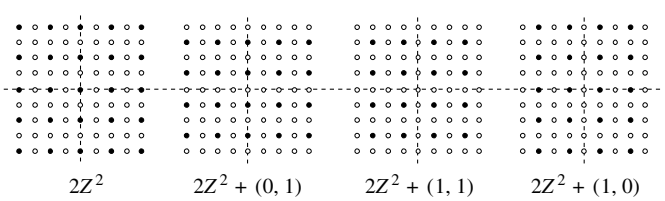


Figure 3.15 The 4 cosets of $2Z^2$ in the partition $Z^2/2Z^2$.

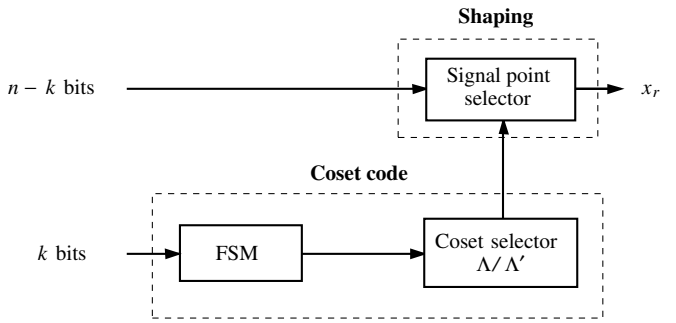


Figure 3.16 Generic trellis encoder block diagram using the lattice notation.

needs to be designed to maximize the minimum distance between sequences of cosets, where the minimum squared 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 point, i.e., $x = 0$).

As shown in Figure 3.16 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*.

3.6 LATTICE FORMULATION OF TRELLIS CODES

A coset code as shown in Figure 3.16 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 $(v_r, v_{r+1}, \ldots)$, that is, 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 that 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, in general, the branch labels of our trellis code are now cosets containing more than one signal point, the minimum distance of the code is $\min(d^2_{\text{free}}, d^2_{\text{min}})$, where d^2_{free} is the minimum free squared Euclidean distance between any two output sequences and d^2_{min} is the minimum squared Euclidean distance between members of Λ', the final sublattice of Λ.

Figure 3.17 shows the generic QAM encoder in lattice formulation, where $v^{(0)}$ selects one of the two cosets of RZ^2 in the partition Z^2/RZ^2, $v^{(1)}$ selects one

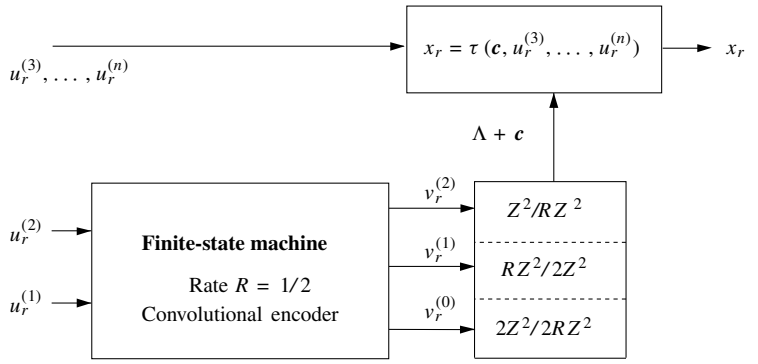


Figure 3.17 Lattice encoder for the two-dimensional rectangular constellations grown from the lattice Z^2.

of the two cosets of $2Z^2$ in the partition $RZ^2/2Z^2$, and $v^{(2)}$ selects one of the two cosets in the partition $2Z^2/2RZ^2$ (compare also Figures 3.10 and 3.11). The final selected coset in the $Z^2/2RZ^2$ partition is given by

$$\boldsymbol{c} = v^{(0)}\boldsymbol{c}^{(0)} + v^{(1)}\boldsymbol{c}^{(1)} + v^{(2)}\boldsymbol{c}^{(2)}, \tag{3.15}$$

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

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; that is, 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 and the use of a group trellis. With the lattice viewpoint we have also introduced an addition operator that allows us to add lattice points or entire lattices. We also note that the lattices themselves are infinite, and there are no boundary problems. We have already remarked that a group code is trivially regular and we do not need to worry about all correct sequences and may choose any convenient one as reference.

Using this new lattice description, trellis codes can now easily be classified. Ungerboeck's original one- and two-dimensional PAM codes are based on the four-way partition $Z/4Z$ and the eight-way partition $Z^2/2RZ^2$, respectively. The one-dimensional codes are used with a rate $R = 1/2$ convolutional code while the two-dimensional codes are used 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 3.5 shows these codes where the two-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

TABLE 3.5 Lattice Partition, Minimum Squared Euclidean Distance, Asymptotic Coding Gain and Number of Nearest Neighbors for the Original Ungerboeck one- and two-Dimensional Trellis Codes

Number of States	Λ	Λ'	$d^2_{\min}$	Asymptotic Coding Gain	N_D
		One-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
		Two-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

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^2_{free} can outperform codes with a larger d^2_{free} but more nearest neighbors.

So far we have only concerned ourselves with two-dimensional lattices as basis for our signal constellations, but in fact, this is an arbitrary restriction and multidimensional trellis codes; that is, trellis codes using lattices of dimensions larger than 2 have been constructed and have a number of advantages. Multidimensional in this context is a theoretical concept, since, in practice, multidimensional signals are transmitted as sequences of one or two-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—that is, 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 four-dimensional signal set, this doubling causes the constituent two-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 two dimensions the signal set expansion costs 3 dB in minimum distance loss, for four-dimensional signals the loss is 1.5 dB, and for eight-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 favor of multidimensional codes is that linear codes with 90° phase invariance can be constructed. This will be explored in more detail in Section 3.7.

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

$$Z^4/D_4/RZ^4/RD_4/2Z^4/2D_4/\cdots \tag{3.16}$$

with minimum distances

$$1/\sqrt{2}/\sqrt{2}/2/2/2\sqrt{2}/\cdots . \tag{3.17}$$

The FSM generating the code trellis, usually a convolutional code, is designed to maximize the intraset 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 binary labels v_r into a distance increment d_i^2, and $d_{\text{free}}^2 = \min_{\boldsymbol{c}(D),\boldsymbol{c}'(D)} \left\|\boldsymbol{c}(D) - \boldsymbol{c}'(D)\right\|^2 = \min_{v(D)} \sum_r d_r^2(v_r)$, where $d_r^2(v_r) = \min_{x\in\Lambda_s(v_r)} \|x\|^2$ is the distance from the origin of the closest point in the coset $\Lambda_s(v_r)$ specified by v_r.

Figure 3.18 illustrates the partitioning tree analogously to Figure 3.10 for the four-dimensional 8-way partition $\Lambda/\Lambda' = Z^4/RD_4$. The cosets at each partition are given as unions of two 2-D cosets; that is, $D_2 \times D_2 \cup \overline{D}_2 \times \overline{D}_2$ 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 coset of D_2.

Figure 3.19 shows a 16-state encoder using a rate $R = 2/3$ convolutional code to generate a trellis with $d_{\text{free}}^2 = d_{\min}^2 = 4$; that is, the parallel transitions are those producing the minimum distance. Since the final lattice $\Lambda' = RD_4$ is a rotated version of the checkerboard lattice D_4, we immediately also know the number of nearest neighbors at $d_{\min}^2 = 4$, which is $N_D = 24$. The addition of the differential encoder allows the code to be made rotationally invariant to 90° phase rotations. This particular encoder will be discussed further in Section 3.7 on rotational invariance.

One philosophy, brought forth by Wei [33], 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 increases the asymptotic coding gain; however,

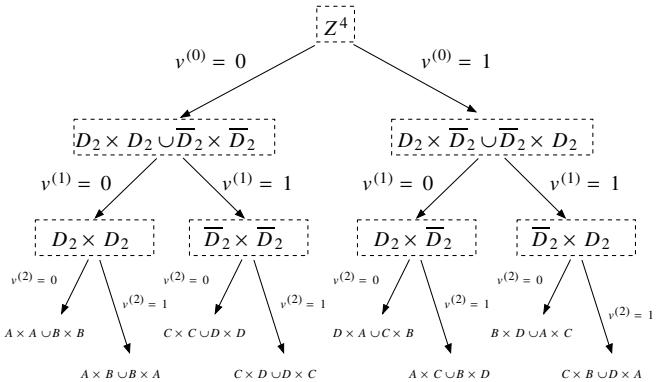


Figure 3.18 Eight-way lattice partition tree of Z_4. The constituent two-dimensional cosets are $A = 2Z^2$, $B = 2Z^2 + (1, 1)$, $C = 2Z^2 + (0, 1)$, and $D = 2Z^2 + (1, 0)$ as depicted in Figure 3.15.

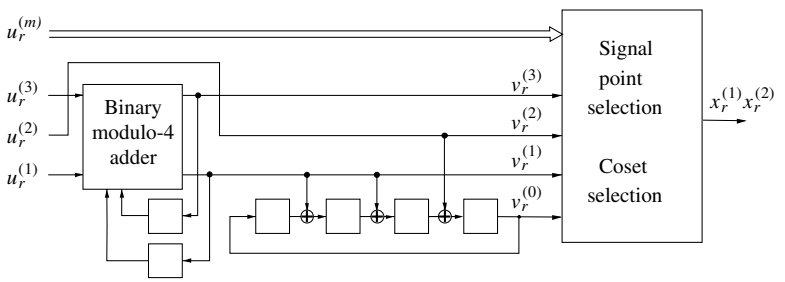


Figure 3.19 Sixteen-state four-dimensional trellis code using the 8-way partition shown in Figure 3.18. The differential encoder on the input bits is used to make the code rotationally invariant.

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.

Table 3.6 summarizes the best-known multidimensional trellis codes found to date. The index in the last column refers to the source of the code. Note that the table also includes eight-dimensional codes, mainly pioneered by Wei [33] and by Calderbank and Sloane [2, 3]. While we now have discussed the construction and classification of multidimensional codes, we have said little about the motivation to use more than two dimensions. Two-dimensional signal constellations seem to be a natural choice since bandpass double-side-band modulated signals naturally generate a pair of closely related dimensions, the in-phase and quadrature channels. The motivation to go to higher-dimensional signal sets is not quite so

TABLE 3.6 Best Multidimensional Trellis Codes Based on Binary Lattice Partitions of 4, 8, and 16 Dimensions[a]

Number of States	Λ	Λ'	$d^2_{\min}$	Asymptotic Coding Gain	N_D	Source
			Four-Dimensional Codes			
8	Z^4	RD_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	RD_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
			Eight-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	RD_8	4	5.27 dB	28	U
32	RD_8	RE_8	8	6.02 dB		W
64	RD_8	RE_8	8	6.02 dB	316	W
128	RD_8	RE_8	8	6.02 dB	124	W
8	E_8	RE_8	8	5.27 dB	764	C-S
16	E_8	RE_8	8	5.27 dB	316	C-S
32	E_8	RE_8	8	5.27 dB	124	C-S
64	E_8	RE_8	8	5.27 dB	60	C-S
			Sixteen-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

[a]The source column indicates the origin of the code; that is, codes marked U were found in ref. 31, those marked W were found in ref. 33, and those marked C-S were found in ref. 3. The Gosset lattice E_8 and the lattice H_{16} are discussed in Chapter 5.

obvious, but there are two very important reasons. The first is the smaller constellation expansion factor discussed previously. The second is that multidimensional codes can be made invariant to phase rotations using linear FSMs.

All the lattices discussed—that is, 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 be reprogrammed to accommodate the lattices without much effort. We simply disallow the signal points in Z^N which are not points in the sublattice in question. Table 3.7 shows these useful sublattices, together with some of their parameters.

TABLE 3.7 Popular Sublattices of Z^N, Their Minimum Distance $d^2_{\min}$, and Their Kissing Number (Nearest Neighbors)

Lattice	Kissing Numbers (Nearest Neighbors)	$d^2_{\min}$	Fundamental Coding Gain $\gamma(\lambda)$
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)	4,320	8	$2^{\frac{3}{2}}$
Λ_{24} (Leech)	196,560	8	4
D_{32} (Barnes–Wall)	208,320	16	4

3.7 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; that is, the received signal, ignoring noise, is given by [Equation (2.32)]

$$s_0(t) = x(t)\sqrt{2}\cos(2\pi f_0 t + \phi). \tag{3.18}$$

Let us assume for purposes of illustration that we are using an MPSK signal set. We can then write (3.18) as

$$s_0(t) = \sqrt{2}\cos(2\pi f_0 t + \phi_m(t) + \phi), \tag{3.19}$$

where $\phi_m(t)$ is the time-varying data phase. For example, for QPSK $\phi_m(t)$ can assume the angles $\pi/2$, π, $3\pi/2$, 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 the received signal to the Mth power. This Mth-order squaring device generates a spectral line $\cos(2\pi Mf_0 + M\phi_m(t) + M\phi)$ at Mf_0. But $M\phi_m(t)$ is always a multiple of 2π for MPSK modulation, and this Mth power spectral line is now free of the data-dependent phase changes, and given by $\cos(2\pi Mf_0 + M\phi)$. A standard PLL can now be 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, and is particularly severe for higher-order constellations [13, 26]. For 8-PSK, for example, this squaring loss with respect to 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 [15] (see also Chapter 12), a concept that has been extended to complete channel estimation [25].

Even with successful phase synchronization, phase offsets by multiples of the 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 has 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 that do not affect decoder operation. This will also assure more rapid resynchronization after temporary signal loss or phase jumps.

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—that is, an angle which rotates the signal constellation into itself. If $\boldsymbol{x}$ is a sequence of coded symbols of a certain trellis code, denote by $\boldsymbol{x}^{\phi}$ the symbol sequence that is obtained by rotating each symbol x_r by the angle ϕ. We now have the following:

Definition 3.4 *A TCM code is rotationally invariant with respect to a rotation by an angle ϕ, if $\boldsymbol{x} \rightarrow \boldsymbol{x}^{\phi}$ is a code sequence for all valid code sequences $\boldsymbol{x}$.*

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; that is, the set of subsets is invariant under these rotations. This latter point is automatically true for one- and two-dimensional constellations [33] (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 [33, 36]:

Theorem 3.2 *A TCM code is rotationally invariant with respect to a rotation by an angle ϕ, if there exists a (bijective) map $f_{\phi} : \mathcal{SP} \mapsto \mathcal{SP}$ with the following*

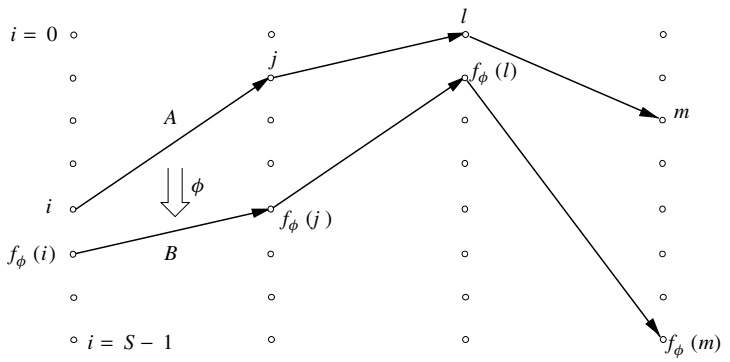


Figure 3.20 Illustration of Theorem 3.2 and its proof.

properties: For each transition from a state i to a state j, denote the associated signal subset by A. Denote by B the subset obtained when A is rotated by the angle ϕ, $A \xrightarrow{\phi} B$. Then B is the subset associated with the transition from state $f_\phi(i)$ to state $f_\phi(j)$.

Proof: The situation in Theorem 3.2 is illustrated in Figure 3.20. It is easy to see that if we concatenate successive trellis sections, for each path $i, j, k, \ldots$, there exists a valid path $f_\phi(i), f_\phi(j), f_\phi(k), \ldots$, 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. First, the code must be rotationally invariant, that is the decoder will still find a valid code sequence after rotation of $\boldsymbol{x}$ by ϕ. Second, the rotated sequence $\boldsymbol{x}^\phi$ 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 group-trellis code in conjunction with two-dimensional signal constellations [16, 20, 21]; hence, nonlinear trellis codes were investigated [34, 35]. Figure 3.21 shows such a nonlinear 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 3.22. This code was adopted in the CCITT V.32 Recommendation [17] and provides an asymptotic coding gain of 4 dB. The three leading uncoded bits that are in plain typeface in Figure 3.22 are not affected by 90° 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 $D_0, \ldots, D_7$. Note that successive 90° phase rotations rotate $D_0 \rightarrow D_1 \rightarrow D_2 \rightarrow D_3 \rightarrow D_0$, 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 nonlinear, rotationally invariant trellis code. This trellis section

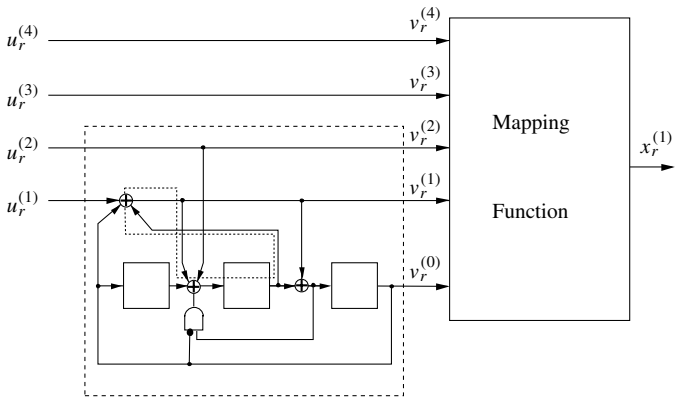


Figure 3.21 Nonlinear rotationally invariant trellis code with eight states for QAM constellations.

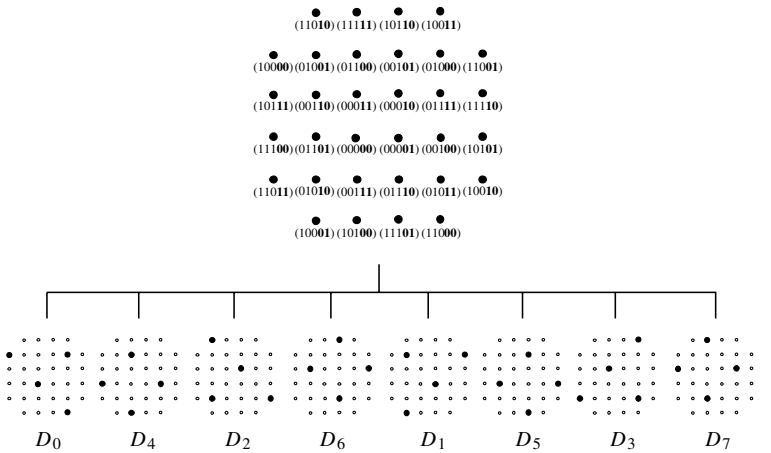


Figure 3.22 Example 32-cross constellation for the nonlinear trellis code of Figure 3.21. The set partitioning of the 32-cross constellation into the eight subsets $D_0, \ldots, D_7$ is also illustrated [31].

is shown in Figure 3.23, together with the state correspondence function $f_\phi(i)$, for successive 90° rotations of the signal constellations (see Theorem 3.2). It is relatively easy to see that if we take an example sequence D_4, D_0, D_3, D_6, D_4 and rotated it by 90° 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 3.2 exists, and the corresponding state relations are illustrated in Figure 3.23.

This then takes care of the rotational invariance of the code with respect to phase rotations of multiples of 90°. However, the bit $v_r^{(1)} = u_r^{(1)}$ changes its

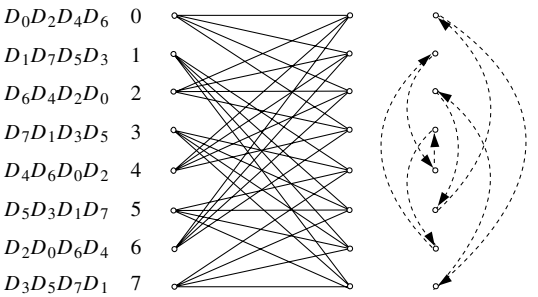


Figure 3.23 Trellis section of the nonlinear eight-state trellis code. The subset labeling is such that the first signal set is the one on the top branch, and the last signal set is on the lowest branch, throughout all the states.

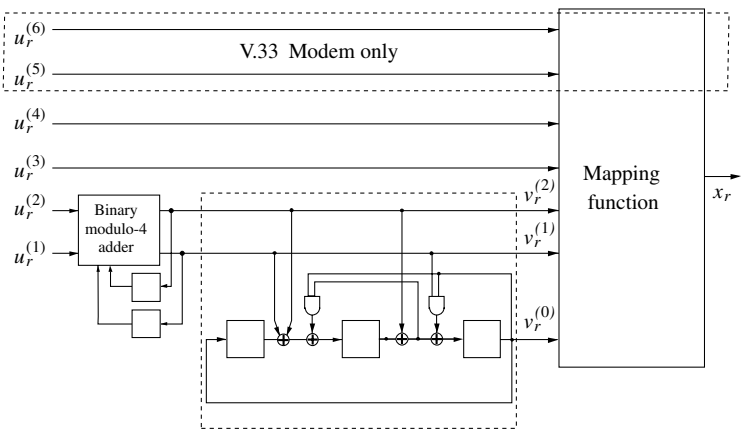


Figure 3.24 Nonlinear eight-state trellis code used in the V.32 and V.33 recommendations.

value through such rotations. This is where the differential encoding comes into play. In Figure 3.21, this function is realized by the XOR gate on the input line $u_r^{(1)}$ and the second delay cell; that is, the differential encoder is integrated into the trellis encoder. This integrated differential encoder is indicated by the dotted loop in Figure 3.21. The coding gain of this code is 4 dB and the number of nearest neighbors is 16 [31].

The actual standards V.32 and V.33 [1] use the trellis code shown in Figure 3.24. The code is an 8-state trellis code with 90° 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 3.25. 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 $\pi/2$.

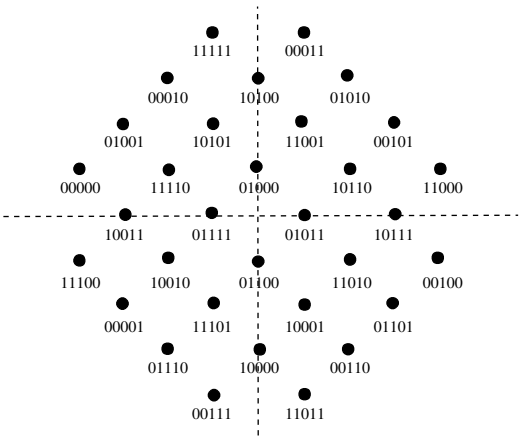


Figure 3.25 The 32-point rotated cross-signal constellation used in the V.32 recommendation for a 9600-bit/s voice-band modem.

The V.33 recommendation allows for data rates of up to 14,400 bits/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 group-trellis codes exist for 2-D constellations that are rotationally invariant, such codes can be found for higher-dimensional signal sets. In fact, the encoder in Figure 3.19 generates such a code. The signal set partition chain for this encoder was given in Figure 3.18, and a 90° phase rotation will cause the following signal set changes[8]: $A \times A \cup B \times B \rightarrow C \times C \cup D \times D \rightarrow A \times A \cup B \times B$, $A \times B \cup B \times A \rightarrow C \times D \cup D \times C \rightarrow A \times B \cup B \times A$, $D \times A \cup C \times B \rightarrow B \times D \cup A \times C \rightarrow D \times A \cup C \times B$, and $A \times C \cup B \times D \rightarrow C \times B \cup D \times A$ (see Figures 3.15 and 3.18). A section of the trellis of this code is shown in Figure 3.26.

The state correspondences are worked out by checking through Theorem 3.2, from where we find that $f_{90^\circ}(0) = 4$, $f_{90^\circ}(4) = 0$, and so on, as indicated in Figure 3.26 by the arrows.

Note that, due to the multidimensional nature of the signal set, 180° phase rotations map the subsets back onto themselves. This is the reason why a linear code suffices, since only one phase ambiguity has to be eliminated. The remaining ambiguities can be taken care of by differential encoding within the subsets.

Wei [36] and Pietrobon and Costello [21] tackle 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.

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

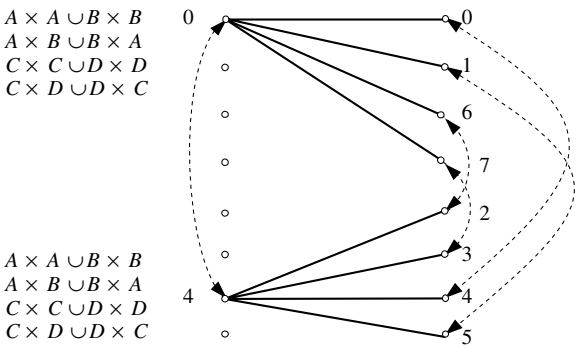


Figure 3.26 Portion of the trellis section of the linear 16-state trellis code, which achieves 90° rotational invariance.

The fastest modems can achieve a data transmission rate of up to 28,800 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. The signal processing techniques that the modems can use adaptively and automatically are selection of the transmission band, selection of the trellis code, constellation shaping, precoding and preemphasis for equalization, and signal set warping. These techniques are summarized in ref. 6. These modems use trellis coding and are described in the standard recommendation V.34, also nicknamed V.fast.

3.8 V.FAST

V.34 [18], or V.fast, is an analog modem standard for full-duplex transmission at rates up to 33.6 kbits/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 and 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 voice-band channels, the V.34 standard supports a large number of different rates, with the highest supportable being negotiated between participating modems at the start of communications session [7, 18]. 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 four-dimensional codes, whose small constellation expansion helps guard against non-Gaussian distortions. Three codes are specified in the standard: a 16-state four-dimensional trellis code [33] with a coding gain of 4.2 dB, a 32-state code [37] with a coding gain of 4.5 dB, and a 64-state code [32] 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 3.5).

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 kbits/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 ref. 12.

Figure 3.27 illustrates the interplay of these different techniques at the transmitter. The differential encoder ensures rotational invariance to multiples of 90° rotations while the shell mapper shapes the signal constellation requiring a constellation expansion of 25%. The four-dimensional trellis code requires a 50% constellation expansion making the total signal constellation expansion 75%.

The precoder and trellis encoder are seemingly switched backwards in this setup, where the trellis encoder determines its next state from the output symbol

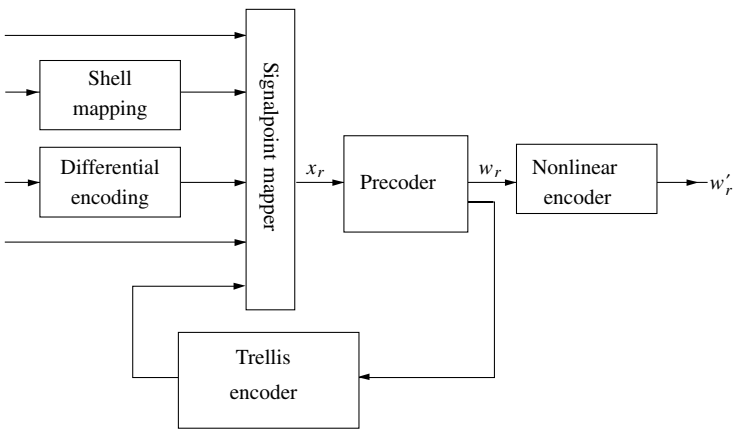


Figure 3.27 Encoder block diagram of a V.fast transmitter.

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 noisy trellis sequence that can be decoded by a standard trellis decoder (Chapter 7). The nonlinear encoder is optional and is designed to counter effects of nonlinear distortion.

V.fast brings together a number of advanced techniques to exploit the information carrying capacity of the telephone voice-band 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 modems in common use today, rely 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 are analog and, not suffering from the quantization noise, have 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 voice-band channel. This is precisely what V.90 does.

3.9 GEOMETRIC UNIFORMITY[9]

The lattice viewpoint of the last sections allowed us to simplify our treatment of trellis codes considerably by making possible the decomposition of the code into a coset code and a signal selector. In this section, we wish to generalize these notions. To do this, we will use the language of geometry. The unique property of the lattice which allowed the elegant treatment of trellis codes was its linearity—that is, the property that the lattice looked identical from every lattice point. Thus appropriate operations such as rotations or translations of the lattice, which moved one signal point into another, left the lattice unchanged. Such operations belong to the class of *isometries*, defined in the following.

Definition 3.5 *An isometry is an operation $T(x)$, where $x, T(x) \in \mathbf{R}^m$, such that*

$$\left\| T(x) - T(x') \right\|^2 = \left\| x - x' \right\|^2 . \tag{3.20}$$

That is, an isometry leaves the Euclidean distance between two points unchanged. Typical examples of isometries are rotations and translations. In fact every isometry in $\mathbf{R}^m$ can be decomposed into a sequence of translations, rotations, and reflections.

[9]This section makes use of some basic group-theoretic concepts, and it can be skipped at a first reading without disrupting continuity.

Isometries are a tool for our treatment of signal sets. Two signal sets S_1 and S_2 are said to be *congruent* if there exists an isometry $T(x)$, which, when applied to S_1, produces S_2, that is, $T(S_1) = S_2$. Furthermore, an isometry, which, when applied to a signal set S, reproduces S is called a *symmetry* of S. Symmetries are therefore isometries which leave a signal set invariant. For example, for a QPSK signal set, the symmetries are the rotations by 0°, 90°, 180°, and 270°, the two reflections by the main axes, and the two reflections about the 45° and the 135° line. These symmetries form a group under composition, called the *symmetry group* of S.

A geometrically uniform signal set is now defined by the following.

Definition 3.6 *A signal set S is geometrically uniform if for any two points x_i, $x_j \in S$, there exists an isometry $T_{i\to j}$ which carries x_i into x_j and leaves the signal set invariant, that is,*

$$\begin{aligned} T_{i\to j}(x_i) &= x_j, \\ T_{i\to j}(S) &= S. \end{aligned} \tag{3.21}$$

Hence a signal set is geometrically uniform if we can take an arbitrary signal point and, through application of isometries $T_{i\to j}$, generate the entire signal constellation S. Another way of saying this is that the symmetry group of S is *transitive*; that is, elements of the symmetry group are sufficient to generate S from an arbitrary starting point.

However, in general, the symmetry group is larger than necessary to generate the signal set. For QPSK, for example, the four rotations by 0°, 90°, 180°, and 270° are sufficient to generate it. Such a subset of the symmetry group is in fact a subgroup, and since it generates the signal set, a transitive subgroup. A transitive subgroup minimally necessary to generate the signal set is also called a *generating group*.

We see the connection to the lattices used earlier. If x_i and x_j are two lattice points in Λ then the translation $T_{i\to j}(\Lambda) = \Lambda + (x_j - x_i)$ is an isometry which carries x_i into x_j but is invariant on the entire lattice.

Geometrically uniform signal sets need not be lattices and Figure 3.28, taken from ref. 11, shows the only four geometrically uniform signal sets in one

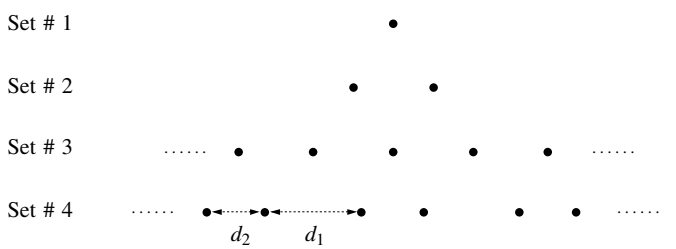


Figure 3.28 The four one-dimensional geometrically uniform signal sets.

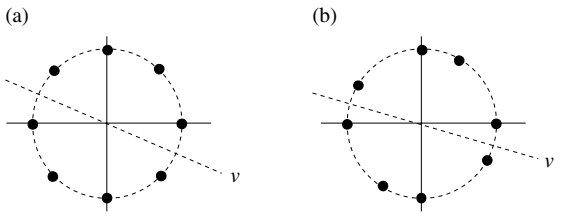


Figure 3.29 Two two-dimensional geometrically uniform constant energy signal sets.

dimension. Of these only set number 3 is a lattice. Loosely speaking, a geometrically uniform signal set is one that "looks" identical from every signal point; that is, it does not depend on which signal point is taken as reference point, and the distances to all other points are independent of this reference. In this sense it is a generalization of a lattice.

Many signal sets used in digital communications are in fact geometrically uniform. There are, for instance, all the lattice-based signal sets if we disregard the restriction to a finite number of signal points. Another class of geometrically uniform signal sets are the MPSK constellations. Figure 3.29 shows an 8-PSK constellation, and it is quite obvious that rotations by multiples of $\pi/4$ leave the signal set unchanged while carrying a given signal point into any other arbitrary signal point. Also shown is an asymmetrical signal set with eight signal points which is also geometrically uniform.

If translations are involved in $T_{i\to j}(S)$, the signal set S must be infinite and periodic (e.g., signal sets number 3 and 4 in Figure 3.28). If only rotations and reflections are involved, the resulting signal points lie on the surface of a sphere. Such signal sets form a *spherical code*.

As mentioned before, a geometrically uniform signal set looks alike from every signal point. This statement can be made precise in a number of ways. The most interesting is that the error probability in additive Gaussian noise is the same for each signal point; that is,

$$P_e = 1 - \int_{I_{x_i}} \frac{1}{(\pi N_0)^{N/2}} \exp\left(-\|n - x_i\|^2 / N_0\right) dn \tag{3.22}$$

is independent of i, where I_{x_i} is the decision region of x_i. This is the case because the decision region, or Voronoi region, has the same shape for each signal point.

Most geometrically uniform signal sets have a generating group.[10] A generating group is obtained from the generating set of isometries of a signal set. A generating set of isometries is a minimal set of isometries which, when applied to an arbitrary initial point x_0, generates all the points of the entire constellation. For example, the set of rotations by the angles $\{0, \pi/4, \pi/2, 3\pi/4, \pi, 5\pi/4, 3\pi/2,$

[10]One would expect that all geometrically uniform signal sets have a generating group, but there exists a counterexample [27].

$7\pi/4\}$, denoted by $\mathcal{R}_8$, generates the 8-PSK signal set of Figure 3.29. Alternatively, the set of rotations by the angles $\{0, \pi/4, \pi/2, 3\pi/4\}$, $\mathcal{R}_4$, together with the reflection V about the axis v, also generates the entire signal set; that is, we have two generating sets of isometries $\mathcal{G}_1 = \mathcal{R}_8$ and $\mathcal{G}_2 = \{V \cdot \mathcal{R}_4\}$. Note that $\mathcal{G}_2$ (but not $\mathcal{G}_1$) also generates the asymmetrical signal set in Figure 3.29. We notice that both sets, $\mathcal{G}_1$ and $\mathcal{G}_2$, have the properties of a group, that is,

(i) They are closed under successive application of any of the isometries in the set. This is true by virtue of the fact that all the isometries in the set are invariant on S, that is, $T_{i\to j}(S) = S$.

(ii) They contain the unit element (rotation by 0).

(iii) They contain an inverse element which is the transformation that carries x_j back into x_i, so that the concatenation of $T_{j\to i}(x_j) \cdot T_{i\to j}(x_i) = x_i$, that is, $T_{j\to i}(x) = T_{i\to j}^{(-1)}(x)$. Such a $T_{j\to i}(x)$ must always exist since the sets we consider are generating sets; that is, they must contain a transformation that takes $x_j \to x_i$. (The initial point can be chosen as x_j.)

From elementary group theory, we know that $\mathcal{G}_1 \equiv Z_8$; that is, $\mathcal{G}_1$ is isomorphic to the additive group of integers modulo 8. We can always find an abstract group which is isomorphic to the generating set $\mathcal{G}$. We call such a group a generating group G.

We will now see how subsets of a signal set S can be associated with subgroups of its generating group G. If G' is a subgroup of G, and we will require that G' is a normal subgroup,[11] then quite obviously G' generates a subset of S. The cosets of G' in G are then associated with the different subsets S' of S. This is expressed by the following theorem, which is the major result in this section:

Theorem 3.3 *Let G' be a normal subgroup of a generating group G of S. Then the subsets of S induced by the cosets of G' in G are geometrically uniform and have G' as a common generating group (with different initial signal point x_0).*

Proof: Consider the subset generated by gG', where $g \in G$. G' generates the subset S' to which we apply the isometry g; that is, $S_g = T_g(G')$ is congruent to S'. Now if G' is normal, then $gG' = G'g$ and $S_g = \bigcup_{g' \in G'} T_{g'}\left(T_g(x_0)\right)$. Therefore S_g is generated by G' taking the initial point $T_g(x_0)$. Q.E.D.

Theorem 3.3 can now be used to generate geometrically uniform signal sets from a given geometrically uniform signal set by studying the normal subgroups of a generating group G. In this manner we have transformed the difficult problem of finding geometrically uniform signal sets into the simpler algebraic problem of finding normal subgroups of the generating group. For example, given

[11] A normal subgroup is one for which the left and right cosets are identical, that is, $gG' = G'g$ for all $g \in G$.

an MPSK signal set and a divisor M' of M, then $\mathcal{R}_{M'}$ is a normal subgroup of $\mathcal{R}_M$.

Or, to use our familiar lattices, if Λ' is a sublattice of Λ, the set of translations generating Λ' is isomorphic to G', which is a normal subgroup of G, the group associated with the set of translations generating Λ. For example, Z^2 is generated by all translations by (i, j), where $i, j \in \mathbf{Z}$. $2Z^2$ is generated by all translations by $(2i, 2j)$. The original generating group G equals $\mathbf{Z} \times \mathbf{Z}$, and the subgroup G' equals $\mathbf{Z}_e \times \mathbf{Z}_e$, where $\mathbf{Z}_e$ is the group of the even integers.

From group theory we know that a normal subgroup G' of G induces a quotient group G/G', whose elements are the cosets of the subgroup G'. Thus the elements of the quotient group G/G' are associated with the subsets S' of S; that is, there is a one-to-one map between them. If we now want to label these subsets, we may choose as labels the elements of any group isomorphic to G/G'. Let us call such a label group L, and we call a labeling with L an *isometric labeling*.

For example, the subsets of the 8-PSK signal constellation containing only one signal point can be labeled either by $\mathbf{Z}_8$, the group of integers mod 8, or by $\mathbf{Z}_2 \times \mathbf{Z}_4$, that is, by vectors (x, y), where $x \in \mathbf{Z}_2$ and $y \in \mathbf{Z}_4$. These two labelings are illustrated in Figure 3.30.

An isometry $T(x)$ can now be associated with an element $l \in L$; that is, applying T to a point $x(l')$ amounts to adding the labels and selecting $x(l \oplus l')$, that is, $T_l(x(l')) = x(l \oplus l')$, where $\oplus$ is the appropriate group operation.

A binary label group is a label group of the form $L = (\mathbf{Z}_2)^n$, and a labeling with the elements of L is called a *binary isometric labeling*. Binary isometric labelings are particularly interesting, since they allow the mapping of strings of bits of length n into signal points.

The importance of isometric labelings lies in the following theorem.

Theorem 3.4 *An isometric labeling of subsets is regular with respect to the label group L.*

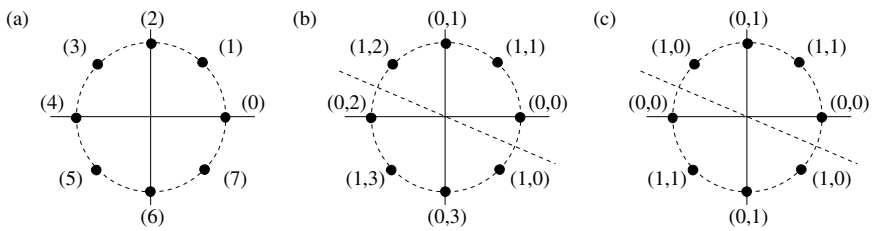


Figure 3.30 (a, b) Two isometric labelings of the 8-PSK signal sets. The first labeling corresponds to rotations by multiples of $\pi/4$; $\mathcal{G}_a = \mathcal{R}_8$. The second labeling corresponds to reflections followed by rotations by multiples of $\pi/2$; $\mathcal{G}_b = V\mathcal{R}_4$. (c) Binary isometric labeling of the 2-PSK *subsets* generated by the subgroup $V\mathcal{R}_2$.

Proof: Let $l_1, l_2 \in L$ and let $x(l_1)$, $x(l_2)$ be their corresponding signal points. Consider $d = \|x(l_1) - x(l_2)\|$. Assume that the signal point $x(0)$ is the one labeled by the unit element (0) of L. Let $T(x(l_2)) = x(0)$ be the isometry that takes $x(l_2) \to x(0)$. Since $T(x)$ is associated with $(-l_2)$, $d = \|x(l_1) - x(l_2)\| = \|T(x(l_1)) - T(x(l_2))\| = \|x(l_1 \oplus (-l_2)) - x(0)\|$ depends only on the label difference $l_1 \oplus (-l_2)$. Q.E.D.

Earlier on we defined regularity as the property that the distance between two signal points depends only on the difference between their labels, where we have used binary label groups (see (3.2)–(3.3)). The above theorem is a straight extension to more general label groups L.

We now understand why our search for a regular 8-PSK mapping using a binary label group $(\mathbf{Z}_2)^3$ was futile. There exists no generating group G for the 8-PSK signal set which is isomorphic to $(\mathbf{Z}_2)^3$.

However, if we look at the subsets containing two signal points generated by $\mathbf{Z}_2$ (rotation by π), the quotient group $G/G' = (V\mathcal{R}_4)/\mathcal{R}_2$ is isomorphic to $(\mathbf{Z}_2)^2$ and there exists an isometric labeling of such 2-PSK subsets, shown in Figure 3.30, where the labels are $(x, y') = (x, (y \bmod 2))$ and $L \equiv (\mathbf{Z}_2)^2$.

The labeling used originally for the 16-QAM signal set (Figure 3.10) is not regular. But using $V_v V_w T_1 T_2$ (where V_v and V_w are reflections about v and w, and T_1 and T_2 are the translations whose vectors are shown in Figure 3.31) as a generating set of the coset representatives of the lattice partition $Z^2/4Z^2$, an isometric labeling of the 16-QAM constellation can be found as shown in Figure 3.31.

Theorem 3.4 allows us now to identify regular trellis codes. Namely, if we combine a group trellis with an isometric labeling, such that the group of branch labels forms the label group, we have what we will call a *geometrically uniform trellis code*:

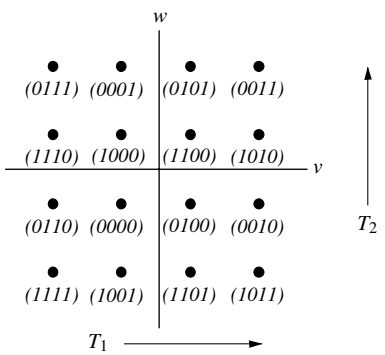


Figure 3.31 Binary isometric labeling of the cosets of $4Z^2$ in the partition $Z^2/4Z^2$. Only the coset leaders are shown which are used as a 16-QAM constellation. The binary isometric labeling is obtained from the generating group which is isomorphic to (V_v, V_w, T_1, T_2).

Definition 3.7 *A geometrically uniform trellis code using a signal constellation*[12] *S is generated by a group trellis, whose branch label group is an isometric labeling of the signal points in S.*

From the above, we immediately have the following.

Theorem 3.5 *A geometrically uniform trellis code is regular; that is, the distance between any two sequences* $d^2(\boldsymbol{x}(\boldsymbol{l}^{(1)}), \boldsymbol{x}(\boldsymbol{l}^{(2)})) = d^2(\boldsymbol{x}(\boldsymbol{l}^{(1)} \oplus (-\boldsymbol{l}^{(2)})), \boldsymbol{x}(\boldsymbol{0}))$, *where* $\boldsymbol{l}^{(1)}$ *and* $\boldsymbol{l}^{(2)}$ *are label sequences.*

The proof of this theorem follows directly from Theorem 3.4 and the additivity property of a group trellis.

While Theorem 3.5 assures that the distance spectrum of a geometrically uniform trellis code is identical for each of the code sequences, more can be said in fact:

Theorem 3.6 *The set of sequences (codewords) of a geometrically uniform trellis code* $\mathcal{C}$ *are geometrically uniform; that is, for any two sequences* $\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)} \in \mathcal{C}$, *there exists an isometry* $T_{i \to j}$ *which carries* $\boldsymbol{x}^{(i)}$ *into* $\boldsymbol{x}^{(j)}$, *and leaves the code* $\mathcal{C}$ *invariant.*

Proof: From Definition 3.7 we know that the sequence $\boldsymbol{x}^{(i)}$ is generated by the label sequence $\boldsymbol{l}^{(i)}$, and $\boldsymbol{x}^{(j)}$ is generated by $\boldsymbol{l}^{(j)}$. Since the generating trellis is a group trellis, $\boldsymbol{l}^{(i-j)} = \boldsymbol{l}^{(i)} \oplus -\boldsymbol{l}^{(j)}$ is also a valid label sequence. We now apply the isometry $T(x(\boldsymbol{l}^{(i-j)}))$ to the code sequence $\boldsymbol{x}^{(i)}$, in the sense that we apply $T(x(l_r^{(i-j)}))$ to $x_r^{(i)}$. By virtue of the branch label group being isomorphic to the generating group G of the signal set S used on the branches, $T(x(l_r^{(i-j)})) = T(x(l_r^{(i)} \oplus -l_r^{(j)})) = T(x(l_r^{(i)})) \cdot T^{(-1)}(x(l_r^{(i)}))$ takes signal $x_r^{(i)}$ to $x_r(0)$ and then to $x_r^{(j)}$. Q.E.D.

Theorem 3.6 is the main theorem of this section, and it states that all code sequences in a geometrically uniform trellis code are congruent, that is, they have the same geometric structure. This includes congruent Voronoi regions, identical error performance over AWGN channels, and regularity, among other properties.

It is now a simple matter to generate geometrically uniform trellis codes. The ingredients are a group code (in our case a group trellis) and a geometrically uniform signal set S, such that the branch label group (or the group of output symbols in the general case) is a generating group of S.

Finally, some examples of geometrically uniform trellis codes are:

(i) All convolutional codes are geometrically uniform.

(ii) Trellis codes using rate 1/2 convolutional codes and QPSK signal constellations with Gray mapping are geometrically uniform. (We need to

[12]Note that this definition could be extended to allow for a different signal constellation S_r at each time interval r in a straightforward way.

use the reflection group V^2 to generate the QPSK set, rather than $\mathcal{R}_4$, hence Gray mapping.)

(iii) The four-state Ungerboeck code from Table 3.1 is geometrically uniform since it uses the labeling from Figure 3.30c. None of the other 8-PSK codes are geometrically uniform—it is the job of quasi-regularity to simplify the error probability calculations.

(iv) The coset codes $\mathcal{C}(\Lambda/\Lambda; C)$ of Section 3.6 are geometrically uniform with a proper binary isometric labeling of the branch label sequences C.

(v) Massey et al. [19] have constructed geometrically uniform trellis codes for MPSK signal sets using $\mathcal{R}_M$ as the label group.

3.10 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 Ungerboeck [28–31] provided the first strong evidence that coding could be used very effectively on band-limited channels. Starting in the 1980s and continuing to this day, numerous researchers have discovered new results and new codes. Multidimensional schemes were first used by Wei [32] and Calderbank and Sloane [2], who also introduced the lattice viewpoint in ref. 3. Much of the material presented in this chapter is adapted from Forney's comprehensive treatments of the subject in refs. 9–11.

BIBLIOGRAPHY

1. Uyless Black, *The V Series Recommendations, Protocols for Data Communications Over the Telephone Network*, McGraw-Hill, New York, 1991.

2. A. R. Calderbank and N. J. A. Sloane, "An eight-dimensional trellis code," *Proc. IEEE,* vol. 74, pp. 757–759, 1986.

3. 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.

4. J. H. Conway and N. J. A. Sloane, *Sphere Packings, Lattices and Groups*, New York, Springer-Verlag, 1988.

5. J. Du and M. Kasahara, "Improvements of the information-bit error rate of trellis code modulation systems," *IEICE, Japan*, vol. E72, pp. 609–614, May 1989.

6. 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.

7. 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.

8. 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.

9. G. D. Forney, "Coset codes—Part I: Introduction and geometrical classification," *IEEE Trans. Inform. Theory,* vol. IT-34, pp. 1123–1151, 1988.

10. G. D. Forney, "Coset codes—Part II: Binary lattices and related codes," *IEEE Trans. Inform. Theory,* vol. IT-34, pp. 1152–1187, 1988.

11. G. D. Forney, "Geometrically uniform codes," *IEEE Trans. Inform. Theory,* vol. IT-37, pp. 1241–1260, 1991.

12. 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.

13. F. M. Gardner, *Phaselock Techniques*, 2nd edition, John Wiley & Sons, New York, 1979.

14. A. Gersho and R. M. Gray, *Vector Quantization and Signal Compression*, Kluwer Academic Publishers, Boston, 1996.

15. 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.

16. 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.

17. 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, Oct. 1983.

18. 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.

19. 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.

20. 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.

21. 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.

22. 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.

23. 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.

24. J. E. Porath and T. Aulin, "Algorithmic construction of trellis codes," *IEEE Trans. Commun.,* vol. COM-41, no. 5, pp. 649–654, May 1993.

25. C. Schlegel and A. Grant, "Differential space-time turbo codes," *IEEE Trans. Inform. Theory*, vol. 49, no. 9, pp. 2298–2306, Sept. 2003.

26. B. Sklar, *Digital Communications: Fundamentals and Applications*, Prentice-Hall, New York, 1988.

27. D. Slepian, "On neighbor distances and symmetry in group codes," *IEEE Trans. Inform. Theory,* vol. IT-17, pp. 630–632, Sep. 1971.

28. G. Ungerboeck and I. Csajka, "On improving data-link performance by increasing channel alphabet and introducing sequence coding," *Int. Sym. Inform. Theory*, Ronneby, Sweden, June 1976.

29. G. Ungerboeck, "Channel coding with multilevel/phase signals," *IEEE Trans. Inform. Theory,* vol. IT-28, no. 1, pp. 55–67, Jan. 1982.

30. G. Ungerboeck, "Trellis-coded modulation with redundant signal sets Part I: Introduction," *IEEE Commun. Mag.,* vol. 25, no. 2, pp. 5–11, Feb. 1987.

31. 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.

32. L. F. Wei, "A new 4D 64-state rate-4/5 trellis code," Cont. D19, ITU-TSG 14, Geneva, Switzerland, Sep. 1993.

33. L. F. Wei, "Trellis-coded modulation with multidimensional constellations," *IEEE Trans. Inform. Theory,* vol. IT-33, pp. 483–501, 1987.

34. 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.

35. 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.

36. L. F. Wei, "Rotationally invariant trellis-coded modulations with multidimensional M-PSK", *IEEE J. Select. Areas Commun.,* vol. SAC-7, no. 9, Dec. 1989.

37. R. G. C. Williams, "A trellis code for V.fast," CCITT V.fast Rapporteur metting, Bath, U.K., Sep. 1992.

38. W. Zhang, "Finite-state machines in communications," Ph.D. thesis, University of South Australia, Australia, 1995.

39. W. Zhang, C. Schlegel, and P. Alexander, "The BER reductions for systematic 8PSK trellis codes by a Gray scrambler," *Proceedings, International Conference on Universal Wireless Access*, Melbourne, Australia, April 1994.

CHAPTER 4

Convolutional Codes

4.1 CONVOLUTIONAL CODES AS BINARY TRELLIS CODES

In Chapter 3 we used convolutional codes as generators of our linear trellis, and the purpose of the finite-state machine was to generate the topological trellis and the linear labeling discussed in Section 3.2. This topological trellis with the linear labeling then served as the foundation for the actual 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 3.

Convolutional codes, on the other hand, have a long history. They were introduced by Elias [6] in 1955. Since then, much theory has evolved to understand convolutional codes. Some excellent references on the subject are refs. 2, 4, 9, 11, and 18. In this chapter, we bring together the main results that have evolved over the years and are specific to convolutional codes. They mainly relate to encoder realizations and classification as well as minimality of finite-state representations. We draw strongly on the elegant summary by Johannesson and Wan [8]. Their work, in turn, mainly draws on the ground-breaking exposition by Forney [5], who presents a rich algebraic structure of convolutional codes and encoders. However, if one is interested mainly in the operational aspects of convolutional codes, this algebraic structure delivers little beyond what was said in Chapter 3 and will be said in Chapters 6 and 7 about the performance and decoding of trellis codes.

We start out by relating convolutional codes to our view of trellis codes. If the trellis mapper function is omitted from the trellis code as shown in Figure 4.1, which is the finite-state machine from Figure 3.1, we have, in fact, a *convolutional encoder* which produces a *convolutional code*—that is, a code which maps a vector $u_r = (u_r^{(2)}, u_r^{(1)})$ of input bits into a vector $v_r = (v_r^{(2)}, v_r^{(1)}, v_r^{(0)})$ 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 see that a convolutional code does not conserve the signal bandwidth, but requires $1/R$ times more bandwidth than uncoded transmission. This is generally the case for traditional error control coding, which was

Trellis and Turbo Coding, by Christian B. Schlegel and Lance C. Pérez
ISBN 0-471-22755-2

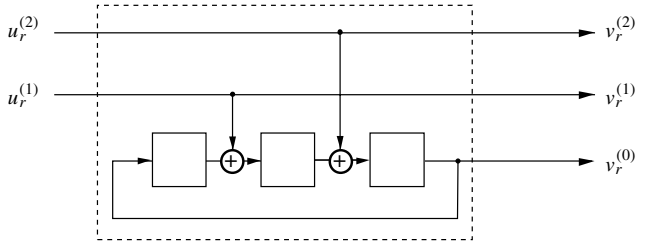


Figure 4.1 Rate $R = 2/3$ convolutional code which was used in Chapter 3, Figure 3.1, to generate an 8-PSK trellis code.

mostly applied to power-limited channels like the deep-space channel, and power efficiency is purchased at the expense of bandwidth, as discussed in Chapter 1.

Convolutional codes are traditionally used with BPSK or QPSK (Gray) mappings. In either case, due to the regularity of the mapper function (Section 3.3), the Euclidean distance between two signal sequences depends only on the Hamming distance $H_d(\boldsymbol{v}^{(1)}, \boldsymbol{v}^{(2)})$ between the two output bit sequences (formerly branch label sequences) $v^{(1)}(D)$ and $v^{(2)}(D)$, 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 Hamming weights, since $H_w(v^{(1)}(D) \oplus v^{(2)}(D)) = H_d(v^{(1)}(D) \oplus v^{(2)}(D), 0) = H_d(v^{(1)}(D), v^{(2)}(D))$.

The convolutional encoder in Figure 4.1 has an alternate "incarnation," which is given in Figure 4.2. The encoder diagram in Figure 4.1 is called the observer canonical form [9], which is also commonly referred to as the systematic feedback realization. The encoder diagram in Figure 4.2 is called the controller canonical nonsystematic 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 4.2, then the controller

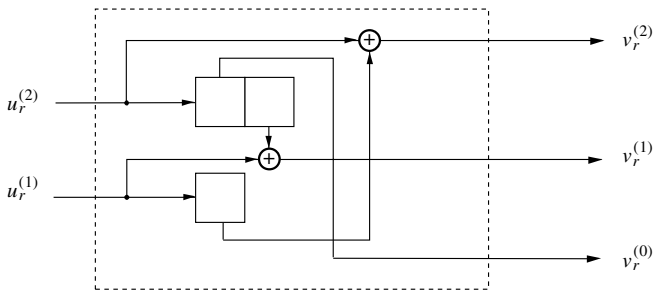


Figure 4.2 Rate $R = 2/3$ convolutional code from above in controller canonical nonsystematic form.

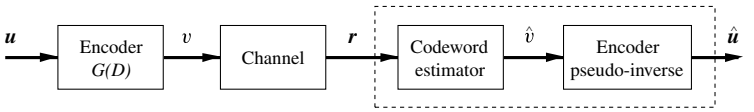


Figure 4.3 Communications system using a convolutional code with the decoder broken up into the codeword estimator and encoder inverse.

canonical form is commonly referred to as the nonsystematic feedforward realization. If feedback is present, the encoder is commonly referred to as the recursive systematic realization. We will see below that equivalent encoders generate the same code, although for a given input bit sequence the different encoders do not generate the same output sequences. In Chapters 10 and 11, we will see that the particular encoder realization chosen for a given code takes on additional significance in parallel and serial concatenated convolutional codes.

We see that convolutional codes are special trellis codes, and everything we said about the latter applies to the former also. The rich structure of convolutional codes, however, makes them an intriguing topic in their own right.

Figure 4.3 shows the place of a convolutional encoder in a communications system: It encodes the input sequence $\boldsymbol{u}$ into the output sequence $\boldsymbol{v}$, which is sent over the channel. This channel includes the modulator and demodulator, as well as the noise process which corrupts the transmitted sequence $\boldsymbol{v} \rightarrow \boldsymbol{r}$. At the receiver side the codeword estimator retrieves the most likely transmitted output sequence $\hat{\boldsymbol{v}}$ and the (pseudo)-inverse of the encoder reproduces the input sequences, or at least the input sequence $\hat{\boldsymbol{u}}$ which corresponds to $\hat{\boldsymbol{v}}$.

The most difficult part of the decoding operation is clearly the codeword estimator, which is a nonlinear operation. We will study the codeword estimator in Chapter 6, and then, particularly, in Chapter 7. This present chapter is mainly concerned with the encoder/encoder inverse, since these are linear functions, and a great deal can be said about them. Often the refinement of the decoder into a codeword estimator and an encoder inverse is not made in the literature and both functions are subsumed into one unit. We call the encoder inverse a pseudo-inverse, since we allow for an arbitrary finite delay to take place in the output sequence $\hat{\boldsymbol{u}}$ with respect to $\boldsymbol{u}$.

4.2 CODES AND ENCODERS

Recall from Section 3.6 the definition of the D-transform

$$u(D) = \sum_{r=s}^{\infty} u_r D^r, \tag{4.1}$$

where $s \in \mathbf{Z}$ guarantees that there are only finitely many negative terms in (4.1). The D-operator can be understood as a unit-delay operator, corresponding to

passing a symbol through one delay element. Formally, (4.1) is a Laurent series with vector coefficients, which we may write as two (k in general) binary Laurent series

$$u(D) = \left(u^{(2)}(D), u^{(1)}(D)\right). \tag{4.2}$$

The binary Laurent series form a field[1] and division is understood to be evaluated by expanding the long division into positive exponents—for example, $(1 + D)/(D^2 + D^3 + D^4) = D^{-2} + 1 + D + D^3 + \cdots$.

We now have enough mathematics to formally define a convolutional code:

Definition 4.1 *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 $v(D) \in F_2^n(D)$, that is,*

$$u(D) \to v(D): \qquad F_2^k(D) \to F_2^n(D). \tag{4.3}$$

From basic linear algebra (see, e.g., ref. 7) we know that any such linear map can be represented by a matrix multiplication; in our case, we write

$$v(D) = u(D)G(D), \tag{4.4}$$

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—that is, 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

$$G_n(D) = D^i G(D), \qquad i \geq 1, \tag{4.5}$$

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.

[1]That 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: $a(D) + b(D) = b(D) + a(D)$. 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: $a(D)\,(b(D) + c(D)) = a(D)b(D) + a(D)c(D)$.

From Definition 4.1 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 4.2 *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 4.2, we can read off quite easily that

$$G_2(D) = \begin{bmatrix} 1 & D^2 & D \\ D & 1 & 0 \end{bmatrix} \tag{4.6}$$

and, upon further examination,[2] that

$$\begin{aligned} v(D) = u(D)G_2(D) &= u(D) \begin{bmatrix} 1 & D^2 \\ D & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & \frac{D}{1+D^3} \\ 0 & 1 & \frac{D^2}{1+D^3} \end{bmatrix} \\ &= u'(D) \begin{bmatrix} 1 & 0 & \frac{D}{1+D^3} \\ 0 & 1 & \frac{D^2}{1+D^3} \end{bmatrix} = u'(D)G_1(D), \end{aligned} \tag{4.7}$$

where $G_1(D)$ is the generator matrix for the encoder shown in Figure 4.1. The set of sequences $\{u(D)\}$ is identical to the set of sequences $\{u'(D)\}$, since

$$T(D) = \begin{bmatrix} 1 & D^2 \\ D & 1 \end{bmatrix} \Longrightarrow T^{-1}(D) = \frac{1}{1+D^3} \begin{bmatrix} 1 & D^2 \\ D & 1 \end{bmatrix} \tag{4.8}$$

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 encoder[3] of Figure 4.1, 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 4.1 and 4.2, 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

[2]Note that all coefficient operations are in the field GF(2); that is, additions are XOR operations, and multiplications are AND operations.

[3]We 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.

amounting to a different choice of bases. Another such choice, or generator, for our code is

$$G_3(D) = \begin{bmatrix} 1+D & 1+D^2 & D \\ D+D^2 & 1+D & 0 \end{bmatrix}. \tag{4.9}$$

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

$$G_3(D) = \begin{bmatrix} 1 & 1 \\ 0 & 1+D \end{bmatrix} G_2(D) = T_3(D)G_2(D), \tag{4.10}$$

and the input sequence

$$u(D) = \left[0, \frac{1}{1+D} = 1 + D + D^2 + \cdots\right] \tag{4.11}$$

generates the output sequence

$$v(D) = \left[0, \frac{1}{1+D}\right] G_3(D) = [D, 1, 0], \tag{4.12}$$

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 4.3 *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

$$T_3^{-1}(D) = \begin{bmatrix} 1 & 1 \\ 0 & 1+D \end{bmatrix}^{-1} = \begin{bmatrix} 1 & \frac{1}{1+D} \\ 0 & \frac{1}{1+D} \end{bmatrix} \tag{4.13}$$

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; that is, 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 inverse[4] $G^{-1}(D)$ 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; that is, they have no negative powers and only a finite number of nonzero 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; that is,

$$G_1^{-1}(D) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix} \tag{4.14}$$

and

$$G_2^{-1}(D) = \begin{bmatrix} 1 & D \\ D & 1+D^2 \\ D^2 & 1+D+D^3 \end{bmatrix} \tag{4.15}$$

and are therefore not catastrophic.

Let us further define the class of basic encoders as follows:

Definition 4.4 *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

$$\nu = \sum_{i=1}^{k} \max_j \left(\deg\left(g_{ij}(D)\right)\right) = \sum_{i=1}^{k} \nu_i . \tag{4.16}$$

Note that $\nu_i = \max_j \left(\deg\left(g_{ij}(D)\right)\right)$, the maximum degree among the polynomials in row i of $G(D)$, corresponds to the maximum number of delay units needed to store the ith input bits $u^{(i)}(D)$ in the controller canonical realization.

Again, we find for $G_2(D)$ that $\nu = 3$; that is, the number of delay elements needed in the controller canonical encoder realization shown in Figure 4.2 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.

[4]Such an inverse must always exist, since we defined a convolutional code to be a *injective* map—that is, one which can be inverted.

Definition 4.5 *A minimal basic encoder is a basic encoder that 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

$$G_4(D) = T_4(D)G_2(D) = \begin{bmatrix} 1 & 1+D+D^2 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & D^2 & D \\ D & 1 & 0 \end{bmatrix} \tag{4.17}$$

$$= \begin{bmatrix} 1+D+D^2+D^3 & 1+D & D \\ D & 1 & 0 \end{bmatrix}, \tag{4.18}$$

whose constraint length $\nu = 4$. Note that since

$$T_4(D)^{-1} = \begin{bmatrix} 1 & 1+D+D^2 \\ 0 & 1 \end{bmatrix}^{-1} = \begin{bmatrix} 1 & 1+D+D^2 \\ 0 & 1 \end{bmatrix}, \tag{4.19}$$

$G_4(D)$ has a polynomial right inverse,

$$G_4^{-1}(D) = G_2^{-1}(D) \begin{bmatrix} 1 & 1+D+D^2 \\ 0 & 1 \end{bmatrix} \tag{4.20}$$

$$= \begin{bmatrix} 1 & D \\ D & 1+D^2 \\ D^2 & 1+D+D^3 \end{bmatrix} \begin{bmatrix} 1 & 1+D+D^2 \\ 0 & 1 \end{bmatrix} \tag{4.21}$$

$$= \begin{bmatrix} 1 & 1+D^2 \\ D & 1+D+D^3 \\ D^2 & 1+D+D^2+D^4 \end{bmatrix}. \tag{4.22}$$

We have seen above that two encoders are equivalent if and only if $T(D)$ has a nonzero determinant and is therefore invertible. We may now strengthen this result for basic encoders in the following.

Theorem 4.1 *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, $\{u(D)G(D)\} = \{u(D)T(D)G'(D)\} = \{u'(D)G'(D)\}$, and $G(D) \equiv G'(D)$.

Conversely, if $G(D) \equiv G'(D)$, then $T^{-1}(D)$ 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; that is, 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 $\{-1, 1\}$, and in $F[D]$ it is only the unit element 1. In the proof of Theorem 4.1, we could also have used the following basic algebraic result [7, (p. 96)].

Theorem 4.2 *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—that is, 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.

4.3 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, that is,

$$G(D) = \tilde{G}(D) + \hat{G}(D) = \tilde{G}(D) + \begin{bmatrix} D^{\nu_1} & & & \\ & D^{\nu_2} & & \\ & & \ddots & \\ & & & D^{\nu_k} \end{bmatrix} G_h, \qquad (4.23)$$

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:

$$G_2(D) = \begin{bmatrix} 1 & 0 & D \\ 0 & 1 & 0 \end{bmatrix} + \begin{bmatrix} D^2 & \\ & D \end{bmatrix} \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}, \qquad (4.24)$$

and

$$G_4(D) = \begin{bmatrix} 1 + D + D^2 & 1 + D & D \\ 0 & 1 & 0 \end{bmatrix} + \begin{bmatrix} D^3 & \\ & D \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}. \qquad (4.25)$$

A minimal basic encoder is then characterized by the following [8].

Theorem 4.3 *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 $\boldsymbol{h}_{i_j}$ of G_h such that

$$\boldsymbol{h}_{i_1} + \boldsymbol{h}_{i_2} + \cdots + \boldsymbol{h}_{i_d} = 0, \tag{4.26}$$

as in (4.25). 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

$$D^{\nu_{i_d}} \left(\boldsymbol{h}_{i_1} + \boldsymbol{h}_{i_2} + \cdots + \boldsymbol{h}_{i_{d-1}} \right) \tag{4.27}$$

to row i_d of $\hat{G}(D)$ reduces it to an all-zero row, and, similarly, adding

$$D^{\nu_{i_d} - \nu_{i_1}} g_{i_1}(D) + D^{\nu_{i_d} - \nu_{i_2}} g_{i_2}(D) + \cdots + D^{\nu_{i_d} - \nu_{i_{d-1}}} g_{i_{d-1}}(D) \tag{4.28}$$

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 4.3 has a constructive proof, there follows a simple algorithm to obtain a minimal basic encoder from any basic encoder, given by:

Step 1: If G_h has full rank, $G(D)$ is a minimal basic encoder and we stop, else

Step 2: Let $\boldsymbol{h}_{i_j}, j \leq k$ be a set of rows of G_h such that

$$\boldsymbol{h}_{i_1} + \boldsymbol{h}_{i_2} + \cdots + \boldsymbol{h}_{i_d} = 0. \tag{4.29}$$

Further, let $g_{i_j}(D)$ be the corresponding rows of $G(D)$, and add

$$D^{\nu_{i_d} - \nu_{i_1}} g_{i_1}(D) + D^{\nu_{i_d} - \nu_{i_2}} g_{i_2}(D) + \cdots D^{\nu_{i_d} - \nu_{i_{d-1}}} g_{i_{d-1}}(D) \tag{4.30}$$

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,

$$G_5(D) = \begin{bmatrix} 1 & D \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & D^2 & D \\ D & 1 & 0 \end{bmatrix} \tag{4.31}$$

$$= \begin{bmatrix} 1 + D^2 & D^2 & D \\ D & 1 & 0 \end{bmatrix} \tag{4.32}$$

is a basic encoder with constraint length $\nu = 3$ equivalent to $G_2(D)$, and both are minimal.

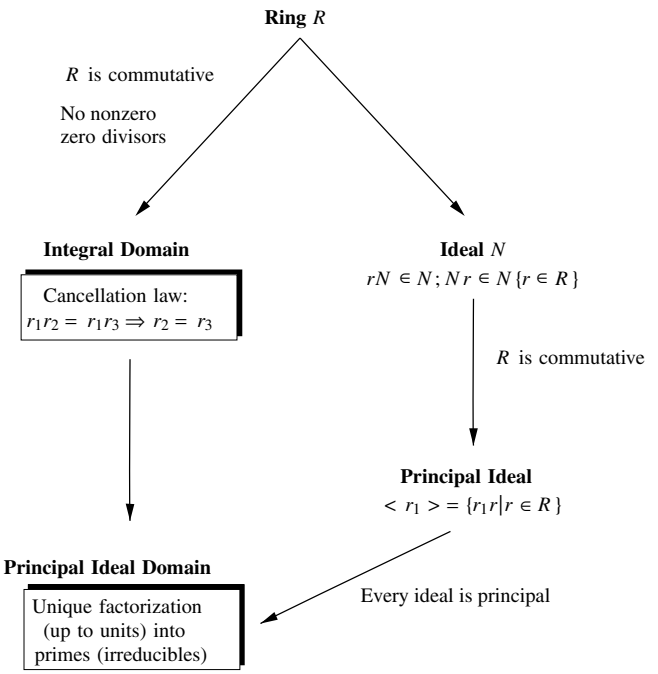


Figure 4.4 Diagram of ring-algebraic concepts. Some examples of rings to which these concepts apply are the integer numbers $\mathbf{Z}$, the integers modulo m, $\mathbf{Z}_m$, and the polynomials in D over the field F, $F[D]$.

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 4.4 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 nonzero zero divisors—that is, there exist no nonzero 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 cancellation 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 $r \in R$, we have $rN \in N$ 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 that is generated by all the multiples of a single element n with R, that is, $< n >= \{rn | r \in R\}$. An example of a principal ideal in $\mathbf{Z}$ is $< 2 >$, 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 4.4 (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., ref. 7, page 181 ff. and ref. 5).

Theorem 4.5 (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*

$$\mathbf{P} = \mathbf{A}\Gamma\mathbf{B}, \tag{4.33}$$

where

$$\Gamma = \begin{bmatrix} \gamma_1 & & & & & & \\ & \gamma_2 & & & 0 & & \\ & & \ddots & & & & \\ & & & \gamma_k & & & \\ & & & & 0 & & \\ & & 0 & & & \ddots & \\ & & & & & & 0 \end{bmatrix}, \tag{4.34}$$

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 $\mathbf{P}$.

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

$$\phi\mathbf{R} = \mathbf{A}\Gamma'\mathbf{B} \Rightarrow \mathbf{R} = \mathbf{A}\Gamma\mathbf{B}, \tag{4.35}$$

where the new entries of Γ 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:

$$G(D) = A(D)\Gamma(D)B(D), \tag{4.36}$$

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, $G(D) \equiv G_b(D)$; that is, $G(D) = A(D)\Gamma(D)G_b(D)$ 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 that is equivalent to the original rational encoding matrix $G(D)$. Hence the following.

Theorem 4.6 *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

$$G(D) = A(D)\Gamma(D)B(D). \tag{4.37}$$

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 presented in Appendix 4.A.

While we now have a pretty good idea about basic encoders and how to obtain a minimal version thereof, the question whether some nonpolynomial 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 that 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 that sets the input sequence to 0 for all non-negative time units, that is, $u(D)P = (\ldots, \boldsymbol{u}_{-2}, \boldsymbol{u}_{-1}, 0, 0, 0, \ldots)$, and let Q be the projection operator which sets the output sequence to zero for all negative time units, that is, $v(D)Q = (\ldots, 0, 0, \boldsymbol{v}_0, \boldsymbol{v}_1, \ldots)$. The abstract state corresponding to an input sequence $u(D)$ is then formally given by

$$v_S(D) = (u(D)PG(D))\,Q. \tag{4.38}$$

Our first result is the following.

Theorem 4.7 *The number of abstract states of a minimal basic encoder is 2^{ν}, equal to the number of physical states.*

Proof: Let

$$s(D) = \left(u^{(k)}_{-\nu_k} D^{-\nu_k} + \cdots + u^{(k)}_{-1} D^{-1}, \ldots, u^{(1)}_{-\nu_1} D^{-\nu_1} + \cdots + u^{(1)}_{-1} D^{-1}\right) \tag{4.39}$$

be a physical state of the basic encoder in its controller canonical form. For example, for the code from Figure 4.2, $v_S(D) = (u^{(2)}_{-2} D^{-2} + u^{(2)}_{-1} D^{-1}, u^{(1)}_{-1} D^{-1})$. 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, that is,

$$v(D)Q = s_1(D)G(D)Q = s_2(D)G(D)Q. \tag{4.40}$$

This is equivalent to

$$(s_1(D) - s_2(D))\,G(D)Q = s(D)G(D)Q = 0; \tag{4.41}$$

that is, there exists a nonzero state $s(D)$ whose corresponding abstract state is 0, according to (4.41). We will now show that the only state that fulfills (4.41) is $s(D) = 0$, and hence $s_1(D) = s_2(D)$. This will prove the theorem by contradiction.

We need to have $v(D) = 0$, that is, $v_i = 0$ for all $i \geq 0$, in order for (4.41) to hold. Now, without loss of generality, assume that

$$\nu_k = \nu_{k-1} = \cdots = \nu_{k-l} > \nu_{k-l-1} \geq \cdots \geq \nu_1. \tag{4.42}$$

The coefficient v_{ν_k} of D^{ν_k} in $v(D)$ is then given by

$$v_{-\nu_k} = \left(0, \ldots, 0, u_{-\nu_k}^{(k-l)}, \ldots, u_{-\nu_k}^{(k)}, \right) G_h(D) = 0, \tag{4.43}$$

but, since $G_h(D)$ has full rank for a minimal basic encoding matrix, we must have $u_{-\nu_k}^{(k)} = u_{-\nu_k}^{(k-l)} = \cdots = u_{-\nu_k}^{(k-l)} = 0$. Continuing by induction, we then prove $v_{-\nu_{k-1}} = 0$, etc., that is, 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 4.6 *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:

$$\begin{aligned} v_S(D) &= u(D)PG(D)Q = u(D)PT(D)G'(D)Q \\ &= u(D)PT(D)\,(P+Q)\,G'(D)Q \\ &= u(D)PT(D)PG'(D)Q + u(D)PT(D)QG'(D)Q, \end{aligned} \tag{4.44}$$

but the first term is an abstract state of $G'(D)$, denoted $v'_S(D)$, and the second term is a codeword, that is,

$$v'(D) = u(D)PT(D)QG'(D)Q \tag{4.45}$$

$$= u(D)PT(D)QG'(D), \tag{4.46}$$

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 that starts at time 0, that is, $u'_j = 0, j < 0$, and since $G'(D)$ must be causal, $v'(D)$ cannot have any nonzero negative components either. Hence the trailing Q-operator in (4.46) is superfluous, and we have the representation given in the following.

Theorem 4.8

$$v_S(D) = v'_S(D) + v'(D), \tag{4.47}$$

where, if $G'(D)$ is a minimal basic encoder, the representation (4.47) is unique.

Proof: To see the uniqueness, assume the contrary, that is,

$$v_{S_{(\mathrm{mb})}}(D) + v(D) = v'_{S_{(\mathrm{mb})}}(D) + v'(D), \tag{4.48}$$

that is,

$$v_{S_{(\mathrm{mb})}}(D) + v'_{S_{(\mathrm{mb})}}(D) = v''_{S_{(\mathrm{mb})}}(D) = v''(D) = v(D) + v'(D), \tag{4.49}$$

and $v''(D)$ is both a codeword and an abstract state $v''_{S_{(\mathrm{mb})}}(D)$ of the minimal basic encoder. But then

$$v''(D) = u''(D)G'(D) \Rightarrow u''(D) = v''(D)[G'(D)]^{-1}, \tag{4.50}$$

and $u''(D)$ is polynomial since $[G'(D)]^{-1}$ is polynomial. Furthermore, since $v''_{S_{(\mathrm{mb})}}(D) = u(D)G'(D)Q$, for some input sequence $u(D)$ with no nonzero terms u_j, $j \geq 0$, and

$$v''_{S_{(\mathrm{mb})}}(D) = u(D)G'(D)Q = u''(D)G'(D), \tag{4.51}$$

we obtain

$$\left(u''(D) + u(D)\right) G'(D)Q = 0. \tag{4.52}$$

Using the same method as in the proof of Theorem 4.7, it can be shown that $\left(u''(D) + u(D)\right) = 0$, which implies $u''(D) = u(D) = 0$, and $v''_{S_{(\mathrm{mb})}} = 0$, leading to a contradiction. This proves the theorem. Q.E.D.

Due to $v_{S_{(\mathrm{mb})}}(D) = v_S(D) + v(D)$ and the uniqueness of (4.47), the map $v_S(D) \rightarrow v_{S_{(\mathrm{mb})}}(D) : v_S(D) = v_{S_{(\mathrm{mb})}}(D) + v'(D)$ is surjective (i.e., onto), and we have our next theorem.

Theorem 4.9 *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 $G_{\mathrm{mb}}(D)$.*

We also immediately notice the following.

Corollary 4.10 *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 4.11 *$G(D)$ 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 *a polynomial right inverse in D^{-1}.*

Proof:

(i) Part (*i*) is obvious from Theorem 4.9.

(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

$$D^{-1}v(D)Q = 0. \tag{4.53}$$

We now show that $u(D)$ must also be polynomial in D^{-1}. Breaking up (4.53) into two components, a past and a future term, we obtain

$$\begin{aligned} D^{-1}v(D)Q = 0 &= D^{-1}u(D)\,(P + Q)\,G(D)Q \\ &= D^{-1}u(D)PG(D)Q + D^{-1}u(D)QG(D)Q, \end{aligned} \tag{4.54}$$

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 (4.49) ff., we conclude that it must be the zero codeword. Since $G(D)$ has full rank, $D^{-1}u(D)QG(D) = 0$ implies

$$D^{-1}u(D)Q = 0, \tag{4.55}$$

that is, $u(D)$ 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}, that is, $u_r = 0$ 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; that is, assume this time that $v(D) = u(D)G(D)$ is polynomial in D. Then

$$v(D) = u(D)PG(D) + u(D)QG(D), \tag{4.56}$$

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

$$u(D)PG(D) = u(D)PG(D)Q, \tag{4.57}$$

is again both a codeword and an abstract state, which implies that it must be the zero codeword, that is, $u(D)PG(D) = 0$. 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 a (pseudo)-inverse which is polynomial in D, such that

$$u(D)D^s = v(D)G^{-1}(D^{-1}), \tag{4.58}$$

and hence, a polynomial $v(D)$ can only generate a polynomial $u(D)$, that is, $u(D)$ 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

$$\begin{aligned} u'(D)G(D)G^{-1}(D^{-1}) &= u(D)G(D)QG^{-1}(D^{-1}), \\ u'(D) &= u(D)G(D)QG^{-1}(D^{-1}). \end{aligned} \tag{4.59}$$

But the left-hand side in (4.59) 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 4.8 and 4.9, 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; that is, we have the following.

Corollary 4.12 *A minimal encoder is not catastrophic.*

Johannesson and Wan pointed out the interesting fact [8] that there are minimal encoders which are not minimal basic. Quoting their example, the basic encoder

$$G(D) = \begin{bmatrix} 1+D & D & 1 \\ 1+D^2+D^3 & 1+D+D^2+D^3 & 0 \end{bmatrix} \tag{4.60}$$

has constraint length $\nu = 4$, but is not minimal basic. In fact

$$G'(D) = \begin{bmatrix} 1+D & D & 1 \\ D^2 & 1 & 1+D+D^2 \end{bmatrix} \tag{4.61}$$

is an equivalent minimal basic encoder with constraint length $\nu' = 3$.

Nevertheless, $G(D)$ has a polynomial right inverse in D^{-1}, given by

$$(G(D^{-1}))^{-1} = \begin{bmatrix} 1+D^{-1}+D^{-2}+D^{-3} & D^{-1} \\ 1+D^{-1}+D^{-3} & D^{-1} \\ D^{-2}+D^{-3} & D^{-1} \end{bmatrix} \tag{4.62}$$

and is therefore, according to Theorem 4.11, a minimal encoder with $2^3 = 8$ abstract states.

4.4 SYSTEMATIC ENCODERS

The encoder in Figure 4.1, whose encoding matrix is

$$G_s(D) = \begin{bmatrix} 1 & 0 & \frac{D}{1+D^3} \\ 0 & 1 & \frac{D^2}{1+D^3} \end{bmatrix} \tag{4.63}$$

is systematic; that is, 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

$$G^{-1}(D) = \big(A(D)\Gamma(D)B(D)\big)^{-1} = B^{-1}(D)\Gamma^{-1}(D)A^{-1}(D), \tag{4.64}$$

and, since $B^{-1}(D)$ and $A^{-1}(D)$ are polynomial, $\Gamma^{-1}(D)$ must be polynomial also. This means that all the invariant factors must be units, that is, $\gamma_i = 1$, 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

$$G_s(D) = T^{-1}(D)G(D) = [\mathbf{I}_k | P(D)], \tag{4.65}$$

where $P(D)$ is a $n-k \times k$ parity check matrix with (possibly) rational entries. For example, for the code from Figure 4.1

$$P(D) = \begin{bmatrix} \frac{D}{1+D^3} \\ \frac{D^2}{1+D^3} \end{bmatrix}. \tag{4.66}$$

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 4.13 *Every systematic encoder for a convolutional code is minimal.*

Proof: The proof of this result comes easily from Theorem 4.11. 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; that is, we find a $k \times k$ submatrix whose determinant is a polynomial which is not a multiple of D. Then simply calculate (4.65) 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 104 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 [15]. Let us take the example of Figure 4.1 again, [Equation (4.63)]. 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 4.5.

We now look at the trellis generated by the systematic encoder, whose first few transitions originating from the zero state are shown in Figure 4.6. We start out by tracing paths from the zero state back to the zero state and map them into the connectors $g^{(j)}_{1,m}$ and $g^{(j)}_{2,m}$. 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 $g^{(2)}_{1,0} = 0, g^{(1)}_{1,0} = 1$ and $g^{(0)}_{1,0} = 0$. The second triple v_1 is generated by setting $g^{(2)}_{1,1} = 1, g^{(1)}_{1,1} = 0$ and $g^{(0)}_{1,1} = 0$. Now we move onto the next path. It

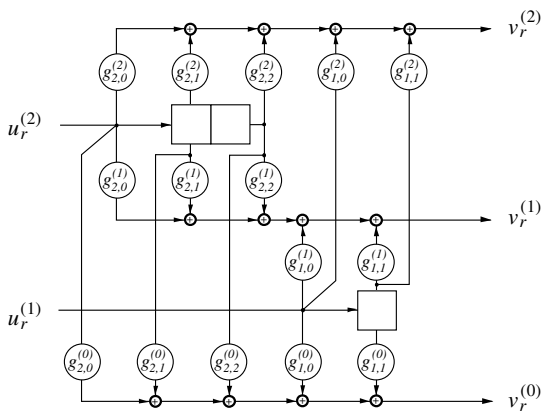


Figure 4.5 Encoder blueprint for the minimal basic encoder equivalent to $G_S(D)$.

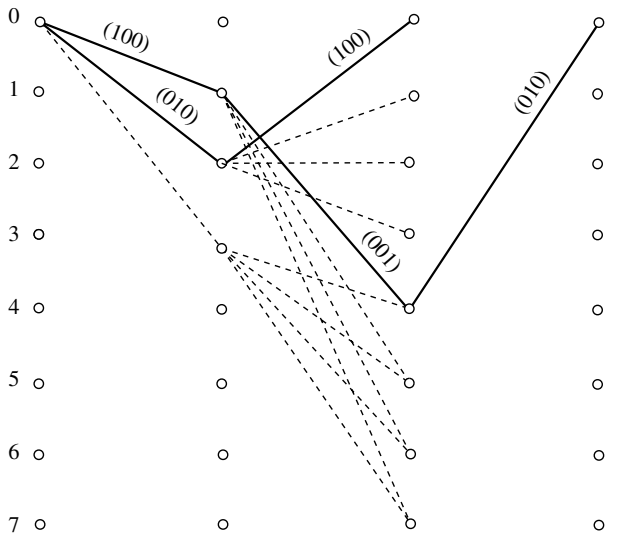


Figure 4.6 Initial trellis section generated by the systematic encoder from Figure 4.1.

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 4.2, 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.

4.5 SYSTEMATIC FEEDBACK AND RECURSIVE SYSTEMATIC ENCODER REALIZATIONS

We now know that every convolutional code has a minimal basic encoder and an equivalent minimal systematic encoder that, in general, has rational entries. For example, the example generator matrix $G_2(D)$ used throughout this chapter is a minimal basic encoder that has the equivalent minimal systematic encoder given by the generator matrix $G_1(D)$. In order to implement a particular generator matrix, it is necessary to develop a corresponding encoder realization—that is, a circuit that performs the matrix multiplication of (4.4), such as those given in Section 4.1. Figure 4.1(a) is a systematic feedback realization of $G_1(D)$ based on the observer canonical form. Figure 4.2 is a nonsystematic feedforward realization

of $G_2(D)$ based on the controller canonical form. It happens that there is a third realization, the recursive systematic realization based on the controller canonical form, that has particular importance in the context of parallel and serial concatenated convolutional codes, see Chapters 10 and 11, and [1].

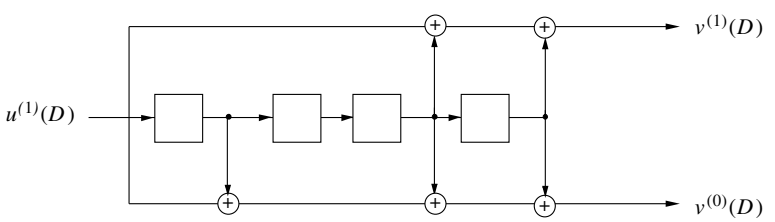
In order to develop this third realization, let us begin with a rate $R = 1/2$ minimal basic encoder

$$G_{FF}(D) = \begin{bmatrix} 1 + D^3 + D^4 & 1 + D + D^3 + D^4 \end{bmatrix} \tag{4.67}$$

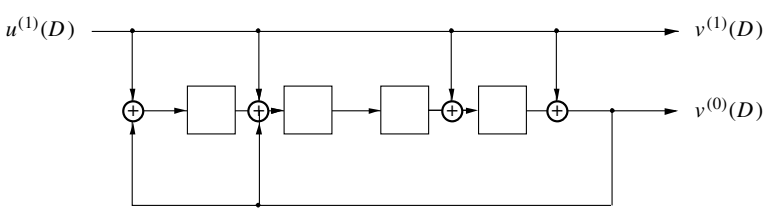
with the obvious nonsystematic feedforward realization shown in Figure 4.7(a). The equivalent systematic encoder has the generator matrix

$$G_{FB}(D) = \begin{bmatrix} 1 & \frac{h_1(D)}{h_0(D)} \end{bmatrix} = \begin{bmatrix} 1 & \frac{1+D+D^3+D^4}{1+D^3+D^4} \end{bmatrix}. \tag{4.68}$$

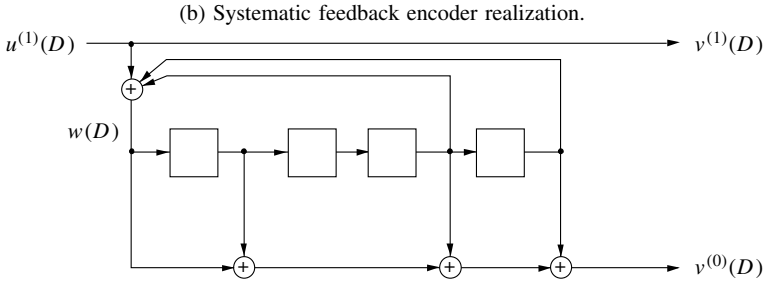
with the obvious systematic feedback realization shown in Figure 4.7(b).



(a) Nonsystematic feedforward encoder realization.



(b) Systematic feedback encoder realization.



(c) Recursive systematic feedback encoder realization.

Figure 4.7 Three encoder realizations for a rate $R = 1/2$ convolutional code.

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

$$(v^{(1)}(D), v^{(0)}(D)) = u^{(1)}(D)G_{FB}(D) = \left(u^{(1)}(D), u^{(1)}(D)\frac{h_1(D)}{h_0(D)}\right). \quad (4.69)$$

Introducing the auxiliary sequence

$$w(D) = \frac{u^{(1)}(D)}{h_0(D)} \quad (4.70)$$

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

$$w(D) = (D^4 + D^3)w(D) + u^{(1)}(D). \quad (4.71)$$

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 4.7(c). The lower part of the recursive systematic realization implements the multiplication by $h_1(D)$ to generate the sequence

$$v^{(0)}(D) = w(D)h_1(D) = \frac{u^{(1)}(D)}{h_0(D)}h_1(D), \quad (4.72)$$

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. That is, they implement the identical mapping from input sequences to output sequences. This then begs the question 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 [9] and, as noted in Section 4.1, the state may be easily set by the input. This feature will be utilized extensively in Chapters 10 and 11.

4.6 MAXIMUM FREE-DISTANCE CONVOLUTIONAL CODES

This chapter has developed many fundamental results concerning 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 [10, 13, 14]. 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^2_{free} 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 equally difficult as finding good trellis codes with large Euclidean free distance. Most often the controller canonical form (Figure 4.2) of an encoder is preferred in these searches; that is, the search 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 $g^{(j)}_{i,m}$, 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 4.1–4.8, were found [3, 10, 11, 16]. They are the rate $R = 1/n$, with $n = 1, 2, \ldots, 8$, 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. However, depending on the application, it may be desirable to optimize other parameters, such as the distance spectrum or the distance profile, or to consider special subclasses of convolutional codes. Extensive tables of a variety of good convolutional codes may be found in Chapter 8 of ref. 9 and in the references therein.

TABLE 4.1 Connectors and Free Hamming Distance of the Best $R = 1/2$ Convolutional Codes [11][a]

ν	$g^{(1)}$	$g^{(0)}$	d_{free}
2	5	7	5
3	15	17	6
4	23	35	7
5	65	57	8
6	133	171	10
7	345	237	10
8	561	753	12
9	1161	1545	12
10	2335	3661	14
11	4335	5723	15
12	10533	17661	16
13	21675	27123	16
14	56721	61713	18
15	111653	145665	19
16	347241	246277	20

[a]The connectors are given in octal notation—for example, $g = 17 = 1111$.

TABLE 4.2 Connectors and Free Hamming Distance of the Best $R = 1/3$ Convolutional Codes [11]

	$g^{(2)}$	$g^{(1)}$	$g^{(0)}$	d_{free}
2	5	7	7	8
3	13	15	17	10
4	25	33	37	12
5	47	53	75	13
6	133	145	175	15
7	225	331	367	16
8	557	663	711	18
9	1117	1365	1633	20
10	2353	2671	3175	22
11	4767	5723	6265	24
12	10533	10675	17661	24
13	21645	35661	37133	26

TABLE 4.3 Connectors and Free Hamming Distance of the Best $R = 2/3$ Convolutional Codes [11]

	$g_2^{(2)}, g_1^{(2)}$	$g_2^{(1)}, g_1^{(1)}$	$g_2^{(0)}, g_1^{(0)}$	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 4.4 Connectors and Free Hamming Distance of the Best $R = 1/4$ Convolutional Codes [10]

	$g^{(3)}$	$g^{(2)}$	$g^{(1)}$	$g^{(0)}$	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 4.5 Connectors and Free Hamming Distance of the Best $R = 1/5$ Convolutional Codes [3]

	$g^{(4)}$	$g^{(3)}$	$g^{(2)}$	$g^{(1)}$	$g^{(0)}$	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 4.6 Connectors and Free Hamming Distance of the Best $R = 1/6$ Convolutional Codes [3]

	$g^{(5)}$	$g^{(4)}$	$g^{(3)}$	$g^{(2)}$	$g^{(1)}$	$g^{(0)}$	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 4.7 Connectors and Free Hamming Distance of the Best $R = 1/7$ Convolutional Codes [3]

	$g^{(6)}$	$g^{(5)}$	$g^{(4)}$	$g^{(3)}$	$g^{(2)}$	$g^{(1)}$	$g^{(0)}$	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 4.8 Connectors and Free Hamming Distance of the Best $R = 1/8$ Convolutional Codes [3]

	$g^{(7)}$	$g^{(6)}$	$g^{(5)}$	$g^{(4)}$	$g^{(3)}$	$g^{(2)}$	$g^{(1)}$	$g^{(0)}$	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

APPENDIX 4.A

We will generate a sequence of matrix operations which turn $\mathbf{P}$ into the diagonal matrix $\mathbf{\Gamma}$ according to Theorem 4.5; see also ref. 7, Section 3.7, or ref. 5.

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.[5] $\mathbf{T}_{ij}(b)$ is invertible in R, since

$$\mathbf{T}_{ij}(b)\mathbf{T}_{ij}(-b) = \big(\mathbf{I} + b\mathbf{E}_{ij}\big)\big(\mathbf{I} + (-b)\mathbf{E}_{ij}\big) = \mathbf{I}. \tag{4.73}$$

Left multiplication of $\mathbf{P}$ by a $k \times k$ matrix $\mathbf{T}_{ij}(b)$ adds b times the jth row to the ith 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 ith column to the jth 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 ith 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 ith row of $\mathbf{P}$ by u, while right multiplication multiplies the ith 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 ith and jth rows of $\mathbf{P}$, while right multiplication interchanges the ith and jth columns, leaving the remainder of the matrix unchanged.

Last, define

$$\mathbf{U} = \begin{bmatrix} x & s & & & & \\ y & t & & & 0 & \\ & & 1 & & & \\ & & & \ddots & & \\ & & 0 & & 1 & \\ & & & & & 1 \end{bmatrix}, \tag{4.74}$$

where x, y, s, and t will be chosen such that the submatrix $\begin{bmatrix} x & s \\ y & t \end{bmatrix}$ is invertible.

Let us start now, and assume $p_{11} \neq 0$ (otherwise bring a nonzero 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 [7], there exist elements $x, y \in R$, such that

$$p_{11}x + p_{12}y = \gcd(p_{11}, p_{12}) = d, \tag{4.75}$$

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 (4.74), which makes it invertible,

[5] It might be helpful to think in terms of integers, that is, $R = \mathbf{Z}$.

that is,

$$\begin{bmatrix} x & p_{12}/d \\ y & -p_{11}/d \end{bmatrix}^{-1} = \begin{bmatrix} p_{11}/d & p_{12}/d \\ y & -x \end{bmatrix}. \tag{4.76}$$

Multiplying $\mathbf{P}$ on the right by this matrix gives a matrix whose first row is $(d, 0, p_{13}, \ldots, p_{1k})$. 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

$$\mathbf{P}'' = \begin{bmatrix} \gamma_1 & 0 \\ 0 & \mathbf{P}''_1 \end{bmatrix}, \tag{4.77}$$

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

$$\mathbf{P}''' = \begin{bmatrix} \gamma_1 & 0 & 0 \\ 0 & \gamma_2 & 0 \\ 0 & 0 & \mathbf{P}'''_2 \end{bmatrix}, \tag{4.78}$$

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 4.5, 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 ref. 7, Section 3.7).

Note that this procedure also provides a proof for Theorem 4.5.

BIBLIOGRAPHY

1. C. Berrou and A. Glavieux, "Near optimum error correcting coding and decoding: turbo-codes," *IEEE Trans. Commun.*, vol. COM-44, pp. 1261–1271, Oct. 1996.
2. G. C. Clark and J. B. Cain, *Error-Correction Coding for Digital Communications,* Plenum Press, New York, 1983.
3. 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.
4. A. Dholakia, *Introduction to Convolutional Codes*, Kluwer Academic Publishers, Boston, 1994.
5. G. D. Forney, "Convolutional codes I: Algebraic structure," *IEEE Trans. Inform. Theory,* vol. IT-16, no. 6, pp. 720–738, Nov. 1970.

6. P. Elias, "Coding for noisy channels," *IRE Conv. Rec.,* pt. 4, pp. 37–47, 1955.

7. N. Jacobson, *Basic Algebra I*, Freeman and Company, New York, 1985.

8. 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.

9. R. Johannesson and K. S. Zigangirov, *Fundamentals of Convolutional Coding*, IEEE Press, New York, 1999.

10. 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.

11. S. Lin and Daniel J. Costello, Jr., *Error Control Coding: Fundamentals and Applications,* Prentice-Hall, Englewood Cliffs, NJ, 1983.

12. J. L. Massey and M. K. Sain, "Inverses of linear sequential circuits," *IEEE Trans. Computers*, vol. C-17, pp. 310–337, April 1968.

13. J. P. Odenwalder, "Optimal decoding of convolutional codes," Ph.D. thesis, University of California, Los Angeles, 1970.

14. 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.

15. J. E. Porath, "Algorithms for converting convolutional codes from feedback to feedforward form and vice versa," *Electron. Lett.*, Vol. 25, no. 15, pp. 1008–1009, July 1989.

16. J. G. Proakis, *Digital Communications*, 3rd edition, McGraw-Hill, New York, 1995.

17. 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.

18. A. J. Viterbi and J. K. Omura, *Principles of Digital Communication and Coding,* McGraw-Hill, New York, 1979.

CHAPTER 5

Link to Block Codes

5.1 PRELIMINARIES

This book is mostly about trellis codes; and while trellis codes have predominantly been used in error control systems for noisy channels, we must not completely ignore block codes. Indeed, most theoretical work in coding theory, starting with Shannon, has been conducted with block codes. As a consequence, the majority of theoretical results known in coding theory have been derived for block codes. Block codes have been used extensively in a number of important applications, most notably in compact disc (CD) storage, digital audio recording (DAT), and high-definition television (HDTV) [16].

Arguably, the single most important result that accounts for the use and popularity of block codes in applications are the algebraic decoding methods for the general class of Goppa codes [30], in particular the Berlekamp–Massey algorithm [3, 27]. 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 in 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 unquantized received symbols (i.e. *soft decision decoding*) cannot be used directly. This may explain why block codes find applications predominantly in systems requiring error correction. Soft-decision decoding of block codes can be done, however, and several algorithms have been proposed [6, 10, 28, 38], but have never found much attention by practitioners of error control coding. More recently, soft-decision decoding of block codes has seen a revival of interest [12–14, 24, 32, 34–37], in particular in connection with the trellis complexity problem of block codes addressed in this chapter.

In Chapter 7 we will discuss a number of powerful decoding techniques that make use of the inherent trellis structure of the codes. But block codes have traditionally been constructed and decoded in algebraic ways, which do not lend

Trellis and Turbo Coding, by Christian B. Schlegel and Lance C. Pérez
ISBN 0-471-22755-2

themselves easily to soft decision decoding. We will see in this chapter that block codes can also be described by a code trellis; therefore, all the decoding algorithms developed for trellis codes find application. We will further see that block codes can be constructed using a trellis approach and that families of long-known block codes reappear in a different light. This trellis view of block codes enabling soft decision, decoding has spurred renewed interest in block codes, and brings block and trellis codes much closer together.

Block codes are extensively discussed in a number of excellent textbooks [4, 7, 23, 30, 31], and the book by MacWilliams and Sloane [31] is arguably the most comprehensive and thorough treatment of block codes to date. It is not the intention of this chapter to give a complete treatise of block codes, but rather to establish the connection between trellis and block codes.

5.2 BLOCK CODE PRIMER

One of the easiest ways to define a linear[1] block code of rate $R = k/n$ is via its *parity-check matrix* $\boldsymbol{H} = [\boldsymbol{h}_1, \boldsymbol{h}_2, \ldots, \boldsymbol{h}_n]$, which is an $n - k \times n$ matrix whose jth column is $\boldsymbol{h}_j$ and has entries $h_{ij} \in \mathrm{GF}(p)$, where $\mathrm{GF}(p)$ is the finite field of the integers modulo p, and p is a prime. In the simplest case the code extends over GF(2)—that is, over {0, 1}—and all operations are modulo 2, or XOR and AND operations.

A linear block code may now be characterized by the following.

Definition 5.1 *A linear block code $\mathcal{C}$ with parity-check matrix $\boldsymbol{H}$ is the set of all n-tuples (vectors), or codewords $\boldsymbol{x}$, such that*

$$\boldsymbol{H}\boldsymbol{x} = \boldsymbol{0}. \tag{5.1}$$

Algebraically, the code words of $\mathcal{C}$ lie in the (right) nullspace of $\boldsymbol{H}$.

For example, the family of single-error correcting binary Hamming codes have as parity-check matrices those $\boldsymbol{H}$ whose columns are all the $2^m - 1$ nonzero 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 $d_{\min} = 3$ [31]. The first such code found by Richard Hamming in 1948 has

$$\boldsymbol{H}_{[7,4]} = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 \end{bmatrix} = [\boldsymbol{h}_1\, \boldsymbol{h}_2\, \boldsymbol{h}_3\, \boldsymbol{h}_4\, \boldsymbol{h}_5\, \boldsymbol{h}_6\, \boldsymbol{h}_7]\,. \tag{5.2}$$

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

[1]We restrict attention to linear block codes since they are the largest and most important class of block codes.

Any parity-check matrix can always be arranged such that $\boldsymbol{H} = [\boldsymbol{A}|\boldsymbol{I}_{n-k}]$ through column and row permutations and linear combinations of rows, where $\boldsymbol{I}_{n-k}$ is the $n-k$ identity matrix and $\boldsymbol{A}$ has dimensions $n-k \times k$. This has already been done in (5.2). From this form we can obtain the systematic $k \times n$ *code generator matrix*

$$\boldsymbol{G} = \left[\boldsymbol{I}_k \middle| \left(-\boldsymbol{A}^T\right)\right] \tag{5.3}$$

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

5.3 TRELLIS DESCRIPTION OF BLOCK CODES

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 [2] in 1974, and later also by Wolf [40] and Massey [26] in 1978. Following the approach in [2, 40] we define the states of the trellis as follows: Let $\boldsymbol{s}_r$ be a vector of size $n-k$ representing the state at time r. Then the state at $r+1$ is defined as

$$\boldsymbol{s}_{r+1} = \boldsymbol{s}_r + x_{r+1}\boldsymbol{h}_{r+1} = \sum_{l=1}^{r} x_l\boldsymbol{h}_l, \tag{5.4}$$

where x_{r+1} runs through all permissible code symbols at time $r+1$. The state at the end of time interval $r+1$, $\boldsymbol{s}_{r+1}$, is calculated from the preceding state $\boldsymbol{s}_r$ according to (5.4). If the states $\boldsymbol{s}_r$ and $\boldsymbol{s}_{r+1}$ 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 (5.4) simply implements the parity-check equation (5.1) in a recursive fashion, since the final zero-state $\boldsymbol{s}_{n+1} = \sum_{l=1}^{n} x_l\boldsymbol{h}_l = \boldsymbol{H}\boldsymbol{x}$ is the complete parity-check equation! Since each intermediate state $\boldsymbol{s}_r = \sum_{l=1}^{r} x_l\boldsymbol{h}_l$ is a vector of length $n-k$ with elements in GF(p), also called the *partial syndrome* at time r, there can be at most p^{n-k} distinct states at time r, since that 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 (5.2). The corresponding trellis is shown in Figure 5.1. The states $\boldsymbol{s}_r$ are labeled as ternary vectors $(\sigma_1, \sigma_2, \sigma_3)^T$, in accordance with (5.4). Note that all the path extensions that lead to states $\boldsymbol{s}_{n+1} \neq (000)^T$ are dashed, since $\boldsymbol{s}_n^{(i)}$ is the final syndrome and must equal $(000)^T$ for $\boldsymbol{x}$ to be a codeword. We therefore force the trellis to terminate in the zero state at time n.

Some further remarks are in order. Contrary to the trellis codes discussed earlier, the trellis of block codes is time-varying; that is, the connections are

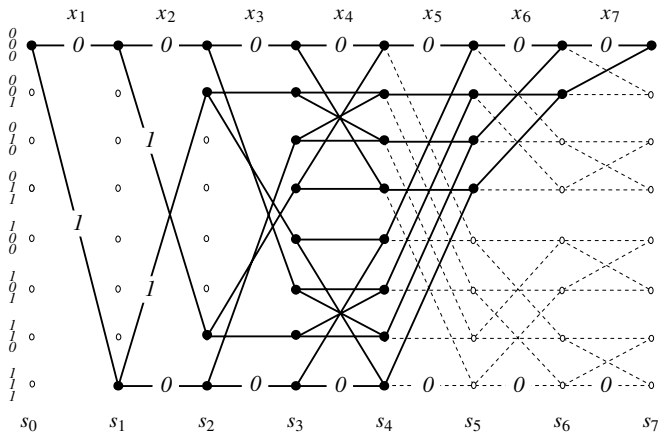


Figure 5.1 Trellis diagram of the [7, 4] Hamming code. The labeling of the trellis is such that a "0" causes a horizontal transition and a "1" causes a sloped transition. This follows from (5.4) since $x_{r+1} = 0$ causes $\boldsymbol{s}_{r+1} = \boldsymbol{s}_r$. Hence, only a few transitions are labeled in the figure.

different for each section of the trellis. The number of states is also varying, with a maximum number of states smaller or equal to p^{n-k}. The codes are regular in the sense of Section 3.3; that is, the error probability is independent of the chosen correct path. This follows directly from the linearity of the block codes considered.

It becomes evident that using a trellis decoding algorithm is quite a different game from algebraic decoding, which operates on the syndrome of the entire code word. There the syndrome for some received symbol sequence $\boldsymbol{y}$, $\boldsymbol{s} = \boldsymbol{H}y$, is calculated, from which the error pattern $\boldsymbol{e}$ is estimated [31]. Then the hypothesized $\hat{\boldsymbol{x}} = \boldsymbol{y} - \boldsymbol{e}$ is generated from $\boldsymbol{y}$ and, in a final step, $\hat{\boldsymbol{u}}$ is decoded from $\hat{\boldsymbol{x}}$. Trellis coding operates differently and we see that, starting at step $r = 4$ in the case of Figure 5.1, hypotheses are being discarded through merger elimination, even before the entire vector $\boldsymbol{y}$ is received. This entails no loss as will be shown rigorously in Chapter 7.

5.4 MINIMAL TRELLISES

In this section we explore the question of optimality of the trellis representation of block codes introduced in the last section. Following the results of McEliece [29], we will show that the parity-check trellis representation (or *PC trellis*) from Section 5.3 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 (5.4) we know that the set of states at time r, denoted by $\mathcal{S}_r$, is given by the mapping $\mathcal{C} \mapsto \mathcal{S}_r$

$$\{\boldsymbol{s}_r(\boldsymbol{x}) : \boldsymbol{s}_r(\boldsymbol{x}) = x_1\boldsymbol{h}_1 + \cdots + x_r\boldsymbol{h}_r\}. \tag{5.5}$$

This mapping is linear due to the linearity of the code and may not be one-to-one. $\boldsymbol{s}_r$ is a vector space with maximum dimension $n-k$, since this is the dimension of the vectors $\boldsymbol{s}_r(x)$, but is, in general, much less (see, e.g., Figure 5.1). We denote its dimension by σ_r, that is, $\sigma_r = \dim \mathcal{S}_r$. 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 with which it is labeled. Formally we may describe the edge space $\mathcal{B}_{r,r+1}$ at time $r+1$ by

$$\left\{\boldsymbol{b}_{r,r+1}(\boldsymbol{x}) : \boldsymbol{b}_{r,r+1}(\boldsymbol{x}) = (\boldsymbol{s}_r(\boldsymbol{x}), \boldsymbol{s}_{r+1}(\boldsymbol{x}), x_{r+1})\,;\, s_{r+1}(x) = s_r(x) + x_{r+1}b_{r+1}\right\}. \tag{5.6}$$

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

Now, since code words are formed via $\boldsymbol{x} = \boldsymbol{G}^T\boldsymbol{u}$,

$$\boldsymbol{s}_r(\boldsymbol{x}) = \boldsymbol{H}_r(x_1, \ldots, x_r) = \boldsymbol{H}_r\boldsymbol{G}_r^T\boldsymbol{u}, \tag{5.7}$$

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

Lemma 5.1 *The state space $\mathcal{S}_r$ at time r is the column space of $\boldsymbol{H}_r\boldsymbol{G}_r^T$, and, consequently, its dimension σ_r is the rank of the matrix $\boldsymbol{H}_r\boldsymbol{G}_r^T$.*

Note that we can of course span the trellis up backwards from the end and then

$$\boldsymbol{s}_r(\boldsymbol{x}) = \overline{\boldsymbol{H}}_r(x_{r+1}, \ldots, x_n)^T = \overline{\boldsymbol{H}}_r\overline{\boldsymbol{G}}_r^T\boldsymbol{u}, \tag{5.8}$$

where $\overline{\boldsymbol{H}}_r$ and $\overline{\boldsymbol{G}}_r$ are matrices made up of the last $n-r$ columns of $\boldsymbol{H}$ and $\boldsymbol{G}$, respectively. The dimensions of $\overline{\boldsymbol{H}}_r\overline{\boldsymbol{G}}_r^T$ are also $n-k \times k$. 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 the following from (5.7), (5.8), and the definitions of $\boldsymbol{H}_r, \boldsymbol{G}_r, \overline{\boldsymbol{H}}_r$, and $\overline{\boldsymbol{G}}_r$.

Lemma 5.2 *The dimension σ_r of the state space $\mathcal{S}_r$ at time r is bounded by*

$$\sigma_r \leq \min(r, n-r, k, n-k). \tag{5.9}$$

Lemma 5.2 can be seen nicely in Figure 5.1.

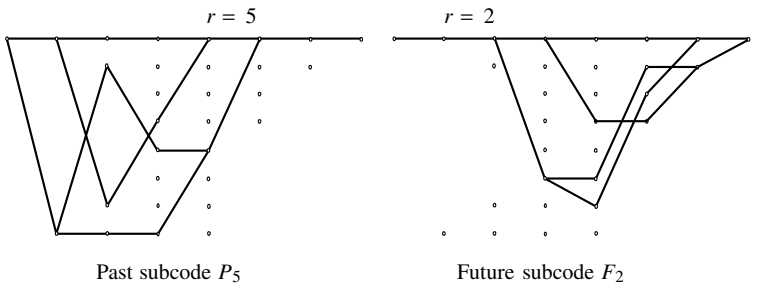


Figure 5.2 Trellis diagram of the past subcode P_5 and the future subcode F_2 of the [7, 4] Hamming code whose full trellis is shown in Figure 5.1.

Now, consider all the code words 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 5.2 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 5.2. Clearly, both of these subcodes are linear subcodes of the original code; that is, they are subgroups of the original group (code) $\mathcal{C}$.

Now, mathematically, $\boldsymbol{s}_r(\boldsymbol{x})$ is a linear map $\mathcal{C} \mapsto \mathcal{S}_r$ from the code space $\mathcal{C}$, which is k-dimensional, onto the state space $\mathcal{S}_r$ at time r, which is σ_r-dimensional. Such a linear map has a kernel, which is the set of codewords which are mapped to the state $\boldsymbol{s}_r(\boldsymbol{x}) = 0$ (i.e., all the codewords whose trellis paths pass through the zero state at time r). From Figure 5.2 we now guess the following.

Lemma 5.3 *The kernel of the map $\boldsymbol{s}_r(\boldsymbol{x})$ is the sum of the past and future subcodes P_r and F_r, that is,*

$$\text{Ker}(\boldsymbol{s}_r) = P_r \oplus F_r. \tag{5.10}$$

The kernel $\text{Ker}(\boldsymbol{s}_r)$ is illustrated in Figure 5.3.

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

Conversely assume $\boldsymbol{s}_r(\boldsymbol{x}) = 0$ and let $\boldsymbol{x} = \boldsymbol{x}^{(p)} + \boldsymbol{x}^{(f)}$ again, where we choose $\boldsymbol{x}^{(p)} = (x_1, \ldots, x_r, 0, \ldots, 0)$ and $\boldsymbol{x}^{(f)} = (0, \ldots, 0, x_{r+1}, \ldots, x_n)$. But due to $\boldsymbol{s}_r(\boldsymbol{x}) = 0$ and (5.7), we conclude that $\boldsymbol{H}\boldsymbol{x}^{(p)} = 0$ and therefore $\boldsymbol{x}^{(p)} \in \mathcal{C}$ and $\boldsymbol{x}^{(p)} \in P_r$. Likewise, applying (5.8) gives $\boldsymbol{H}\boldsymbol{x}^{(f)} = 0$ and $\boldsymbol{x}^{(f)} \in \mathcal{C}$ and $\boldsymbol{x}^{(f)} \in F_r$. Hence $\text{Ker}(\boldsymbol{s}_r) \subseteq P_r \oplus F_r$ also. Q.E.D.

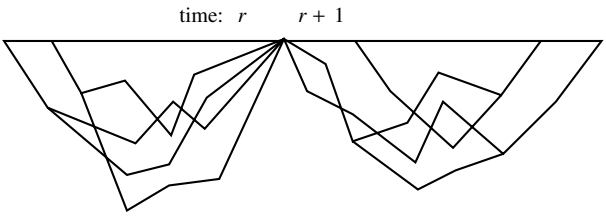


Figure 5.3 Illustration of the kernel of the map $\mathcal{C} \mapsto \mathcal{S}_r$ as the set of codewords which pass through $\boldsymbol{s}_r(\boldsymbol{x}) = 0$.

As is the case with $\boldsymbol{s}_r(\boldsymbol{x})$, $\boldsymbol{b}_{r,r+1}(\boldsymbol{x})$ 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 5.4 *The kernel of the map $\boldsymbol{b}_{r,r+1}(\boldsymbol{x})$ is the sum of the past and future subcodes P_r and F_{r+1}, that is,*

$$\mathrm{Ker}(\boldsymbol{b}_{r,r+1}) = P_r \oplus F_{r+1}. \tag{5.11}$$

Proof: From the definition of the edge mapping (5.6) we see that $\mathrm{Ker}(\boldsymbol{b}_{r,r+1})$ must be contained in $\mathrm{Ker}(\boldsymbol{s}_r)$ and $\mathrm{Ker}(\boldsymbol{s}_{r+1})$, 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 of binary codes. (codes over $GF(p)$ are analogous).

Theorem 5.5 *The number of states at depth r in the PC trellis for a binary code is*

$$|\mathcal{S}_r| = 2^{k-p_r-f_r}, \tag{5.12}$$

and the number of edges at depth r is given by

$$|\mathcal{B}_{r,r+1}| = 2^{k-p_r-f_{r+1}}. \tag{5.13}$$

Proof: Since $\boldsymbol{s}_r$ is a linear map from $\mathcal{C} \to \mathcal{S}_r$, we may apply the rank theorem of linear algebra [33], i.e.,

$$\dim \mathrm{Ker}(\boldsymbol{s}_r) + \dim |\mathcal{S}_r| = \dim \mathcal{C}, \tag{5.14}$$

which leads to $p_r + f_r + \sigma_r = k$, and hence to (5.12). The proof of (5.13) 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 5.6 *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 binary linear block code $\mathcal{C}$ are bounded by*

$$|\mathcal{S}_r| \geq 2^{k-p_r-f_r}, \tag{5.15}$$

$$|\mathcal{B}_{r,r+1}| \geq 2^{k-p_r-f_{r+1}}. \tag{5.16}$$

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 $\boldsymbol{s}_r(\boldsymbol{x})$ is a state at depth r in this trellis. Now let $\mathcal{C}_{s_r}$ be the subset of codewords which pass through $\boldsymbol{s}_r$. Since every codeword must pass through at least one state at time r, we have

$$\mathcal{C} = \bigcup_{\boldsymbol{s}_r} \mathcal{C}_{s_r}. \tag{5.17}$$

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

Now, since $\boldsymbol{x}$ and $\boldsymbol{x}^{(1)}$ both pass through $\boldsymbol{s}_r$, $\boldsymbol{x}' = \left[\boldsymbol{x}_p \boldsymbol{x}_f^{(1)}\right]^T$ and $\boldsymbol{x}'' = \left[\boldsymbol{x}_p^{(1)} \boldsymbol{x}_f\right]^T$ are two codewords which also pass through $\boldsymbol{s}_r$. Hence,

$$\boldsymbol{x}' - \boldsymbol{x} = \begin{bmatrix} 0 \\ \boldsymbol{x}_f^{(1)} - \boldsymbol{x}_f \end{bmatrix} \tag{5.18}$$

is $\in F_r$, and

$$\boldsymbol{x}'' - \boldsymbol{x} = \begin{bmatrix} \boldsymbol{x}_p^{(1)} - \boldsymbol{x}_p \\ 0 \end{bmatrix} \tag{5.19}$$

is $\in P_r$. We conclude that

$$\boldsymbol{x}^{(1)} - \boldsymbol{x} = \begin{bmatrix} \boldsymbol{x}_p^{(1)} - \boldsymbol{x}_p \\ \boldsymbol{x}_f^{(1)} - \boldsymbol{x}_f \end{bmatrix} = \boldsymbol{x}' + \boldsymbol{x}'' - 2\boldsymbol{x} \Rightarrow \boldsymbol{x} = \underbrace{\boldsymbol{x}' + \boldsymbol{x}''}_{\text{is in } P_r \oplus F_r} - \boldsymbol{x}^{(1)}. \tag{5.20}$$

The codeword $\boldsymbol{x}' + \boldsymbol{x}'' \in P_r \oplus F_r$ however; hence, $\boldsymbol{x}$ is in the coset $P_r \oplus F_r - \boldsymbol{x}^{(1)}$. But this coset's size 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 (5.15).

The proof of the second part of the theorem is analogous. Q.E.D.

5.5 MINIMUM-SPAN GENERATOR MATRICES

We restrict attention to binary codes from now on, noting that codes over $GF(p)$ can be handled analogously.

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 [29]. Let us consider the example of the [7, 4] Hamming code from the last section again, whose generator matrix is given by

$$\boldsymbol{G}_{[7,4]} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}. \tag{5.21}$$

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)

$$\begin{aligned} \boldsymbol{G}_{[7,4]} &\equiv \begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 1 & \\ & 1 & 0 & 0 & 1 & 1 & \\ & & 1 & 0 & 1 & 0 & 1 \\ & & & 1 & 0 & 1 & 1 \end{bmatrix} \equiv \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & & \\ & 1 & 0 & 0 & 1 & 1 & \\ & & 1 & 0 & 1 & 0 & 1 \\ & & & 1 & 0 & 1 & 1 \end{bmatrix} \\ &\equiv \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & & \\ & 1 & 0 & 0 & 1 & 1 & \\ & & 1 & 1 & 1 & 1 & \\ & & & 1 & 0 & 1 & 1 \end{bmatrix} \equiv \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & & \\ & 1 & 1 & 1 & & & \\ & & 1 & 1 & 1 & 1 & \\ & & & 1 & 0 & 1 & 1 \end{bmatrix}. \end{aligned} \tag{5.22}$$

What we have generated is an equivalent minimum span generator matrix (MSGM) [29] 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 $\boldsymbol{x}$ is defined as $\text{Span}(\boldsymbol{x}) = R(\boldsymbol{x}) - L(\boldsymbol{x})$, where $R(\boldsymbol{x})$ is the index of the rightmost nonzero entry of $\boldsymbol{x}$, and $L(\boldsymbol{x})$ is the index of the leftmost nonzero entry of $\boldsymbol{x}$.

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

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

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

Step 3: Let $\boldsymbol{x}^{(i)} = \boldsymbol{x}^{(i)} + \boldsymbol{x}^{(j)}$; that is, replace the row $\boldsymbol{x}^{(i)}$ 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 5.7 *The above algorithm always generates a 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

$$\begin{aligned} L(\boldsymbol{x}^{(i)}) &\neq L(\boldsymbol{x}^{(j)}) \\ R(\boldsymbol{x}^{(i)}) &\neq R(\boldsymbol{x}^{(j)}) \end{aligned} \tag{5.23}$$

for all rows $\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)}; \boldsymbol{x}^{(i)} \neq \boldsymbol{x}^{(j)}$ 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 $\boldsymbol{x}^{(1)\prime}, \ldots, \boldsymbol{x}^{(k)\prime}$ be the rows of G'. Then, since G and G' are equivalent,

$$\boldsymbol{x}^{(j)\prime} = \sum_{\boldsymbol{x}^{(i)} \in \mathcal{I}_j} \boldsymbol{x}^{(i)}, \tag{5.24}$$

for every j, where $\mathcal{I}_j$ is some subset of the set of rows of G. But due to (5.23) the sum in (5.24) can produce no cancellations at the endpoints of any member of $\mathcal{I}_j$, and therefore

$$\mathrm{Span}(\boldsymbol{x}^{(j)\prime}) = \max_{\boldsymbol{x}^{(i)} \in \mathcal{I}_j} R(\boldsymbol{x}^{(i)}) - \min_{\boldsymbol{x}^{(i)} \in \mathcal{I}_j} L(\boldsymbol{x}^{(i)}) \geq \max_{\boldsymbol{x}^{(i)} \in \mathcal{I}_j} \mathrm{Span}(\boldsymbol{x}^{(i)}). \tag{5.25}$$

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

$$\mathrm{Span}(\boldsymbol{x}^{(j)\prime}) \geq \mathrm{Span}(\boldsymbol{x}^{(i)}) \tag{5.26}$$

for every j, and hence also

$$\sum_{j=1}^{k} \mathrm{Span}(\boldsymbol{x}^{(j)\prime}) \geq \sum_{i=1}^{k} \mathrm{Span}(\boldsymbol{x}^{(i)}), \tag{5.27}$$

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); that is, it has the smallest span set. From this we immediately deduce the following.

Corollary 5.8 *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 5.9 *Given an MSGM G, we have*

$$p_r = |i : R(\boldsymbol{x}^{(i)}) \leq r|, \tag{5.28}$$

that is, p_r equals the number of rows in G for which the rightmost nonzero entry is at position r or before, and

$$f_r = |i : L(\boldsymbol{x}^{(i)}) \geq r + 1|, \tag{5.29}$$

that is, f_r equals the number of rows in G for which the leftmost nonzero entry is at position $r + 1$ or later.

Proof: p_r is the dimension of the past subcode P_r—that is, the set of all codewords $\in \mathcal{C}$ 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 (5.23), no other row, or linear combination of rows, can be $\in P_r$, since then a codeword would have nonzero entries x_j, for $j > r$, and therefore every codeword in P_r is a linear combination of said rows. There are exactly $|i : R(\boldsymbol{x}^{(i)}) \leq r|$ qualifying rows. This, together with Corollary 5.8, proves (5.28).

Analogously, the rows of G which start at position $r + 1$ or later generate the subcode F_r. Since there are $|i : L(\boldsymbol{x}^{(i)}) \geq r + 1|$ independent rows with that property, this proves (5.29). 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 [17–19].

We now use Theorem 5.28 to read the values of p_r and f_r off the MSGM $\boldsymbol{G}_{[7,4]}$, given in (5.22). 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
$\lvert\mathcal{S}_r\rvert$	1	2	4	8	8	4	2	1
$\lvert\mathcal{B}_{r,r+1}\rvert$	2	4	8	16	8	4	2	—

These values correspond exactly with those in Figure 5.1, where we have constructed the trellis explicitly.

5.6 CONSTRUCTION OF THE PC TRELLIS

The construction of the PC trellis in Section 5.3 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 ref. 29. We will use the fact that the rows of G form bases for P_r and F_r according to Theorems 5.9 and 5.5, that is, $|\mathcal{S}_r| = 2^{k-p_r-f_r}$. Let us start with an example and consider the MSGM G for the extended $[8, 4]$ Hamming code, given by

$$\boldsymbol{G}_{[8,4]} = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}. \tag{5.30}$$

At $r = 0$ we have a single starting state, that is, $|\mathcal{S}_0| = 1$. By inspection we see that from $r = 1$ to $r = 3$, the first row $\boldsymbol{x}^{(1)} = (11110000)$ is *active*. By that we mean that any codeword which contains $\boldsymbol{x}^{(1)}$ will be affected by it in positions $r = 1$ to $r = 3 + 1$. Likewise, the second row $\boldsymbol{x}^{(2)}$ is active for $r \in [2, 6]$, $\boldsymbol{x}^{(3)}$ is active for $r \in [3, 5]$, and $\boldsymbol{x}^{(4)}$ 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 ith component labels the ith active row. It is set to one if the active row is selected, and is set to zero otherwise. These vectors are given for $\boldsymbol{G}_{[8,4]}$ above as:

$\boldsymbol{w}_0$	$\boldsymbol{w}_1$	$\boldsymbol{w}_2$	$\boldsymbol{w}_3$	$\boldsymbol{w}_4$	$\boldsymbol{w}_5$	$\boldsymbol{w}_6$	$\boldsymbol{w}_7$	$\boldsymbol{w}_8$
—	$[w^{(1)}]$	$\begin{bmatrix} w^{(1)} \\ w^{(2)} \end{bmatrix}^T$	$\begin{bmatrix} w^{(1)} \\ w^{(2)} \\ w^{(3)} \end{bmatrix}^T$	$\begin{bmatrix} w^{(2)} \\ w^{(3)} \end{bmatrix}^T$	$\begin{bmatrix} w^{(2)} \\ w^{(3)} \\ w^{(4)} \end{bmatrix}^T$	$\begin{bmatrix} w^{(2)} \\ w^{(4)} \end{bmatrix}^T$	$[w^{(4)}]$	—

We now let the states $\mathcal{S}_r$ be the vectors $\boldsymbol{w}_r$ with $w_r^{(i)} \in \{0, 1\}$. This gives the states of the trellis in Figure 5.4. Two states in this trellis are connected if

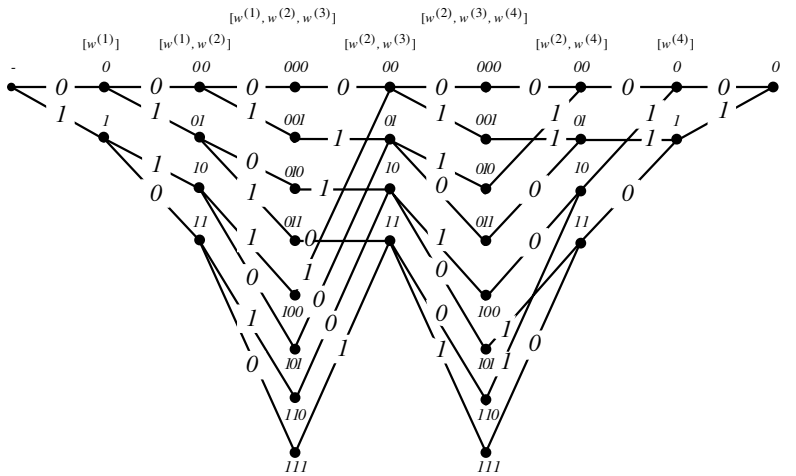


Figure 5.4 PC trellis for the [8, 4] extended Hamming code, constructed in a systematic way.

either $\boldsymbol{w}_r \in \boldsymbol{w}_{r+1}$ or $\boldsymbol{w}_{r+1} \in \boldsymbol{w}_r$; for example, $\boldsymbol{w}_2 = (01)$ and $\boldsymbol{w}_3 = (011)$ are connected since $\boldsymbol{w}_2 = (w^{(1)} = 0, w^{(2)} = 1)^T$ is contained in $\boldsymbol{w}_3 = (w^{(1)} = 0, w^{(2)} = 1, w^{(3)} = 1)$.

The trellis branch labels at time r are determined by the active rows that are selected at both times r and $r + 1$.

First, let $\tilde{\boldsymbol{w}}$ be a k-vector padded with zeros and with entries from w_r in the appropriate places (e.g., $\tilde{\boldsymbol{w}}_6 = (0, w_2, 0, w_4)$). The label x_r is now given by

$$x_r = (\tilde{\boldsymbol{w}}_r \text{ OR } \tilde{\boldsymbol{w}}_{r+1}) \cdot \boldsymbol{g}_r{}^T, \tag{5.31}$$

where OR is the bitwise logic OR and $\boldsymbol{g}_r$ is the rth column in the generator matrix. For instance, x_4 on the transition $(111) \rightarrow (11)$ is given by $((1110) \text{ OR } (0110)) \cdot \boldsymbol{g}_r{}^T = (1110) \cdot (1110)^T = 1$. Since the state vector $\boldsymbol{w}_r$ indicates which active rows are present in the codewords which pass through $\boldsymbol{w}_r$, it should be easy to see that Equation (5.31) simply equals the symbol which these active rows generate at time r.

It is interesting to note that if we combine pairs of branches into single branches with two symbols as branch labels, the resulting new trellis has a maximum number of states which is only four—that is, half that of the original PC trellis. This new trellis is shown in Figure 5.5, and we will construct this trellis again in Section 5.9 in our discussion on Reed–Muller codes. It becomes apparent that there are many ways of representing a block code, and it is not always straightforward to determine which is the best method. However, the PC trellis has minimal complexity among all terminated trellises.

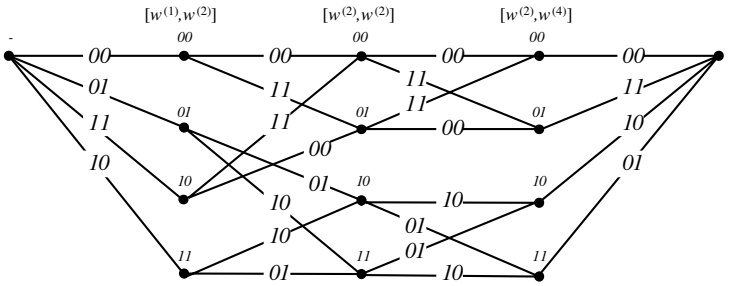


Figure 5.5 Contracted PC trellis for the [8, 4] extended Hamming code, obtained by taking pairs of symbols as branches.

5.7 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 unconventional 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 it often results in trellises that have significantly smaller complexities than their conventional counterparts. In fact, the *square root bound* [39] 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 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 and defines where each row is active, with the only extension that the active span may extend from the end to the beginning. Consider the following slightly altered generator matrix for the extended [8, 4] Hamming code, given by

$$\boldsymbol{G'}_{[8,4]} = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \end{bmatrix}, \tag{5.32}$$

which can be constructed by moving the fourth row from (5.30) into third place, and constructing the fourth row in (5.32) 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 it may start at any nonzero element and extend, possibly wrapping around, to cover the last nonzero element. For the following we define the spans in (5.32) as indicated below by

the star entries:

$$\boldsymbol{G}'_{[8,\,4]} = \begin{bmatrix} \star & \star & \star & \star & & & & \\ & & \star & \star & \star & \star & & \\ & & & & \star & \star & \star & \star \\ \star & \star & \star & & & \star & \star & \star \end{bmatrix}; \tag{5.33}$$

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

$\boldsymbol{w}_0 = \boldsymbol{w}_8$	$\boldsymbol{w}_1$	$\boldsymbol{w}_2$	$\boldsymbol{w}_3$	$\boldsymbol{w}_4$	$\boldsymbol{w}_5$	$\boldsymbol{w}_6$	$\boldsymbol{w}_7$	$\boldsymbol{w}_8$
$\left[w^{(4)}\right]^T$	$\begin{bmatrix} w^{(1)} \\ w^{(4)} \end{bmatrix}^T$	$\begin{bmatrix} w^{(1)} \\ w^{(4)} \end{bmatrix}^T$	$\begin{bmatrix} w^{(1)} \\ w^{(2)} \end{bmatrix}^T$	$\left[w^{(2)}\right]^T$	$\begin{bmatrix} w^{(2)} \\ w^{(3)} \end{bmatrix}^T$	$\begin{bmatrix} w^{(3)} \\ w^{(4)} \end{bmatrix}^T$	$\begin{bmatrix} w^{(3)} \\ w^{(4)} \end{bmatrix}^T$	$\left[w^{(4)}\right]^T$

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:

There exist exactly two nonisomorphic such minimal trellises for the [8,4] extended Hamming code [5].

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 5.6. 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 5.7, a contracted trellis with two branch symbols on every branch results. Of course this new trellis describes a code that is equivalent to the original code only up to symbol permutations.

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. The following theorem is called the *cut-set lower bound* and is due to Wiberg et al. [5, 39].

Theorem 5.10 *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*

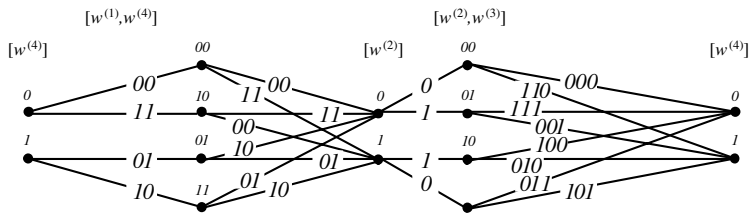


Figure 5.6 Contracted tail-biting trellis for the [8, 4] extended Hamming code.

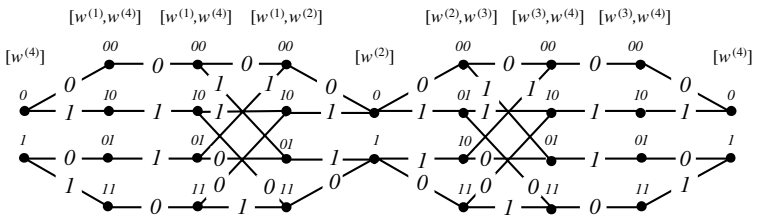


Figure 5.7 Tail-biting trellis for the [8, 4] extended Hamming code.

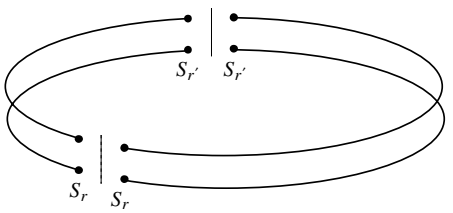


Figure 5.8 Illustration of the cut-set lower bound.

trellis for the same code, that is,

$$|S_r||S_{r'}| \geq |S_c| = \max_r |S_r|. \tag{5.34}$$

Proof: We cut the tail-biting trellis in two disjoint sections by splitting the states at r and r' into two and cutting the connection between them as shown in Figure 5.8.

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 in the middle of a conventional two-section trellis for the code.

Let the set of branches from the initial state to state $S_c = (S_r, S_{r'})$ in the middle 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 state $(S_{r'}, S_r)$ to the final state be the set of paths from S_r to $S_{r'}$ in the future code, that is, the conventional trellis remembers the different states $S_{r'}$ not at the beginning, but in the middle by extending the state space. 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 $|S_c| = |S_r||S_{r'}|$, and the trellis just constricted cannot have a smaller state space S_c than the best trellis. Q.E.D.

The following corollary is known as the *square-root bound*, and it is a straightforward consequence of Theorem 5.10:

Corollary 5.11 *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, that is,*

$$|S_r| \geq \sqrt{|S_c|}. \tag{5.35}$$

An application of Theorem 5.10 to the trellis of the [8, 4] extended Hamming code in Figure 5.4 shows that a tail-biting trellis with a state-space profile (2, 4, 2, 4, 2, 4, 2, 4, 2) meets the bound. However, Calderbank et al. [5] 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 5.7. 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.

As a last point we wish to note that the trellis of a *trellis code* [25] (as opposed to the block codes discussed in this chapter) 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 has 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 slightly higher rate than a terminated trellis code.

5.8 THE SQUARING CONSTRUCTION AND THE TRELLIS OF LATTICES

In this section we take a reversed approach to that of the previous sections, that is, we construct lattices and block codes by constructing their trellises. 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 [8].

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, let $d(S)$ be the minimum squared Euclidean distance between any two elements of S, that is, $d(S) = \min_{\substack{s_1, s_2 \in S \\ (s_1 \neq s_2)}} |s_1 - s_2|^2$, and let $d(T_j)$ be the minimum squared distance between any two elements of the subset T_j. Define $d(T) = \min_j d(T_j)$ 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* [11] as the set of signals U, given by the following definition.

Definition 5.2 *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 $s_1, s_2 \in T_j$, that is, 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 5.9. 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.

Continuing our lattice example, let us apply the squaring construction to the lattice partition $Z^2/RZ^2 = Z^2/D_2$ (compare Figure 3.14, 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 5.10, and we see that this construction yields D_4, the Schläfli lattice, whose points have an integer 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; that is, any two points $d_1, d_2 \in D_4$ can be added as four-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 5.10. 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.

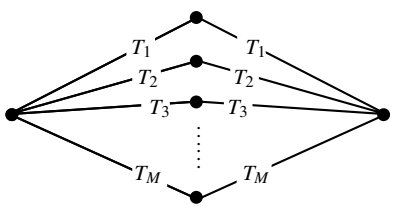


Figure 5.9 Trellis diagram of the squaring construction.

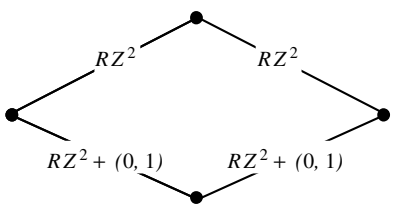


Figure 5.10 Using the squaring construction to generate D_4 from the partition Z^2/D_2.

The squaring construction is an important tool for the following reason:

Lemma 5.12 *Given the distances $d(T)$ and $d(S)$ in the partition S/T, the minimum distance $d(U)$ of U obeys*

$$d(U) = \min[d(T), 2d(S)]. \tag{5.36}$$

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, that is, $u_1 = (t_1, s_1)$, $u_2 = (t_2, s_2)$, 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, that is, $s_1 \in T_j$ 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 5.11; that is, 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 two-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 5.11, we are led to the two-level representation of the set U shown in Figure 5.12. From the above, it should be straightforward to see that Figures 5.9 and 5.12 show the same object with different degrees of detail.

There is another way of looking at Figure 5.11. The paths to the first node at the second stage (solid paths in the figure) are in fact the set obtained by

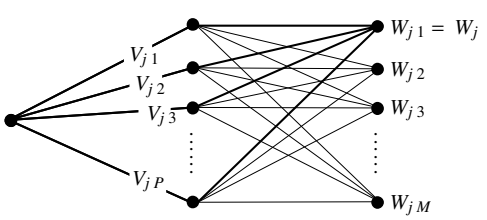


Figure 5.11 Trellis representation of the set $|T_j|^2 = T_j \times T_j$, using the second level partition T/V.

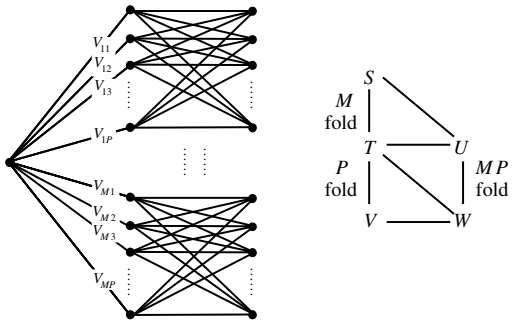


Figure 5.12 Representation of the original squaring construction refined to the two-level partition $S/T/V$ and schematic representation of the induced partition U/W.

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*.

This then induces an MP partition of the set U, denoted by U/W, as shown in Figure 5.12; that is, the set U is partitioned into MP disjoint sets W_{ji}. The right-hand side of Figure 5.12 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 may now apply the squaring construction to the new partition U/W and obtain a *two-level squaring construction* depicted in Figure 5.13. This is done by concatenating back to back two trellis sections of the type in Figure 5.12, 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 5.12 twice we obtain

$$\begin{aligned} d(|S/T/V|^4) &= \min[d(W), 2d(U)] \\ &= \min[d(V), 2d(T), 4d(S)]. \end{aligned} \tag{5.37}$$

As an example of the two-level squaring construction, let us start with the binary lattice partition chain of the integer lattice, $Z^2/RZ^2/2Z^2$. The resulting lattice[2] has eight dimensions, and its 4-state trellis diagram is shown in Figure 5.14. 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 [8].

This game of two-level squaring constructions may now be continued with the newly constructed lattice; that is, we can recursively construct the lattices

[2]The fact that the resulting union set is also a lattice is argued analogously to the case of D_4.

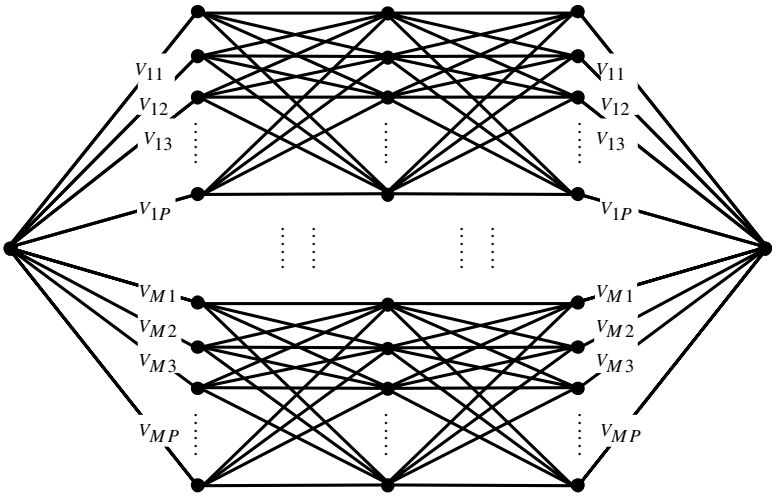


Figure 5.13 Two-level squaring construction for the partition $S/T/V$.

$\Lambda(n) = |\Lambda(n-1)/R\Lambda(n-1)|^2 = |\Lambda(n-2)/R\Lambda(n-2)/2\Lambda(n-2)|^4$ 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 $2^n = N/2$ [11, 8].

The minimum squared distance results from the squaring construction, that is $d(\Lambda_N) = \min\big(d(R\Lambda_{N/2}),\ 2d(\Lambda_{N/2})\big) = \min(d(2\Lambda_{N/4}),\ 2d(R\Lambda_{N/4}),\ 4d(\Lambda_{N/4}))$. 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 general fact that $d(R\Lambda) = 2d(\Lambda)$, that is, $d(\Lambda_N) = 2d(\Lambda_{N/2}) = 4d(\Lambda_{N/4})$, and hence $d(\Lambda_N) = N/2$.

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 5.15, 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 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

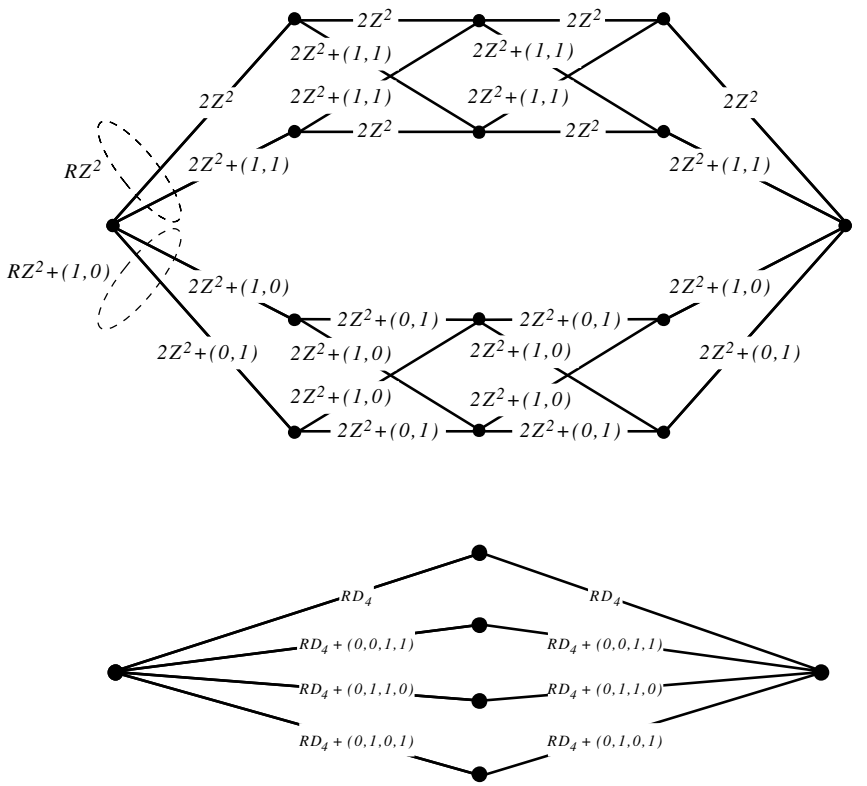


Figure 5.14 Gosset lattice E_8 obtained via the two-level squaring construction. The subsets are expressed as cosets $Z_2 + \mathbf{c}$, where $\mathbf{c}$ is the coset representative (see also Section 3.5). The lower figure shows E_8 as the single-level squaring construction from the partition $U/V = D_4/RD_4$.

Λ_{16}, and then 256, 65,536, and 2^{64} states for Λ_{32}, Λ_{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 two-dimensional signals to make up one eight-dimensional codeword 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 5.10 that a decoder for the E_8 lattice becomes rather simple indeed. Further aspects

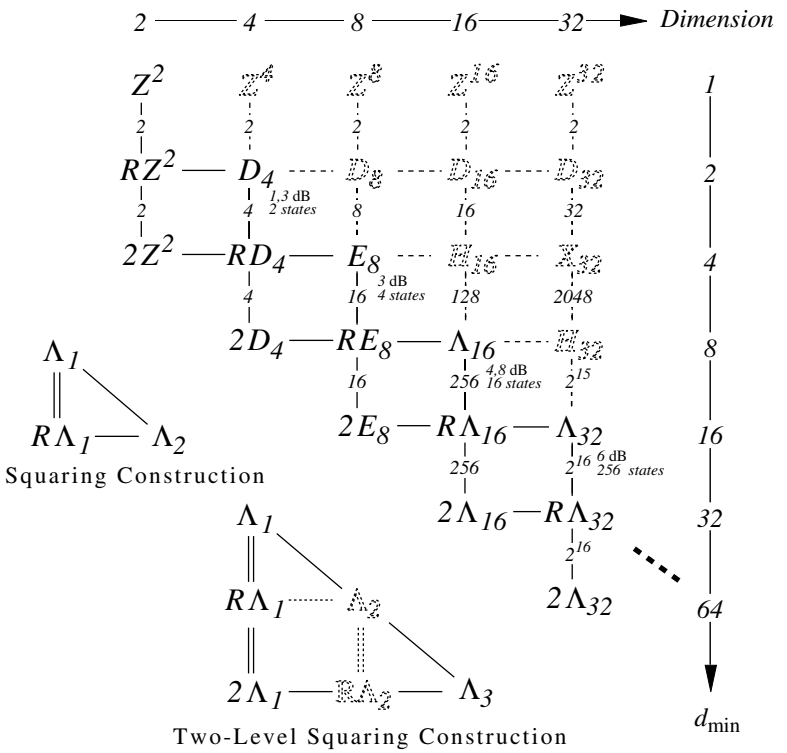


Figure 5.15 Schematic representation of the relationships of the infinite sequence of Barnes–Walls and related lattices. The small numbers indicate the order of the lattice partition $|\Lambda/\Lambda'|$. The asymptotic coding gain and the number of trellis states is also given for the main sequence of lattices.

of implementation of lattice modems are discussed by Lang and Longstaff [21] and by Conway and Sloane [9].

5.9 THE CONSTRUCTION OF REED–MULLER CODES

In this section 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 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 GF(2), $(2, 2)/(2, 1)$ is a true partition, and the same holds for $(2, 1)/(2, 0)$. We now define [11] the Reed–Muller codes $\mathrm{RM}(r, n)$ with parameters r, n recursively as

$$\mathrm{RM}(r, n) = |\mathrm{RM}(r, n-1)/\mathrm{RM}(r-1, n-1)|^2, \tag{5.38}$$

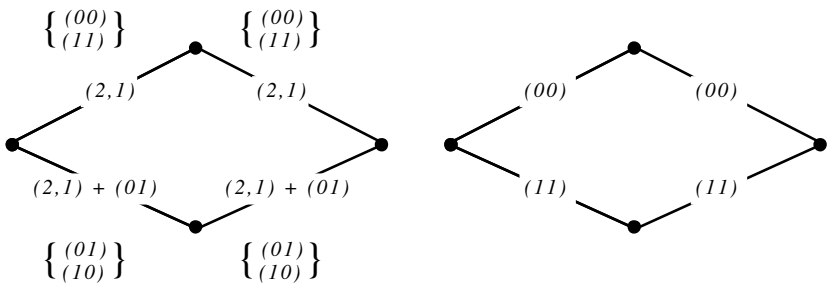


Figure 5.16 The construction of RM(1, 2) and RM(0, 2) via the squaring construction from the binary codes (2, 2), (2, 1), and (2, 0).

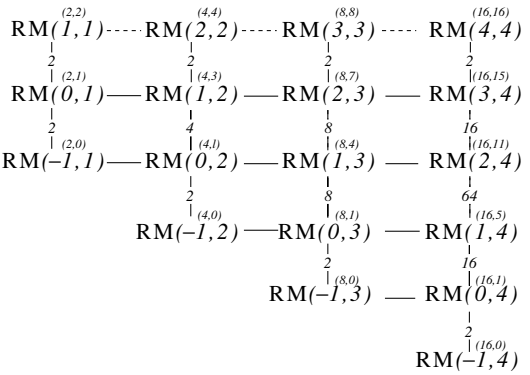


Figure 5.17 Diagram for the construction of the Reed–Muller codes via the squaring and two-level squaring constructions.

and start the recursion with RM(−1, 1) = (2, 0), RM(0, 1) = (2, 1) and RM(1, 1) = (2, 2). The first two codes constructed are illustrated in Figure 5.16. The code RM(1, 2) = $|(2, 2)/(2, 1)|^2$ is a (4, 3) single parity-check code, and the code RM(0, 2) = $|(2, 1)/(2, 0)|^2$ is the (4, 1) repetition code.

Figure 5.17 shows the construction table for the Reed–Muller codes. Every column is completed at the top with RM(n, n), the binary $(2^n, 2^n)$ code consisting of all length-2^n binary vectors, and at the bottom with RM($-1, n$), the binary code consisting of the single codeword $(0, \ldots, 0)$ of length 2^n. The partition orders are also indicated in the figure.

We now need to determine the code parameters of the RM(r, n) codes. Their length N equals 2^n, and from Lemma 5.12 we establish recursively the minimum distance $d_{\min} = 2^{n-r}$. The code rate is found as follows. The partition order of $|\mathrm{RM}(r, n)/\mathrm{RM}(r-1, n)|$, denoted by $m(r, n)$, follows from the squaring construction and obeys the recursion

$$m(r, n) = m(r, n-1)m(r-1, n-1). \tag{5.39}$$

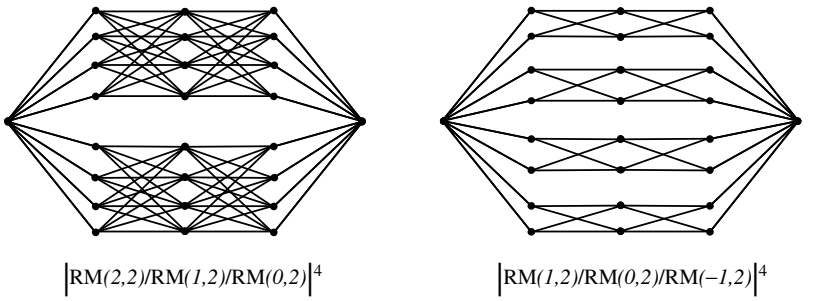


Figure 5.18 The trellis diagrams of RM(2, 4) and RM(1, 4), constructed via the two-level squaring construction.

Starting this recursion with $m(1, 1) = 2$ and $m(0, 1) = 2$ leads to the partition numbers in Figure 5.17. Converting to logarithms—that is, $M(r, n) = \log_2(m(r, n))$—(5.39) becomes

$$M(r, n) = M(r, n-1) + M(r-1, n-1), \tag{5.40}$$

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 RM(r, n) codes are now found easily. The rate of RM(r, n) is given by the rate of RM$(r - 1, n)$ plus $M(r, n)$. This generates the code rates indicated in Figure 5.17.

Among the general class of Reed–Muller codes we find the following special classes:

- RM$(n - 1, n)$ are the single parity-check codes of length 2^n.
- RM$(n-2, n)$ is the family of extended Hamming codes of length 2^n with minimum distance $d_{\min} = 4$ [11, 8].
- RM$(1, n)$ are the first-order Reed–Muller codes of length $N = 2^n$, rate $R = (n + 1)/N$, and minimum distance $d_{\min} = N/2$ [8].
- RM$(0, n)$ are the repetition codes of length 2^n.

Figure 5.18 below shows the four-section trellis diagrams of RM(2, 4), a [16, 11] code, and of RM(1, 4), a [16, 5] code.

5.10 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 5.9 above have $N/2$ states, while the PC trellises of the original and the extended Hamming codes, constructed according to Section 5.3, 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 RM(1, 3), whose trellis is shown in Figure 5.19. Note that the trellis is the same as that of the Gosset lattice E_8 in Figure 5.14 and, after rearranging the states at time $r = 1$, identical to the trellis in Figure 5.5. Let us assume that, as is the usual practice in telecommunications, that zeros are mapped into -1's and 1's are retained; that is, we map the output signals $\{0, 1\}$ of the code into a BPSK signal set and we obtain the modified codewords $\boldsymbol{x}'$ from the original $\boldsymbol{x}$. With this the decoder now operates as follows. For each of the signals y_i from the received signal vector $\boldsymbol{y} = (y_1, \ldots, y_8)$, 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 $m_1 = -y_i$ by expanding the squares and canceling common terms. We see that m_0 and m_1 are negatives of each other, and only one must be calculated. Since the all-one vector $\boldsymbol{x} = (1, \ldots, 1)$ is a codeword, the negative of every modified codeword $\boldsymbol{x}'$ is itself a code word, as can be seen by applying the linearity of the code and adding $\boldsymbol{x}$ to that modified codeword. The sign of the metric sum can now be used to identify the sign of the codeword.

The operation of the decoder is illustrated in Figure 5.20, where the operator $\mathcal{O}$ produces the sum and difference of the inputs; that is, 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

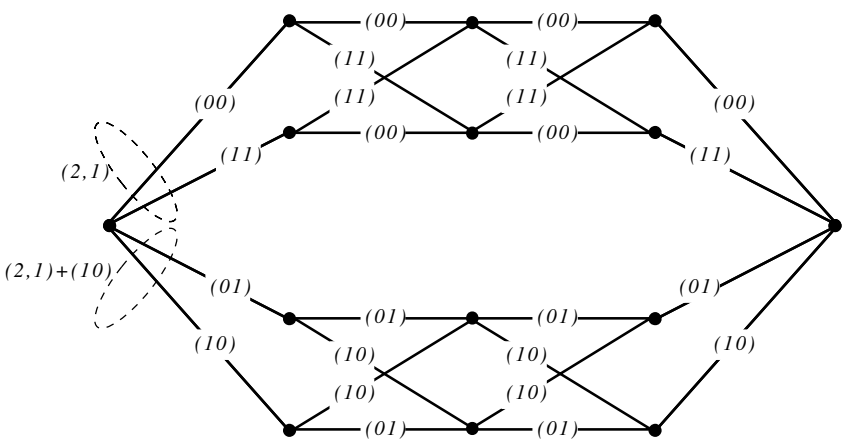


Figure 5.19 Trellis of the [8, 4] extended Hamming code. This is also the trellis of the Gosset lattice from Figure 5.14 as can be seen by comparing Figures 5.15 and 5.17.

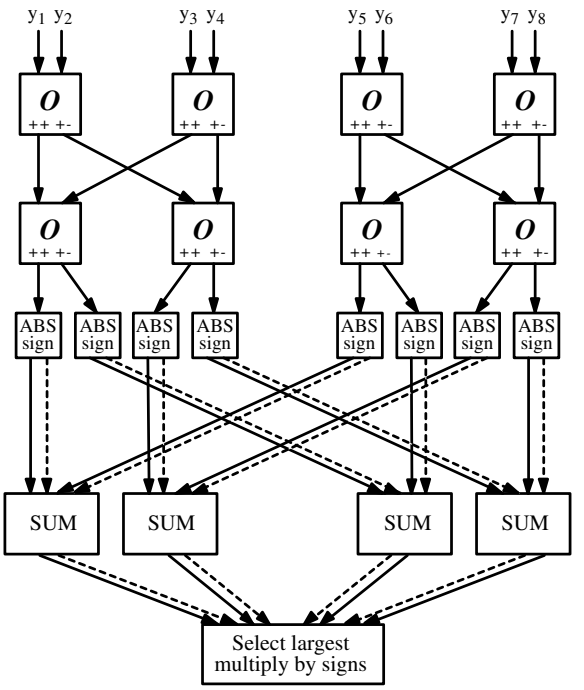


Figure 5.20 Schematic diagram of an efficient decoder for the [8, 4] extended Hamming code. The operator $\mathcal{O}$ produces the sum $y_i + y_j$ and the difference $y_i - y_j$ of the inputs, ABS forms the absolute value and SUM adds $y_i + y_j$.

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 5.14 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 that is identical to that of the extended Hamming code. The decoded sublattice sequence determines now if the odd or even integers are retained, that is, 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.

Finally we wish to refer the interested reader to more in-depth material on this subject, such as recent books by Lin [22] and by Honary and Markarian [15].

BIBLIOGRAPHY

1. J. B. Anderson and S. M. Hladik, "Tailbiting MAP decoders," *IEEE J. Select. Areas Commun.,* vol. SAC-16, pp. 297–302, Feb. 1998.
2. 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.
3. E. R. Berlekamp, *Algebraic Coding Theory*, McGraw-Hill, New York, 1968.
4. R. E. Blahut, *Principles and Practice of Information Theory*, Addison-Wesley, Reading, MA, 1987.
5. 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.
6. 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.
7. G. C. Clark and J. B. Cain, *Error-Correction Coding for Digital Communications*, Plenum Press, New York, 1983.
8. J. H. Conway and N. J. A. Sloane, *Sphere Packings, Lattices and Groups*, Springer-Verlag, New York, 1988.
9. 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.
10. G. D. Forney, Jr., "Generalized minimum distance decoding," *IEEE Trans. Inform. Theory,* vol. IT-12, pp. 125–131, April 1966.
11. 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.
12. 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.
13. 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.
14. F. Hemmati, "Closest coset decoding of $|u|u + v|$ codes," *IEEE J. Select. Areas Commun.,* vol. SAC-7, pp. 982–988, Aug. 1989.
15. B. Honary and G. Markarian, *Trellis Decoding of Block Codes: A Practical Approach*, Kluwer Academic Publishers, Boston, 1997.
16. H. Imai et al. *Essentials of Error-Control Coding Techniques*, Academic Press, New York, 1990.
17. 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.
18. T. Kasami et al., "On the trellis structure of block codes," *IEEE Trans. Inform. Theory,* vol. IT-39, pp. 1057–1064, 1993.

19. A. B. Kiely, S. J. Dolinar, Jr., R. J. McEliece, L. L. Ekroot, and W. Lin, "Trellis complexity of linear block codes," *IEEE Trans. Inform. Theory,* vol. IT-42, no. 6, Nov. 1996.

20. F. Kshischang and V. Sorokine, "On the trellis structure of block codes," *IEEE Trans. Inform. Theory,* vol. IT-41, pp. 1924–1937, Nov. 1995.

21. G. R. Lang and F. M. Longstaff, "A Leech lattice modem," *IEEE J. Select. Areas Commun.,* vol. SAC-7, no. 6, Aug. 1989.

22. S. Lin (Editor) *Trellises and Trellis-Based Decoding Algorithms for Linear Block Codes*, Kluwer Academic Publishers, Boston, 1998.

23. S. Lin and D. J. Costello, Jr., *Error Control Coding*, Prentice-Hall, Englewood Cliffs, NJ, 1983.

24. 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.

25. J. H. Ma and J. K. Wolf, "On tailbiting convolutional codes," *IEEE Trans. Commun.,* vol. COM-34, pp. 104–111, Feb. 1986.

26. J. L. Massey, "Foundations and methods of channel coding," *Proc. Int. Conf. Inform. Theory and Systems, NTG-Fachberichte*, vol. 65, pp. 148–157, 1978.

27. J. L. Massey, "Shift-register synthesis and BCH decoding," *IEEE Trans. Inform. Theory,* vol. IT-15, pp. 122–127, Jan. 1969.

28. 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.

29. R. J. McEliece, "On the BCJR trellis for linear block codes," *IEEE Trans. Inform. Theory,* vol. IT-42, no. 4, July 1996.

30. R. McEliece, *The Theory of Information and Coding* Encyclopedia of Mathematics and Its Applications, Addison-Wesley, Reading, MA, 1977.

31. F. J. MacWilliams and N. J. A. Sloane, *The Theory of Error Correcting Codes*, North-Holland, New York, 1988.

32. 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.

33. G. Strang, *Linear Algebra and Its Applications*, Harcourt Brace Jovanovich, San Diego, 1988.

34. 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.

35. 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.

36. 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.

37. 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.

38. 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.

39. 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.

40. 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 6

Performance Bounds

6.1 ERROR ANALYSIS

The error performance analysis of trellis codes is almost exclusively based on a code's distance spectrum used in union bounding techniques. One of the purposes of this chapter is to show that 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.

In this chapter we treat only trellis codes, and we extend the methods to turbo codes later after introducing the principles of turbo coding in Chapter 10.

6.2 THE ERROR EVENT PROBABILITY

As discussed in Chapter 3, the encoder for a trellis 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 (optimal decoding is discussed in the next chapter). The same principles also apply to block codes, whose trellis, however, is not generated by a FSM.

Possibly the single most important measure of interest is the information bit error rate (BER) associated with our decoder. Unfortunately, it turns out that the calculation of the BER is extremely complex, and no efficient methods exist. We therefore look for more accessible ways of obtaining a measure of performance. We will proceed in the same way as is usual for block codes and 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.

Trellis and Turbo Coding, by Christian B. Schlegel and Lance C. Pérez
ISBN 0-471-22755-2

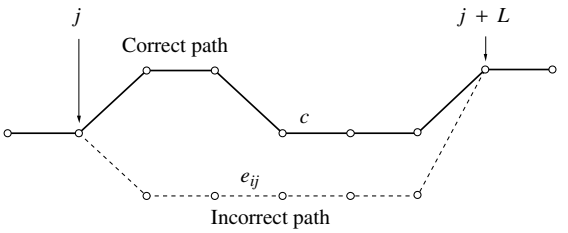


Figure 6.1 The correct path and an error path of length L shown in the code trellis.

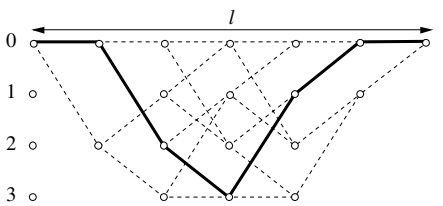


Figure 6.2 Trellis diagram showing a given correct path and the set of possible error paths.

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 6.1, where the decoder starts an error at time j by following an incorrect path which remerges with the correct sequence at time $j + L$.

In general there are many overlapping error paths possible; Figure 6.2 shows an example trellis where the correct path is indicated by a solid line, and all possible paths (i.e., all error paths) are indicated 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.

The probability of error, P, is the probability that any of the dotted paths in Figure 6.2 is chosen; that is, P is the probability of the union of the individual errors e_{ij}, given by

$$P = \Pr\left(\bigcup_j \bigcup_i e_{ij} \middle| c\right), \tag{6.1}$$

where e_{ij} is the ith 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, that is

$$\overline{P} = \sum_c p(c) \Pr\left(\bigcup_j \bigcup_i e_{ij} \middle| c\right), \tag{6.2}$$

where $p(c)$ is the probability that the *encoder* chooses path c.

The probability in (6.2) is still difficult to calculate, and we use the union bound[1] to simplify the expression further to obtain

$$\overline{P} \leq \sum_{c} p(c) \sum_{j} \Pr\left(\bigcup_{i} e_{ij} | c\right). \tag{6.3}$$

If the total length l of the encoded sequence is very large, $\overline{P}$ will be very large; in fact, it will approach 1 as $l \to \infty$, and its probability is therefore not a good measure for the performance of a trellis code. We therefore normalize $\overline{P}$ per unit time and consider

$$\overline{P_e} = \lim_{l \to \infty} \frac{1}{l} \overline{P}. \tag{6.4}$$

Since an infinite trellis looks identical at every time unit, we can eliminate the sum over j in (6.3) to obtain

$$\overline{P_e} \leq \sum_{c} p(c) \Pr\left(\bigcup_{i} e_i | c\right), \tag{6.5}$$

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 (6.5) can also be interpreted as the *first event error probability*—that is the probability that the decoder starts its first error event at node j.

In order to make the bound manageable, we apply the union bound again to obtain

$$\overline{P_e} \leq \sum_{c} p(c) \sum_{e_i} \Pr\left(e_i | c\right). \tag{6.6}$$

Let us denote $\Pr\left(e_i | c\right)$ by $P_{c \to e_i}$, which, since there are only two hypotheses involved, is easily evaluated as [see Equation (2.14)]

$$P_{c \to e_i} = Q\left(\sqrt{d_{ci}^2 \frac{RE_b}{2N_0}}\right), \tag{6.7}$$

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 (6.7) is commonly called the *pairwise error probability*.

[1]The 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, that is $\Pr(\bigcup E_i) \leq \sum_i \Pr(E_i)$.

The upper bound on $\overline{P_e}$ now becomes

$$\overline{P_e} \leq \sum_{c} p(c) \sum_{e_i | c} Q\left(\sqrt{d_{ci}^2 \frac{RE_b}{2N_0}}\right), \tag{6.8}$$

which can be rearranged into

$$\overline{P_e} \leq \sum_{\substack{i \\ (d_i^2 \in \mathcal{D})}} A_{d_i^2} Q\left(\sqrt{d_i^2 \frac{RE_b}{2N_0}}\right), \tag{6.9}$$

by counting how often each of the squared Euclidean distances d_i^2 occurs in (6.8) 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, 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_{ci}^2 with respect to their error paths. The smallest d_i^2 that 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 6.3 shows the distance spectrum of the 16-state 8-PSK code from Table 3.1, whose free squared Euclidean distance d_{free}^2 equals 5.17.

From the first error event probability we can obtain a bound on the average bit error probability by the following reasoning. Each error event $c \to e_i$ 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

$$\overline{P_b} \leq \sum_{\substack{i \\ (d_i^2 \in \mathcal{D})}} \frac{1}{k} B_{d_i^2} Q\left(\sqrt{d_i^2 \frac{RE_b}{2N_0}}\right). \tag{6.10}$$

Figure 6.3 also shows the values B_{d^2} for this code.

Figure 6.4 is a plot of the bound (6.10) for some popular 8-PSK Ungerboeck trellis codes (solid curves) and compares it to results obtained by simulation (dashed curves). This illustrates the tightness of (6.10) which is a good measure of a code's performance for $\overline{P_b} \lessapprox 10^{-2}$.

Algorithms to compute a code's distance spectrum can be generated relatively easily from any one of the decoding algorithms described in Chapter 7, by changing the decoding metric to d_{ci}^2 and keeping account of all the merged distances and their multiplicities.

$\{d^2\}$	A_{d^2}	B_{d^2}
5.172	2.25	11.5
5.757	4.625	21.875
6	1	4
6.343	6.063	41.25
6.586	4	15
6.929	17.656	116.219
7.172	6.75	39
7.515	31.891	234.906
7.757	26.375	169.625
8	8.5	42
8.101	62.602	526.289
8.343	58.375	408.875
8.586	16.625	94.375
8.686	130.691	1163.28
8.929	134.719	1107.84
9.172	60.75	400.75
9.272	253.697	2480.54
9.414	4	17
9.515	338.688	2948.34
9.757	168.219	1253.53
9.858	511.915	5385.71
10	16.5	106

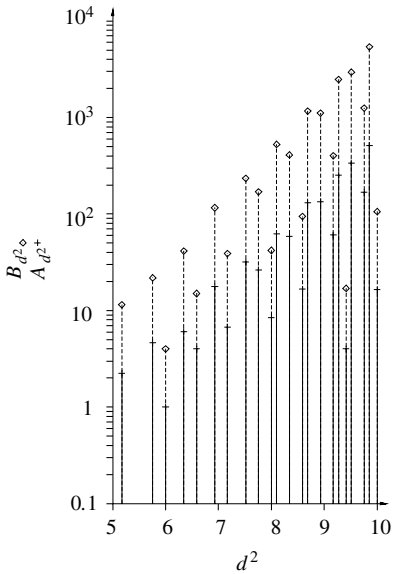


Figure 6.3 Distance spectrum of the Ungerboeck 16-state 8-PSK trellis code with generators $h^{(0)} = 23$, $h^{(1)} = 4$, and $h^{(2)} = 16$. [12]

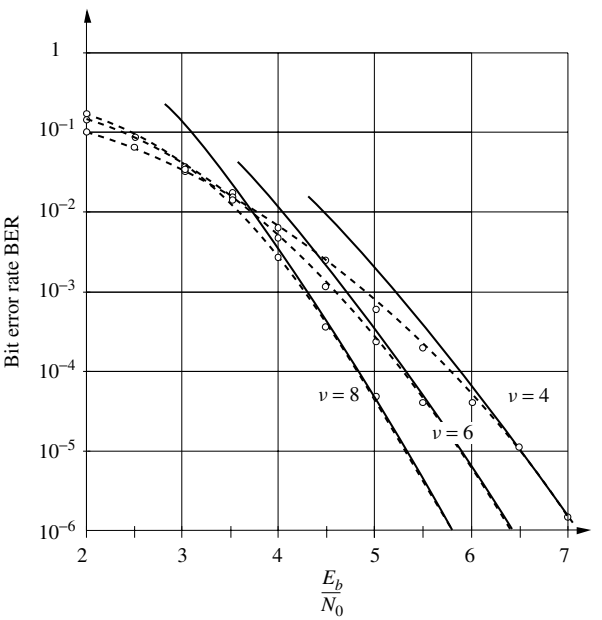


Figure 6.4 Bound on the bit error probability and simulation results for the three 8-PSK Ungerboeck trellis codes with $\nu = 4, 6, 8$ from Table 3.1. The simulation results are from ref. 18.

6.3 FINITE-STATE MACHINE DESCRIPTION OF ERROR EVENTS

If a trellis code is regular (or geometrically uniform), the averaging over c in (6.8) 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 $\delta((p, q) \to (p_1, q_1)) = d^2_{(p,q),(p_1,q_1)}$, where $d^2_{(p,q),(p_1,q_1)}$ 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 $(p, q) \to (p_1, q_1)$ is not possible, $\delta((p, q) \to (p_1, q_1)) = \infty$, 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 $p \to p_1 \to p_2 \cdots \to p_{L-1} \to p_L$ and the error path leads through the states $q \to q_1 \to q_2 \cdots \to q_{L-1} \to q_L$, where $q = p$ and $q_L = p_L$. This is illustrated in Figure 6.5 for $L = 5$.

If we now multiply the exponentials of the distance increments $\delta((p_{i-1}, q_{i-1}) \to (p_i, q_i))$ of $\mathcal{M}$, we obtain

$$\prod_{i=1}^{L} X^{\delta((p_{i-1},q_{i-1})\to(p_i,q_i))} = X^{\sum_{i=1}^{L} d^2_{(p_{i-1},q_{i-1}),(p_i,q_i)}} = X^{d^2_{ci}}, \tag{6.11}$$

that is, the total squared Euclidean distance between c and e_i appears in the exponent of our dummy base X.

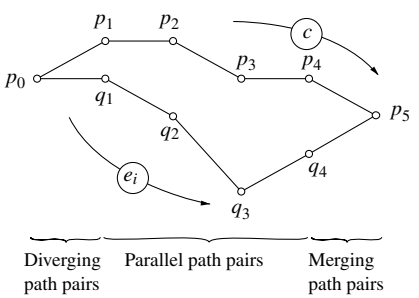


Figure 6.5 The correct path and an error path of length $L = 5$.

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 $1/2^k$.

We can now define the output transition matrix associated with our FSM $\mathcal{M}$, whose entries are given by

$$\mathbf{B} = \{b_{(pq)(p_1q_1)}\} = \left\{\frac{1}{2^k} X^{\delta((p,q)\to(p_1,q_1))}\right\} = \left\{\frac{1}{2^k} X^{d^2_{(p,q),(p_1,q_1)}}\right\}, \qquad (6.12)$$

and it is quite straightforward to see that the $((p, p), (q, q))$-entry of the Lth 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 $p \to p_1$. 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.

It is worth mentioning here that the size of the matrix $\mathbf{B}$ quickly becomes unmanageable, since it grows with the square of the number of states in the code FSM M. As an example, the matrix $\mathbf{B}$ for the 4-state 8-PSK trellis code with $h^{(0)} = 5, h^{(1)} = 4, h^{(2)} = 2$ already has 16×16 entries[2] and is given by

$$\mathbf{B} = \frac{1}{4}\begin{array}{c} \\ 00 \\ 01 \\ 02 \\ 03 \\ 10 \\ 11 \\ 12 \\ 13 \\ 20 \\ 21 \\ 22 \\ 23 \\ 30 \\ 31 \\ 32 \\ 33 \end{array}\begin{array}{c} \begin{array}{cccccccccccccccc} 00 & 01 & 02 & 03 & 10 & 11 & 12 & 13 & 20 & 21 & 22 & 23 & 30 & 31 & 32 & 33 \end{array} \\ \begin{pmatrix} 1 & X^2 & X^4 & X^2 & X^2 & 1 & X^2 & X^4 & X^4 & X^2 & 1 & X^2 & X^2 & X^4 & X^2 & 1 \\ X^{3.4} & X^{0.6} & X^{0.6} & X^{3.4} & X^{3.4} & X^{3.4} & X^{0.6} & X^{0.6} & X^{0.6} & X^{3.4} & X^{3.4} & X^{0.6} & X^{0.6} & X^{0.6} & X^{3.4} & X^{3.4} \\ X^2 & 1 & X^2 & X^4 & 1 & X^2 & X^4 & X^2 & X^2 & X^4 & X^2 & 1 & X^4 & X^2 & 1 & X^2 \\ X^{0.6} & X^{3.4} & X^{3.4} & X^{0.6} & X^{3.4} & X^{3.4} & X^{0.6} & X^{0.6} & X^{3.4} & X^{0.6} & X^{0.6} & X^{3.4} & X^{0.6} & X^{0.6} & X^{3.4} & X^{3.4} \\ X^{3.4} & X^{3.4} & X^{0.6} & X^{0.6} & X^{0.6} & X^{3.4} & X^{3.4} & X^{0.6} & X^{0.6} & X^{0.6} & X^{3.4} & X^{3.4} & X^{3.4} & X^{0.6} & X^{0.6} & X^{3.4} \\ 1 & X^2 & X^4 & X^2 & X^2 & 1 & X^2 & X^4 & X^4 & X^2 & 1 & X^2 & X^2 & X^4 & X^2 & 1 \\ X^{3.4} & X^{3.4} & X^{0.6} & X^{0.6} & X^{3.4} & X^{0.6} & X^{0.6} & X^{3.4} & X^{0.6} & X^{0.6} & X^{3.4} & X^{3.4} & X^{0.6} & X^{3.4} & X^{3.4} & X^{0.6} \\ X^2 & 1 & X^2 & X^4 & 1 & X^2 & X^4 & X^2 & X^2 & X^4 & X^2 & 1 & X^4 & X^2 & 1 & X^2 \\ X^2 & 1 & X^2 & X^4 & 1 & X^2 & X^4 & X^2 & X^2 & X^4 & X^2 & 1 & X^4 & X^2 & 1 & X^2 \\ X^{3.4} & X^{3.4} & X^{0.6} & X^{0.6} & X^{3.4} & X^{0.6} & X^{0.6} & X^{3.4} & X^{0.6} & X^{0.6} & X^{3.4} & X^{3.4} & X^{0.6} & X^{3.4} & X^{3.4} & X^{0.6} \\ 1 & X^2 & X^4 & X^2 & X^2 & 1 & X^2 & X^4 & X^4 & X^2 & 1 & X^2 & X^2 & X^4 & X^2 & 1 \\ X^{3.4} & X^{3.4} & X^{0.6} & X^{0.6} & X^{0.6} & X^{3.4} & X^{3.4} & X^{0.6} & X^{0.6} & X^{0.6} & X^{3.4} & X^{3.4} & X^{3.4} & X^{0.6} & X^{0.6} & X^{3.4} \\ X^{0.6} & X^{3.4} & X^{3.4} & X^{0.6} & X^{3.4} & X^{3.4} & X^{0.6} & X^{0.6} & X^{3.4} & X^{0.6} & X^{0.6} & X^{3.4} & X^{0.6} & X^{0.6} & X^{3.4} & X^{3.4} \\ X^2 & 1 & X^2 & X^4 & 1 & X^2 & X^4 & X^2 & X^2 & X^4 & X^2 & 1 & X^4 & X^4 & X^2 & X^2 \\ X^{3.4} & X^{0.6} & X^{0.6} & X^{3.4} & X^{3.4} & X^{3.4} & X^{0.6} & X^{0.6} & X^{0.6} & X^{3.4} & X^{3.4} & X^{0.6} & X^{0.6} & X^{0.6} & X^{3.4} & X^{3.4} \\ 1 & X^2 & X^4 & X^2 & X^2 & 1 & X^2 & X^4 & X^4 & X^2 & 1 & X^2 & X^2 & X^4 & X^2 & 1 \end{pmatrix} \end{array}$$

[2]The 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

$$\mathbf{D} = \frac{1}{4} \begin{array}{c} \\ 00 \\ 11 \\ 22 \\ 33 \end{array} \begin{array}{c} \begin{array}{cccccccccccc} 01 & 02 & 03 & 10 & 12 & 13 & 20 & 21 & 23 & 30 & 31 & 32 \end{array} \\ \begin{pmatrix} X^2 & X^4 & X^2 & X^2 & X^2 & X^4 & X^4 & X^2 & X^2 & X^2 & X^4 & X^2 \\ X^2 & X^4 & X^2 & X^2 & X^2 & X^4 & X^4 & X^2 & X^2 & X^2 & X^4 & X^2 \\ X^2 & X^4 & X^2 & X^2 & X^2 & X^4 & X^4 & X^2 & X^2 & X^2 & X^4 & X^2 \\ X^2 & X^4 & X^2 & X^2 & X^2 & X^4 & X^4 & X^2 & X^2 & X^2 & X^4 & X^2 \end{pmatrix} \end{array} \tag{6.13}$$

$$\mathbf{P} = \frac{1}{4} \begin{array}{c} \\ 01 \\ 02 \\ 03 \\ 10 \\ 12 \\ 13 \\ 20 \\ 21 \\ 23 \\ 30 \\ 31 \\ 32 \end{array} \begin{array}{c} \begin{array}{cccccccccccc} 01 & 02 & 03 & 10 & 12 & 13 & 20 & 21 & 23 & 30 & 31 & 32 \end{array} \\ \begin{pmatrix} X^{0.6} & X^{0.6} & X^{3.4} & X^{3.4} & X^{0.6} & X^{0.6} & X^{0.6} & X^{3.4} & X^{0.6} & X^{0.6} & X^{0.6} & X^{3.4} \\ 1 & X^2 & X^4 & 1 & X^4 & X^2 & X^2 & X^4 & 1 & X^4 & X^2 & 1 \\ X^{3.4} & X^{3.4} & X^{0.6} & X^{3.4} & X^{0.6} & X^{0.6} & X^{3.4} & X^{0.6} & X^{3.4} & X^{0.6} & X^{0.6} & X^{3.4} \\ X^{3.4} & X^{0.6} & X^{0.6} & X^{0.6} & X^{3.4} & X^{0.6} & X^{0.6} & X^{0.6} & X^{3.4} & X^{3.4} & X^{0.6} & X^{0.6} \\ X^{3.4} & X^{0.6} & X^{0.6} & X^{3.4} & X^{0.6} & X^{3.4} & X^{0.6} & X^{0.6} & X^{3.4} & X^{0.6} & X^{3.4} & X^{3.4} \\ 1 & X^2 & X^4 & 1 & X^4 & X^2 & X^2 & X^2 & 1 & X^4 & X^2 & 1 \\ 1 & X^2 & X^4 & 1 & X^4 & X^2 & X^2 & X^4 & 1 & X^4 & X^2 & 1 \\ X^{3.4} & X^{0.6} & X^{0.6} & X^{3.4} & X^{0.6} & X^{3.4} & X^{0.6} & X^{0.6} & X^{3.4} & X^{0.6} & X^{3.4} & X^{3.4} \\ X^{3.4} & X^{0.6} & X^{0.6} & X^{0.6} & X^{3.4} & X^{0.6} & X^{0.6} & X^{0.6} & X^{3.4} & X^{3.4} & X^{0.6} & X^{0.6} \\ X^{3.4} & X^{3.4} & X^{0.6} & X^{3.4} & X^{0.6} & X^{0.6} & X^{3.4} & X^{0.6} & X^{3.4} & X^{0.6} & X^{0.6} & X^{3.4} \\ 1 & X^2 & X^4 & 1 & X^4 & X^2 & X^2 & X^4 & 1 & X^4 & X^4 & X^2 \\ X^{0.6} & X^{0.6} & X^{3.4} & X^{3.4} & X^{0.6} & X^{0.6} & X^{0.6} & X^{3.4} & X^{0.6} & X^{0.6} & X^{0.6} & X^{3.4} \end{pmatrix} \end{array} \tag{6.14}$$

$$\mathbf{M} = \frac{1}{4} \begin{array}{c} \\ 01 \\ 02 \\ 03 \\ 10 \\ 12 \\ 13 \\ 20 \\ 21 \\ 23 \\ 30 \\ 31 \\ 32 \end{array} \begin{array}{c} \begin{array}{cccc} 00 & 11 & 22 & 33 \end{array} \\ \begin{pmatrix} X^{3.4} & X^{3.4} & X^{3.4} & X^{3.4} \\ X^2 & X^2 & X^2 & X^2 \\ X^{0.6} & X^{3.4} & X^{0.6} & X^{3.4} \\ X^{3.4} & X^{3.4} & X^{3.4} & X^{3.4} \\ X^{3.4} & X^{0.6} & X^{3.4} & X^{0.6} \\ X^2 & X^2 & X^2 & X^2 \\ X^2 & X^2 & X^2 & X^2 \\ X^{3.4} & X^{0.6} & X^{3.4} & X^{0.6} \\ X^{3.4} & X^{3.4} & X^{3.4} & X^{3.4} \\ X^{0.6} & X^{3.4} & X^{0.6} & X^{3.4} \\ X^2 & X^2 & X^2 & X^2 \\ X^{3.4} & X^{3.4} & X^{3.4} & X^{3.4} \end{pmatrix} \end{array}, \tag{6.15}$$

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 6.5 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

$$\mathbf{G}_L = \mathbf{D}\mathbf{P}^{L-2}\mathbf{M}; \qquad L \geq 2. \tag{6.16}$$

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. Equation (6.16) can now be used to find the distance spectrum of a code up to any desired length. While (6.16) is a convenient way of describing the distance spectrum, its calculation is best performed using an adaptation of one of the decoding algorithms described in Chapter 7.

6.4 THE TRANSFER FUNCTION BOUND

The union bound (6.9) and (6.10) are very tight for error probabilities smaller than about 10^{-2} as evidenced in Figure 6.4. Its disadvantage is that we need to evaluate the distance spectrum and also need some way of terminating the sum since (6.9) and (6.10) 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 ref. 2. The first step we take is loosening (6.9) by applying the inequality (see e.g., ref. 15, page 83)

$$Q\left(\sqrt{d_i^2 \frac{RE_b}{2N_0}}\right) \leq \exp\left(-d_i^2 \frac{RE_b}{4N_0}\right). \tag{6.17}$$

Now, using (6.16), which is essentially a table of the squared distances between path pairs of length L, it is easy to see that

$$\overline{P_e} \leq \sum_{\substack{i \\ (d_i^2 \in \mathcal{D})}} A_{d_i^2} \exp\left(-d_i^2 \frac{RE_b}{2N_0}\right) \tag{6.18}$$

$$= \frac{1}{N} \sum_{L=2}^{\infty} \mathbf{1}^T \mathbf{D}\mathbf{P}^{L-2}\mathbf{M}\ \mathbf{1}\Big|_{X=\exp\left(-\frac{RE_b}{4N_0}\right)}, \tag{6.19}$$

where $\mathbf{1} = (1, 1, \ldots, 1)^T$ 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

$$\overline{P_e} \leq \frac{1}{N}\mathbf{1}^T\mathbf{D}\sum_{L=2}^{\infty}\mathbf{P}^{L-2}\mathbf{M}\ \ \mathbf{1}\Big|_{X=\exp\left(-\frac{RE_b}{4N_0}\right)} \tag{6.20}$$

$$= \frac{1}{N}\mathbf{1}^T\mathbf{D}\,(\mathbf{I}-\mathbf{P})^{-1}\,\mathbf{M}\ \ \mathbf{1}\Big|_{X=\exp\left(-\frac{RE_b}{4N_0}\right)}. \tag{6.21}$$

The above equation involves the inversion[3] of an $(N^2-N)\times(N^2-N)$ matrix, which might be a sizeable task. In going from (6.20) to (6.21) we assumed that the infinite series in (6.20) converges. Using matrix theory, it can be shown that this series converges, if the largest eigenvalue of $\mathbf{P}$, $\lambda_{\max} < 1$ [11], and hence $(\mathbf{I}-\mathbf{P})^{-1}$ exists. This is always the case for noncatastrophic 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.

Since using (6.16) 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 (6.16) into two components] and obtain

$$\overline{P_e} \leq \sum_{L=1}^{L_{\max}}\sum_{c_k^{(L)}} p(c_k^{(L)}) \sum_{e_i^{(L)}|c_k^{(L)}} Q\left(\sqrt{d_{ci}^2\frac{RE_b}{2N_0}}\right)$$

$$+ \quad \frac{1}{N}\mathbf{1}^T\mathbf{D}\sum_{L=L_{\max}+1}^{\infty}\mathbf{P}^{L-2}\mathbf{M}\ \ \mathbf{1}\Big|_{X=\exp\left(-\frac{RE_b}{4N_0}\right)}, \tag{6.22}$$

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

$$\epsilon = \frac{1}{N}\mathbf{1}^T\mathbf{D}\sum_{L=L_{\max}+1}^{\infty}\mathbf{P}^{L-2}\mathbf{M}\ \ \mathbf{1}\Big|_{X=\exp\left(-\frac{RE_b}{4N_0}\right)} \leq \frac{\lambda_{\max}^{L_{\max}-1}}{1-\lambda_{\max}}, \tag{6.23}$$

where $\lambda_{\max}$ is the largest eigenvalue[4] of $\mathbf{P}$.

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

[3]The matrix $\mathbf{I}-\mathbf{P}$ is sparse in most cases of interest and can therefore be inverted efficiently numerically [1].

[4]The Perron–Frobenious theorem from linear algebra tells us that a non-negative matrix has a non-negative largest eigenvalue [11]. A matrix is called non-negative, if every entry is a non-negative real number, which is the case for $\mathbf{P}$.

$\mathbf{I} - \mathbf{P}$ and finding $\lambda_{\max}$ are comparable. However, there are many efficient ways of bounding $\lambda_{\max}$. 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 $\lambda_{\max}$ we will use now is obtained by applying Gerschgorin's circle theorem (see Appendix 6.A). Doing this we find that

$$\lambda_{\max} \leq \max_i \sum_j p_{ij} = g; \tag{6.24}$$

that is $\lambda_{\max}$ is smaller than the the maximum row sum of $\mathbf{P}$. A quick look at (6.14) 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 (6.24) is not useful as a bound in (6.23).

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 $\lambda_{\max}^2$. Let us then apply the Gerschgorin bound to $\mathbf{P}^2$, and we obtain

$$\lambda_{\max} \leq \sqrt{\max_i \sum_j p_{ij}^{(2)}} = \sqrt{g^{(2)}}. \tag{6.25}$$

Figure 6.6 shows plots of

$$\frac{\left(\sqrt{g^{(2)}}\right)^{L_{\max}-1}}{\left(1-\sqrt{g^{(2)}}\right)}$$

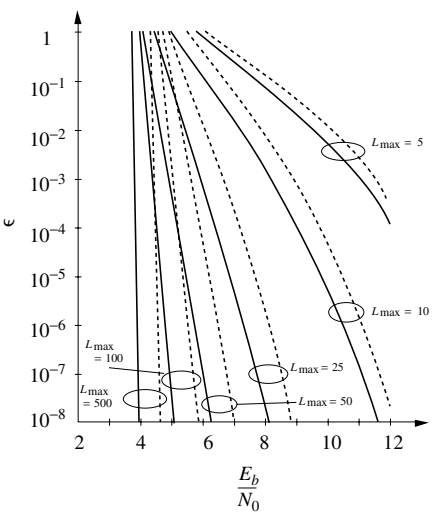


Figure 6.6 Plots of the bound on ϵ (6.23) using the exact value of $\lambda_{\max}$ (solid curves) as well as the Gerschgorin bound (dashed curves) for the 4-state 8-PSK trellis code for $L_{\max} = 5, 10, 25, 50, 100$, and 500.

versus the signal-to-noise ratio $\frac{E_b}{N_0}$ 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.

6.5 REDUCTION THEOREMS

The reader may well wonder at this stage about the complexity of our error calculation, since N^2 grows quickly beyond convenience and makes the evaluation of (6.9) or (6.21) a task for supercomputers. Fortunately, there are large classes of trellis codes for which this error computation can be simplified considerably.

For convolutional codes, for instance, we know that linearity (see Definition 4.1 and preceding discussion) implies that the choice of the correct sequence c does not matter. The same is true for the geometrically uniform codes discussed in Section 3.8. In fact, for any regular code, the distance spectrum is the same for every correct path (codeword) and we may therefore assume c to be any convenient path (codeword), say the all-zero path, and do this without loss of generality. This corresponds to a reduction of the size of the matrix $\mathbf{P}$ to $N - 1 \times N - 1$, since averaging over c in (6.8) is not required.

For general trellis codes, this simplification is not so obvious and in order to proceed we need the notion of equivalence of finite-state machines and their states. The idea behind this approach is to replace our $(N^2 - N)$-state FSM $\mathcal{M}$ by a possibly much smaller FSM—that is by one with fewer states.

The following two definitions will be used later on:

Definition 6.1 *Two states U and V of a FSM $\mathcal{M}$ are output equivalent, denoted by $U \equiv V$, if and only if:*

(i) For every transition $U \to U'$, there exists a transition $V \to V'$ such that the outputs $\delta(U \to U') = \delta(V \to V')$ and vice versa.

(ii) The corresponding successor states are also equivalent, that is $U' \equiv V'$.

Definition 6.2 *Two finite-state machines $\mathcal{M}_1$ and $\mathcal{M}_2$ are called* output equivalent *if and only if their output sequences are identical for identical input sequences and equivalent starting states.*

Definition 6.1 guarantees that if $U \equiv V$, the set of possible output sequences from U and V are identical. We may therefore combine U and V into a single state W which accounts for both U and V and thus generate an equivalent FSM with one state less than $\mathcal{M}$. For the general case there exists a partition algorithm which converges in a finite number of steps to an equivalent FSM having the minimum number of states. This algorithm is not needed for our further discussion here and is described in Appendix 6.B.

We are now ready for our first reduction, using the quasi-regularity of certain signal sets, as discussed in Chapter 3, Definition 3.2.

Theorem 6.1 *The number of states in the FSM $\mathcal{M}$ generating the distances d_i^2 and their multiplicities $A_{d_i^2}$ in group-trellis codes which use quasi-regular signal sets can be reduced from N^2 to N.*

Proof: The output signal of a TCM code is given by the mapper function $\tau(u, s_i)$. Since we assume group-trellis codes (Section 3.2), the set of states s_i forms an additive group and we may pick $p = s_i$ and $q = s_i \oplus e_s$. The output δ of $\mathcal{M}$ from pair-state $(p, q) = (s_i, s_i \oplus e_s)$ with inputs u and $u \oplus e_u$ is the Euclidean distance $d^2 = \|\tau(u, s_i) - \tau(u \oplus e_u, s_i \oplus e_s)\|^2$, as shown in Figure 6.7.

The set of distances $\{d_i^2\}$ from the state $(s_i, s_i \oplus e_s)$ as we vary over all u is $\{d_{e_u, e_s}^2\}$, and it is independent of s_i for quasi-regular signal sets from Definition 3.2. For quasi-regular signal sets, the same set of distances originates from state $(m, n) = (s_k, s_k \oplus e_s)$ as from (p, q), where s_k is arbitrary, establishing (i) in Definition 6.1.

Since $\{d_{e_u, e_S}^2\}$ is independent of s_i there exists for each u a unique v such that

$$\delta' = \|\tau(u, s_i) - \tau(u \oplus e_u, s_i \oplus e_s)\|^2 = \|\tau(v, s_k) - \tau(v \oplus e_u, s_k \oplus e_s)\|^2. \tag{6.26}$$

The successors of the associated transitions are $(p', q') = (\sigma(u, s_i), \sigma(u \oplus e_u, s_i \oplus e_s))$ and $(m', n') = (\sigma(v, s_k), \sigma(v \oplus e_u, s_k \oplus e_s))$ (see Figure 6.8), where $\sigma(u, s_i)$ is the state transition function which gives a successor state as a function of the previous state s_i and the input u.

But due to the additivity of the trellis, $(p', q') = (\sigma(u, s_i), \sigma(u \oplus e_u, s_i \oplus e_s)) = (s_i', s_i' \oplus e_{s'})$ implies $(m', n') = (\sigma(s_k, v), \sigma(s_k \oplus e_s, v \oplus e_u) = (s_k', s_k' \oplus e_{s'})$. Therefore $(s_i, s_i \oplus e_s) \equiv (s_k, s_k \oplus e_s)$ since their successors $(p', q') \equiv (m', n')$ for every output distance δ' by induction on u and the length of the trellis.

Since there are $N - 1$ states $s_k \neq s_i$ there are N equivalent pair states $(s_k, s_k \oplus e_s), s_k \in M$. Each class of equivalent states can be represented by e_s whose outputs are the sets $\{d_{e_u}^2, e_s\}$ and whose successors are $(\sigma(e_u, e_s))$. Q.E.D.

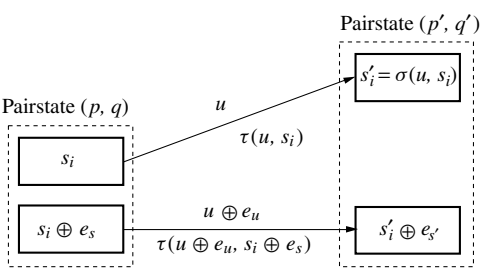


Figure 6.7 Inputs and outputs for the states s_i and $s_i \oplus e_s$ of a quasi-regular trellis code.

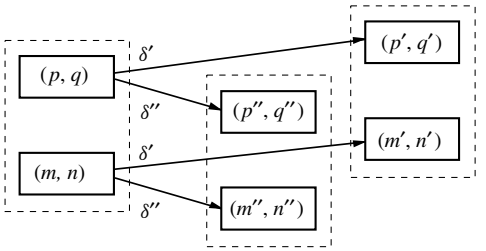


Figure 6.8 Illustration of state equivalence.

Corollary 6.2 *The number of states in the FSM $\mathcal{M}$ for the MPSK and QAM codes in Section 3.4 (Ungerboeck TCM codes) can be reduced from N^2 to N.*

Proof: Ungerboeck codes are group-trellis codes from Section 3.2, using quasi-regular signal constellations. We have already shown in Theorem 3.1 that MPSK signal sets are quasi-regular. Quasi-regularity can also be shown for the QAM-signal sets used in the Ungerboeck codes. This is a tedious, albeit straightforward, exercise. Q.E.D.

As an example, consider again the 4-state trellis code from page 161. The states $00 \equiv 11 \equiv 22 \equiv 33$ are all equivalent, and their equivalent state is $e_s = 0$. Similarly, $01 \equiv 10 \equiv 23 \equiv 32$, $02 \equiv 13 \equiv 20 \equiv 31$, and $03 \equiv 12 \equiv 21 \equiv 30$ with equivalent states $e_s = 1, 2$ and 3, respectively.

The reduced FSM can now be obtained via the following simple procedure. Any equivalent state can be used to describe the distances for the entire equivalence class. We therefore eliminate all but one equivalent state in each class, and we choose this state to be $e_s = (0, e_s)$. This amounts to canceling the corresponding rows in $\mathbf{B}$. Next, mergers into a canceled state have to be handled by its equivalent state. We therefore add all the columns of canceled equivalent states to the one column of e_s, producing as entries polynomials in X with exponents from the set $\{d_{e_s,e_u}\}$.

Doing this with the 16×16 matrix (6.3), we obtain the reduced transfer matrix

$$\mathbf{B}_r = \begin{pmatrix} 1 & X^2 & X^4 & X^2 \\ X^{3.41} & \frac{1}{2}X^{0.59}+\frac{1}{2}X^{3.41} & X^{0.59} & \frac{1}{2}X^{0.59}+\frac{1}{2}X^{3.41} \\ X^2 & 1 & X^2 & X^4 \\ \frac{1}{2}X^{0.59}+\frac{1}{2}X^{3.41} & X^{3.41} & \frac{1}{2}X^{0.59}+\frac{1}{2}X^{3.41} & X^{0.59} \end{pmatrix}, \tag{6.27}$$

which we can likewise partition into a reduced diverging matrix $\mathbf{D}_r$, a reduced parallel matrix $\mathbf{P}_r$, and a reduced merging matrix $\mathbf{M}_r$, obtaining

$$\mathbf{D}_r = \begin{pmatrix} X^2 & X^4 & X^2 \end{pmatrix}, \tag{6.28}$$

$$\mathbf{P}_r = \begin{pmatrix} X^2 & X^4 & X^2 \\ \frac{1}{2}X^{0.59}+\frac{1}{2}X^{3.41} & X^{0.59} & \frac{1}{2}X^{0.59}+\frac{1}{2}X^{3.41} \\ 1 & X^2 & X^4 \\ X^{3.41} & \frac{1}{2}X^{0.59}+\frac{1}{2}X^{3.41} & X^{0.59} \end{pmatrix}, \tag{6.29}$$

$$\mathbf{M}_r = \begin{pmatrix} X^{3.41} \\ X^{0.59} \\ \frac{1}{2}X^{0.59}+\frac{1}{2}X^{3.41} \end{pmatrix}. \tag{6.30}$$

The transfer function bound (6.19) can now be simplified to

$$\overline{P_e} \leq \sum_{L=2}^{\infty} \mathbf{D}_r \mathbf{P}_r^{L-2} \mathbf{M}_r \Big|_{X=\exp\left(-d_i^2 \frac{RE_b}{4N_0}\right)}; \tag{6.31}$$

that is, the summation over all starting and terminating states could be eliminated. Finding the distance spectrum of Ungerboeck codes becomes essentially as simple as finding the distance spectrum of regular or geometrically uniform codes. Remember that convolutional codes, which are linear in the output space, are the best-known representatives of regular codes.

Theorem 6.1 is valid for the distance spectrum as used in (6.9). The following corollary assures us that the bit error multiplicities can also be calculated by a reduced FSM.

Corollary 6.3 *The number of states in the FSM $\mathcal{M}$ describing the distances d_i^2 and bit error multiplicities $B_{d_i^2}$ in Ungerboeck TCM codes can be reduced from N^2 to N.*

Proof: In order to keep track of the bit errors, we have to add the bit error weight to the output of $\mathcal{M}$; that is, the output is given by the pair $\delta((p,q) \to (p_1,q_1)) = \left(d^2_{(p,q),(p_1,q_1)}, w(e_u)\right)$, where $w(e_u)$ is the Hamming weight of e_u and e_u is the bit error pattern between the transitions $p \to p_1$ and $q \to q_1$. But extending (6.26) leads to

$$\begin{aligned} &\left(\|\tau(s_i,u) - \tau(s_i \oplus e_s, u \oplus e_u)\|^2, w(e_u)\right) \\ &= \left(\|\tau(s_k,u') - \tau(s_k \oplus e_s, u' \oplus e_u)\|^2, w(e_u)\right) \\ &= (\delta_a, w(e_u)), \end{aligned} \tag{6.32}$$

which is also independent of s_i as we vary over all u. The rest of the proof is exactly as in Theorem 6.1. Q.E.D.

The approach chosen in this section does not reflect the actual historical development of these results. Methods to overcome the nonlinearity of Ungerboeck

trellis codes were first presented by Zehavi and Wolf [16] and independently by Rouanne and Costello [8] with generalizations in ref. 9 and appeared under the suggestive term "quasi-regularity." Later a slightly different but equivalent approach was given by Biglieri and McLane [3], who called the property necessary for reduction "uniformity."

The state reduction point of view is novel and has been chosen because it appeared to us to be both easier to understand and more general. Reduction theorems have also been proven for CPFSK [18] and complementary convolutional codes [17], where the number of equivalent states could be reduced to less than the number of states in the code FSM M.

6.6 RANDOM CODING BOUNDS

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 [10], 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 decoder[5] which selects the hypothesized transmitted sequence $\hat{\boldsymbol{x}}$ according to the highest probability metric $p(\boldsymbol{y}|\boldsymbol{x})$, where $\boldsymbol{y}$ is the received sequence $\boldsymbol{x} + \boldsymbol{n}$, $\boldsymbol{x}^{(c)}$ is the transmitted sequence, and $\boldsymbol{n}$ is a noise sequence (see Section 2.5). This decoder makes an error if it decodes a sequence $\boldsymbol{x}'$, given that the transmitted sequence was $\boldsymbol{x}$. For an ML decoder, this happens if and only if $p(\boldsymbol{y}|\boldsymbol{x}) \leq p(\boldsymbol{y}|\boldsymbol{x}')$.

[5]Note 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 [10].

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. Furthermore, 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 total probability than the correct path c over their unmerged segments of the trellis.

If we are using a group-trellis code, 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, $\boldsymbol{x}$ is the sequence of signals assigned along the correct path c and $\boldsymbol{x}'$ is the sequence of signals assigned to e.

Using this notation we rewrite (6.5) as

$$\overline{P_e} \leq \sum_{c} p(c) \int_{\boldsymbol{y}} p(\boldsymbol{y}|\boldsymbol{x}) \mathcal{I}\left(\bigcup_{e \in \mathcal{E}} e(p(\boldsymbol{y}|\boldsymbol{x}') \geq p(\boldsymbol{y}|\boldsymbol{x})) \right) d\boldsymbol{y}, \tag{6.33}$$

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 $e(p(\boldsymbol{y}|\boldsymbol{x}') \geq p(\boldsymbol{y}|\boldsymbol{x}))$ is a path e for which $p(\boldsymbol{y}|\boldsymbol{x}') \geq p(\boldsymbol{y}|\boldsymbol{x})$. This then also implies that we are using an ML decoder (see Section 2.4). $\mathcal{I}\left(\bigcup_{e \in \mathcal{E}} e(p(\boldsymbol{y}|\boldsymbol{x}') \geq p(\boldsymbol{y}|\boldsymbol{x}))\right)$ 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 group-trellis code to be the one shown in Figure 6.9. The binary input vector at time r, u_r, has k components and enters a feed-forward shift 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, that is, $x_r = \tau(u_r, s_r) = \tau(u_r, \ldots, u_{r-\nu-1})$.

The particular realization of the group-trellis code as in Figure 6.9 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 for some of them. Note that this structure is a group-trellis encoder as introduced in Figure 3.4.

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; that is, the

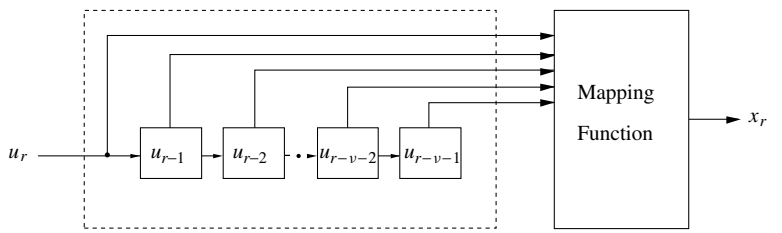


Figure 6.9 Group-trellis encoder structure used for the random coding bound arguments.

output symbol y is described by the conditional probability distribution $p(y|x)$, and the conditional probability of a symbol vector $p(\boldsymbol{y}|\boldsymbol{x})$ 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 $u'_j, \cdots, u'_{j+L}$ associated with e must equal the last $\nu-1$ entries in the information sequences $u_j, \cdots, u_{j+L}$ 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 (6.33) as the sum over sequences of length L, and rewrite it as

$$\overline{P_e} \leq \sum_{L=\nu}^{\infty} \sum_{c^{(L)} \in \mathcal{C}^{(L)}} p(c^{(L)}) \int_{\boldsymbol{y}} p(\boldsymbol{y}|\boldsymbol{x}) \mathcal{I}\left(\bigcup_{e^{(L)} \in \mathcal{E}^{(L)}} e^{(L)} (p(\boldsymbol{y}|\boldsymbol{x}) \leq p(\boldsymbol{y}|\boldsymbol{x}')) \right) d\boldsymbol{y}, \tag{6.34}$$

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 we have

$$p(\boldsymbol{y}|\boldsymbol{x}') \geq p(\boldsymbol{y}|\boldsymbol{x}). \tag{6.35}$$

Therefore, for any real parameter $\gamma \geq 0$ we obtain

$$\sum_{e^{(L)} \in \mathcal{E}^{(L)}} \left[\frac{p(\boldsymbol{y}|\boldsymbol{x}')}{p(\boldsymbol{y}|\boldsymbol{x})} \right]^{\gamma} \geq 1. \tag{6.36}$$

We can raise both sides of (6.36) to the power of some non-negative parameter $\rho \geq 0$ and preserve the inequality, that is,

$$\left(\sum_{e^{(L)} \in \mathcal{E}^{(L)}} \left[\frac{p(\boldsymbol{y}|\boldsymbol{x}')}{p(\boldsymbol{y}|\boldsymbol{x})} \right]^{\gamma} \right)^{\rho} \geq 1. \tag{6.37}$$

This Chernoff bounding technique was introduced by Gallager [6] to derive random coding theorems for block codes, and then applied to convolutional codes by Viterbi [13].

We now use (6.37) to overbound the indicator function $\mathcal{I}(\cdot)$ by an exponential, that is,

$$\mathcal{I}\left(\bigcup_{e^{(L)}\in\mathcal{E}^{(L)}} e^{(L)}(p(\boldsymbol{y}|\boldsymbol{x}) \le p(\boldsymbol{y}|\boldsymbol{x}'))\right) \le \left(\sum_{e^{(L)}\in\mathcal{E}^{(L)}} \left[\frac{p(\boldsymbol{y}|\boldsymbol{x}')}{p(\boldsymbol{y}|\boldsymbol{x})}\right]^{\gamma}\right)^{\rho}. \quad (6.38)$$

Using (6.38) in (6.34), we obtain

$$\overline{P_e} \le \sum_{L=\nu}^{\infty}\sum_{c^{(L)}} p(c^{(L)}) \int_{\boldsymbol{y}} p(\boldsymbol{y}|\boldsymbol{x}) \left(\sum_{e^{(L)}\in\mathcal{E}^{(L)}} \left[\frac{p(\boldsymbol{y}|\boldsymbol{x}')}{p(\boldsymbol{y}|\boldsymbol{x})}\right]^{\gamma}\right)^{\rho} d\boldsymbol{y}. \quad (6.39)$$

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 (6.6). 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 (6.39) to

$$\overline{P_e} \le \sum_{L=\nu}^{\infty}\sum_{c^{(L)}} p(c^{(L)}) \int_{\boldsymbol{y}} p(\boldsymbol{y}|\boldsymbol{x})^{1/(1+\rho)} \left(\sum_{e^{(L)}\in\mathcal{E}^{(\mathcal{L})}} p(\boldsymbol{y}|\boldsymbol{x}')^{1/(1+\rho)}\right)^{\rho} d\boldsymbol{y}. \quad (6.40)$$

$\overline{P_e}$ is the event error probability of a particular code since it depends on the signal sequences $\boldsymbol{x}$ and $\boldsymbol{x}'$ 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, that is,

$$\text{Avg}\left\{\overline{P_e}\right\} \le$$

$$\text{Avg}\left\{\sum_{L=\nu}^{\infty}\sum_{c^{(L)}} p(c^{(L)}) \int_{\boldsymbol{y}} p(\boldsymbol{y}|\boldsymbol{x})^{1/(1+\rho)} \left(\sum_{e^{(L)}\in\mathcal{E}^{(\mathcal{L})}} p(\boldsymbol{y}|\boldsymbol{x}')^{1/(1+\rho)}\right)^{\rho} d\boldsymbol{y}\right\}, \quad (6.41)$$

where 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

$$\text{Avg}\left\{\overline{P_e}\right\} \le$$

$$\sum_{L=\nu}^{\infty} \frac{1}{2^{kL}} \text{Avg}\left\{\sum_{c^{(L)}} \int_{\boldsymbol{y}} p(\boldsymbol{y}|\boldsymbol{x})^{1/(1+\rho)} \left(\sum_{e^{(L)}\in\mathcal{E}^{(\mathcal{L})}} p(\boldsymbol{y}|\boldsymbol{x}')^{1/(1+\rho)}\right)^{\rho} d\boldsymbol{y}\right\}. \quad (6.42)$$

To continue let $\mathcal{X} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N\}$ be a possible assignment of signal sequences of length L associated with the paths $c^{(L)}$; that is, $\mathcal{X}$ is a particular

code and let $q_{LN}(\{\mathcal{X}\}) = q_{LN}(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N)$ 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—that is, over all assignments $\mathcal{X}$. Doing this we obtain

$$\text{Avg}\left\{\overline{P_e}\right\} \leq \sum_{L=\nu}^{\infty} \frac{1}{2^{kL}} \sum_{c^{(L)}} \int_{\boldsymbol{y}} \sum_{\boldsymbol{x}_1} \cdots \sum_{\boldsymbol{x}_N} q_{LN}(\mathcal{X}) p(\boldsymbol{y}|\boldsymbol{x})^{1/(1+\rho)} \times \left(\sum_{e^{(L)} \in \mathcal{E}^{(\mathcal{L})}} p(\boldsymbol{y}|\boldsymbol{x}')^{1/(1+\rho)} \right)^{\rho} d\boldsymbol{y}, \tag{6.43}$$

where $\boldsymbol{x}$ is the signal sequence on $c^{(L)}$ and $\boldsymbol{x}'$ is the one on $e^{(L)}$.

We can rewrite $q_{LN}(\mathcal{X}) = q_{L(N-1)}(\mathcal{X}'|\boldsymbol{x})$, where $\mathcal{X}' = \mathcal{X}\backslash\{\boldsymbol{x}\}$ is the set of signal sequences without $\boldsymbol{x}$ and $q_{L(N-1)}(\mathcal{X}|\boldsymbol{x})$ is the probability of $\mathcal{X}\backslash\{\boldsymbol{x}\}$, conditioned on $\boldsymbol{x}$. 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

$$\text{Avg}\left\{\overline{P_e}\right\} \leq \sum_{L=\nu}^{\infty} \frac{1}{2^{kL}} \sum_{c^{(L)}} \int_{\boldsymbol{y}} \sum_{\boldsymbol{x}} q(\boldsymbol{x}) p(\boldsymbol{y}|\boldsymbol{x})^{1/(1+\rho)} \times \left(\sum_{e^{(L)} \in \mathcal{E}^{(\mathcal{L})}} \underbrace{\sum_{\boldsymbol{x}_1} \cdots \sum_{\boldsymbol{x}_N}}_{(\boldsymbol{x}_i \neq \boldsymbol{x})} q_{L(N-1)}(\mathcal{X}'|\boldsymbol{x}) p(\boldsymbol{y}|\boldsymbol{x}')^{1/(1+\rho)} \right)^{\rho} d\boldsymbol{y}. \tag{6.44}$$

But the inner term in (6.44) depends only on $\boldsymbol{x}'$ and we can reduce (6.44) by summing over all other signal assignments, that is,

$$\text{Avg}\left\{\overline{P_e}\right\} \leq \sum_{L=\nu}^{\infty} \frac{1}{2^{kL}} \sum_{c^{(L)}} \int_{\boldsymbol{y}} \sum_{\boldsymbol{x}} q(\boldsymbol{x}) p(\boldsymbol{y}|\boldsymbol{x})^{1/(1+\rho)} \times \left(\sum_{e^{(L)} \in \mathcal{E}^{(\mathcal{L})}} \sum_{\boldsymbol{x}'} q(\boldsymbol{x}'|\boldsymbol{x}) p(\boldsymbol{y}|\boldsymbol{x}')^{1/(1+\rho)} \right)^{\rho} d\boldsymbol{y}. \tag{6.45}$$

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—that is, $q(\boldsymbol{x}'|\boldsymbol{x}) = q(\boldsymbol{x}')$—and that each signal is chosen independently, making $q(\boldsymbol{x}) = \prod_{i=j}^{L-1} q(x_j)$ 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 $\boldsymbol{x}$ and $\boldsymbol{x}'$ 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

$$\text{Avg}\left\{\overline{P_e}\right\} \leq \sum_{L=v}^{\infty} \frac{1}{2^{kL}} \sum_{c^{(L)}} \int_{\boldsymbol{y}} \sum_{\boldsymbol{x}} \prod_{i=1}^{L} q(x_i) p(y_i|x_i)^{1/(1+\rho)} \left(\sum_{e^{(L)} \in \mathcal{E}^{(\mathcal{L})}} \sum_{\boldsymbol{x}'} \prod_{i=1}^{L} q(x_i') p(y_i|x_i')^{1/(1+\rho)} \right)^{\rho} d\boldsymbol{y}, \tag{6.46}$$

where we have assumed that $p(\boldsymbol{y}|\boldsymbol{x}) = \prod_{i=1}^{L} p(y_i|x_i)$; 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

$$\begin{aligned}
\text{Avg}\left\{\overline{P_e}\right\} &\leq \sum_{L=v}^{\infty} \int_{\boldsymbol{y}} \left(\sum_{x} q(x) p(y|x)^{1/(1+\rho)} \right)^{L} \\
&\quad \times \left(N_p \left(\sum_{x'} q(x') p(y|x')^{1/(1+\rho)} \right)^{L} \right)^{\rho} d\boldsymbol{y} \\
&= \sum_{L=v}^{\infty} N_p^{\rho} \int_{\boldsymbol{y}} \left(\left(\sum_{x} q(x) p(y|x)^{1/(1+\rho)} \right)^{1+\rho} \right)^{L} d\boldsymbol{y} \qquad (6.47) \\
&\leq \sum_{L=v}^{\infty} (2^k - 1)^{\rho} 2^{\rho k (L-v)} \left(\int_{y} \left(\sum_{x} q(x) p(y|x)^{1/(1+\rho)} \right)^{1+\rho} dy \right)^{L}, \qquad (6.48)
\end{aligned}$$

where we have broken up the integral over $\boldsymbol{y}$ into individual integrals over y in the last step.

Let us now define the error exponent

$$E_0(\rho, \mathbf{q}) \equiv -\log_2 \int_{y} \left(\sum_{x} q(x) p(y|x)^{1/(1+\rho)} \right)^{1+\rho} dy, \tag{6.49}$$

where $\mathbf{q} = (q_1, \cdots, q_A)$ 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:

$$\text{Avg}\left\{\overline{P_e}\right\} \leq (2^k - 1)^{\rho} 2^{-v E_0(\rho, \mathbf{q})} \sum_{L=0}^{\infty} 2^{\rho k L} 2^{-L E_0(\rho, \mathbf{q})} \tag{6.50}$$

$$= \frac{(2^k - 1)^{\rho} 2^{-v E_0(\rho, \mathbf{q})}}{1 - 2^{-(E_0(\rho, \mathbf{q}) - \rho k)}}, \qquad \rho k < E_0(\rho, \mathbf{q}), \tag{6.51}$$

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:

$$\text{Avg}\left\{\overline{P_e}\right\} = \frac{(2^R - 1)^\rho 2^{-\nu E_0(\rho,\mathbf{q})}}{1 - 2^{-(E_0(\rho,\mathbf{q})-\rho R)}}, \qquad \rho < E_0(\rho, \mathbf{q})/R. \tag{6.52}$$

Using (6.50), 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 $k(L - \nu + 1)$ bit errors, we obtain (compare (6.10))

$$\text{Avg}\left\{\overline{P_b}\right\} \leq \frac{1}{k}(2^R - 1)^\rho 2^{-\nu E_0(\rho,\mathbf{q})} k \sum_{L=0}^{\infty} (L + 1) 2^{\rho R L} 2^{-L E_0(\rho,\mathbf{q})} \tag{6.53}$$

$$= \frac{(2^R - 1)^\rho 2^{-\nu E_0(\rho,\mathbf{q})}}{\left(1 - 2^{-(E_0(\rho,\mathbf{q})-\rho k)}\right)^2}, \qquad \rho < E_0(\rho, \mathbf{q})/R, \tag{6.54}$$

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 nontrivial. Furthermore, we have the following lemma.

Lemma 6.4 *$E_0(\rho, \mathbf{q})$ is a monotonically increasing function of ρ.*

Proof: We will apply the inequality[6]

$$\left(\sum_i p_i \alpha_i^r\right)^{\frac{1}{r}} \leq \left(\sum_i p_i \alpha_i^s\right)^{\frac{1}{s}}, \qquad 0 < r < s; \alpha_i \geq 0, \tag{6.55}$$

which holds with equality if for some constant c: $p_i \alpha_i = c p_i$ for all i. Using (6.55) in the expression for the error exponent (6.49), we obtain

$$E_0(\rho, \mathbf{q}) = -\log_2 \int_y \left(\sum_x q(x) p(y|x)^{1/(1+\rho)}\right)^{1+\rho} dy$$

[6]This inequality is easily derived from Hölder's inequality

$$\sum_i \beta_i \gamma_i \leq \left(\sum_i \beta_i^{\frac{1}{\mu}}\right)^\mu \left(\sum_i \gamma_i^{\frac{1}{1-\mu}}\right)^{1-\mu}, \qquad 0 < \mu < 1; \beta_i, \gamma_i \geq 0,$$

by letting $\beta_i = p_i^\mu \alpha_i^r$, $\gamma_i = p_i^{1-\mu}$, and $\mu = r/s$.

$$\leq -\log_2 \int_y \left(\sum_x q(x) p(y|x)^{1/(1+\rho_1)} \right)^{1+\rho_1} dy$$

$$= E_0(\rho_1, \mathbf{q}), \tag{6.56}$$

for $\rho_1 > \rho > -1$. Equality holds in (6.56) 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 6.4 tells us that in order to obtain the best bound, we want to choose ρ as large as possible, given the constraint in (6.52); that is, we choose

$$\rho = \frac{E_0(\rho, \mathbf{q})}{R}(1 - \epsilon), \tag{6.57}$$

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 6.5 *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 ref. 13, Appendix 3A.3. We therefore know that $\frac{\partial^2}{\partial \rho^2} E_0(\rho, \mathbf{q}) \leq 0$; that is, the function $\frac{\partial}{\partial \rho} E_0(\rho, \mathbf{q})$ is strictly monotonically decreasing. Let us then consider

$$\frac{\partial}{\partial \rho} \frac{E_0(\rho, \mathbf{q})}{\rho} = \frac{1}{\rho^2} \left(\rho \frac{\partial E_0(\rho, \mathbf{q})}{\partial \rho} - E_0(\rho, \mathbf{q}) \right). \tag{6.58}$$

For any given ρ the point $(\rho, \rho \frac{\partial E_0(\rho, \mathbf{q})}{\partial \rho})$, lies on a straight line through the origin with slope $\frac{\partial E_0(\rho, \mathbf{q})}{\partial \rho}$. By virtue of the $\cap$-convexity of $E_0(\rho, \mathbf{q})$, the point $(\rho, E_0(\rho, \rho))$ lies below $(\rho, \rho \frac{\partial E_0(\rho, \mathbf{q})}{\partial \rho})$, and the derivative (6.58) 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 $E_0(\rho, \mathbf{q}) \to 0$ as $\rho \to 0$, which can easily be seen from (6.49). Let us find out at what limiting rate R the error exponent approaches its value of zero; that is, let us calculate

$$\lim_{\rho \to 0} R(\rho) = \lim_{\rho \to 0} \frac{E_0(\rho, \mathbf{q})}{\rho}. \tag{6.59}$$

Since this limit is of the form $\frac{0}{0}$, we can use the rule of de l'Hôpital and obtain the limit

$$\lim_{\rho \to 0} R(\rho) = \left. \frac{\partial E_0(\rho, \mathbf{q})}{\partial \rho} \right|_{\rho=0}. \tag{6.60}$$

This derivative is easily evaluated, and to our great surprise and satisfaction we obtain

$$\left. \frac{\partial E_0(\rho, \mathbf{q})}{\partial \rho} \right|_{\rho=0} = \int_y \sum_x q(x) p(y|x) \log_2 \left(\frac{p(y|x)}{\sum_{x'} q(x') p(y|x')} \right) dy = C(\mathbf{q}), \tag{6.61}$$

the Shannon channel capacity using the signal set $\mathcal{A}$ and the input probability distribution $\mathbf{q}$! (See also ref. 4 and 13.) Maximizing over the input probability distribution $\mathbf{q}$ will then achieve the channel capacity C.

We have now proved the important fact that group-trellis codes can achieve the Shannon capacity, and, as we will see, with a very favorable error exponent (6.49). The bounds thus derived are an important confirmation of our decision to use group-trellis codes, since they tell us that these codes can 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

$$p(y|x) = \frac{1}{2\pi N_0} \exp\left(-\frac{|y-x|^2}{2N_0} \right). \tag{6.62}$$

Furthermore, for $\rho = 1$, $E_0(\rho, \mathbf{q})$ can be simplified considerably by evaluating the integral over y, that is,

$$\begin{aligned} E_0(1, \mathbf{q}) &= -\log_2 \int_y \left(\sum_x q(x) \frac{1}{\sqrt{2\pi N_0}} \exp\left(-\frac{|y-x|^2}{4N_0} \right) \right)^2 dy \\ &= -\log_2 \sum_x \sum_{x'} q(x) q(x') \int_y \frac{1}{2\pi N_0} \exp\left(-\frac{|y-x|^2}{4N_0} - \frac{|y-x'|^2}{4N_0} \right) dy \\ &= -\log_2 \sum_x \sum_{x'} q(x) q(x') \exp\left(-\frac{|x-x'|^2}{8N_0} \right) = R_0(\mathbf{q}), \end{aligned} \tag{6.63}$$

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 [7, 14] that R_0 represents a "practical" limit for coded systems. Furthermore, in Chapter 7 we will see that certain decoding algorithms will exhibit an unbounded computation time for rates which exceed R_0. However, with the appearance of turbo codes the significance of R_0 as a measure of practical importance has evaporated.

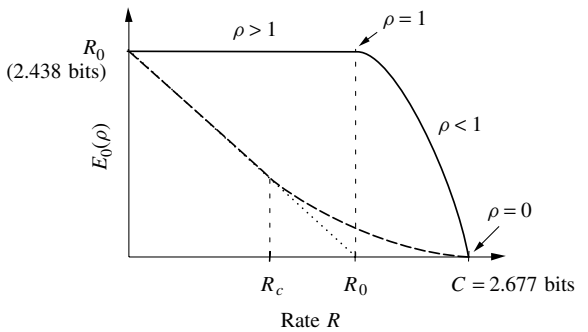


Figure 6.10 Error exponent as a function of the rate R for an 8-PSK constellation on an AWGN channel at a signal-to-noise ratio of $E_s/N_0 = 10$ dB.

On AWGN channels the uniform input symbol distribution $\mathbf{q} = 1/A, \ldots, 1/A$ is particularly popular, and we define $E_0(\rho, \text{uniform distribution}) = E_0(\rho)$ and $R_0(\text{uniform distribution}) = R_0$; that is, we simply omit the probability distribution vector $\mathbf{q}$.

Figure 6.10 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 exhibits the typical behavior, known from error bounds for convolutional codes [13], having a constant value of R_0 up to the rate $R = R_0$ and then rapidly dropping to zero as $R \to C$. We have also shown the error exponent for block codes[7] (dashed curve) in Figure 6.10. A very thorough discussion about block code error exponents can be found in either ref. 13 or ref. 6.

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 block code error exponent multiplies the block code length, this is taken as an indication that the constraint length ν needed to obtain a prescribed error probability $\overline{P_b}$ is much smaller than the block length 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.

[7]The block error probability of the ensemble of block codes of length N is given by (ref. 13, Section 3.1)

$$\overline{P_B} < 2^{-NE(R)}, \qquad E(R) = \max_{\mathbf{q}} \max_{0 \le \rho \le 1} \left[E_0(\rho, \mathbf{q}) - \rho R \right].$$

APPENDIX 6.A

The following theorem is also known as *Gerschgorin's circle theorem*:

Theorem 6.6 *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 $r_i = \sum_{j \neq i} |p_{ij}|$; that is, r_i is the ith absolute row sum without the diagonal element.*

Proof: $\mathbf{P}\boldsymbol{x} = \lambda \boldsymbol{x}$ immediately leads to

$$(\lambda - p_{ii})x_i = \sum_{j \neq i} p_{ij} x_j. \tag{6.64}$$

Taking absolute values on both sides and applying the triangle inequality we obtain

$$|\lambda - p_{ii}| \leq \sum_{j \neq i} |p_{ij}||x_j|/|x_i|. \tag{6.65}$$

Now let x_i be the largest component of $\boldsymbol{x}$, then $|x_j|/|x_i| \leq 1$, and $|\lambda - p_{ii}| \leq r_i$. Q.E.D.

In the application in Section 6.4, all entries p_{ij} are greater than or equal to 0, and hence

$$(\lambda - p_{ii}) \leq \sum_{j \neq i} p_{ij} \Rightarrow \lambda \leq \sum_{j} p_{ij}. \tag{6.66}$$

Now, the largest eigenvalue $\lambda_{\max}$ must be smaller than the largest row sum, that is,

$$\lambda_{\max} \leq \max_i \sum_j p_{ij}. \tag{6.67}$$

APPENDIX 6.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 it 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 that produce identical sets of outputs $\delta(U \rightarrow U')$ as we go through all transitions $U \rightarrow U'$ (compare Definition 6.1).

Step 2: Obtain the $(l+1)$th partition P_{l+1} from the lth 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.

(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—that is, 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}, and so on, 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 FSMs. The interested reader will find a more complete discussion in ref. 5, Chapter 4.

BIBLIOGRAPHY

1. F. L. Alvarado, "Manipulation and visualization of sparse matrices," *ORSA J. Comput.*, vol. 2, no. 2, Spring 1990.
2. E. Biglieri, "High-level modulation and coding for nonlinear satellite channels," *IEEE Trans. Commun.,* vol. COM-32, pp. 616–626, May 1984.
3. 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.
4. T. M. Cover and J. A. Thomas, *Elements of Information Theory*, John Wiley & Sons, New York, 1991.
5. P. J. Denning et al., *Machines, Language and Computation*, Prentice-Hall, Englewood Cliffs, NJ, 1978.
6. R. G. Gallager, *Information Theory and Reliable Communication,* John Wiley & Sons, New York, 1968.
7. J. L. Massey, "Coding and modulation in digital communications," *Proceedings, International Zürich Seminar on Digital Communication*, Zürich, Switzerland, March 1974, pp. E2(1)–E2(4).
8. 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.
9. 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.
10. C. E. Shannon, "A mathematical theory of communications," *Bell Syst. Tech. J.,* vol.27, pp. 379–423, July 1948.

11. G. Strang, *Linear Algebra and Its Applications*, Harcourt Brace Jovanovich, San Diego, 1988.
12. G. Ungerboeck, "Channel coding with multilevel/phase signals," *IEEE Trans. Inform. Theory,* vol. IT-28, no. 1, pp. 55–67, Jan. 1982.
13. A. J. Viterbi and J. K. Omura, *Principles of Digital Communication and Coding,* McGraw-Hill, New York, 1979.
14. 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.
15. J. M. Wozencraft and I. M. Jacobs, *Principles of Communication Engineering*, John Wiley & Sons, New York, 1965.
16. 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.
17. 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.
18. W. Zhang, "Finite-state systems in mobile communications," Ph.D. dissertation, University of South Australia, June 1995.

CHAPTER 7

Decoding Strategies

7.1 BACKGROUND AND INTRODUCTION

There are a great variety of decoding algorithms for trellis codes, 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 by so-called sequence decoders. These sequence decoders fall into two main groups: the tree decoders and the 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 Viterbi algorithm.

Recently, and in conjunction with the emergence of Turbo coding, symbol probability decoding algorithms have become prominent. They calculate the reliability of individual transmitted or information symbols, rather than decoding sequences. Symbol probability decoding algorithms are essential for the iterative algorithms used to decode large concatenated codes, such as turbo codes, and their importance will eclipse that of sequence decoders. 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 with the trellis of the code, and is discussed in detail in Section 7.7.

Let us now set the stage for the discussion of these decoding algorithms. In Chapters 2 and 3 we have discussed how a trellis encoder generates a sequence $\boldsymbol{x}^{(i)} = (x_{-l}^{(i)}, \ldots, x_l^{(i)})$ 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

$$s^{(i)}(t) = \sum_{r=-l}^{l} x_r^{(i)} p(t - rT). \tag{7.1}$$

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

Trellis and Turbo Coding, by Christian B. Schlegel and Lance C. Pérez
ISBN 0-471-22755-2

select the message which corresponds to the signal that produces the largest sampled output value.

The impulse response of the matched filter for $s^{(i)}(t)$ is given by

$$s^{(i)}(-t) = \sum_{r=-l}^{l} x_r^{(i)} p(-t - rT); \tag{7.2}$$

and if $r(t)$ is the received signal, the sampled response of the matched filter (7.2) to $r(t)$ is given by

$$\begin{aligned} \boldsymbol{r} \cdot \boldsymbol{s}^{(i)} &= \int_{-\infty}^{\infty} r(t) s^{(i)}(t)\, dt \\ &= \sum_{r=-l}^{l} x_r^{(i)} y_r = \boldsymbol{x}^{(i)} \cdot \boldsymbol{y}, \end{aligned} \tag{2.21}$$

where $y_r = \int_{-\infty}^{\infty} r(\alpha) p(\alpha - rT)$ is the output of the filter matched to the pulse $p(t)$ sampled at time $t = rT$ as discussed in Section 2.5 [Equation (2.24)], and $\boldsymbol{y} = (y_{-l}, \ldots, y_l)$ 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

$$\begin{aligned} |\boldsymbol{s}^{(i)}|^2 &= \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} s^{(i)}(\alpha) s^{(i)}(\beta)\, d\alpha d\beta \\ &= \sum_{r=-l}^{l} |x_r^{(i)}|^2, \end{aligned} \tag{7.3}$$

and, from (2.19), the maximum likelihood receiver will select the sequence $\boldsymbol{x}^{(i)}$ which maximizes

$$J^{(i)} = 2 \sum_{r=-l}^{l} \mathrm{Re}\left\{ x_r^{(i)} y_r^* \right\} - \sum_{r=-l}^{l} |x_r^{(i)}|^2, \tag{7.4}$$

where we have extended (2.19) to complex signals.

$J^{(i)}$ in Equation (7.4) is called the *metric* of the sequence $\boldsymbol{x}^{(i)}$, and this metric is to be maximized over all allowable choices of $\boldsymbol{x}^{(i)}$.

7.2 TREE DECODERS

From (7.4) we define the partial metric at time n as

$$J_n^{(i)} = 2 \sum_{r=-l}^{n} \mathrm{Re}\left\{ x_r^{(i)} y_r^* \right\} - \sum_{r=-l}^{n} |x_r^{(i)}|^2, \tag{7.5}$$

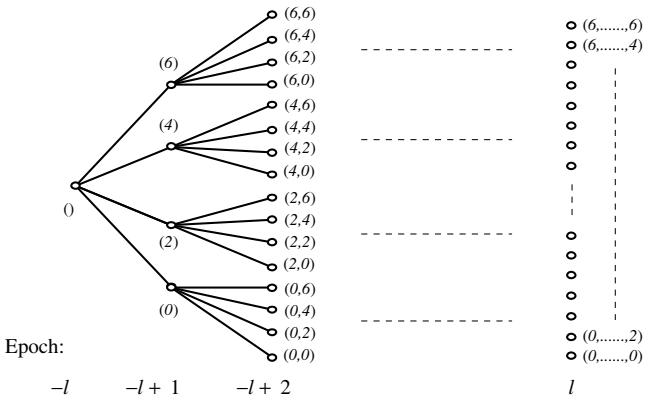


Figure 7.1 Code tree extending from time $-l$ to time l for the code from Figure 3.1, Chapter 3.

which allows us to rewrite (7.4) in the recursive form

$$J_n^{(i)} = J_{n-1}^{(i)} + 2\mathrm{Re}\left\{x_n^{(i)} y_n^*\right\} - |x_n^{(i)}|^2. \tag{7.6}$$

Equation (7.6) implies a tree structure that can be used to evaluate the metrics for all the allowable signal sequences as illustrated in Figure 7.1 for the trellis 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 and k is the number of bits carried by x_n. Each node is labeled with the hypothesized partial sequence[1] $\tilde{\boldsymbol{x}}^{(i)} = (x_{-l}^{(i)}, \ldots, x_n^{(i)})$ 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 (7.6) until time unit $n = l$, at which time the largest accumulated metric identifies the most likely sequence $\boldsymbol{x}^{(i)}$.

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—that is, short encoded sequences. We therefore need to reduce the complexity of decoding in some appropriate way, and this can be done 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 $\tilde{\boldsymbol{x}}^{(i)}$ with lengths $\{n_i\}$, and let $n_{\max} = \max\{n_1, \ldots, n_M\}$ be the maximum length among the M partial

[1]We denote partial sequences by tildes to distinguish them from complete sequences or code words.

sequences. The decoder must make its likelihood ranking of the paths based on the partial received sequence $\tilde{\boldsymbol{y}}$ of length $n_{\max}$.

From (2.10) we know that an optimum receiver would choose the $\tilde{\boldsymbol{x}}^{(i)}$ which maximizes

$$P[\tilde{\boldsymbol{x}}^{(i)}|\tilde{\boldsymbol{y}}] = P[\tilde{\boldsymbol{x}}^{(i)}]\frac{\prod_{r=-l}^{n_i} p_n(y_r - x_r)\prod_{n_i+1}^{n_{\max}} p(y_r)}{p(\tilde{\boldsymbol{y}})}, \tag{7.7}$$

where the second product reflects the fact that we have no hypotheses $x_r^{(i)}$ for $r > n_i$, since $\tilde{\boldsymbol{x}}^{(i)}$ extends only up to n_i. We therefore have to use the a priori probabilities $p(y_r|x_r^{(i)}) = p(y_r)$ for $r > n_i$. Using $p(\tilde{\boldsymbol{y}}) = \prod_{r=-l}^{n_{\max}} p(y_r)$, Equation (7.7) can be rewritten as

$$P[\tilde{\boldsymbol{x}}^{(i)}|\tilde{\boldsymbol{y}}] = P[\tilde{\boldsymbol{x}}^{(i)}] \prod_{r=-l}^{n_i} \frac{p_n(y_r - x_r^{(i)})}{p(y_r)}, \tag{7.8}$$

and we see that we need not be concerned with the tail samples not affected by $\tilde{\boldsymbol{x}}^{(i)}$. Taking logarithms gives the "additive" metric

$$L(\tilde{\boldsymbol{x}}^{(i)}, \tilde{\boldsymbol{y}}) = \sum_{r=-l}^{n_i} \log \frac{p_n(y_r - x_r^{(i)})}{p(y_r)} - \log \frac{1}{P[\tilde{\boldsymbol{x}}^{(i)}]}. \tag{7.9}$$

Since $P[\tilde{\boldsymbol{x}}^{(i)}] = (2^{-k})^{n_i+l}$ is the a priori probability of the partial sequence $\tilde{\boldsymbol{x}}^{(i)}$, assuming that all the inputs to the trellis encoder have equal probability, (7.9) becomes

$$L(\tilde{\boldsymbol{x}}^{(i)}, \tilde{\boldsymbol{y}}) = L(\tilde{\boldsymbol{x}}^{(i)}, \boldsymbol{y}) = \sum_{r=-l}^{n_i} \left[\log \frac{p_n(y_r - x_r^{(i)})}{p(y_r)} - k \right], \tag{7.10}$$

where we have extended $\tilde{\boldsymbol{y}} \to \boldsymbol{y}$ since (7.10) ignores the tail samples $y_r; r > n_i$ anyhow. The metric (7.10) was introduced for decoding tree codes by Fano [15] in 1963, and was analytically derived by Massey [29] in 1972.

Since Equation (2.11) explicitly gives the conditional probability distribution $p_n(y_r - x_r)$, the metric in (7.10) can be specialized for additive white Gaussian noise channels to

$$\begin{aligned} L(\tilde{\boldsymbol{x}}^{(i)}, \boldsymbol{y}) &= \sum_{r=-l}^{n_i} \left[\log \frac{\exp\left(-|x_r^{(i)} - y_r|^2/N_0\right)}{\sum_{x\in\mathcal{A}} p(x)\exp\left(-|x - y_r|^2/N_0\right)} - k \right] \\ &= -\sum_{r=-l}^{n_i} \frac{|x_r^{(i)} - y_r|^2}{N_0} - c_r(y_r), \end{aligned} \tag{7.11}$$

where $c_r(y_r) = \log\left(\sum_{x\in\mathcal{A}} p(x)\exp\left(-|x-y_r|^2/N_0\right)\right) + k$ 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 $-\sum_{r=-l}^{n} c_r$ in their metrics, which therefore may be discarded from all the metrics. This allows us to simplify (7.11) to

$$L(\tilde{\boldsymbol{x}}^{(i)}, \boldsymbol{y}) \equiv \sum_{r=-l}^{n} 2\mathrm{Re}\left\{x_r^{(i)} y_r^*\right\} - |x_r^{(i)}|^2 = J_n^{(i)}, \tag{7.12}$$

by neglecting terms common to all the metrics. The metric (7.12) is equivalent to the accumulated squared Euclidean distance between the received partial sequence $\tilde{\boldsymbol{y}}$ and the hypothesized symbols on the ith path to up to length n. The restriction to path of equal length makes this metric much simpler than the general metric (7.10) [and (7.11)] and finds application in the so-called *breadth-first* decoding algorithms that we will discuss in subsequent sections.

7.3 THE STACK ALGORITHM

The stack algorithm is one of the many variants of what has become known as *sequential decoding* of trellis codes. Sequential decoding was introduced by Wozencraft and Reiffen [40] for convolutional codes and 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 (7.10) [or (7.11)] 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 [45] and Jelinek [25]. This basic algorithm is as follows:

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 $L((), \boldsymbol{y}) = 0$.

Step 2: Extend the node corresponding to the top entry $\left\{\tilde{\boldsymbol{x}}_{\text{top}}, L(\tilde{\boldsymbol{x}}_{\text{top}}, \boldsymbol{y})\right\}$ by forming $L(\tilde{\boldsymbol{x}}_{\text{top}}, \boldsymbol{y}) - |x_r - y_r|^2/N_0 - c_r$ for all 2^k extensions of $\tilde{\boldsymbol{x}}_{\text{top}} \to (\tilde{\boldsymbol{x}}_{\text{top}}, x_r) = \tilde{\boldsymbol{x}}^{(i)}$. Delete $\left\{\tilde{\boldsymbol{x}}_{\text{top}}, L(\tilde{\boldsymbol{x}}_{\text{top}}, \boldsymbol{y})\right\}$ from the stack.

Step 3: Place the new entries $\left\{\tilde{\boldsymbol{x}}^{(i)}, L(\tilde{\boldsymbol{x}}^{(i)}, \boldsymbol{y})\right\}$ from Step 2 onto the stack and resort the stack such that the largest metric is at the top.

Step 4: If the top entry of the stack is a path to one of the terminal nodes at depth l, stop and select $\boldsymbol{x}_{\text{top}}$ 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 that the algorithm 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 7.9. 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—that is, 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

$$j\Delta \leq L(\tilde{\boldsymbol{x}}^{(i)}, \boldsymbol{r}) \leq (j+1)\Delta, \tag{7.13}$$

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 losing 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 refs. 2, 3, 20, and 24.

7.4 THE FANO ALGORITHM

Unlike the stack algorithm, the Fano algorithm is a depth-first tree search procedure in its purest form. Introduced by Fano [15] 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 [21], but for moderate rates the Fano algorithm decodes faster than the stack algorithm [22]. 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 nonconnected nodes; that is, 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 tradeoff 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 7.2 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 7.2 is used to illustrate the values of the metrics for each node. Also assume that the paths shown are those with the best metrics; that is, 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

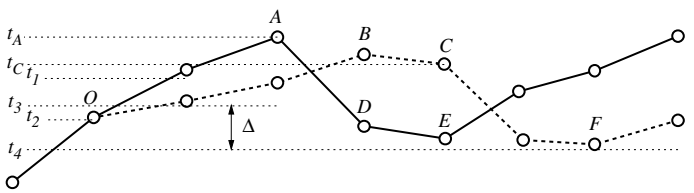


Figure 7.2 Illustration of the operation of the Fano algorithm when choosing between two competing paths.

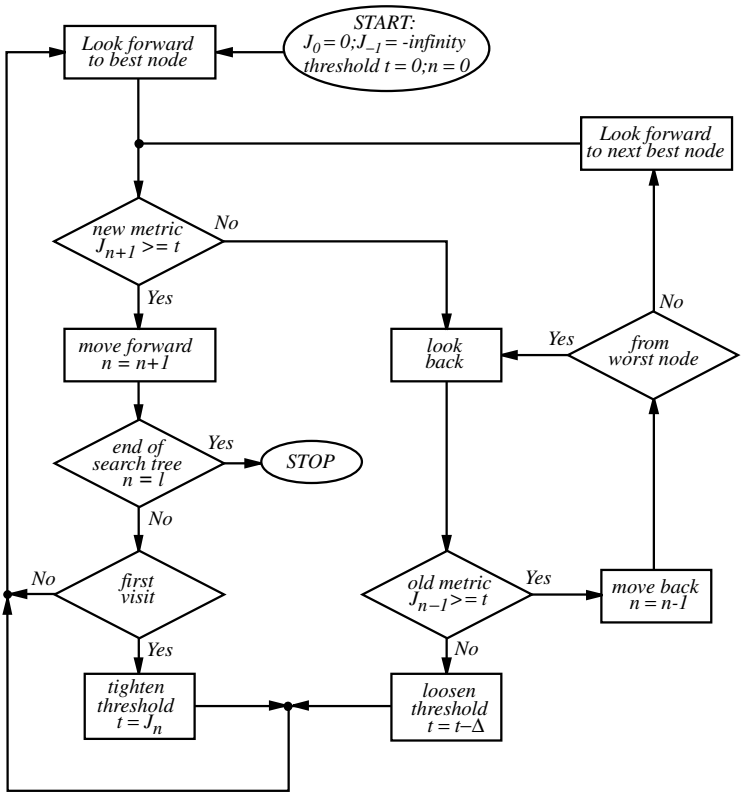


Figure 7.3 Flowchart of the Fano algorithm. The initialization of $J_{-1} = -\infty$ has the effect that the algorithm can lower the threshold for the first step, if necessary.

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.

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 that 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 7.3 shows the flowchart of the Fano algorithm.

7.5 THE M-ALGORITHM

The M-algorithm is a purely breadth-first synchronous algorithm that 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 (7.12) can be used. The algorithm is very simple:

Step 1: Initialize an empty list L of candidate paths and their metrics. Deposit the zero-length path () with its metric $L((), \boldsymbol{y}) = 0$ in the list. Set $n = -l$.

Step 2: Extend 2^k partial paths $\tilde{\boldsymbol{x}}^{(i)} \rightarrow (\tilde{\boldsymbol{x}}^{(i)}, x_r^{(i)})$ from each of the at most M paths $\tilde{\boldsymbol{x}}^{(i)}$ 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 extensions[2] 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.

The M-algorithm appeared in the literature for the first time in a paper by Jelinek and Anderson [26], 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 [3], along with ref. 1, collects a lot of this material and 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 $L(\tilde{\boldsymbol{x}}^{(i)}, \boldsymbol{y}) \geq \lambda_t - T$ 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.

In Chapter 3 we have discussed that the performance of trellis codes using a maximum-likelihood detector was governed by the distance spectrum of the

[2]Note that from two or more extensions leading to the same state (see Section 7.6), all but the one with the best metric may be discarded. This will improve performance slightly by eliminating some paths that cannot be the ML path.

code, where the minimum free Euclidean distance d_{free} played a particularly important role. Since the M-algorithm is a suboptimum decoding algorithm, its performance is additionally affected by other criteria. The major criterion is the probability 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. We will see that the probability of correct path loss has no direct connection with the distance spectrum of a code.

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 d_{free} is appropriately large. If this is the case, the correct path loss becomes the dominant error event 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 7.4. 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 $\tilde{\boldsymbol{x}}^{(c)}$. The optimal strategy of the decoder will then be to retain that particular set of M candidates which maximizes

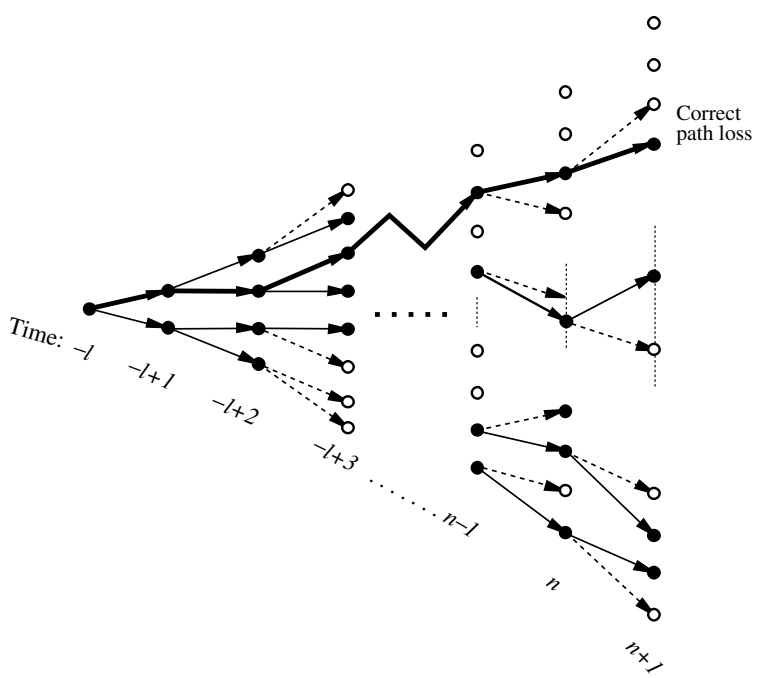


Figure 7.4 Extension of $2^k M = 2 \cdot 4$ paths from the list of the best $M = 4$ paths. The solid paths are those retained by the algorithm, the path indicated by the heavy line corresponds to the correct transmitted sequence.

the probability of containing $\tilde{\boldsymbol{x}}^{(c)}$. Let $\mathcal{C}_p$ be one of the $\binom{M2^k}{M}$ possible sets of M candidates at time n. We wish to maximize

$$\max_p \Pr\left\{\tilde{\boldsymbol{x}}^{(c)} \in \mathcal{C}_p | \tilde{\boldsymbol{y}}\right\}. \tag{7.14}$$

Since all the partial paths $\tilde{\boldsymbol{x}}^{(p_j)} \in \mathcal{C}_p$, $j = 1, \ldots, M$ are distinct, the events $\{\tilde{\boldsymbol{x}}^{(c)} = \tilde{\boldsymbol{x}}^{(p_j)}\}$ are all mutually exclusive for different j; that is, the correct path can be at most only one of the M different candidates $\tilde{\boldsymbol{x}}^{(p_j)}$. Equation (7.14) can therefore be evaluated as

$$\max_p \Pr\left\{\tilde{\boldsymbol{x}}^{(c)} \in \mathcal{C}_p | \tilde{\boldsymbol{y}}\right\} = \max_p \sum_{j=1}^{M} \Pr\left\{\tilde{\boldsymbol{x}}^{(c)} = \tilde{\boldsymbol{x}}_j^{(p_j)} | \tilde{\boldsymbol{y}}\right\}. \tag{7.15}$$

From (7.7), (7.9), and (7.12) we know that

$$\Pr\left\{\tilde{\boldsymbol{x}}^{(c)} = \tilde{\boldsymbol{x}}^{(p_j)} | \tilde{\boldsymbol{y}}\right\} \propto \exp\left(-\sum_{r=-l}^{-l+n}\left(2\mathrm{Re}\left\{x_r^{(p_j)} y_r^*\right\} - |x_r^{(p_j)}|^2\right)\right), \tag{7.16}$$

where the proportionality constant is independent of $\tilde{\boldsymbol{x}}^{(p_j)}$. The maximization in (7.14) now becomes equivalent to (considering only the exponent from above)

$$\begin{aligned}\max_p \Pr\left\{\tilde{\boldsymbol{x}}^{(c)} \in \mathcal{C}_p | \tilde{\boldsymbol{y}}\right\} &\equiv \max_p \sum_{j=1}^{M} \sum_{r=-l}^{-l+n}\left(2\mathrm{Re}\left\{x_r^{(p_j)} y_r^*\right\} - |x_r^{(p_j)}|^2\right) \\ &= \max_p J_n^{(p_j)},\end{aligned} \tag{7.17}$$

that is, we simply collect the M paths with the best partial metrics $J_n^{(p_j)}$ at time n. This argument was derived by Aulin [4]. Earlier we showed that the total metric can be broken up into the recursive form of (7.6), 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 $\tilde{\boldsymbol{x}}^{(p_j)}$ with maximum partial metrics is the optimal strategy.

The probability of correct path loss, denoted by Pr(CPL), can now be addressed. Following the methodology of Aulin [4], we need to evaluate the probability that the correct path $\tilde{\boldsymbol{x}}^{(c)}$ is not among the M candidate paths. This will happen if M paths $\tilde{\boldsymbol{x}}^{(p_j)} \neq \tilde{\boldsymbol{x}}^{(c)}$ have a partial metric $J_n^{(p_j)} \geq J_n^{(c)}$, or equivalently if all the M metric differences

$$\delta_n^{(j,c)} = J_n^{(c)} - J_n^{(p_j)} = \sum_{r=-l}^{-l+n}\left(|x_r^{(p_j)}|^2 - |x_r^{(c)}|^2 - 2\mathrm{Re}\left\{x_r^{(p_j)} - x_r^{(c)}\right\} y_r^*\right) \tag{7.18}$$

are smaller than or equal to zero. That is,

$$\Pr(\text{CPL}|\mathcal{C}_\text{p}) = \Pr\{\boldsymbol{\delta}_n \leq \mathbf{0}\}, \qquad \tilde{\boldsymbol{x}}^{(p_j)} \in \mathcal{C}_p, \tag{7.19}$$

where $\boldsymbol{\delta}_n = \left(\delta_n^{(1,c)}, \ldots, \delta_n^{(M,c)}\right)$ 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 $\tilde{\boldsymbol{x}}^{(c)}$. $\Pr(\text{CPL}|\mathcal{C}_\text{p})$ depends on the correct path $\boldsymbol{x}^{(c)}$ and, strictly speaking, has to be averaged over all correct paths. We shall be satisfied with the correct path which produces the largest $\text{P}(\text{CPL}|\mathcal{C}_\text{p})$.

In Appendix 7.A we show that the probability of losing the correct path decreases exponentially with the signal-to-noise ratio and is overbounded by

$$\Pr(\text{CPL}|\mathcal{C}_\text{p}) \leq Q\left(\sqrt{\frac{d_l^2}{2N_0}}\right). \tag{7.20}$$

The parameter d_l^2 depends on $\mathcal{C}_p$ and is known as the *Vector Euclidean distance* [4] of the path $\tilde{\boldsymbol{x}}^{(c)}$ with respect to the M error paths $\tilde{\boldsymbol{x}}^{(p_i)} \in \mathcal{C}_p$. It is important to note here that (7.20) is an upper bound of the probability that M specific error paths have a metric larger than $\tilde{\boldsymbol{x}}^{(c)}$. Finding d_l^2 involves a combinatorial search.

Equation (7.20) demonstrates that the probability of correct path loss is an exponential error integral, and can thus be compared to the probability of the maximum likelihood decoder [Equations (6.8 and (6.9)]. 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 [4] has studied this problem and gives several rejection rules which alleviate the complexity of finding d_l^2, but the problem remains complex and is in need of further study.

Note that d_l^2 is a nondecreasing function of M, the decoder complexity, and one way of selecting M is to choose it such that

$$\min(d_l^2) \geq d_\text{free}^2. \tag{7.21}$$

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 (6.5)]. Then

$$\begin{aligned} \overline{P_e(M)} &\leq \overline{P_e}\,(1 - \Pr(\text{CPL})) + \Pr(\text{CPL}) \\ &\leq \overline{P_e} + \Pr(\text{CPL}), \end{aligned} \tag{7.22}$$

where $\overline{P_e}$ is of course the probability that a maximum-likelihood decoder starts an error event (Chapter 6). For high values of the signal-to-noise ratio, Equation

(7.22) can be approximated by

$$\overline{P_e(M)} \approx N_{d_{\text{free}}} Q\left(\frac{d_{\text{free}}}{\sqrt{2N_0}}\right) + \kappa Q\left(\frac{\min(d_l)}{\sqrt{2N_0}}\right), \tag{7.23}$$

where κ is some constant, which, however, is difficult to determine in the general case. Now, if (7.21) holds, the suboptimality does not exponentially dominate the error performance for high signal-to-noise ratios.

Aulin [4] has analyzed this situation for 8-PSK trellis codes and found that, in general, $M \approx \sqrt{S}$ will fulfill condition (7.21), where S is the number of states in the code trellis.

Figure 7.5 shows the simulated performance of the M-algorithm versus M for the 64-state trellis code from Table 3.1 with $d^2_{\text{free}} = 6.34$. $M = 8$ meets (7.21) according to ref. 4, but from Figure 7.5 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^2_{free} nor $\min(d_l^2)$ are necessarily dominating the error performance.

Figures 7.6 and 7.7 show the empirical probability of correct path loss P(CPL) and the BER for two convolutional codes and various values of M. Figure 7.6

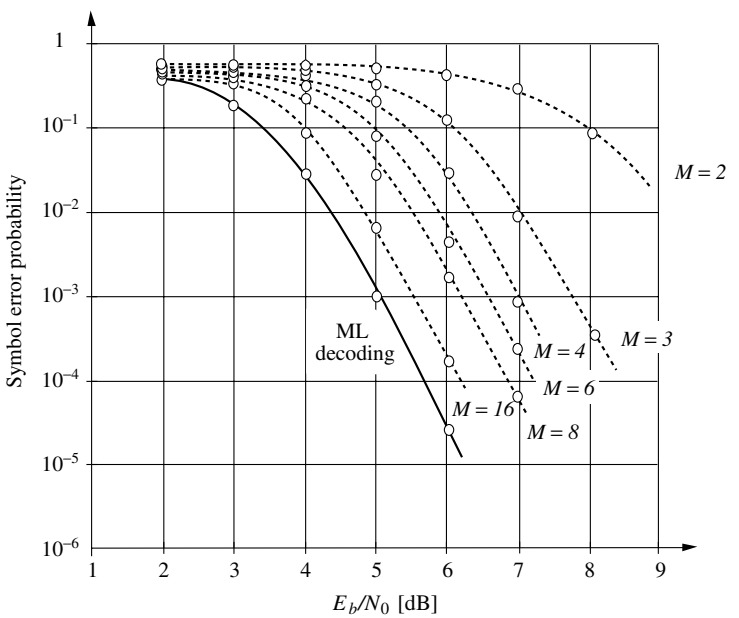


Figure 7.5 Simulation results for the 64-state optimal distance 8-PSK trellis codes decoded with the M-algorithm, using $M = 2, 3, 4, 5, 6, 8$ and 16. The performance of maximum likelihood decoding is also included in the figure (Aulin [4]).

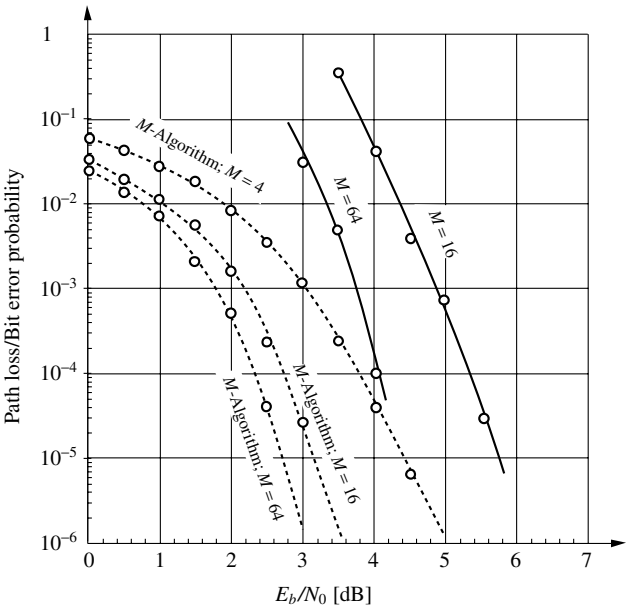


Figure 7.6 Simulation results for the 2048-state, rate $R = 1/2$ convolutional code using the M-algorithm, for $M = 4$, 16, and 64. The dashed curves are the path loss probability, and the solid curves are BERs.

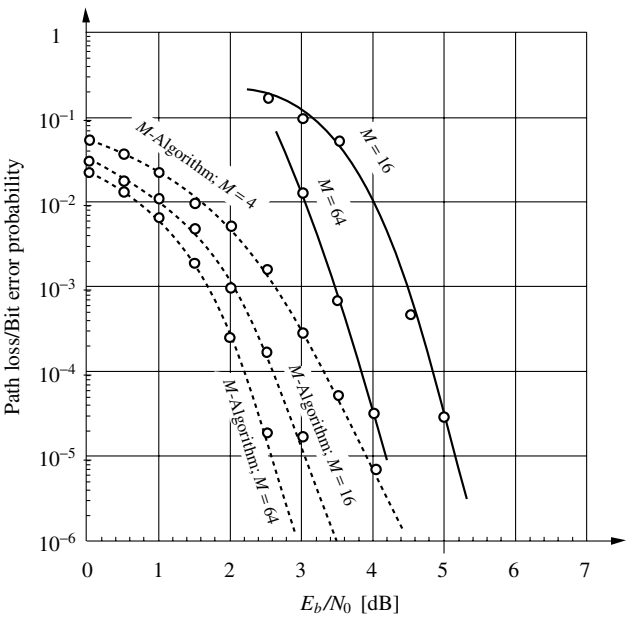


Figure 7.7 Same simulation results for the $\nu = 15$, $R = 1/2$ convolutional code.

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 7.7 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 relatively long time before it finds it again—that is, before the correct node in the code tree is again among the M retained nodes. Correct path recovery is a very complex problem, and no complete analytical results have been found to date. There are only a few theoretical approaches to the recovery problem, such as ref. 5. This difficulty suggests that insight into the operation of the decoder during a recovery has to be gained through simulation studies. Note that correct path recovery is somewhat of a misnomer since the algorithm only rejoins the correct path. It cannot recover the lost section since it does not back track.

Figure 7.8 shows the simulated average number of steps taken for the algorithm to recover the correct path. The simulations were done for the 2048-state, $\nu = 11$ code, whose error performance is shown in Figure 7.6. 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 [27].

Figure 7.9 shows the average number of steps until recovery for the rate $1/2$, $\nu = 11$ systematic convolutional code with generator polynomials $g^{(0)} =$

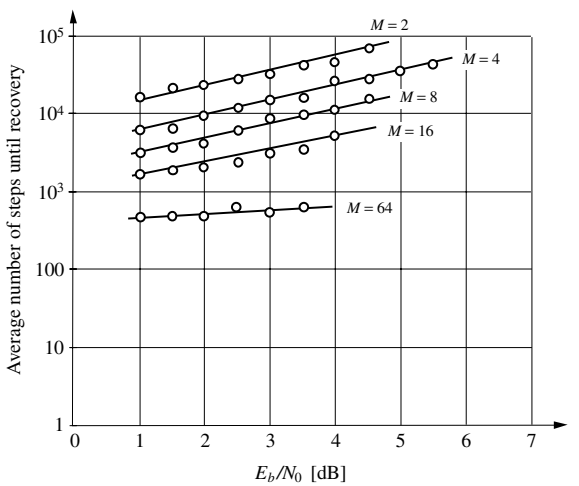


Figure 7.8 Average number of steps until recovery of the correct path for the code from Figure 7.6 (Ma [27]).

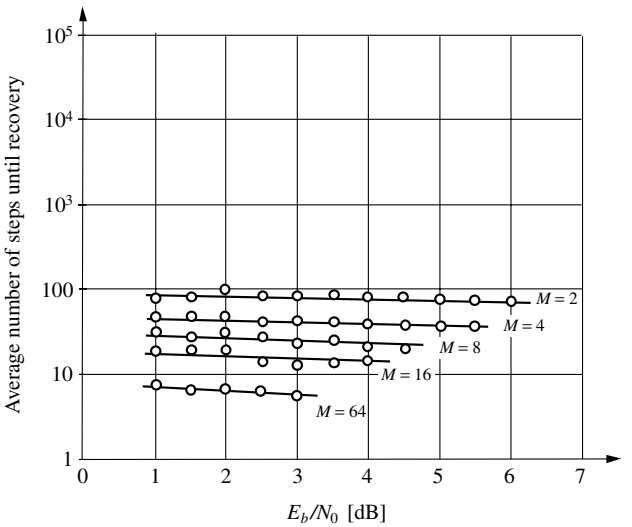


Figure 7.9 Average number of steps until recovery of the correct path for the systematic convolutional code with $\nu = 11$ (Ma [27]).

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 nonsystematic code. In fact, the average number of steps until recovery is independent of the signal-to-noise ratio, while it increases approximately linearly with E_b/N_0 for the nonsystematic code. This rapid recovery results in superior error performance of the systematic code compared to the nonsystematic code, shown in Figure 7.10, even though its free distance is significantly smaller. What is true, however, and can be seen clearly in Figure 7.10 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 nonsystematic codes for error rates $P_b \gtrapprox 10^{-6}$ has also been made by Osthoff et al. [30]. The reason for this difference lies in the *return barrier* phenomenon, which can be explained with the aid of Figure 7.11. 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 with a connection to the correct state at the next time interval. In Figure 7.11 we assume that the all-zero sequence is the correct sequence, and hence the all-zero state is the correct state for all time intervals. 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 $(0, \ldots, 0, 1)$, 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 7.11.

For a nonsystematic, maximum free distance code on the other hand, the two branch signals are (11) and (00), respectively. Since the correct branch signal is

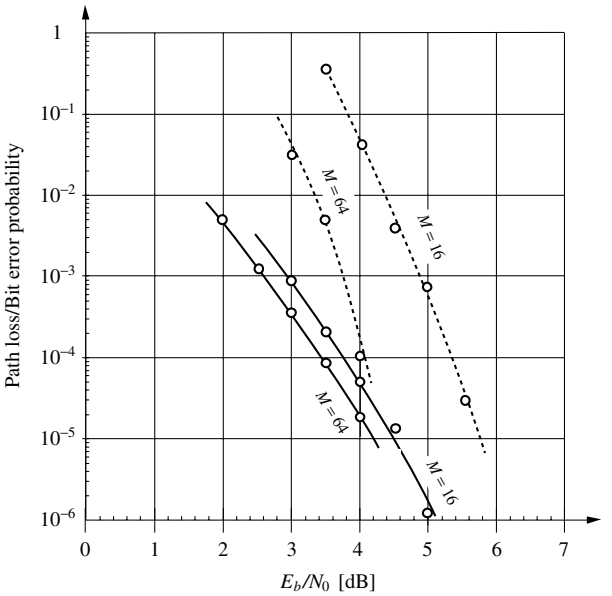


Figure 7.10 Simulation results for the superior 2048-state systematic code using the M-algorithm. The dashed curves are the error performance of the same constraint length nonsystematic code from Figure 7.6 (Ma [27]).

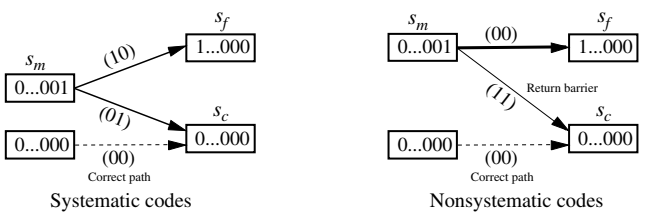


Figure 7.11 Heuristic explanation of the return barrier phenomenon in the M-algorithm.

(00), the probability that the metric of s_f (for **f**ailed) exceeds the metric of s_c equals $1/2$ for the systematic code, since both branch signals are equidistant from the correct branch signal. For the nonsystematic 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 nonsystematic 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 [1–3, 30], was presented by Zigangirov and Kolesnik [46], while Simmons and Wittke [36], Aulin [6], and Balachandran [8], 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 [32].

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 nonbinary modulations such as CPM, coded modulation, and code-division multiple access, where it may have a place in practical implementations.

7.6 MAXIMUM LIKELIHOOD DECODING

The difficulty in decoding trellis codes 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 Section 3.

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 $\tilde{\boldsymbol{x}}^{(i)}$ and $\tilde{\boldsymbol{x}}^{(j)}$ of length $n-1$, respectively. Let the encoder states which correspond to $\tilde{\boldsymbol{x}}^{(i)}$ and $\tilde{\boldsymbol{x}}^{(j)}$ 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 $\tilde{\boldsymbol{x}}^{(i)} \rightarrow (\tilde{\boldsymbol{x}}^{(i)}, x_n^{(i)})$ and $\tilde{\boldsymbol{x}}^{(j)} \rightarrow (\tilde{\boldsymbol{x}}^{(j)}, x_n^{(j)})$ is such that $s_n^{(i)} = s_n^{(j)}$; that is, the encoder states at time n are identical. See also Figure 7.12.

Now we propose to merge the two nodes $(\tilde{\boldsymbol{x}}^{(i)}, x_n^{(i)})$ and $(\tilde{\boldsymbol{x}}^{(j)}, x_n^{(j)})$ 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 7.1 (Theorem of Nonoptimality) *The procedure of merging nodes, which correspond to identical encoder states, and discarding the path with the smaller metric never eliminates the maximum-likelihood path.*

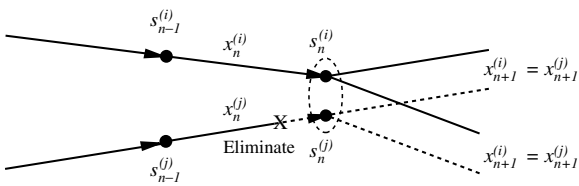


Figure 7.12 Merging nodes.

Theorem 7.1 is sometimes referred to as the theorem of nonoptimality 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 $n + r$ for path i can be written as

$$J_{n+r}^{(i)} = J_n^{(i)} + \sum_{h=1}^{r} \beta_{n+h}^{(i)} \tag{7.24}$$

for every future time index $n + r, 0 < r \leq l - n$, where $\beta_n^{(i)} = 2\text{Re}\left\{x_n^{(i)} y_n^*\right\} - |x_n^{(i)}|^2$ 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 ith path $(x_{n+1}^{(i)}, \ldots, x_{n+r}^{(i)})$ a corresponding identical extension $(x_{n+1}^{(j)}, \ldots, x_{n+r}^{(j)})$ of the jth path. Let us then assume without loss of generality that the ith path accumulates the largest metric at time l, that is, $J_l^{(i)} \geq J_l^{(j)}$, Therefore

$$J_n^{(i)} + \sum_{h=1}^{n-l} \beta_{n+h}^{(i)} \geq J_n^{(j)} + \sum_{h=1}^{n-l} \beta_{n+h}^{(j)}, \tag{7.25}$$

and $\sum_{h=1}^{n-l} \beta_{n+h}^{(i)}$ is the maximum metric sum for the extensions from node $(\tilde{\boldsymbol{x}}^{(i)}, x_n^{(i)})$. (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, states 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 $\tilde{\boldsymbol{x}}^{(i)}$, called the *survivors*. The most convenient labeling convention is that each state is labeled by the corresponding encoder state, plus the survivor that leads to it. This trellis is an exact replica of the encoder trellis discussed in Chapter 3, and the task of the decoder is to retrace the path the encoder traced through this trellis. Theorem 7.1 guarantees that this procedure is optimal. This method was introduced by Viterbi in 1967 [31, 38] in the context of analyzing convolutional codes and has since become widely known as the *Viterbi algorithm* [17]:

Step 1: Initialize the S states of the decoder with a metric $J_{-l}^{(i)} = -\infty$ and survivors $\tilde{\boldsymbol{x}}^{(i)} = \{\}$. 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

$$\beta_n = 2\mathrm{Re}\left\{x_n y_n^*\right\} - |x_n|^2 \tag{7.26}$$

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 $\tilde{\boldsymbol{x}}^{(i)}$ 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 $\boldsymbol{x}^{(i)}$ 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 [18, 33], multi-user optimal detectors [37], 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., refs. 9, 20, 33, and 39). 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 $\tilde{\boldsymbol{x}}^{(i)}$ 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 (ref. 11, page 182), but the argument can easily be extended to general trellis 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 $\tilde{\boldsymbol{x}}^{(i)}$ 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 $\boldsymbol{x}^{(i)}$ which maximizes $J_l^{(i)}$.

We recognize that we may now let $l \to \infty$; that is, 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 Chapter 6 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; that is, for a linear trellis there are $S2^k$ branches per time epoch. The complexity requirements of the Viterbi algorithm can then be captured by the following [28].

Theorem 7.2 *The Viterbi algorithm requires a complexity which is linear in the number of edges E; that is, 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 multiplications 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, and 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 $\rho(s)$, step 3 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.

7.7 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 invented by Bahl, Cocke, Jelinek, and Raviv [7] in 1972 and was used to maximize the probability of each symbol being correct, referred to as the maximum a posteriori probability (MAP) algorithm. As mentioned, this algorithm was not widely used since it provided no significant improvement over maximum-likelihood decoding and was significantly more complex.

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 on the 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 10.

Due to its importance, we will first give a functional description of the algorithm before deriving the formulas in detail. Figure 7.13 shows the example

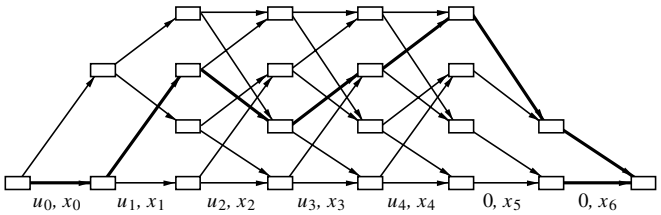


Figure 7.13 Example trellis of a short terminated trellis code.

trellis of a short terminated trellis code with seven sections. The transmitted signal is $\boldsymbol{x} = [x_0, \ldots, x_6]$, and the information symbols are $\boldsymbol{u} = [u_0, \ldots, u_4, u_5 = 0, u_6 = 0]$; that is, there are two tail bits that drive the encoder back into the zero state.

The ultimate purpose of the algorithm is the calculation of a posteriori probabilities, such as $\Pr[u_r|\boldsymbol{y}]$ or $\Pr[x_r|\boldsymbol{y}]$, where $\boldsymbol{y}$ is the received sequence observed at the output of a channel, whose input is the transmitted sequence $\boldsymbol{x}$. However, conceptually, it is more immediate to calculate the probability that the encoder traversed a specific transition in the trellis; that is, $\Pr[s_r = i, s_{r+1} = j|\boldsymbol{y}]$, 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:

$$\begin{aligned}\Pr[s_r = i, s_{r+1} = j|\boldsymbol{y}] &= \frac{1}{\Pr(\boldsymbol{y})}\Pr[s_r = i, s_{r+1} = j, \boldsymbol{y}] \\ &= \frac{1}{\Pr(\boldsymbol{y})}\alpha_{r-1}(i)\gamma_r(j,i)\beta_r(j). \end{aligned} \tag{7.27}$$

The α-values are internal variables of the algorithm and are computed by the *forward recursion*

$$\alpha_{r-1}(i) = \sum_{\text{states } l} \alpha_{r-2}(l)\gamma_{r-1}(i,l). \tag{7.28}$$

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 7.14. 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.

The β-values are calculated by an analogous procedure, called the *backward recursion*

$$\beta_r(j) = \sum_{\text{states } k} \beta_{r+1}(k)\gamma_{r+1}(k,j), \tag{7.29}$$

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 7.15.

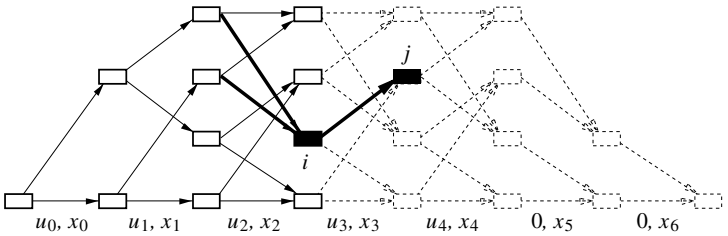


Figure 7.14 Illustration of the forward recursion of the APP algorithm.

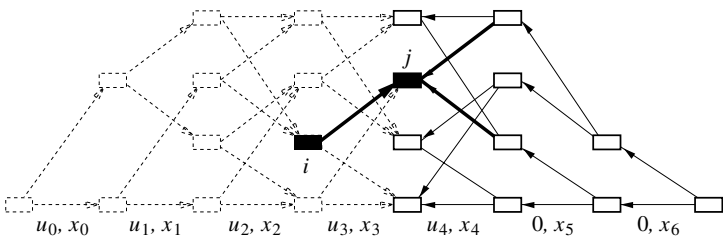


Figure 7.15 Illustration of the forward recursion of the APP algorithm.

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; it is calculated as

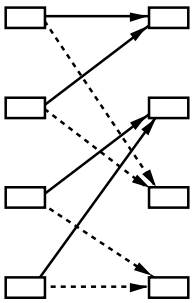
$$\gamma_r(j, i) = \Pr(s_{r+1} = j, y_r | s_r = i) = \Pr[s_{r+1} = j | s_r = i] \Pr(y_r | x_r). \tag{7.30}$$

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

$$p_{ij} = \Pr[s_{r+1} = j | s_r = i] = \Pr[u_r]. \tag{7.31}$$

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 $i \to j$.

The a posteriori symbol probabilities $\Pr[u_r | \boldsymbol{y}]$ can now be calculated from the a posteriori transition probabilities (7.27) by summing over all transitions corresponding to $u_r = 1$ and, separately, by summing over all transitions corresponding to $u_r = 0$, to obtain



$$p[u_r = 1|\boldsymbol{y}] = \frac{1}{\Pr(\boldsymbol{y})} \sum_{\text{solid}} \Pr[s_r = i, s_{r+1} = j, \boldsymbol{y}] \tag{7.32}$$

$$p[u_r = 0|\boldsymbol{y}] = \frac{1}{\Pr(\boldsymbol{y})} \sum_{\text{dashed}} \Pr[s_r = i, s_{r+1} = j, \boldsymbol{y}]. \tag{7.33}$$

The solid transition correspond to $u_r = 1$, and the dashed transitions correspond to $u_r = 0$ as illustrated above.

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. [7].

In the general case we will have need for the probability

$$q_{ij}(x) = \Pr(\tau(u_r, s_r) = x|s_r = i, s_{r+1} = j), \tag{7.34}$$

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

$$\alpha_r(j) = \Pr(s_{r+1} = j, \tilde{\boldsymbol{y}}), \tag{7.35}$$

the joint probability of the partial sequence $\tilde{\boldsymbol{y}} = (y_{-l}, \ldots, y_r)$ up to and including time epoch r and state $s_{r+1} = j$; and

$$\beta_r(j) = \Pr((y_{r+1}, \ldots, y_l)|s_{r+1} = j), \tag{7.36}$$

the conditional probability of the remainder of the received sequence $\boldsymbol{y}$ given that the state at time $r + 1$ is j.

With the above we now calculate

$$\begin{aligned} \Pr(s_{r+1} = j, \boldsymbol{y}) &= \Pr(s_{r+1} = j, \tilde{\boldsymbol{y}}, (y_{r+1}, \ldots, y_l)) \\ &= \Pr(s_{r+1} = j, \tilde{\boldsymbol{y}}) \Pr((y_{r+1}, \ldots, y_l)|s_{r+1} = j, \tilde{\boldsymbol{y}}) \\ &= \alpha_r(j)\beta_r(j), \end{aligned} \tag{7.37}$$

where we have used the fact that $\Pr((y_{r+1}, \ldots, y_l)|s_{r+1} = j, \tilde{\boldsymbol{y}}) = \Pr((y_{r+1}, \ldots, y_l)|s_{r+1} = j)$; that is, if $s_{r+1} = j$ is known, events after time r are independent of the history $\tilde{\boldsymbol{y}}$ up to s_{r+1}.

In the same way we calculate via Bayes' expansion

$$\begin{aligned}
&\Pr(s_r = i, s_{r+1} = j, \boldsymbol{y}) \\
&= \Pr(s_r = i, s_{r+1} = j, (y_{-l}, \ldots, y_{r-1}), y_r, (y_{r+1}, \ldots, y_l)) \\
&= \Pr(s_r = i, (y_{-l}, \ldots, y_{r-1}))\Pr(s_{r+1} = j, y_r|s_r = i)\Pr((y_{r+1}, \ldots, y_l)|s_{r+1} = j) \\
&= \alpha_{r-1}(i)\gamma_r(j, i)\beta_r(j). \qquad (7.38)
\end{aligned}$$

Now, again applying Bayes' rule and $\sum_b p(a, b) = p(a)$, we obtain

$$\begin{aligned}
\alpha_r(j) &= \sum_{\text{states } i} \Pr(s_r = i, s_{r+1} = j, \tilde{\boldsymbol{y}}) \\
&= \sum_{\text{states } i} \Pr(s_r = i, (y_{-l}, \ldots, y_{r-1}))\Pr(s_{r+1} = j, y_r|s_r = i) \\
&= \sum_{\text{states } i} \alpha_{r-1}(i)\gamma_r(j, i). \qquad (7.39)
\end{aligned}$$

For a trellis code started in the zero state at time $r = -l$ we have the starting conditions

$$\alpha_{-l-1}(0) = 1, \alpha_{-l-1}(j) = 0, \qquad j \neq 0. \qquad (7.40)$$

As above, we similarly develop an expression for $\beta_r(j)$, that is,

$$\begin{aligned}
\beta_r(j) &= \sum_{\text{states } i} \Pr(s_{r+2} = i, (y_{r+1}, \ldots, y_l)|s_{r+1} = j) \\
&= \sum_{\text{states } i} \Pr(s_{r+2} = i, y_{r+1}|s_{r+1} = j)\Pr((y_{r+2}, \ldots, y_l)|s_{r+2} = i) \\
&= \sum_{\text{states } i} \beta_{r+1}(i)\gamma_{r+1}(i, j). \qquad (7.41)
\end{aligned}$$

The boundary condition for $\beta_r(j)$ is

$$\beta_l(0) = 1, \beta_l(j) = 0, \qquad j \neq 0, \qquad (7.42)$$

for a trellis code which is terminated in the zero state.

Furthermore, the general form of the γ values is given by

$$\begin{aligned}
\gamma_r(j, i) &= \sum_{x_r} \Pr(s_{r+1} = j|s_r = i)\Pr(x_r|s_r = i, s_{r+1} = j)\Pr(y_r|x_r) \\
&= \sum_{x_r} p_{ij}q_{ij}(x_r)p_n(y_r - x_r), \qquad (7.43)
\end{aligned}$$

where we have used the conditional density function of the AWGN channel from (2.11), that is, $\Pr(y_r|x_r) = p_n(y_r - x_r)$. The calculation of $\gamma_r(j, i)$ is not very complex and can most easily be implemented by a table look-up procedure.

Equations (7.39) and (7.41) 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 nonzero 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 (7.43) and (7.39).

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 (7.41). Calculate $\Pr(s_{r+1} = j, \boldsymbol{y})$ from (7.37), and calculate $\Pr(s_r = i, s_{r+1} = j; \boldsymbol{y})$ from (7.27).

Step 5: If $r > -l$, let $r = r - 1$ and go to Step 4.

Step 6: Terminate the algorithm and output all the values $\Pr(s_{r+1} = j, \boldsymbol{y})$ and $\Pr(s_r = i, s_{r+1} = j, \boldsymbol{y})$.

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 $2^{kv}2l$ storage locations since there are 2^{kv} 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 v 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—that is, 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 $\Pr(s_r = i, s_{r+1} = j|\boldsymbol{y})$, we simply sum over all transitions $i \to j$ which are caused by $u_r = u$. Denoting these transitions by $A(u)$, we obtain

$$\Pr(u_r = u) = \sum_{(i,j)\in A(u)} \Pr(s_r = i, s_{r+1} = j|\boldsymbol{y}). \tag{7.44}$$

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

$$\begin{aligned}\Pr(x_r = x) &= \sum_{(i,j)\in B(x)} \Pr(x|y_r)\Pr(s_r = i, s_{r+1} = j|\boldsymbol{y}) \\ &= \sum_{(i,j)\in B(x)} \frac{p_n(y_r - x_r)}{p(y_r)} q_{ij}(x)\Pr(s_r = i, s_{r+1} = j|\boldsymbol{y}),\end{aligned} \tag{7.45}$$

where the a priori probability of y_r can be calculated via

$$p(y_r) = \sum_{\substack{x' \\ ((i,j)\in B(x))}} p(y_r|x')q_{ij}(x'), \tag{7.46}$$

and the sum extends over all transitions $i \to j$.

Equation (7.45) can be much simplified if there is only one output symbol on the transition $i \to j$ as in the introductory discussion. In that case the transition automatically determines the output symbol, and

$$\Pr(x_r = x) = \sum_{(i,j)\in B(x)} \Pr(s_r = i, s_{r+1} = j|\mathbf{y}). \tag{7.47}$$

One problem we have to address is that of numerical stability. The α and β values in (7.39) and (7.41) decay rapidly and will underflow in any fixed precision implementation. We therefore normalize both values at each epoch, that is,

$$\alpha_r(i) \to \frac{\alpha_r(i)}{\sum_s \alpha_r(s)}, \qquad \beta_r(i) \to \frac{\beta_r(i)}{\sum_s \beta_r(s)}. \tag{7.48}$$

This normalization has no effect on our final results such as (7.45), since these are similarly normalized. In fact, this normalization allows us to ignore the division by $p(y_r)$ in (7.45) and ignore the division by $\Pr(\mathbf{y})$ in (7.27), (7.32), and (7.33).

7.8 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 (7.28), (7.39) into the logarithm domain using the definitions

$$A_r(i) = \log(\alpha_r(i)), \qquad \Gamma_r(i,l) = \log(\gamma_r(i,l)) \tag{7.49}$$

to obtain the *log-domain* forward recursion

$$A_{r-1}(i) = \log\left(\sum_{\text{states } l} \exp(A_{r-2}(l) + \Gamma_{r-1}(i,l))\right). \tag{7.50}$$

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

$$B_r(j) = \log\left(\sum_{\text{states } k} \exp\left(B_{r+1}(l) + \Gamma_{r+1}(k, j)\right)\right). \tag{7.51}$$

The product in (7.27) and (7.38) now turns into the simple sum

$$\alpha_{r-1}(i)\gamma_r(j, i)\beta_r(j) \to A_{r-1}(i) + \Gamma_r(j, i) + B_r(j). \tag{7.52}$$

Unfortunately, Equations (7.50) and (7.51) 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; that is, there are only two branches involved at each state. Therefore there are only two terms in the sums (7.50) and (7.51). The logarithm of such a binary sum can be expanded as

$$\begin{aligned}\log(\exp(a) + \exp(b)) &= \log(\exp(\max(a, b))(1 + \exp(-|a - b|))) \\ &= \max(a, b) + \log((1 + \exp(-|a - b|))).\end{aligned}$$

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

$$\log((1 + \exp(-x)), \qquad x = |a - b|, \tag{7.53}$$

in a ROM look-up table. Given an example quantization of 4 bits, this is a 16×16 value look-up table, which is very manageable. Figure 7.16 shows the signal flow of this binary log-domain operation on the example of a node operation in the forward recursion.

Finally, to decode binary codes the algorithm computes the log-likelihood ratio (LLR) $\lambda(u_r)$ of the information bits u_r using the a posteriori probabilities (7.44) as

$$\lambda(u_r) = \log\left(\frac{\Pr(u_r = 1)}{\Pr(u_r = 0)}\right) = \log\left(\frac{\displaystyle\sum_{(i,j)\in A(u=1)} \alpha_{r-1}(i)\gamma_r(j, i)\beta_r(j)}{\displaystyle\sum_{(i,j)\in A(u=0)} \alpha_{r-1}(i)\gamma_r(j, i)\beta_r(j)}\right)$$

$$\lambda(u_r) = \log\left(\frac{\displaystyle\sum_{(i,j)\in A(u=1)} \exp(A_{r-1}(i) + \Gamma_r(j, i) + B_r(j))}{\displaystyle\sum_{(i,j)\in A(u=0)} \exp(A_{r-1}(i) + \Gamma_r(j, i) + B_r(j))}\right). \tag{7.54}$$

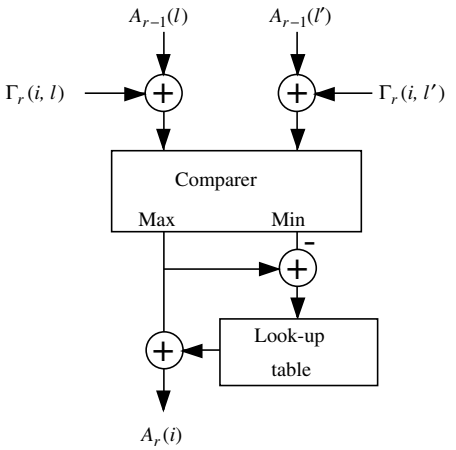


Figure 7.16 Signal flow diagram of the node calculation of the Log-APP algorithm.

The LLR is the quantity that 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 look-up table in Figure 7.16 [34]. This has the effect of approximating the forward and backward recursions by

$$A_{r-1}(i) = \log\left(\sum_{\text{states } l} \exp(A_{r-2}(l) + \Gamma_{r-1}(i, l))\right)$$
$$\approx \max_{\text{states } l}(A_{r-2}(l) + \Gamma_{r-1}(i, l)) \tag{7.55}$$

and

$$B_r(j) = \log\left(\sum_{\text{states } k} \exp(B_{r+1}(l) + \Gamma_{r+1}(k, j))\right)$$
$$\approx \max_{\text{states } k}(B_{r+1}(l) + \Gamma_{r+1}(k, j)). \tag{7.56}$$

It is very interesting to note that (7.55) is nothing else than our familiar Viterbi algorithm for maximum-likelihood sequence decoding. Furthermore, Equation (7.56) is also a Viterbi algorithm, but operated in the reverse direction.

Analogously, the final LLR calculation in (7.54) is approximated by

$$\lambda(u_r) \approx \max_{(i,j)\in A(u=1)}(A_{r-1}(i) + \Gamma_r(j, i) + B_r(j))$$
$$- \max_{(i,j)\in A(u=0)}(A_{r-1}(i) + \Gamma_r(j, i) + B_r(j)). \tag{7.57}$$

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 expressing the LLR of u_r in its basic form, that is,

$$\lambda(u_r) = \log\left(\frac{\displaystyle\sum_{\boldsymbol{x};(u_r=1)} \exp\left(-\frac{|\boldsymbol{y}-\boldsymbol{x}|^2}{N_0}\right)}{\displaystyle\sum_{\boldsymbol{x};(u_r=0)} \exp\left(-\frac{|\boldsymbol{y}-\boldsymbol{x}|^2}{N_0}\right)}\right), \tag{7.58}$$

where the sum in the numerator extends over all coded sequences $\boldsymbol{x}$ which correspond to information bit $u_r = 1$, and the denominator sum extends over all $\boldsymbol{x}$ 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

$$\lambda(u_r) \approx \min_{\boldsymbol{x};(u_r=0)} \frac{|\boldsymbol{y}-\boldsymbol{x}|^2}{N_0} - \min_{\boldsymbol{x};(u_r=1)} \frac{|\boldsymbol{y}-\boldsymbol{x}|^2}{N_0}, \tag{7.59}$$

that is, the metric difference between the nearest path to $\boldsymbol{y}$ with $u_r = 0$ and the nearest path with $u_r = 1$. For constant energy signals this simplifies to

$$\lambda(u_r) \approx \frac{(\boldsymbol{x}_r^{(1)} - \boldsymbol{x}_r^{(0)}) \cdot \boldsymbol{y}}{N_0/2}, \tag{7.60}$$

where $\boldsymbol{x}_r^{(1)} = \arg\min_{\boldsymbol{x};(u_r=1)} |\boldsymbol{y}-\boldsymbol{x}|^2$, and $\boldsymbol{x}_r^{(0)} = \arg\min_{\boldsymbol{x};(u_r=0)} |\boldsymbol{y}-\boldsymbol{x}|^2$.

For high-speed turbo decoding applications requiring up to 10 iterations, evaluation of Equation (7.53) may be too complex, yet one is not readily willing to accept the 0.5-dB loss entailed by using the Max-Log-APP. A very effective way of approximating (7.53) is [23]

$$\begin{aligned} &\max(a,b) + \log((1+\exp(-|a-b|))) \\ &\qquad \approx \max(a,b) + \begin{cases} 0 & \text{if } |a-b| > T \\ C & \text{if } |a-b| \le T. \end{cases} \end{aligned} \tag{7.61}$$

This simple threshold approximation is called *constant-Log-APP* algorithm. It is used in the UMTS turbo code [14], and leads to a degradation with respect to the full Log-APP of only 0.03 dB on this code [12], where the optimal parameters for this code are determined to be $C = 0.5$ and $T = 1.5$. This reduces the ROM look-up table of the Log-APP to a simple comparator circuit.

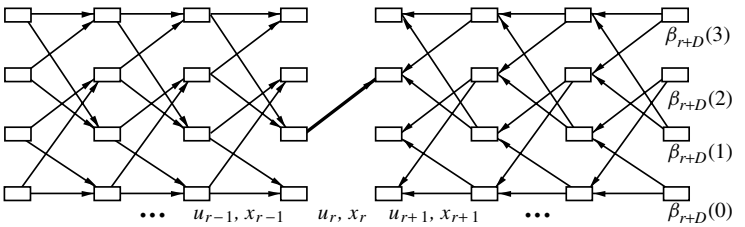


Figure 7.17 Sliding window approximation to the APP algorithm.

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 in 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 7.17.

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 are 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.

7.9 RANDOM CODING ANALYSIS OF SEQUENTIAL DECODING

In Section 6.6 we presented random coding performance bounds for trellis codes. In that section we implicitly assumed that we were using a maximum likelihood decoder. Since sequential decoding is not a maximum likelihood decoding method, the results in Section 6.6 do not apply.

The error analysis of sequential decoding is very difficult, and, again, we find it easier to generate results for the ensemble of all trellis 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.

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 7.18 shows an example of the search procedure of sequential decoding. The algorithm explores at each node an entire set of

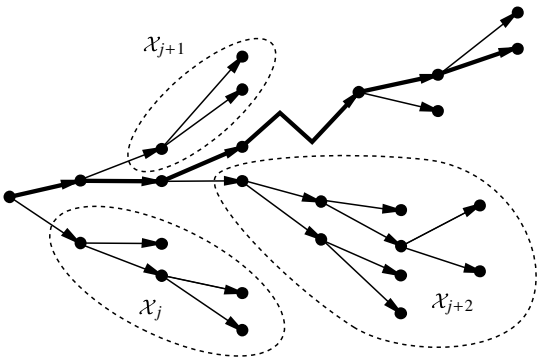


Figure 7.18 Incorrect subsets explored by a sequential decoder. The solid path is the correct one.

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 $\tilde{\boldsymbol{x}}'$ diverging at node j, which are searched by the algorithm. Furthermore, 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 done 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 7.18).

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 $\tilde{\boldsymbol{x}}$ be the partial correct path through the trellis and $\tilde{\boldsymbol{x}}'_j$ be a partial incorrect path which diverges from $\tilde{\boldsymbol{x}}$ at node j. Furthermore, let $L_n(\tilde{\boldsymbol{x}}') = L(\tilde{\boldsymbol{x}}', \boldsymbol{y})$ be the metric of the incorrect path at node n, and let $L_m(\tilde{\boldsymbol{x}})$ 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; hence, if $L_n(\tilde{\boldsymbol{x}}') < L_m(\tilde{\boldsymbol{x}})$, the incorrect path is not searched further until the metric of $\tilde{\boldsymbol{x}}$ falls below $L_n(\tilde{\boldsymbol{x}}')$. If

$$\min_{m \geq j} L_m(\tilde{\boldsymbol{x}}) = \lambda_j > L_n(\tilde{\boldsymbol{x}}'), \tag{7.62}$$

the incorrect path $\tilde{\boldsymbol{x}}'$ 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

$$\Pr(C_j \geq N_c) \leq \sum_{\boldsymbol{x}} p(\boldsymbol{x}) \int_{\boldsymbol{y}} p(\boldsymbol{y}|\boldsymbol{x}) \mathcal{B}\left(\left|e(p(\boldsymbol{y}|\boldsymbol{x}') \geq \lambda_j)\right| \geq N_c\right) d\boldsymbol{y}, \tag{7.63}$$

where $e(p(\boldsymbol{y}|\boldsymbol{x}') \geq \lambda_j)$ 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 that equals 1 if the expression is true and 0 otherwise. The function $\mathcal{B}(\star)$ in (7.63) 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 Chapter 6, by realizing that $\mathcal{B}(\star) = 1$ if at least N_c error paths have a metric such that

$$L_n(\tilde{\boldsymbol{x}}') \geq \lambda_j, \tag{7.64}$$

and, hence,

$$\left(\frac{1}{N_c} \sum_{\boldsymbol{x}' \in \mathcal{X}'_j} \exp\left(\alpha \left(L_n(\tilde{\boldsymbol{x}}') - \lambda_j\right)\right) \right)^{\rho} \geq 1, \tag{7.65}$$

where α and ρ are arbitrary positive constants. Note that we have extended the sum in (7.65) over all error sequences as is customary in random coding analysis. We may now use (7.65) to overbound the indicator function $\mathcal{B}(\star)$, and we obtain

$$\Pr(C_j \geq N_c) \leq N_c^{-\rho} \sum_{\boldsymbol{x}} p(\boldsymbol{x}) \int_{\boldsymbol{y}} p(\boldsymbol{y}|\boldsymbol{x}) \left(\sum_{\boldsymbol{x}' \in \mathcal{X}'_j} \exp\left(\alpha \left(L_n(\tilde{\boldsymbol{x}}') - \lambda_j\right)\right) \right)^{\rho} d\boldsymbol{y}, \tag{7.66}$$

and, due to (7.62),

$$\exp\left(-\alpha\rho\lambda_i\right) \leq \sum_{m=j}^{\infty} \exp\left(-\alpha\rho L_m(\tilde{\boldsymbol{x}})\right), \tag{7.67}$$

and we have

$$\Pr(C_j \geq N_c) \leq N_c^{-\rho} \sum_{\boldsymbol{x}} p(\boldsymbol{x}) \int_{\boldsymbol{y}} p(\boldsymbol{y}|\boldsymbol{x}) \left(\sum_{\boldsymbol{x}' \in \mathcal{X}'_j} \exp\left(\alpha L_n(\tilde{\boldsymbol{x}}')\right) \right)^{\rho} \times \sum_{m=j}^{\infty} \exp\left(-\alpha\rho L_m(\tilde{\boldsymbol{x}})\right) d\boldsymbol{y}. \tag{7.68}$$

Analogously to Chapter 5, let c be the correct path and 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 $\boldsymbol{x}'_j$. Again, $\boldsymbol{x}$ and $\boldsymbol{x}'$ are, strictly taken, the signal sequences assigned to the correct and incorrect path, respectively, and $\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{x}}'$ 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, that is,

$$\text{Avg}\{\Pr(C_j \geq N_c)\} \leq N_c^{-\rho} \overline{\sum_{c} p(c) \int_{\boldsymbol{y}} p(\boldsymbol{y}|\boldsymbol{x}) \sum_{m=j}^{\infty} \exp\left(-\alpha\rho L_m(\tilde{\boldsymbol{x}})\right)} \times \left(\sum_{e \in \mathcal{E}'_j} \overline{\exp\left(\alpha L_n(\tilde{\boldsymbol{x}}')\right)} \right)^{\rho} d\boldsymbol{y}. \tag{7.69}$$

Note that we have used Jensen's inequality to pull the averaging into the second sum, which restricts ρ to $0 \le \rho \le 1$. Since we are using time-varying random trellis codes, (7.69) 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 inequality[3]

$$\left(\sum a_i\right)^{\rho} \le \sum a_i^{\rho}; \qquad a_i \ge 0, \quad 0 \le \rho \le 1, \tag{7.70}$$

we obtain[4]

$$\begin{aligned}\text{Avg}\{\Pr(C_0 \ge N_c)\} &\le N_c^{-\rho} \sum_c p(c) \overline{\int_{\boldsymbol{y}} p(\boldsymbol{y}|\boldsymbol{x}) \sum_{m=0}^{\infty} \exp\left(-\alpha\rho L_m(\tilde{\boldsymbol{x}})\right)} \\ &\quad \times \sum_{n=0}^{\infty} 2^{kn\rho} \left(\overline{\exp\left(\alpha L_n(\tilde{\boldsymbol{x}}')\right)}\right)^{\rho} d\boldsymbol{y} \qquad (7.71) \\ &= N_c^{-\rho} \sum_c p(c) \sum_{m=0}^{\infty} \sum_{n=0}^{\infty} 2^{kn\rho} \\ &\quad \times \overline{\int_{\boldsymbol{y}} p(\boldsymbol{y}|\boldsymbol{x}) \exp\left(-\alpha\rho L_m(\tilde{\boldsymbol{x}})\right) \left(\overline{\exp(\alpha L_n(\tilde{\boldsymbol{x}}'))}\right)^{\rho} d\boldsymbol{y}}. \qquad (7.72)\end{aligned}$$

Now we substitute the metrics [see (7.10)]

$$L_m(\tilde{\boldsymbol{x}}) = \sum_{r=0}^{m} \log\left(\frac{p(y_r|x_r)}{p(y_r)}\right) - k, \tag{7.73}$$

$$L_n(\tilde{\boldsymbol{x}}') = \sum_{r=0}^{n} \log\left(\frac{p(y_r|x'_r)}{p(y_r)}\right) - k, \tag{7.74}$$

into the expression (7.72) and use $\alpha = \frac{1}{1+\rho}$. This let's us rewrite the exponentials in (7.72) as

$$2^{kn\rho} \overline{\int_{\boldsymbol{y}} p(\boldsymbol{y}|\boldsymbol{x}) \exp\left(-\alpha\rho L_m(\tilde{x})\right) \left(\overline{\exp\left(\alpha L_n(\tilde{\boldsymbol{x}}')\right)}\right)^{\rho}} = \begin{cases} 2^{-(m-n)E_c(\rho) - m(E_{ce}(\rho) - k\rho)} & \text{if } m \ge n \\ 2^{-(n-m)(E_e(\rho) - k\rho) - n(E_{ce}(\rho) - k\rho)} & \text{if } n \ge m. \end{cases} \tag{7.75}$$

[3]This inequality is easily shown, that is,

$$\frac{\sum a_i^{\rho}}{\left(\sum a_i\right)^{\rho}} = \sum \left(\frac{a_i}{\sum a_i}\right)^{\rho} \ge \sum \left(\frac{a_i}{\sum a_i}\right) = 1,$$

where the inequality resulted from the fact that each term in the sum is ≤ 1 and $\rho \le 1$.

[4]Note that it is here that we need the time-varying assumption of the codes (compare also Section 6.6 and Figure 6.9).

The exponents used above are given by

$$\begin{aligned}
2^{-E_c(\rho)} &= \int_v \sum_x q(x)p(v|x)\left(\frac{p(v|x)}{p(v)}2^{-k}\right)^{-\frac{\rho}{1+\rho}} \\
&= 2^{k\frac{\rho}{1+\rho}} \int_v \sum_x q(x)p(v|x)^{\frac{1}{1+\rho}} p(v)^{\frac{\rho}{1+\rho}} \\
&\le 2^{k\frac{\rho}{1+\rho}} \left(\int_v \left(\sum_x q(x)p(v|x)^{\frac{1}{1+\rho}}\right)^{1+\rho}\right)^{\frac{1}{1+\rho}} \\
&= 2^{k\frac{\rho}{1+\rho}-\frac{1}{1+\rho}E_0(\rho,\mathbf{q})} = f_c, \qquad (7.76)
\end{aligned}$$

where we have used Hölder's inequality above (see Chapter 6) 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 Chapter 6, Equation (6.49)! Analogously, we also find

$$\begin{aligned}
2^{-(E_e(\rho)-k\rho)} &= 2^{k\rho} \int_v p(v)\left(\sum_{x'} q(x')\left(\frac{p(v|x')}{p(v)}2^{-k}\right)^{\frac{1}{1+\rho}}\right)^{\rho} \\
&= 2^{k\frac{\rho^2}{1+\rho}} \int_v p(v)^{\frac{1}{1+\rho}}\left(\sum_{x'} q(x')p(v|x')^{\frac{1}{1+\rho}}\right)^{\rho} \\
&\le 2^{k\frac{\rho^2}{1+\rho}} \left(\int_v \left(\sum_{x'} q(x')p(v|x')^{\frac{1}{1+\rho}}\right)^{1+\rho}\right)^{\frac{\rho}{1+\rho}} \\
&= 2^{k\frac{\rho^2}{1+\rho}-\frac{\rho}{1+\rho}E_0(\rho,\mathbf{q})} = f_e, \qquad (7.77)
\end{aligned}$$

where $\lambda = \frac{\rho}{1+\rho}$. Finally we obtain

$$\begin{aligned}
2^{-(E_{ce}(\rho)-k\rho)} &= 2^{k\rho} \int_v \sum_x q(x)p(v|x)\left(\sum_{x'} q(x')\left(\frac{p(v|x')}{p(v|x)}\right)^{\frac{1}{1+\rho}}\right)^{\rho} \\
&= 2^{k\rho} \int_v \sum_x q(x)p(v|x)^{\frac{1}{1+\rho}}\left(\sum_{x'} q(x')p(v|x')^{\frac{1}{1+\rho}}\right)^{\rho} \\
&= 2^{k\rho-E_0(\rho,\mathbf{q})}. \qquad (7.78)
\end{aligned}$$

Note now that since $1 = \frac{\rho}{1+\rho} + \frac{1}{1+\rho}$, we have

$$2^{k\rho-E_0(\rho,\mathbf{q})} = 2^{k\frac{\rho^2}{1+\rho}-\frac{\rho}{1+\rho}E_0(\rho,\mathbf{q})}2^{k\frac{\rho}{1+\rho}-\frac{1}{1+\rho}E_0(\rho,\mathbf{q})} = f_e f_c, \qquad (7.79)$$

where the two factors f_e and f_c are defined in (7.76) and (7.77).

With this we can rewrite (7.72) as

$$\text{Avg}\{\Pr(C_0 \geq N_c)\} \leq N_c^{-\rho} \sum_c p(c) \sum_{m=0}^{\infty} \sum_{n=0}^{\infty} f_e^n f_c^m. \tag{7.80}$$

The double infinite sum in (7.80) converges if $f_e, f_c < 1$; hence from (7.79), if $\rho k < E_0(\rho, \mathbf{q})$ we obtain

$$\text{Avg}\{\Pr(C_0 \geq N_c)\} \leq N_c^{-\rho} \frac{1}{(1 - f_e)(1 - f_c)}. \tag{7.81}$$

Similarly, it can be shown [39] that there exists a lower bound on the number of computations, given by

$$\text{Avg}\{\Pr(C_j \geq N_c)\} \geq N_c^{-\rho} (1 - o(N_c)). \tag{7.82}$$

Together, (7.81) and (7.82) characterize the computational behavior of sequential decoding. It is interesting to note that if $\rho \leq 1$, the expectation of (7.81) and (7.82)—that is, the expected number of computations—becomes unbounded, since

$$\sum_{N_c=1}^{\infty} N_c^{-\rho} \tag{7.83}$$

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. This is why R_0 is often also referred to as the *computational cutoff rate*.

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 [41], who constructed random trellis codes for 8-PSK and 16-QAM constellations which achieve R_0 with constraint lengths of 15 and 16—that is, very realizable codes.

7.10 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, ref. 32—or, for convolutional codes, ref. 39—is an excellent and inexhaustible source of information. Many of the random coding arguments in ref. 39 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. First, it is a "sequential" algorithm; that is, modern pipelining and parallelizing is very difficult, if not impossible, to accomplish. Second, the metric used in sequential decoding contains the "bias" term accounting for the different paths 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., ref. 42).

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 ref. 11, 13, and 16. 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 ref. 35 discusses an interesting implementation that 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 that can also be stated as a trellis (tree) search [43], and to the decoding of block codes.

In all likelihood the importance of all these decoding algorithms, with the likely exception of the Viterbi algorithm, will fade compared to that of the APP decoder in its different forms. The impact of iterative decoding of large error control codes (see Chapters 10–12 on turbo coding and related topics) has been so revolutionary as to push other strategies into obscurity.

APPENDIX 7.A

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 (7.19) is a vector of Gaussian random variables (see also Section 2.6), we can easily write down its probability density function [44], namely,

$$p(\boldsymbol{\delta}_n) = \frac{1}{(2\pi)^{M/2}|\mathbf{R}|^{1/2}} \exp\left(-\frac{1}{2}(\boldsymbol{\mu} - \boldsymbol{\delta}_n)^T \mathbf{R}^{-1}(\boldsymbol{\mu} - \boldsymbol{\delta}_n)\right), \tag{7.84}$$

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 $r_{ij} = E\left[(\delta_n^{(i,c)} - \mu_i)\right.$

$(\delta_n^{(j,c)} - \mu_j)\Big]$ can be evaluated as

$$r_{ij} = \begin{cases} 2N_0\left(d_i^2 + d_j^2 - d_{ij}^2\right) & \text{if } i \neq j \\ 4N_0 d_i^2 & \text{if } i = j \end{cases} \tag{7.85}$$

and where

$$d_{ij}^2 = \left|\tilde{\boldsymbol{x}}^{(p_i)} - \tilde{\boldsymbol{x}}^{(p_j)}\right|^2 \quad \text{and} \quad d_i^2 = \left|\tilde{\boldsymbol{x}}^{(p_i)} - \tilde{\boldsymbol{x}}^{(c)}\right|. \tag{7.86}$$

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

$$\Pr(\text{CPL}|\mathcal{C}_\text{p}) = \int_{\boldsymbol{\delta}_n \leq \mathbf{0}} p(\boldsymbol{\delta}_n)\, d\boldsymbol{\delta}_n. \tag{7.87}$$

Equation (7.87) 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 (7.87) 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)]

$$\Pr(\text{CPL}|\mathcal{C}_\text{p}) \leq Q\left(\sqrt{\frac{d_l^2}{2N_0}}\right), \tag{7.20}$$

where d_l^2, the vector Euclidean distance, is given by

$$d_l^2 = 2N_0 \min_{\boldsymbol{y} \leq \mathbf{0}} (\boldsymbol{\mu} - \boldsymbol{y})^T \mathbf{R}^{-1} (\boldsymbol{\mu} - \boldsymbol{y}), \tag{7.88}$$

and $\boldsymbol{y}$ is simply a dummy variable of minimization.

The problem of calculating (7.20) has now been transformed into the geometric problem of finding the point on the surface of the convex polytope $\boldsymbol{y} \leq \mathbf{0}$ which is closest to $\boldsymbol{\mu}$ using the distance measure of (7.88). This situation is illustrated in Figure 7.19 for a two-dimensional scenario. The minimization in (7.88) is a constrained minimization of a quadratic form. Obviously, some of the constraints $\boldsymbol{y} \leq \mathbf{0}$ will be met with equality. These constraints are called the *active constraints*; that is, if $\boldsymbol{y} = (\boldsymbol{y}^{(a)}, \boldsymbol{y}^{(p)})^T$ is the partitioning of $\boldsymbol{y}$ into active and passive components, $\boldsymbol{y}^{(a)} = \mathbf{0}$. This minimum is the point $\boldsymbol{y}_0$ in Figure 7.19. The right-hand side of Figure 7.19 also shows the geometric

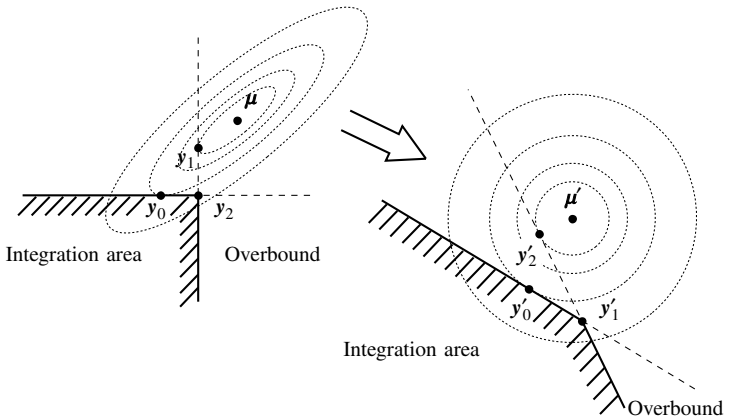


Figure 7.19 Illustration of the concept of the vector Euclidean distance with $M = 2$. The distance between $\boldsymbol{y}_0$ and $\boldsymbol{\mu}$ is d_l^2. The right-hand side shows the space after decorrelation, and d_l^2 equals the standard Euclidean distance between $\boldsymbol{y}_0'$ and $\boldsymbol{\mu}'$.

configuration when the decorrelating linear transformation $\boldsymbol{\delta}_n' = \sqrt{\mathbf{R}^{(-1)}}\boldsymbol{\delta}_n$ is applied. The vector Euclidean distance (7.88) is invariant to such a transformation, but $E\left[(\delta_n'^{(i,c)} - \mu_i')(\delta_n'^{(j,c)} - \mu_j')\right] = \delta_{ij}$; that is, 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 (Chapter 2, ref. 44) that the probability that $\boldsymbol{\mu}'$ is carried into the shaded region of integration can be overbounded by integrating over the halfplane not containing $\boldsymbol{\mu}'$, as illustrated in the figure. This leads to (7.20).

We now have to minimize

$$d_l^2 = 2N_0 \min_{\boldsymbol{y}^{(p)} \leq 0} \left(\begin{pmatrix} \boldsymbol{\mu}^{(p)} \\ \boldsymbol{\mu}^{(a)} \end{pmatrix} - \begin{pmatrix} \boldsymbol{y}^{(p)} \\ \mathbf{0} \end{pmatrix} \right)^T \begin{pmatrix} \mathbf{R}^{(pp)} & \mathbf{R}^{(pa)} \\ \mathbf{R}^{(ap)} & \mathbf{R}^{(aa)} \end{pmatrix}^{-1} \left(\begin{pmatrix} \boldsymbol{\mu}^{(p)} \\ \boldsymbol{\mu}^{(a)} \end{pmatrix} - \begin{pmatrix} \boldsymbol{y}^{(p)} \\ \mathbf{0} \end{pmatrix} \right), \tag{7.89}$$

where we have partitioned $\boldsymbol{\mu}$ and $\mathbf{R}$ analogously to $\boldsymbol{y}$. After some elementary operations we obtain

$$\boldsymbol{y}^{(p)} = \boldsymbol{\mu}^{(p)} + \left(\mathbf{X}^{(pp)}\right)^{-1} \mathbf{X}^{(pa)} \boldsymbol{\mu}^{(a)} \leq \mathbf{0}, \tag{7.90}$$

and

$$d_l^2 = 2N_0 \boldsymbol{\mu}^{(a)T} \left(\mathbf{X}^{(aa)}\right)^{-1} \boldsymbol{\mu}^{(a)}, \tag{7.91}$$

where[5]

$$\mathbf{X}^{(pp)} = \left[\mathbf{R}^{(pp)} - \mathbf{R}^{(pa)}\left(\mathbf{R}^{(aa)}\right)^{-1}\mathbf{R}^{(ap)}\right]^{-1},$$
$$\mathbf{X}^{(aa)} = \left(\mathbf{R}^{(aa)}\right)^{-1}\left[\mathbf{I} + \mathbf{R}^{(ap)}\mathbf{X}^{(pp)}\mathbf{R}^{(pa)}\right],$$
$$\mathbf{X}^{(pa)} = -\mathbf{X}^{(pp)}\mathbf{R}^{(pa)}\left(\mathbf{R}^{(aa)}\right)^{-1}. \tag{7.92}$$

We are now presented with the problem of finding the active components in order to evaluate (7.91). This is a combinatorial problem; that is, we must test all $2^M - 1 = \sum_{i=1}^{M}\binom{M}{i}$ possible combinations of active components from the M entries in $\boldsymbol{y}$ for compatibility with (7.90). This gives us the following procedure:

Step 1: Select all $2^M - 1$ combinations of active components and set $\boldsymbol{y}^{(a)} = \mathbf{0}$ for each.

Step 2: For each combination for which $\boldsymbol{y}^{(p)} \leq \mathbf{0}$, store the resulting d_l^2 from (7.91) 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 7.19. The $2^2 - 1 = 3$ combinations correspond to the points $\boldsymbol{y}_0, \boldsymbol{y}_1$ and $\boldsymbol{y}_2$. The point $\boldsymbol{y}_1$ does not qualify, since it violates (7.90). The minimum is chosen between $\boldsymbol{y}_0$ and $\boldsymbol{y}_2$. This process might be easier to visualize in the decorrelated space $\boldsymbol{y}'$, 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 $\boldsymbol{y}$ are linearly dependent on the other entries. In the context of the restriction $\boldsymbol{y} \leq \mathbf{0}$, 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 that produce a singular $\mathbf{R}^{(aa)}$ from further consideration without affecting d_l^2.

[5]These 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

$$E = \left(A - BD^{-1}C\right)^{-1}; F = -EBD^{-1}; G = -D^{-1}CE.$$

and

$$H = D^{-1} + D^{-1}CEBD^{-1}.$$

BIBLIOGRAPHY

1. J. B. Anderson, "Limited search trellis decoding of convolutional codes," *IEEE Trans. Inform. Theory,* vol. IT-35, no. 5, pp. 944–955, Sept. 1989.
2. 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.
3. J. B. Anderson and S. Mohan, *Source and Channel Coding: An Algorithmic Approach*, Kluwer Academic Publishers, Boston, MA, 1991.
4. T. Aulin, "Breadth first maximum likelihood sequence detection," *IEEE Trans. Commun.,* vol. COM-47, no. 2, pp. 208–216, Feb. 1999.
5. T. Aulin, "Recovery properties of the SA(B) algorithm," Technical Report No. 105, Chalmers University of Technology, Sweden, Feb. 1991.
6. 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.
7. 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, March 1974.
8. K. Balachandran, "Design and performance of constant envelope and non-constant envelope digital phase modulation schemes," Ph.D. thesis, ECSE Department. Rensselaer Polytechnic Institute, Troy, NY, Feb. 1992.
9. R. E. Blahut, *Principles and Practice of Information Theory*, Addison-Wesley, Reading, MA, 1987.
10. P. R. Chevillat and D. J. Costello, Jr., "A multiple stack algorithm for erasure-free decoding of convolutional codes," *IEEE Trans. Commun.,* vol. COM-25, pp. 1460–1470, Dec. 1977.
11. G. C. Clark and J. B. Cain, *Error-Correction Coding for Digital Communications*, Plenum Press, New York, 1983.
12. B. Classen, K. Blankenship, and V. Desai, "Turbo decoding with the constant-log-MAP algorithm," *Proceedings, Second International. Symposium on Turbo Codes and Related Applications*, (Brest, France), pp. 467–470, Sept. 2000.
13. 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.
14. 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, Sept. 23, 2000.
15. R. M. Fano, "A heuristic discussion of probabilistic decoding," *IEEE Trans. Inform. Theory,* vol. IT-9, pp. 64–74, April 1963.
16. 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.
17. G. D. Forney, Jr. "The Viterbi algorithm," *Proc. IEEE,* vol. 61, pp. 268–278, 1973.
18. 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.

19. 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.

20. S. Lin and D. J. Costello, Jr., *Error Control Coding*, Prentice-Hall, Englewood Cliffs, NJ, 1983.

21. J. M. Geist, "An empirical comparison of two sequential decoding algorithms," *IEEE Trans. Commun.,* vol. COM-19, pp. 415–419, Aug. 1971.

22. J. M. Geist, "Some properties of sequential decoding algorithms," *IEEE Trans. Inform. Theory,* vol. IT-19, pp. 519–526, July 1973.

23. 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.

24. 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.

25. F. Jelinek, "A fast sequential decoding algorithm using a stack," *IBM J. Res. Dev.*, vol. 13, pp. 675–685, Nov. 1969.

26. F. Jelinek and A. B. Anderson, "Instrumentable tree encoding of information sources," *IEEE Trans. Inform. Theory,* vol. IT-17, Jan. 1971.

27. L. Ma, "Suboptimal decoding strategies," MSEE thesis, University of Texas at San Antonio, May 1996.

28. 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.

29. J. L. Massey, "Variable-length codes and the Fano metric," *IEEE Trans. Inform. Theory,* vol. IT-18, pp. 196–198, Jan. 1972.

30. 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.

31. J. K. Omura, "On the Viterbi decoding algorithm," *IEEE Trans. Inform. Theory,* vol. IT-15, pp. 177–179, Jan. 1969.

32. 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.

33. J. G. Proakis, *Digital Communications*, McGraw-Hill, New York, 1989.

34. 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.

35. 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.

36. S. J. Simmons and P. Wittke, "Low complexity decoders for constant envelope digital modulation," *Conference Records*, GlobeCom, Miami, FL, pp. E7.7.1–E7.7.5, Nov. 1982.

37. S. Verdú, "Minimum probability of error for asynchronous Gaussian multiple-access channels," *IEEE Trans. Inform. Theory,* vol. IT-32, pp. 85–96, Jan. 1986.

38. A. J. Viterbi, "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm," *IEEE Trans. Inform. Theory,* vol. IT-13, pp. 260–269, April 1969.

39. A. J. Viterbi and J. K. Omura, *Principles of Digital Communication and Coding*, McGraw-Hill, New York, 1979.

40. J. M. Wozencraft and B. Reiffen, *Sequential Decoding*, MIT Press, Cambridge, MA, 1961.

41. 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.

42. 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, pp. 691–700, June 1997.

43. L. Wei and C. Schlegel, "Synchronous DS-SSMA with improved decorrelating decision-feedback multiuser detection," *IEEE Trans. Veh. Technol.,* vol. VT-43, no. 3, pp. 767–772, Aug. 1994.

44. J. M. Wozencraft and I. M. Jacobs, *Principles of Communication Engineering*, John Wiley & Sons, New York, 1965.

45. K. Sh. Zigangirov, "Some sequential decoding procedures," *Prob. Pederachi Inform.,* vol. 2, pp. 13–25, 1966.

46. K. S. Zigangirov and V. D. Kolesnik, "List decoding of trellis codes," *Problems of Control and Information Theory*, no. 6, 1980.

CHAPTER 8

Factor Graphs

WITH SHERYL HOWARD

8.1 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 purposes 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 [26, 27] 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. [30] 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 [29] established such bipartite graphs, which he termed "Tanner graphs," as a model applicable to a wide variety of decoding algorithms, including the Viterbi [28] or min-sum algorithm and the BCJR [2], forward–backward [4], or APP algorithm, discussed in Chapter 7.

Trellis and Turbo Coding, by Christian B. Schlegel and Lance C. Pérez
ISBN 0-471-22755-2

The artificial intelligence community also developed graphical methods of solving probabilistic inference problems, termed *probability propagation*. Pearl [22] 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 [15] 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 [5], which is based on the APP algorithm, is an application of belief propagation to a graph with cycles [9, 19].

In ref. 14, Kschischang et al. introduced factor graph formalism, showing that belief propagation 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 8.3 after examining graphical models for probabilistic inference.

8.2 GRAPHICAL FUNCTION REPRESENTATION

Figure 8.1 shows the Bayesian network and the factor graph for a simple four-state Markov chain, that is, for the probability function $P(A, B, C, D) = P(A)P(B|A)P(C|B)P(D|C)$. 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 8.1 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.

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 [22, 23], uses the Bayesian network representation. Graphical decoding via sum-product

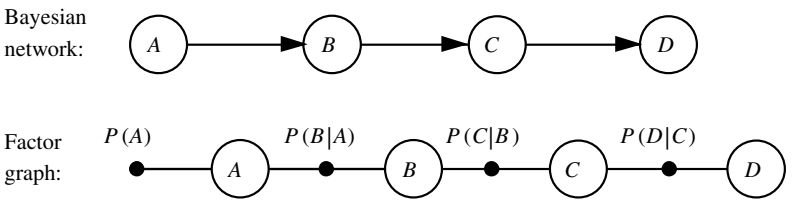


Figure 8.1 Graphical models for a four-state Markov chain.

decoding [26, 30], functionally equivalent to belief propagation, utilizes factor graphs or their bipartite precursors. Both representations 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 8.1. We wish to calculate the marginal probability distribution of D, given by

$$\begin{aligned} P(D) &= \sum_A \sum_B \sum_C P(A, B, C, D) \\ &= \sum_A \sum_B \sum_C P(A)P(B|A)P(C|B)P(D|C), \end{aligned} \tag{8.1}$$

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 (8.1) allows us to execute the summations locally:

$$P(D) = \sum_C P(D|C) \underbrace{\sum_B P(C|B) \underbrace{\sum_A P(B|A)P(A)}_{P(B)}}_{P(C)}. \tag{8.2}$$

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 (8.1). 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

$$P(C|D = d) = \frac{\sum_{A,B} P(A, B, C, D = d)}{P(D = d)} = K \times P(C, D = d), \tag{8.3}$$

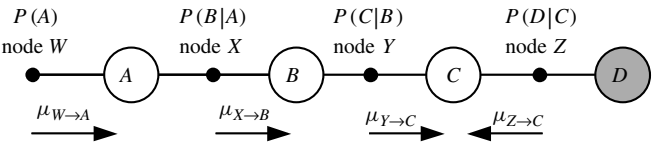


Figure 8.2 Illustration of message passing in a four-state Markov chain.

where we need to sum out the intermediate variables A and B. We now factor the global joint distribution into local distributions and obtain

$$\begin{aligned} P(C, D = d) &= \sum_A P(A) \sum_B P(B|A)P(C|B)P(D = d|C) \\ &= \underbrace{P(D = d|C)}_{\mu_{Z\to C}} \underbrace{\sum_B P(C|B) \underbrace{\sum_A P(B|A) \underbrace{P(A)}_{\mu_{W\to A}}}_{\mu_{X\to B}}}_{\mu_{Y\to C}}. \end{aligned} \tag{8.4}$$

As illustrated in Figure 8.2, Equation (8.4) can be visualized in the factor graph of the Markov chain as passing messages between nodes. The messages $\mu_{\text{function}\to\text{variable}}$ 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 $P(C, D = d) = \text{BEL}(C) = \mu_{Y\to C}\ \mu_{Z\to C}$. 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 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 $\mu_{X\to B} = P(B) = [P(B = -1), P(B = 1)]$ for $B \in \{-1, 1\}$.

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

$$\mu_{X\to B} = \sum_A P(B|A)\mu_{W\to A}. \tag{8.5}$$

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.

8.3 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 8.2, the message passed from node X to node B is $P(B) = \sum_A P(B|A)\mu_{W\to A}$.
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 $\mu_{W\to A} = P(A)$, and of the latter, $\mu_{Z\to C} = P(D = d|C)$. At a variable node, incoming messages are multiplied to generate the outgoing messages.

For an acyclic graph—that is, for a tree graph that 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 BEL(node) $= \prod \mu_{\text{function}\to\text{variable}}$. We will show later that BEL(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.

The first occurrence of the sum-product algorithm was in Gallager's work on low-density parity-check codes (LDPCs), presented in 1962 [11, 12]. The algorithm that he developed to decode LDPCs was a case of the sum-product algorithm specific to LDPCs. A proof of the optimality of the sum-product algorithm on acyclic graphs was first given by Tanner [26] in 1981.

8.4 ITERATIVE PROBABILITY PROPAGATION

Let us now study the more complex factor graph of the [8, 4] extended Hamming code. This code has the systematic parity-check matrix [20]

$$\mathbf{H}_{[8,4]} = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}. \tag{8.6}$$

The code has 4 information bits, which we associate with variable nodes $U_1, \ldots, U_4$; it also has 4 parity bits, which are associated with the variable nodes $X_5, \ldots, X_8$. There are also 8 observed output symbols, nodes $Y_1, \ldots, Y_8$, which are the noisy received variables $U_1, \ldots, X_8$. The function nodes $V_1, \ldots, V_4$ are the four parity-check equations described by (8.6), 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 it 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 in Figure 8.3.

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 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.

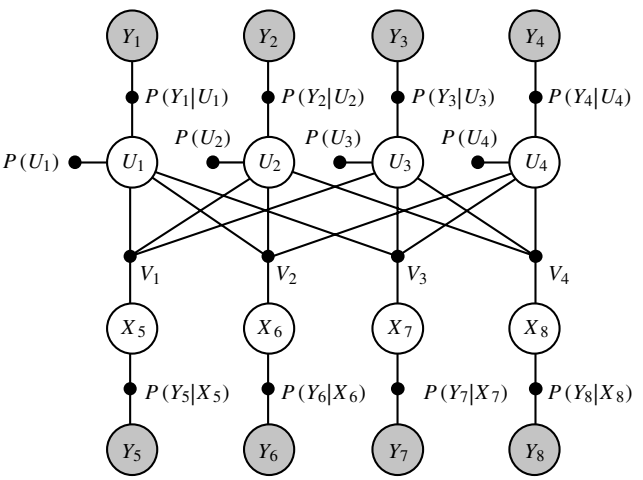


Figure 8.3 [8, 4] Hamming code factor graph representation.

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 $\mu_{Y_k \to U_k} = P(Y_k = y_k | U_k)$ from Y_k to U_k and $\mu_{Y_j \to X_j} = P(Y_j = y_j | X_j)$ from Y_j to X_j. For an AWGN channel, $P(Y_k = y_k | U_k = u) = \frac{1}{\sqrt{\pi N_0}} \exp(-(y_k - u)^2 / N_0)$.

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 $\mu_{X_j \to V_j} = P(Y_j = y_j | X_j)$ 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 8.4.

Step 5: Send the messages $\mu_{U_k \to V_j}$ from the nodes U_k to the nodes V_j with which they are connected. These messages are given by

$$\mu_{U_k \to V_j} = \mu_{U_k} \mu_{Y_k \to U_k} \prod_{i \neq j} \mu_{V_i \to U_k}. \tag{8.7}$$

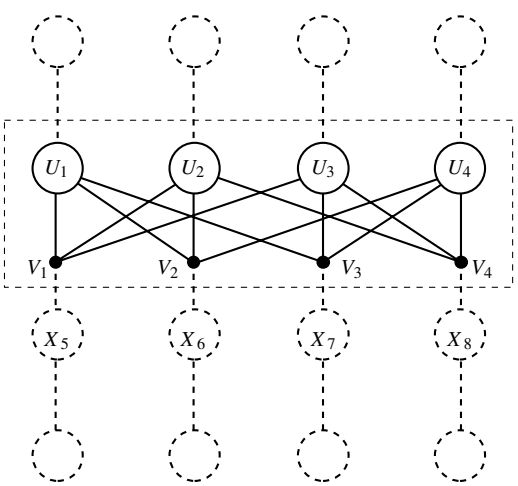


Figure 8.4 Inner nodes participating in the iterative part of the decoding process.

In the first iteration, no messages from the inner function nodes have been received yet, and $\mu_{U_k \to V_j} = \mu_{U_k} \mu_{Y_k \to U_k}$.

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 $\mu_{V_j \to U_k}$ 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

$$\begin{aligned} \text{BEL}(U_k) &= \mu_{U_k} \mu_{Y_k \to U_k} \prod_j \mu_{V_j \to U_k} = P(U_k, Y_1, \ldots, Y_8) \\ &\propto P(U_k | Y_1, \ldots, Y_8). \end{aligned}$$

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 between iterations, $\text{BEL}(U_{k,i}) - \text{BEL}(U_{k,i-1}) \leq \epsilon, \forall U_k$, or a maximum number of iterations $i = i_{\max}$.

Figure 8.5 compares the performance of the sum-product algorithm used to decode the [8, 4] extended Hamming code with the performance of optimal decoding, performed by exhaustively searching the 2^4 codewords. While the sum-product algorithm performs very well, there is a clear loss in performance versus optimal decoding observable, in particular for large values of the signal-to-noise ratio E_b/N_0. This is not particularly surprising in the light of the fact

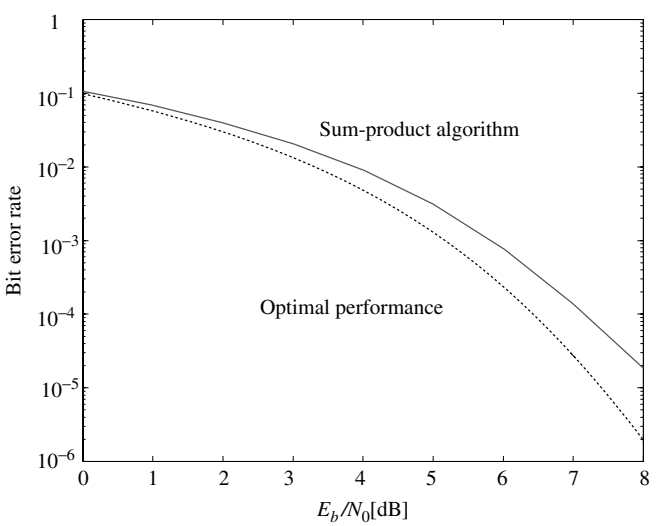


Figure 8.5 Performance of the min-sum algorithm operated on the parity-check factor graph of the [8, 4] extended Hamming code.

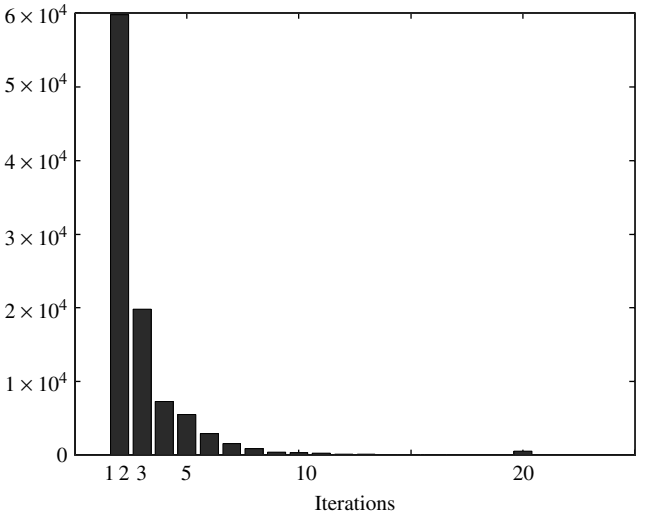


Figure 8.6 Histogram of the iterations required for convergence of the sum-product algorithm for the [8, 4] extended Hamming code at $E_b/N_0 = 0$ dB.

that the sum-product algorithm operated on the graph in Figure 8.3 does not exploit the rich code structure of the code. Later we will develop more complex graphical representations of this code; and we will see that operating the sum-product algorithm on a factor graph, which captures the code structure better, will give significantly better performance.

Figure 8.6 shows a histogram of the numbers of iterations required to achieve an accuracy of $\epsilon = 0.01$ at $E_b/N_0 = 0$ dB. The maximum number of iterations was set to $i_{\max} = 20$, which explains the accumulation of events at $i = 20$ for all those cases where no convergence is achieved.

8.5 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 7 (Section 7.7) and the sum-product algorithm. This will establish factor graphs as a general and convenient high-level description of trellis codes.

Figure 8.7 shows the trellis of the code used in Chapter 7 to discuss the APP algorithm. In the light of factor graph terminology, we now realize that the states $S_1, \ldots, S_8$ are random variables, and we represent these random variables by variable nodes as shown in the equivalent factor graph representation of Figure 8.8. These state nodes are no longer binary random variables as was

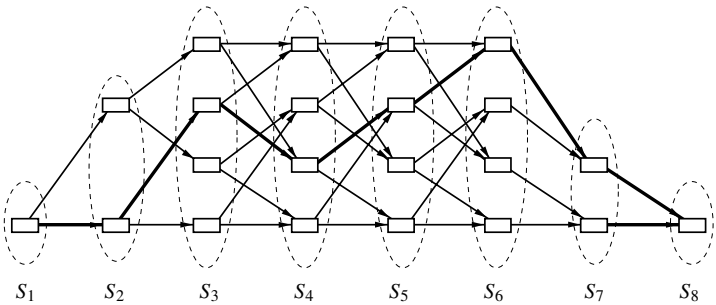


Figure 8.7 Example trellis of the short trellis code from Chapter 7.

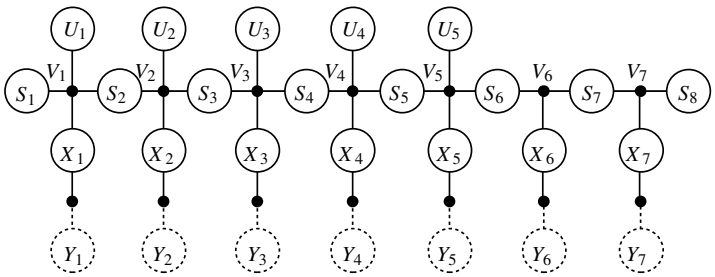


Figure 8.8 Factor graph of the trellis code in Figure 8.7.

the case in the factor graph of the [8, 4] Hamming code of Figure 8.3, 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.

We now establish the equivalence of the APP algorithm with the sum-product algorithm. The first three steps in the latter, according to page 233, 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.

Step 4 in the sum-product algorithm generates the messages

$$\mu_{U_r \to V_r} = P(U_r = u_r), \tag{8.8}$$

$$\mu_{X_r \to V_r} = P(Y_r = y_r | X_r), \tag{8.9}$$

and

$$\mu_{S_r \to V_r} = P(S_r, Y_1, \ldots, Y_{r-1})) = [\alpha_r(0), \ldots, \alpha_r(S-1)], \tag{8.10}$$

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, (8.8) has two entries for $u_r = \{0, 1\}$, (8.9) has four entries for $\{00, 01, 10, 11\}$, and (8.10) 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

$$\begin{aligned} \boldsymbol{\alpha}_{r+1} &= [\alpha_r(0), \ldots, \alpha_r(S-1)] \\ &= \sum_{U_r, S_r} \mu_{S_r \to V_r} \mu_{X_r \to V_r} \mu_{U_r \to V_r} P(S_{r+1} | S_r, U_r, X_r), \end{aligned} \tag{8.11}$$

where

$$\alpha_{r+1}(j) = \sum_i \alpha_r(i) \sum_{U_r} P(Y_r = y_r | X_r) P(U_r) P(S_{r+1} = j, X_r | S_r = i, U_r). \tag{8.12}$$

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 (7.29):

$$\alpha_{r+1}(j) = \sum_i \alpha_r(i) \gamma_{r+1}(i, j), \tag{8.13}$$

where $\gamma_{r+1}(i, j) = P(S_{r+1} = j, Y_r = y_r | S_r = i)$ as for APP decoding.

Figure 8.9 illustrates the processing that takes place at function node V_j.

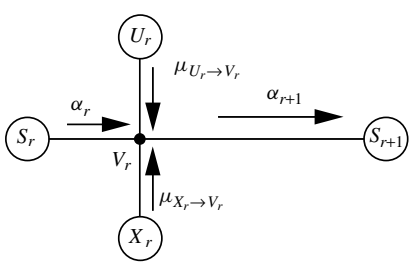


Figure 8.9 Function node processing during the "forward" phase.

Analogously, on the backward sweep of algorithm, the incoming messages are marginalized, multiplied, and forwarded to the previous state variable S_r as

$$\boldsymbol{\beta}_r = [\beta_r(0), \ldots, \beta_r(S-1)], \tag{8.14}$$

where

$$\beta_r(j) = \sum_i \beta_{r+1}(i) \sum_{U_r} P(Y_r = y_r|X_r)P(U_r)P(S_{r+1}, X_r|S_r, U_r). \tag{8.15}$$

As before, we can replace the last terms with γ_{r+1} to get

$$\beta_r(j) = \sum_i \beta_{r+1}(i)\gamma_{r+1}(j, i). \tag{8.16}$$

This is the backward recursion for β developed in Chapter 7 for APP decoding (7.41).

According to Step 7, the belief or conditional probability of state S_r is calculated as the product of the incoming messages:

$$P(S_r = i, Y_1 = y_1, \ldots, Y_L = y_L) = \text{BEL}(S_r = i) = \alpha_r(i)\beta_r(i). \tag{8.17}$$

From the derivation in Chapter 7 for (7.37), 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 (7.44) and (7.45).

8.6 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 7.7, 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 8.10 shows an edge in a general factor graph, connecting a variable node X with a function V. The function node V is a function of the variables X, $Z_1, \ldots, Z_N$, and the variable X is an argument in the functions $V, W_1, \ldots, W_I$. The nodes Z_n send the messages $\mu_{Z_n \to V}$, 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

$$\mu_{V \to X} = \sum_{Z_1, \ldots, Z_N} f_V(X, Z_1, \ldots, Z_N) \prod_{n=1}^{N} \mu_{Z_n \to V}. \tag{8.18}$$

From the right-hand side, the function nodes W_i send the messages $\mu_{W_i \to X}$ 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

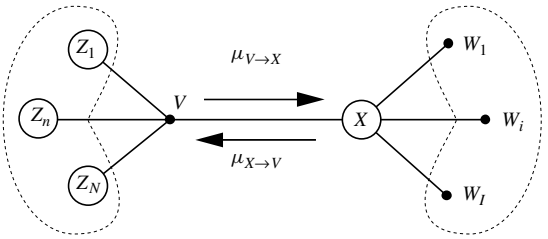


Figure 8.10 General edge in a factor graph.

node W_i. At node X, these incoming messages are multiplied componentwise and forwarded to the function node V as

$$\mu_{X\to V} = \prod_{i=1}^{I} \mu_{W_i\to X}. \tag{8.19}$$

Note that both $\mu_{V\to X}$ and $\mu_{X\to V}$ 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

$$\mathrm{BEL}(X) = \mu_{V\to X} \prod_{i=1}^{I} \mu_{W_i\to X} = \mu_{V\to X}\mu_{X\to V}, \tag{8.20}$$

where the product is again performed componentwise.

We will now show that the belief calculated in (8.20) 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 8.11. 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 $P(Q, Y|S) = P(Q|Y)P(Y|S)$, 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.

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 (8.19), 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

$$\begin{aligned} &\sum_{S_1,\ldots,S_L} P(S_1|T_r)P(\text{observation}|S_1)\cdots P(S_L|T_r)P(\text{observation}|S_L) \\ &\quad = \quad P(\text{observation}|T_r). \end{aligned} \tag{8.21}$$

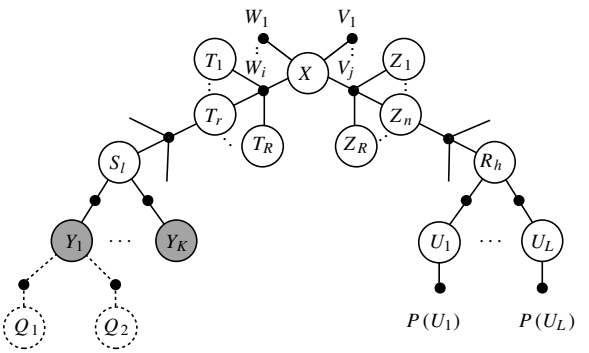


Figure 8.11 Structure of a tree factor graph.

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 $U_1, \ldots, U_L$ who pass a priori information to node R_h, which forwards the message

$$\sum_{U_1,\ldots,U_L} P(U_1)P(R_h|U_1)\cdots P(U_L)P(R_h|U_L) = P(R_h) \tag{8.22}$$

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 $\mu_{V_j\to X} = \sum_{Z_1\cdots Z_N} P(X|Z_1, \ldots, Z_N)\prod_n P(Z_n)$ to node X. Node X in turn multiplies the incoming messages and calculates

$$\begin{aligned}
\text{BEL}(X) &= \prod_i \mu_{W_i\to X} \prod_j \mu_{V_j\to X} \\
&= \prod_i \prod_{r=1}^{R} \sum_{T_r} P(\text{observation}, T_r|X) \\
&\quad \times \prod_j \sum_{Z_1} \cdots \sum_{Z_N} P(X|Z_1, \ldots, Z_N) \prod_{n=1}^{N} P(Z_n) \\
&= \prod_i \prod_j \prod_{r=1}^{R} \sum_{T_r} \sum_{Z_1} \cdots \sum_{Z_N} P(\text{observation}, T_r, Z_1, \ldots, Z_N, X) \\
&= P(X, \text{observation}),
\end{aligned} \tag{8.23}$$

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

$$\sum_{R_3 \cdots R_H} P(Z_n|R_1 = r_1)P(Z_n|R_2 = r_2)P(Z_n|R_3)P(R_3) \cdots P(Z_n|R_H)P(R_H)$$
$$= P(Z_n|R_1 = r_1, R_2 = r_2) = P(Z_n|O), \tag{8.24}$$

and the conditioning on observation O simply proceeds up to node X.

The sum-product algorithm does not produce exact a posteriori probabilities for loopy networks—that is, networks with cycles. Exact probability propagation in a loopy network is known to be NP-complete [6]. 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 is the topic of much current research interest.

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 that 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 8.12, where we have contracted the four information nodes into a single

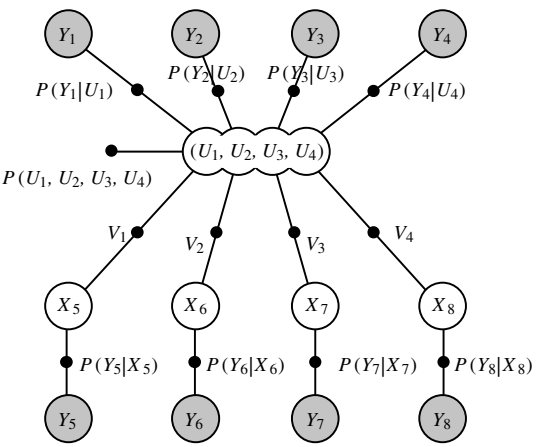


Figure 8.12 [8, 4] Hamming code factor graph representation as a tree.

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—that is, more complex than the simple binary factor graph of Figure 8.3.

8.7 BINARY FACTOR GRAPHS

We now consider binary factor graphs, where all variables participating in the sum-product computations are binary [14]. The [8, 4] extended Hamming code of Section 8.4 is an example of a binary factor graph. The variables $\mathbf{U}$ and $\mathbf{X}$ are binary. The observed variables $\mathbf{Y}$ are real-valued but fixed to their received value; thus their possible range of values plays no part in the sum-product algorithm.

If we limit all function nodes to parity-check equations, excluding single-variable functions that describe the probabilistic conditional dependence, as in $P(Y_k|X_k)$ and $P(U_k)$, and thus pass straightforward messages, we may further simplify the messages passed in the sum-product algorithm.

An important class of codes that satisfies these limitations are low-density parity-check codes (LDPCs), introduced by Gallager [12]. LDPCs, which are decoded by the sum-product algorithm, exhibit excellent performance rivaling that of turbo codes. In fact, irregular LDPCs have currently achieved the closest performance to capacity [25]. LDPCs, discussed in detail in Chapter 9, are represented by bipartite factor graphs that separate variable nodes.

A binary random variable x, $x \in \{0, 1\}$, has an associated probability mass function that can be represented by the vector $[p_0(x = 0|O), p_1(x = 1|O)]$, where $p_0 + p_1 = 1$. This probability message has cardinality 2.

Let us examine the messages sent from variable nodes and parity-check function nodes in a binary format. Initially, we consider nodes of degree 3, where the degree denotes the number of edges incident to the node.

Variable Node Messages

If a variable node Z is of degree 3, connected to function nodes A, B, and C, its outgoing message to A will be the product of the incoming messages from B and C:

$$\begin{aligned} \mu_{B\to Z} &= [P(Z = 0|O_B), P(Z = 1|O_B)] = [p_0, p_1], \\ \mu_{C\to Z} &= [P(Z = 0|O_C), P(Z = 1|O_C)] = [q_0, q_1], \end{aligned} \tag{8.25}$$

where we multiply the incoming messages componentwise and normalize by the componentwise sum so that $z_0 + z_1 = 1$. The normalized outgoing message is

$$\begin{aligned} [z_0, z_1] &= [P(Z = 0|O_{B,C}), P(Z = 1|O_{B,C})], \\ \mu_{Z\to A} &= [z_0, z_1] = \left[\frac{p_0 q_0}{p_0 q_0 + p_1 q_1}, \frac{p_1 q_1}{p_0 q_0 + p_1 q_1}\right]. \end{aligned} \tag{8.26}$$

Variable nodes of degree >3 send outgoing messages $[z_0, z_1]$ that are the normalized products of the corresponding incoming messages; that is, z_0 is the normalized product of all incoming probabilities p_0, and z_1 is defined likewise.

Parity-Check Node Messages

A parity-check function node A of degree 3 with incoming messages from variable nodes X and Y enforces the parity-check equation $f(X, Y, Z) = [Z \oplus Y \oplus Z = 0]$. The incoming messages to A are

$$\begin{aligned} \mu_{Z\to A} &= [P(X=0|O_x),\, P(X=1|O_x)] = [p_0,\, p_1], \\ \mu_{Y\to A} &= [P(Y=0|O_y),\, P(Y=1|O_y)] = [q_0,\, q_1] \end{aligned} \tag{8.27}$$

and A sends the outgoing message in the form

$$\mu_{A\to Z} = [P(Z=0|O_{x,y}),\, P(Z=1|O_{x,y})] = [z_0,\, z_1]. \tag{8.28}$$

The parity-check equation is satisfied for $Z = 0$ when $X + Y \mod 2 = 0$, and it is satisfied for $Z = 1$ when $X + Y \mod 2 = 1$. The outgoing message from the parity- check node is thus

$$\mu_{A\to X} = [z_0, z_1] = [p_0q_0 + p_1q_1,\, p_0q_1 + p_1q_0]. \tag{8.29}$$

Check nodes of degree > 3 send outgoing messages $[z_0, z_1]$, where z_0 is the sum of all even-parity solutions and z_1 is the sum of all odd-parity solutions to the node's parity-check equation. These messages are the same as those developed by Gallager [12] to decode LDPCs back in 1962.

Log Likelihood Ratio (LLR)

Instead of $[p_0, p_1]$, we may use the fact that $p_0 + p_1 = 1$ to simplify our representation to a single parameter, for example, p_0/p_1. A commonly used representation is the log likelihood ratio (LLR) $\lambda = \ln(p_0/p_1)$. The probability messages change as follows when using LLR representations.

LLR Variable Node Messages

Variable node Z receives LLR messages from function nodes B and C of the form

$$\mu_{B\to Z} = \ln(p_0/p_1) = \lambda_B, \qquad \mu_{C\to Z} = \ln(q_0/q_1) = \lambda_C.$$

The probability that $Z = 0$ still equals $\frac{p_0q_0}{p_0q_0+p_1q_1}$; however, the message format is now $\ln(z_0/z_1)$.

$$\begin{aligned} z_0 &= P(z=0|O_{B,C}) = \frac{p_0q_0}{p_0q_0 + p_1q_1}, \\ z_1 &= P(z=1|O_{B,C}) = \frac{p_1q_1}{p_0q_0 + p_1q_1}, \\ \ln(z_0/z_1) &= \ln\left(\frac{p_0q_0}{p_1q_1}\right) = \ln(p_0/p_1) + \ln(q_0/q_1). \end{aligned} \tag{8.30}$$

The outgoing message from node Z to function node A is now

$$\mu_{Z\to A} = \ln(z_0/z_1) = \ln(p_0/p_1) + \ln(q_0/q_1) = \mu_{B\to Z} + \mu_{C\to Z}; \tag{8.31}$$

that is, the LLR messages are simply added.

LLR Check Node Messages

Check node A receives LLR messages from variable nodes X and Y of the form

$$\mu_{Z\to A} = \ln(p_0/p_1) = \lambda_x, \qquad \mu_{Y\to A} = \ln(q_0/q_1) = \lambda_y. \tag{8.32}$$

Again appealing to the parity-check constraint $f(x, y, z) = [X \oplus Y \oplus Z = 0]$, we observe $z_0/z_1 = \frac{p_0q_0+p_1q_1}{p_0q_1+p_1q_0}$, and the outgoing LLR message from check node A to variable Z is

$$\mu_{A\to Z} = \lambda_z = \ln(z_0/z_1) = \ln\left(\frac{p_0q_0 + p_1q_1}{p_0q_1 + p_1q_0}\right). \tag{8.33}$$

However, we want to be able to express the outgoing LLR message λ_z in terms of the incoming LLR messages λ_x and λ_y. This requires some trigonometric manipulations. Noting that $p_0/p_1 = e^{\lambda_x}$ and $q_0/q_1 = e^{\lambda_y}$ and dividing both z_0 and z_1 by p_1q_1, we obtain

$$\lambda_z = \ln\left(\frac{e^{\lambda_x}e^{\lambda_y} + 1}{e^{\lambda_x} + e^{\lambda_y}}\right). \tag{8.34}$$

Factoring out $e^{\frac{\lambda_x}{2}}e^{\frac{\lambda_y}{2}}$ from numerator and denominator gives

$$\lambda_z = \ln\left(\frac{\cosh\left(\frac{\lambda_x+\lambda_y}{2}\right)}{\cosh\left(\frac{\lambda_x-\lambda_y}{2}\right)}\right). \tag{8.35}$$

Using the expansion for $\cosh(x \pm y)$ and dividing numerator and denominator by $\cosh\left(\frac{\lambda_x}{2}\right)\cosh\left(\frac{\lambda_y}{2}\right)$ gives

$$\lambda_z = \ln\left(\frac{1 + \tanh\left(\frac{\lambda_x}{2}\right)\tanh\left(\frac{\lambda_y}{2}\right)}{1 - \tanh\left(\frac{\lambda_x}{2}\right)\tanh\left(\frac{\lambda_y}{2}\right)}\right), \tag{8.36}$$

as used in ref. 3. With the relation $\tanh^{-1}(z) = \frac{1}{2}\ln\left(\frac{1+z}{1-z}\right)$, we obtain

$$\lambda_z = 2\tanh^{-1}\left(\tanh\left(\frac{\lambda_x}{2}\right)\tanh\left(\frac{\lambda_y}{2}\right)\right), \tag{8.37}$$

as in ref. 13.

The check node LLR message can be approximated by using the fact that for $x \gg 1$, $\ln(\cosh(x)) \approx |x| - \ln(2)$. Thus,

$$\lambda_z \approx \left|\frac{\lambda_x + \lambda_y}{2}\right| - \left|\frac{\lambda_x - \lambda_y}{2}\right| = \operatorname{sgn}(\lambda_x)\operatorname{sgn}(\lambda_y)\min\left(|\lambda_x|, |\lambda_y|\right). \tag{8.38}$$

This happens to be the min-sum update rule, as used in the Viterbi algorithm [28] presented in Chapter 7.

For variable nodes of degree higher than 3, the outgoing LLR message is simply the sum of all incoming LLR messages. If variable node Z has degree i, the message it sends to function node A will be

$$\mu_{z\to A} = \lambda_z = \sum_{\substack{j=1 \\ (j\neq A)}}^{i} \lambda_j \tag{8.39}$$

For parity-check nodes of degree higher than 3, the incoming LLR messages multiply together through the tanh transformation. Check node A of degree i sends to variable node Z the message

$$\mu_{A\to Z} = \lambda_A = 2\tanh^{-1}\left(\prod_{\substack{j=1 \\ (j\neq Z)}}^{i} \tanh\left(\frac{\lambda_j}{2}\right)\right). \tag{8.40}$$

Binary factor graphs thus allow for a significant simplification of the computations required at the nodes.

Current research efforts to develop analog implementations of binary sum-product algorithms [18, 21, 31] are producing promising results with potentially far-reaching implications. By exploiting the nonlinear tanh response inherent in basic transistor behavior, a binary factor graph processor that performs the sum-product algorithm may be designed from a simple analog building block, the Gilbert multiplier. The real-valued probability messages become currents in such an analog decoder, and LLRs correspond to voltages. Iterative decoding is replaced by continuous-time processing, where the network settles into a steady-state solution. Analog probabilistic decoders are estimated to outperform their digital counterparts in speed and/or power consumption by two orders of magnitude [16], as also discussed in Chapter 1.

8.8 NORMAL FACTOR GRAPHS

Normal graphs, defined by Forney in [7], are an extension of factor graphs. As described in Section 8.2, factor graphs have two types of nodes, variable and function nodes, and direct connections are only allowed between different node types—that is, between variable and function nodes. No connections between the

Figure 8.13 Factor graph and normal graph of a Markov chain.

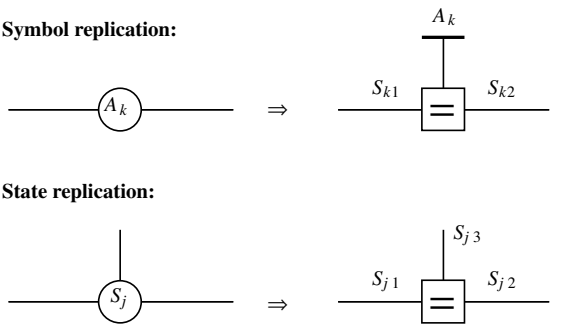


Figure 8.14 Symbol and state variable replication [7].

same type of node are allowed. Normal graphs, however, allow direct connections between function nodes by representing variables as edges rather than nodes.

Normal graphs define all variables to be either symbol variables (external or leaf edges) or state variables (internal edges). Symbol variables are of degree 1; that is, they have one incident edge connecting to one function node. State variables are of degree 2, connecting to exactly two function nodes. However, function nodes themselves are not restricted as to degree. Figure 8.13 shows a Markov chain, which meets the normal requirements for variable degree, as a factor graph and as a normal graph. Variables of degree 2 simply pass messages along, so it is reasonable to represent them as an edge. Note the "half-edge" symbol on the normal graph variable C, denoting it as an edge of degree 1. The function nodes are expanded from a small node in the factor graph to a larger square as they become the sole computational nodes.

If the original factor graph representation does not meet the variable degree requirements, it may be converted to a normal graph through the following replication of variables of higher degree and use of the "equality" function node. We use the same terminology as Forney [7]. See Figure 8.14 for the transformation from factor graph to normal graph through symbol and state variable replication.

Symbol Variable Replication

The factor graph symbol variable node A_k of degree >1 is replicated, with each of its incident edges replaced by state variable edges $S_{ki} = A_k$ of degree 2. The symbol variable node itself is represented by a half-edge A_k of degree 1. All edges S_{ki} and A_k connect to an "equality" function node that defines all

connecting variable edges to be equal. The equality function node takes the node position of the original factor graph variable A_k.

State Variable Replication

The factor graph state variable node S_j of degree >2 is replicated by replacing each of its incident edges with a state variable edge S_{ji} of degree 2, $S_{ji} = S_j$. All variable edges S_{ji} connect to an "equality" function node that is located at the original node position of S_j. The equality constraint does not set the messages along each incident edge equal; it merely sets the value of the variable represented by each incident edge equal.

Any factor graph may be converted to a normal graph by use of replication for variable nodes of higher degree and direct replacement of all variable nodes of appropriate degree by edges, half-edges for symbol variables, or ordinary edges for state variables.

The sum-product algorithm discussed in Section 8.3 operates on normal graphs in exactly the same manner as on factor graphs. As with factor graphs, the sum-product algorithm is exact for cycle-free normal graphs, while it is iterative and only approximate on cyclic normal graphs.

The normal graph provides clear separation between variable edges and function nodes. Symbol variable edges are used for observable, I/O nodes. State variable edges, being of degree 2 as in the Markov chain example discussed in Section 8.2, are used simply for message-passing. All computations of the sum-product algorithm, therefore, occur at the function nodes on normal graphs. The incoming variable edges supply messages that are multiplied at the function node together with the function itself and summed over the incoming variables. This separation of function and use of the "equality" function node provides for a direct conversion from the normal graph to an analog transistor-level representation, as in ref. 17.

BIBLIOGRAPHY

1. S. M. Aji and R. J. McEliece, "The generalized distributive law," *IEEE Trans. Inform. Theory,* vol. IT-46, pp. 325–343, March 2000.
2. 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, March 1974.
3. G. Battail, M. C. Decouvelaere, and P. Godlewski, "Replication decoding," *IEEE Trans. Inform. Theory,* vol. IT-25, pp. 332–345, May 1979.
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. 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. 1060–1074, 1993.

6. G. F. Cooper, "The computational complexity of probabilistic inference using Bayesian belief networks," *Artificial Intell.*, vol. 42, pp. 393–405, 1990.

7. G. D. Forney, Jr., "Codes on graphs: Normal realizations," *IEEE Trans. Inform. Theory,* vol. IT-47, pp. 520–548, Feb. 2001.

8. B. J. Frey, *Graphical Models for Machine Learning and Digital Communication*, MIT Press, Cambridge, MA; 1998.

9. B. J. Frey and F. R. Kschischang, "Probability Propagation and Iterative Decoding," *Proceedings, 34th Allerton Conference on Communication, Control and Computing 1996*, Champaign-Urbana, Illinois, 1996.

10. B. J. Frey and D. J. C. MacKay, "A Revolution: Belief Propagation in Graphs with Cycles," *Neural Information Processing Systems Conference*, Denver, Colorado, Dece. 1997.

11. R. G. Gallager, "Low-density parity-check codes," *IRE Trans. Inform. Theory,* vol. IT-8, pp. 21–28, Jan. 1962.

12. R. G. Gallager, *Low-Density Parity-Check Codes*, MIT Press, Cambridge, MA, 1963.

13. J. Hagenauer, E. Offer, and L. Papke, "Iterative Decoding of Binary Block and Convolutional Codes," *IEEE Trans. Inform. Theory,* vol. IT-42, pp. 429–445, March 1996.

14. F. R. Kschischang, B. J. Frey, and H.-A. Loeliger, "Factor Graphs and the Sum-Product Algorithm," *IEEE Trans. Inform. Theory,* vol. IT-47, pp. 498–519, Feb. 2001.

15. S. L. Lauritzen and D. J. Spiegelhalter, "Local computations with probabilities on graphical structures and their application to expert systems," *J. R. Stat. Soc. B*, vol. 50, pp. 157–224, 1988.

16. H.-A. Loeliger, F. Tarköy, F. Lustenberger and M. Helfenstein, "Decoding in analog VLSI," *IEEE Commun. Mag.*, pp. 99–101, April 1999.

17. F. Lustenberger, *On the Design of Analog VLSI Iterative Decoders*, Swiss Federal Institute of Technology, Zurich, doctoral dissertation, 2000.

18. L. Lustenberger, M. Helfenstein, H.-A. Loeliger, F. Tarköy, and G. S. Moschytz, "All-analog decoder for a binary (18, 9, 5) tail-biting trellis code," *Proceedings European Solid-State Circuits Conference*, pp. 362–365, Duisberg, Sept. 1999.

19. 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.

20. F. J, MacWilliams and N. J. A. Sloane, *The Theory of Error-Correcting Codes*, North-Holland Mathematical Library, vol. 16, North-Holland, Amsterdam, 1988.

21. M. Moerz, T. Gabara, R. Yan, and J. Hagenauer, "An analog 0.25 μm BiCMOS tailbiting MAP decoder," *IEEE Proceedings, Int. Solid-State Circuits Conference*, pp. 356–357, San Francisco, Feb. 2000.

22. J. Pearl, "Fusion, propagation, and structuring in belief networks," *Artificial Intell.*, vol. 29, pp. 241–288, 1986.

23. J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers, San Francisco, 1988.

24. J. G. Proakis, *Digital Communications*, 3rd edition, McGraw-Hill, New York, 1995.

25. T. Richardson, A. Shokrollahi, and R. Urbanke, "Design of capacity-approaching irregular low-density parity-check codes," *IEEE Trans. Inform. Theory,* vol. IT-47, pp. 619–637, Feb. 2001.

26. R. M. Tanner, "A recursive approach to low complexity codes," *IEEE Trans. Inform. Theory,* vol. IT-27, pp. 533–547, Sept. 1981.

27. R. M. Tanner, Patent no. 4,295,218, "Error-correcting coding system," Oct. 13, 1981.

28. 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.

29. N. Wiberg, "Codes and Decoding on General Graphs," Linköping University, Linköping, Sweden, doctoral dissertation, 1996.

30. 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.

31. C. Winstead, J. Dai, S. Yu, S. Little, C. Myers, R. Harrison, and C. Schlegel, "CMOS analog MAP decoder for (8,4) hamming code," *IEEE J. Solid State Cir.*, Jan. 2004.

CHAPTER 9

Low-Density Parity-Check Codes

WITH CHRISTOPHER WINSTEAD

9.1 INTRODUCTION

The emphasis of the book to this point has been on trellis codes and their decoding algorithms. The primary advantage of trellis codes is that they have convenient trellis representations and soft-decision decoders. In Chapters 5 and 8, it was shown that block codes also had a structure that enabled iterative soft-decision decoding. With this background, we now consider a class of linear block codes, referred to as low-density parity-check codes, that offer performance spectacularly close to theoretical limits when decoded using the iterative soft-decision decoding algorithm based on the factor graphs of Chapter 8. Indeed, a rate $R = 1/2$ LDPC code with a blocklength of 10^7 bits has been shown to perform within 0.0045 dB of the Shannon limit for the binary input additive white Gaussian noise channel at a bit error rate of 10^{-6} [4], discussed in more detail in Section 9.7.

Low-density parity-check (LDPC) codes and a corresponding iterative decoding algorithm were first introduced by Gallager more than 40 years ago [8, 9]. However, for the next several decades LDPC codes were largely forgotten primarily because computers of the time could not simulate the performance of these codes with meaningful block lengths at low error rates. Following the discovery of turbo codes (see Chapter 10), LDPC codes were rediscovered through the work of McKay and Neal [16, 17] and have become a major research topic. LDPC codes significantly differ from the trellis codes and block codes discussed so far. First, they are constructed in a random manner; second, they have a decoding algorithm whose complexity is linear in the block length of the code. When combined with their spectacular performance, this makes LDPC codes a compelling class of codes for many applications.

In this chapter, the basic results concerning low-density parity-check codes are discussed. We begin with the basic construction of regular LDPC codes and their representation as bipartite graphs. The graphical representation of LDPC

Trellis and Turbo Coding, by Christian B. Schlegel and Lance C. Pérez
ISBN 0-471-22755-2

codes leads to the notion of irregular LDPC codes and to the density evolution analysis technique which predicts the ultimate 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 [16] 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 code length. The chapter ends with a discussion of irregular repeat-accumulate (RA) codes that also have a graphical representation and offer near-capacity performance.

9.2 LDPC CODES AND GRAPHS

It was shown in Section 5.2 that a linear block code $\mathcal{C}$ of rate $R = k/n$ can be defined in terms of a $(n-k) \times n$ parity-check matrix $\boldsymbol{H} = [\boldsymbol{h}_1, \boldsymbol{h}_2, \ldots, \boldsymbol{h}_n]$. Each entry h_{ij} of $\boldsymbol{H}$ 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 $\boldsymbol{x}$ that lie in the (right) nullspace of $\boldsymbol{H}$, that is, $\boldsymbol{H}\boldsymbol{x} = 0$. Given a parity-check matrix $\boldsymbol{H}$, we can find a corresponding $k \times n$ generator matrix $\boldsymbol{G}$ such that $\boldsymbol{G}\boldsymbol{H}^T = \boldsymbol{0}$. The generator matrix can be used as an encoder according to $\boldsymbol{x}^T = \boldsymbol{u}^T\boldsymbol{G}$.

In its simplest guise, an LDPC code is a linear block code with a parity-check matrix that is "sparse"; that is, it has a small number of nonzero entries. In ref. 8 and 9, Gallager proposed constructing LDPC codes by randomly placing 1's and 0's in a $m \times n$ parity-check matrix $\boldsymbol{H}$ subject to the constraint that each row of $\boldsymbol{H}$ had the same number d_c of 1's and each column of $\boldsymbol{H}$ had the same number d_v of 1's. For example, the $m = 15 \times n = 20$ parity-check matrix, shown in Figure 9.1 has $d_v = 3$ and $d_c = 4$ and defines an LDPC code with length $n = 20$. Codes of this form are referred to as *regular* (d_v, d_c) LDPC codes of length n. In a (d_v, d_c) LDPC code each information bit is involved in d_v parity checks and each parity-check bit involves d_c information bits. The fraction of 1's in the

$$\boldsymbol{H}_1 = \left[\begin{array}{cccccccccccccccccccc}
1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\
\hline
1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\
\hline
1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1
\end{array}\right]$$

Figure 9.1 Gallager-type low-density parity-check matrix $\boldsymbol{H}_1$.

parity-check matrix of a regular LDPC code is $\frac{md_c}{mn} = \frac{d_c}{n}$, which approaches zero as the block length gets large and leads to the name low-density parity-check codes.

We must now determine the rate of the code defined by $\boldsymbol{H}_1$. Since the parity-check matrix is randomly constructed, there is no guarantee that the rows are linearly independent and the matrix is full rank. Indeed, the rank of $\boldsymbol{H}_1$ 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, and it is also convenient to refer to $1 - \frac{m}{n} = 1 - \frac{d_v}{d_c}$ as the *designed* rate of the code.

Having defined 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, $md_c = nd_v$ and that d_v must be less than d_c in order to have a rate less than $R = 1$. Assuming that the block length n and the code rate are determined by the application, it remains to determine appropriate values for d_v and d_c. In ref. 9, Gallager showed that the minimum distance of a typical regular LDPC code increased linearly with n, provided that $d_v \geq 3$. Therefore, most regular LDPC codes are constructed with d_v and d_c on the order of 3 or 4, subject to the above constraints. For large block lengths, the random placement of 1's in $\boldsymbol{H}$ 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 [16, 17].

An important advance in the theory of LDPC codes occurred when Tanner [21] used bipartite graphs to provide a graphical representation of the parity-check matrix. (As a consequence, the bipartite graph of a LDPC code is sometimes referred to as a Tanner graph.) As discussed in Chapter 8, 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 referred to as *variable* nodes and *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 $\boldsymbol{H}$. An edge exists between the ith variable node and the jth check node if and only if $h_{ij} = 1$. The bipartite graph corresponding to the parity-check matrix $\boldsymbol{H}_1$ is shown in Figure 9.2. In a graph, 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. This is the current approach to addressing LDPC codes. Note that the pair (d_v, d_c), together with the code length n, specifies an *ensemble* of codes, rather than any particular code. This ensemble is denoted by

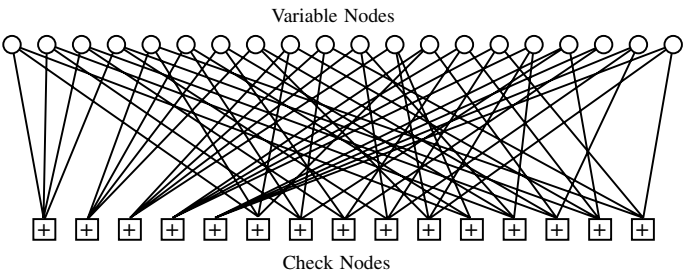


Figure 9.2 Bipartite graph for a $(3, 4)$ regular LDPC code with parity-check matrix $\boldsymbol{H}_1$.

$C^n(d_v, d_c)$. Once the degrees of the nodes are chosen, we are still free to choose which particular connections are made in the graph.

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. 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. An individual edge is specified by the pair $(i, \pi(i))$, which indicates that the ith variable node socket is connected to the $\pi(i)$th check-node socket.

Selecting a random code from the ensemble $C^n(d_v, d_c)$ therefore amounts to randomly selecting a permutation on nd_v elements. Many permutations will result in a graph that contains *parallel edges*—that is, 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.

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:

$$\gamma(x) = \sum_i \gamma_i x^{i-1}, \tag{9.1}$$

such that $\gamma(1) = 1$. The coefficients γ_i denote the fraction of *edges* in the graph which are connected to a node of degree i. We will also use the notation $\int \gamma$ to denote the inverse average node degree, given by

$$\int \gamma \;\Rightarrow\; \int_0^1 \gamma(x)\, dx = \sum_i \frac{\gamma_i}{i}. \tag{9.2}$$

Let d be the average node degree corresponding to the degree distribution γ. To show that $\int \gamma = \frac{1}{d}$, let n be the total number of nodes, and let n_i be the number of nodes of degree i. The total number of edges in the graph is $d \cdot n$. Because γ_i is the fraction of edges connected to a node of degree i, we conclude that $\frac{\gamma_i}{i} = \frac{n_i}{d \cdot n}$. Completing the sum of (9.2), we arrive at $\int \gamma = \frac{1}{d}$.

The code length 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. A graph G from this ensemble will have n variable nodes. The number of check nodes, m, is given by

$$m = n\frac{\int \rho}{\int \lambda}. \tag{9.3}$$

The number of degree-i variable nodes in G is

$$n\widetilde{\lambda_i} = n\frac{\lambda_i}{i\int \lambda}. \tag{9.4}$$

where $\tilde{\lambda_i}$ denotes the fraction of variable nodes of degree i. Similarly, the number of degree-i check nodes in G is

$$m\widetilde{\rho_i} = m\frac{\rho_i}{i\int \rho}, \tag{9.5}$$

where $\tilde{\rho_i}$ denotes the fraction of check nodes of degree i. And the design rate of the code represented by G is

$$r = \frac{n-m}{n}. \tag{9.6}$$

We can once again enumerate the variable-node and check-node sockets of the irregular code graph. Selection of a code from the ensemble is a selection of a permutation on N_E elements, where N_E is the number of edges in the graph, given by

$$N_E = \frac{n}{\int \lambda}. \tag{9.7}$$

We will want to rule out parallel edges in irregular codes as well.

9.3 MESSAGE PASSING DECODING ALGORITHMS

It was stated in the introduction that a principal advantage of low-density parity-check codes is that they can be decoded using an iterative decoding algorithm whose complexity grows linearly with the block length of the code. Now that we have a graphical representation of LDPC codes, we can decode them using the

belief propagation decoding algorithm developed in Section 8.7. Belief propagation is one instance of a broad class of message passing algorithms on graphs as discussed in [1, 7, 12, 22]. Similar decoding algorithms for the binary erasure channel and the binary symmetric channel are discussed in Sections 9.5 and 9.6, respectively. All message passing algorithms must, however, respect the following rule, which was introduced with turbo codes.

Rule 9.1 (Extrinsic Information Principle) *A message sent from a node n along an edge e cannot depend on any message previously received on edge e.*

Before stating the algorithm, it is necessary to formulate the decoding problem. An LDPC code is constructed in terms of its parity-check matrix $\boldsymbol{H}$ which is used for decoding. To encode the information sequence $\boldsymbol{u}$, it is necessary to derive a generator matrix $\boldsymbol{G}$ such that $\boldsymbol{G}\boldsymbol{H}^T = \boldsymbol{0}$. Finding a suitable $\boldsymbol{G}$ is greatly simplified if we first convert $\boldsymbol{H}$ into the equivalent systematic parity-check matrix $\boldsymbol{H}_S = \left[\boldsymbol{A}|\boldsymbol{I}_{n-k}\right]$. As shown in Section 5.2, the systematic generator matrix is now given by

$$\boldsymbol{G}_s = \left[\boldsymbol{I}_k|\boldsymbol{A}^T\right],$$

and the information sequence is encoded as $\boldsymbol{x}^T = \boldsymbol{u}^T\boldsymbol{G}_s$. It is worth noting that encoding by matrix multiplication has complexity $\mathcal{O}(n^2)$ and that, in general, LDPC codes have linear decoding complexity, but quadratic *encoding* complexity. Methods for reducing the encoding complexity are therefore of great interest, and some of them are discussed in Section 9.8.

The codeword $\boldsymbol{x}$ is transmitted across the additive white Gaussian noise (AWGN) channel described in Chapter 2 using BPSK modulation resulting in the received sequence $\boldsymbol{r} = (2\boldsymbol{x} - 1) + \boldsymbol{n}$. The optimal decoder for the AWGN channel is the MAP decoder derived in Chapter 7 that computes the log-likelihood ratio

$$\Lambda(x_r) = \log\left(\frac{P(x_r = 1|\boldsymbol{y})}{P(x_r = 0|\boldsymbol{y})}\right)$$

and makes a decision by comparing this LLR to the threshold zero. As shown in Chapter 8, belief propagation on a graph with cycles can closely approximate the MAP algorithm, and we can state the decoding algorithm for LDPC codes on the AWGN channel using these results.

Let $\mathcal{A} = \{-1, +1\}$ denote the message alphabet, let $r_i \in \mathcal{R}$ denote the received symbol at variable node i, and let $\Lambda_i \in \mathcal{R}$ denote the decision at variable node i. A message from variable node i to check node j is represented by $\mu_{i\to j} \in \mathcal{R}$, and a message from check node j to variable node i is $\beta_{j\to i} \in \mathcal{R}$. Let $C_{j\setminus i}$ be the set of variable nodes which connect to check node j, excluding variable node i. Similarly, let $V_{i\setminus j}$ be the set of check nodes which connect to variable node i, excluding check node j. The decoding algorithm is then as follows:

Step 1: Initialize $\Lambda_i = \frac{2}{\sigma^2}r_i$ for each variable node. ($\sigma^2 = N_0/2$).
Step 2: Variable nodes send $\mu_{i\to j} = \lambda_i$ to each check node $j \in V_i$.

Step 3: Check nodes connected to variable node i send

$$\beta_{j\to i} = 2\tanh^{-1}\left(\prod_{l\in C_{j\setminus i}} \tanh\left(\frac{\Lambda_l}{2}\right)\right), \tag{9.8}$$

as shown in Section 8.7.

Step 4: Variable nodes connected to check nodes j send

$$\mu_{i\to j} = \sum_{l\in V_{i\setminus j}} \beta_{l\to i} \tag{9.9}$$

as shown in Section 8.7.

Step 5: When a fixed number of iterations have been completed or the estimated code word $\hat{\boldsymbol{x}}$ satisfies the syndrome constraint $\boldsymbol{H}\hat{\boldsymbol{x}} = \boldsymbol{0}$, stop. Otherwise return to Step 3.

In order to illustrate the efficacy of this algorithm, we present several simulation results. Figure 9.3 shows simulations results for several regular LDPC codes with a range of block lengths. This figure shows the superior performance of these codes with increasing block length. Figure 9.4 gives simulations results for a regular LDPC code, an irregular LDPC code and a turbo code (see Chapter 10) of

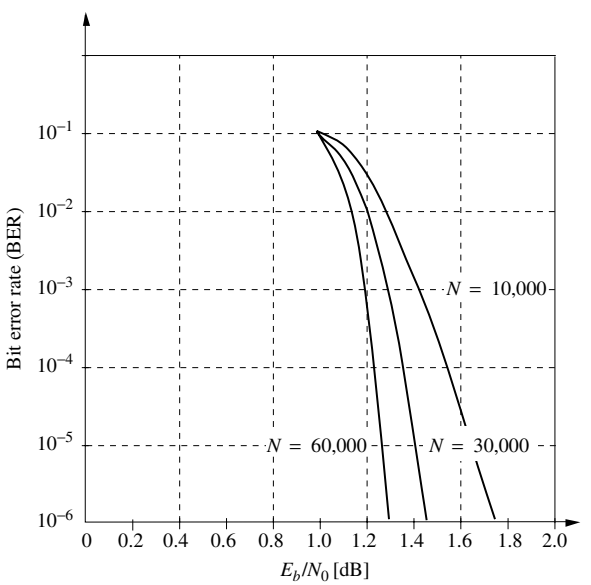


Figure 9.3 Simulation results for randomly constructed rate $R = 1/2$ regular (3,6) LDPC codes with varying block lengths.

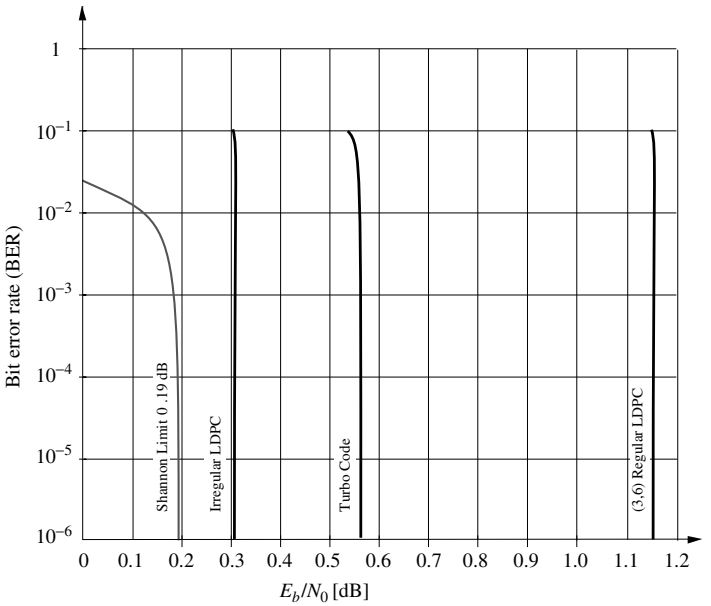


Figure 9.4 Simulation results for a regular (3,6) LDPC, an optimized irregular LDPC and a turbo code. All three codes are of rate $R = 1/2$ and have block length 10^6 [18].

similar complexity on the AWGN channel [18]. This figure clearly demonstrates the advantages of using irregular LDPC codes in terms of performance. The theoretical motivation and design methodology for irregular codes is discussed in the next section. The irregular node degree, however, can greatly increase the implementation complexity, especially if very high node degrees are used (see later in this chapter).

The check node's rule (9.8) 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 some extra combinational logic and an adder. Such decoders provide very good performance with only a few bits of precision. If we want to keep the complexity even simpler, we can use the max-Log approximation, in which we can replace the above rule (see Chapter 8) with

$$\beta_{j\to i} \approx \min_{l\in C_{j\setminus i}} (\{|\lambda_l|\}) \prod_{l\in C_{j\setminus i}} \operatorname{sign}(\Lambda_l)\,. \tag{9.10}$$

Some performance loss will result from this approximation, but it may be justified by the overall savings in decoder complexity.

In contrast to LDPC codes, as we will see in Chapter 10, turbo codes use two component APP trellis decoders joined through a pseudo-random interleaver. The APP algorithm can itself be viewed as an instance of belief propagation between nodes which represent individual trellis sections. As noted in [7], normal graphs provide an interesting comparison between a number of turbo-style codes. The

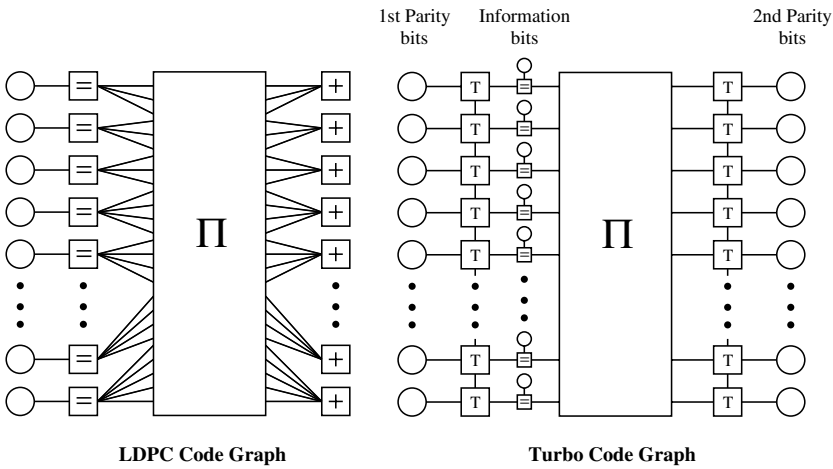


Figure 9.5 Normal code graphs for LDPC and turbo codes.

graphs in Figure 9.5 illustrate the normal graphs for LDPC codes and standard turbo codes. Both codes (and many others) can be represented as graph structures joined by a randomly selected permutation Π. This graph representation also exposes the turbo decoding algorithm as an instance of message-passing on code graphs.

9.4 DENSITY EVOLUTION

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 ref. 18 and [19] use an approach called "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 [8, 9] discussed in Section 9.6, and the modern extensions 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 $\mathcal{C}(\sigma)$ has an associated *threshold* σ^*. The threshold is analogous to the Shannon capacity in that the error probability for transmission on $\mathcal{C}(\sigma)$ for a randomly chosen (λ, ρ)-code can be made arbitrarily small for a growing block 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.

The error probability over the specified channel for a randomly chosen (λ, ρ)-code can be made arbitrarily small if the channel's parameter is better than the threshold. Threshold results for several ensembles on various channels are reported in ref. 19. The qualities of different ensembles of regular and irregular LDPC codes are studied and optimized via density evolution in ref. 5 and 18.

In designing good codes, we choose the ensemble with the best threshold, from which we select a code with the largest length which can be accommodated by our implementation. As 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.

9.5 DENSITY EVOLUTION FOR BINARY ERASURE CHANNELS

We first illustrate the method of density evolution using the binary erasure channel (BEC), shown in Figure 9.6. 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 [3]. The channel parameter for the BEC is the erasure probability ε.

Let $\mathcal{A} = \{-1, 0, +1\}$ denote the message alphabet, let $r_i \in \mathcal{A}$ be the received symbol at variable node i, and let $d_i \in \mathcal{A}$ be the decision at variable node i. A message from variable node i to check node j is represented by $\mu_{i\to j} \in \mathcal{A}$, and a message from check node j to variable node i is $\beta_{j\to i} \in \mathcal{A}$. Let $C_{j\setminus i}$ be the set of variable nodes which connect to check node j, excluding variable node i. Similarly, let $V_{i\setminus j}$ be the set of check nodes which connect to variable node i, excluding check node j. The decoding algorithm for the BEC proceeds as follows [15]:

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 $\mu_{i\to j} = d_i$ to each check node $j \in V_i$.

Step 3: Check nodes connected to variable node i send $\beta_{j\to i} = \prod_{l\in C_{j\setminus i}} \mu_{l\to j}$ 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 $\beta_{j\to i} \neq 0$, set $d_i = \beta_{j\to i}$ and declare variable i to be *known*. (Note that known variables will never have to be changed anymore for the simple BEC.)

Step 5: When all variables are known, stop. Otherwise go back to Step 2.

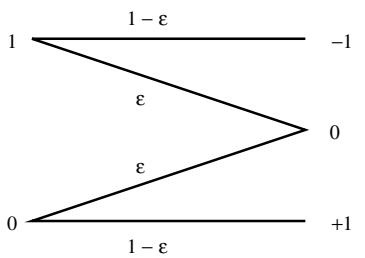


Figure 9.6 Binary erasure channel model.

Under this algorithm, a check node can only output a nonzero message on an edge if all incoming messages on other edges are nonzero. If there are not enough known variables when decoding begins, then the algorithm will fail.

Density evolution analysis for this channel and algorithm 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 $l \to \infty$. 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:

$$\begin{aligned} p_u^{(l-1)} &= 1 - \left[1 - p_v^{(l-1)}\right]^{d_c - 1}, \\ p_v^{(l)} &= p_0 \left[p_u^{(l-1)}\right]^{d_v - 1}, \end{aligned} \tag{9.11}$$

where $p_0 = \varepsilon$ is the initial density of erasures. Equation (9.11) contains an important assumption, which is that the probabilities of node u or v do not depend on themselves, that is, that there are no cycles in the local subgraph which could introduce such dependencies. This condition is best illustrated by Figure 9.7.

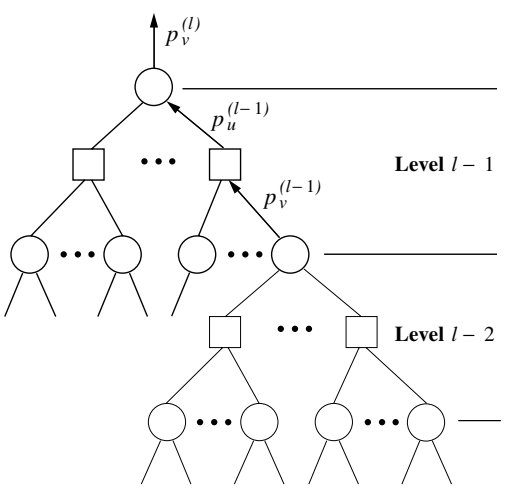


Figure 9.7 Local subtree structure of a large LDPC code.

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, at least approximately, the case for large LDPC codes whose loops in the code's graph are large. Hence the necessity to design LDPC codes with large loops, since otherwise the dependencies introduced in the message passing will, in general, degrade performance.

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 (9.11) becomes [14]

$$\begin{aligned} p_u^{(l-1)} &= 1 - \sum_{i=1}^{d_c} \rho_i \left[1 - p_v^{(l-1)}\right]^{i-1} = 1 - \rho\left(1 - p_v^{(l-1)}\right) \\ p_v^{(l)} &= p_0 \sum_{j=1}^{d_v} \lambda_j \left[p_u^{(l-1)}\right]^{j-1} = p_0 \lambda\left(p_u^{(l-1)}\right) \end{aligned} \tag{9.12}$$

It can be derived by the following simple argument: the probability that a check node's output is given by

$$\begin{aligned} P\left(\beta^{(l-1)} \neq 0\right) &= \sum_{i=1}^{d_c} P\ (\text{all } i-1 \text{ inputs known} \mid \text{node deg} = i)\ P\ (\text{node deg} = i) \\ &= \sum_{i=1}^{d_c} P\left(\mu^{(l-1)} \neq 0\right)^{i-1} P\ (\text{node deg} = i) \end{aligned} \tag{9.13}$$

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 line of (9.12) is an immediate consequence.

The same reasoning is used to obtain the second line of (9.12):

$$\begin{aligned} \mathrm{P}\left(\mu^{(l-1)} = 0\right) &= \sum_{j=1}^{d_v} P\ (j-1 \text{ inputs unknown} \mid \text{node deg} = j\ ) \\ &= \sum_{j=1}^{d_v} P\left(\beta^{(l-1)} = 0\right)^{j-1} P\ (\text{node deg}\ = j) \end{aligned} \tag{9.14}$$

Finally, we assume that the probability that a check node has degree i is equal to ρ_i. Similarly, the probability for 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 (9.12).

Figure 9.8 shows the evolution of the erasure probability for the ensemble of (3,6) LDPC codes for a number of input erasure values $p_0 = \varepsilon$. The zigzag curve

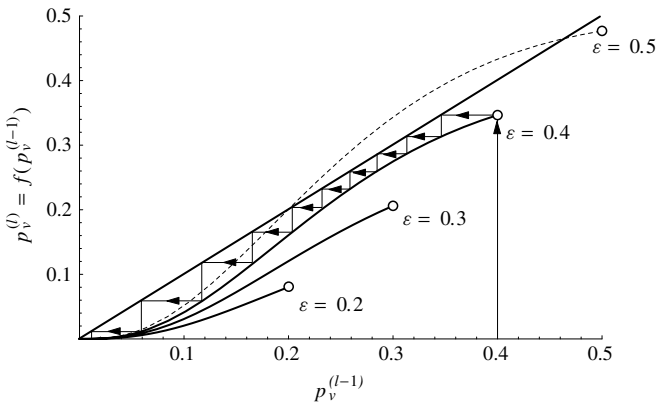


Figure 9.8 Graphical illustration of the evolution of the erasure probability $p_v^{(l)}$.

shows the values $p_v^{(l)}$ for $\varepsilon = 0.4$ according to Equation (9.11). The solid curves express the function $p_v^{(l)} = f(p_v^{(l-1)})$, computed by combining the formulas in (9.11), and illustrate the evolution of $p_v^{(l)}$ over iterations. Evidently, if ε is larger than a certain value, the *decoding threshold*, there exists a nonzero fixed point $f(\varepsilon, x) = x$, and the erasure probabilities no longer converge to zero.

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 $l \to \infty$. Based on (9.12) and appealing to Figure 9.8, this threshold happens where the probability evolution function touches the straight line at a nonzero point—that is, where $f(\varepsilon, x) = x; x \neq 0$. This threshold has to be expressed as a supremum, and for irregular LDPC codes it is given by [3]

$$\varepsilon^* = \sup\{\varepsilon : f(\varepsilon, x) < x, \forall x, 0 < x \leq \varepsilon\} \tag{9.15}$$

where $f(\varepsilon, x) = \varepsilon\lambda(1 - \rho(1 - x))$.

Error-free decoding is possible if and only if

$$x = \varepsilon\lambda[1 - \rho(1 - x)] \tag{9.16}$$

has no positive solutions for $0 < x \leq \varepsilon$.

Since any fixed-point solution x corresponds to a unique value of ε:

$$\varepsilon(x) = \frac{x}{\lambda[1 - \rho(1 - x)]}, \tag{9.17}$$

this allows us to rewrite the threshold as

$$\varepsilon^* = \min\{\varepsilon(x) : \varepsilon(x) \geq x\}. \tag{9.18}$$

For a regular ensemble (d_v, d_c), this minimum can be solved analytically. Substitute $x = 1 - y$ and consider the derivative of $\varepsilon(1 - y)$, given by

$$\begin{aligned}\frac{d}{dy}\left\{\frac{1-y}{\left[1-y^{d_c-1}\right]^{d_v-1}}\right\} &= \left[(d_c-1)(d_v-1)-1\right] y^{d_c-1} \\ &\quad -(d_c-1)(d_v-1)y^{d_c-2}+1 \\ &= 0, \end{aligned} \tag{9.19}$$

which we set to zero in order to find the maximum.

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 (9.19) therefore has no more than two roots. One of those roots is $y = 1$, the original fixed point at $x = 0$. Dividing (9.19) by $(y - 1)$ yields

$$\left[(d_v-1)(d_c-1)-1\right] y^{d_c-2} - \sum_{i=0}^{d_c-3} y^i = 0. \tag{9.20}$$

If s is the positive real root of (9.20), then the threshold is

$$\varepsilon^* = \frac{1-s}{\left(1-s^{d_c-1}\right)^{d_v-1}}. \tag{9.21}$$

As an example, the threshold for the $(3, 6)$ regular ensemble is $\varepsilon^* = 0.4294$, which is also evident from Figure 9.8. This code has nominal rate $R = 0.5$. The capacity limit for the BEC is $\varepsilon_C = 1 - r$, which is 0.5 for this example.

This example illustrates the general methodology of the density evolution method, which we outline here again as follows:

Step 1: The channel is specified in terms of a single parameter σ, and a decoding algorithm is specified.

Step 2: Assume the limit of infinitely long codes, in which the corresponding code graph is cycle-free; that is, every neighborhood is given by a tree graph. In this limit, we assume that messages are independent and identically distributed.

Step 3: Represent the decoder as a tree T_l. Note that the subtree T_{l-1} is indistinguishable from T_l, as illustrated in Figure 9.7. The root of the tree is a variable node in its lth iteration.

Step 4: Based on the decoding algorithm, find a recursive transformation $\mathcal{F}$ which relates the probability density function of a message at iteration l to the density at iteration $l - 1$. Thus if X is a message and $p[X]$ is the probability density function of X, then $p\left[m_v^{(l)}\right] = \mathcal{F}\left\{p\left[m_v^{(l-1)}\right]\right\}$. The initial density $p\left[m_v^{(0)}\right]$ is determined by the channel.

Step 5: Use the relation $\mathcal{F}$ to determine the set of parameters σ for which the density of incorrect messages converges to zero. The boundary of this set, σ^*, is the ensemble's threshold.

9.6 BINARY SYMMETRIC CHANNELS AND THE GALLAGER ALGORITHMS

The BEC is admittedly a somewhat artificial channel model, mostly popular with information theorists to find closed-form solutions that are harder to find for other channel models. A more realistic channel model is that of the binary symmetric channel (BSC) shown in Figure 9.9, which has two input, $\{0, 1\}$ and two outputs $\{0, 1\}$. A transmission is successful if input equals 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.

The following analysis has already been presented by Gallager [8, 9], but had apparently been forgotten like LDPC codes themselves until their recent rediscovery. Gallager's basic algorithm, called Gallager A, operates as follows:

Step 1: Initialize $d_i = r_i$ for each variable node.

Step 2: Variable nodes send $\mu_{i \to j} = d_i$ to each check node $j \in V_i$.

Step 3: Check nodes connected to variable node i send $\beta_{j \to i} = \prod_{l \in C_{j \setminus i}} \mu_{l \to j}$ 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 checks $\beta_{j \to i}$ disagree with d_i, change the value of variable node i to its opposite value, that is, $d_i = d_i \oplus 1$.

Step 5: Stop when no more variables are changing, or after a fixed number of iterations have been executed. Otherwise go back to Step 2.

After specifying the algorithm, we can progress to the density evolution analysis. 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*

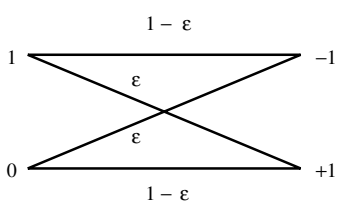


Figure 9.9 Binary symmetric channel model.

number of errors occur in its checked symbols. This happens with probability[1]

$$1 - p_v^{(l)} = \frac{1 + (1 - 2p_v^{(l)})^{d_c - 1}}{2}. \tag{9.22}$$

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

$$p_v^{(l)} \sum_{t=b}^{d_v-1} \binom{d_v - 1}{t} \left(\frac{1 + (1 - 2p_v^{(l)})^{d_c-1}}{2} \right)^t \left(\frac{1 - (1 - 2p_v^{(l)})^{d_c-1}}{2} \right)^{d_v-1-t}, \tag{9.23}$$

where the two power terms simply express the probability of t correct parity checks, and $d_v - 1 - t$ incorrect checks, respectively.

The new probability of error in a variable node is now given by

$$\begin{aligned} p_v^{(l+1)} &= p_v^{(l)} \\ &- p_v^{(l)} \sum_{t=b}^{d_v-1} \binom{d_v - 1}{t} \left(\frac{1 + (1 - 2p_v^{(l)})^{d_c-1}}{2} \right)^t \left(\frac{1 - (1 - 2p_v^{(l)})^{d_c-1}}{2} \right)^{d_v-1-t} \\ &+ (1 - p_v^{(l)}) \sum_{t=b}^{d_v-1} \binom{d_v - 1}{t} \left(\frac{1 - (1 - 2p_v^{(l)})^{d_c-1}}{2} \right)^t \left(\frac{1 + (1 - 2p_v^{(l)})^{d_c-1}}{2} \right)^{d_v-1-t}, \end{aligned} \tag{9.24}$$

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.

For a simple (3,6) LDPC code, the error probability evolution equation (9.24) becomes very simple. Its shape is illustrated in Figure 9.10, which is similar to the erasure probability evolution for the BEC channel, just for much smaller values, since the BEC channel is much more severe.

From Figure 9.10, or numerical computations, it is evident that the threshold value for a large (3,6) LDPC code on a BSC is $\varepsilon^* = 0.04$, while the channel capacity threshold for rate $R = 0.5$ is $\varepsilon = 0.11$, illustrating that it is quite a bit harder to approach capacity on a more severe channel.

Gallager [8, 9] fine-tuned his algorithms by allowing a maximization over the "vote" parameter b in (9.24). This improves the threshold for the rate $R = 0.5$ LDPC codes to $\varepsilon^* = 0.052$ for (4,8) LDPC codes.

[1] Equation (9.22) can be derived by considering the binomial expansions of $(1+p)^{d_c-1}$ and $(1-p)^{d_c-1}$ and adding them.

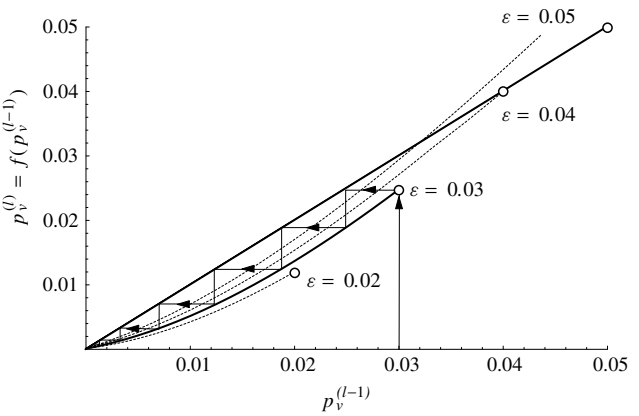


Figure 9.10 Graphical illustration of the evolution of the variable node error probability $p_v^{(l)}$.

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 the major constructive contribution 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; that is, if b_j check nodes disagree with the variable node, it is changed to the new value in Step 4 in the decoding algorithm. A check node will now signal a correct check back with probability

$$1 - p_v^{(l)} = \frac{1 + \rho\left(1 - 2p_v^{(l)}\right)}{2}, \tag{9.25}$$

which is a straightforward extension of (9.22) treating the node degrees as random according to $\rho(x)$.

Analogously, the probability that an erroneous variable is corrected is extended to

$$p_v^{(l)} \sum_{j=1}^{d_v} \lambda_j \sum_{t=b_j}^{j-1} \binom{j-1}{t} \underbrace{\left(\frac{1 + \rho\left(1 - 2p_v^{(l)}\right)}{2}\right)^t \left(\frac{1 - \rho\left(1 - 2p_v^{(l)}\right)}{2}\right)^{j-1-t}}_{g(t,j)}, \tag{9.26}$$

and the new iterated variable node error probability is

$$p_v^{(l+1)} = p_v^{(l)} - \sum_{j=1}^{d_v} \lambda_j \left[\sum_{t=b_j}^{j-1} \binom{j-1}{t} \left(p_v^{(l)} g(t, j) + (1 - p_v^{(l)}) g(j, t)\right)\right]. \tag{9.27}$$

The best "vote" parameter can be found to satisfy [14]

$$\frac{1-\varepsilon}{\varepsilon} \leq \left[\frac{1+\rho\left(1-2p_v^{(l)}\right)}{1-\rho\left(1-2p_v^{(l)}\right)}\right]^{2b_j-j+1}, \tag{9.28}$$

where we note that $2b_j - j + 1 = b_j - (j - 1 - b_j) = \Delta b$ is the difference between the check nodes that disagree with the variable node and those that agree. Equation (9.28) 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 ref. 14 the authors present the improved irregular codes shown in Table 9.1.

TABLE 9.1 Some Irregular LDPC Codes Which Outperform Regular Gallager Codes on the BEC[a] at Rate $R = 0.5$

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

[a]The capacity threshold is at 0.111.

9.7 THE AWGN CHANNEL

The additive white Gaussian noise channel discussed in Chapter 2 is probably the most important representative of practical channels, especially for wireless communications. In this section, we apply the method of density evolution to the AWGN channel. An exact solution for its density evolution is possible, but involved. Furthermore, the inevitable numerical treatment of probability densities gives little insight into the process [19]. A close approximation can be found with little effort if all messages in the decoder are assumed to have a Gaussian density. This is in general not the case, but the Gaussian approximation greatly reduces the complexity and gives results that are close to the exact solution [5].

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-zero codeword, since the LDPC code is linear. A correct message in the decoder is therefore one with positive sign, and an incorrect message is one with negative sign. The probability density function of log-likelihood messages at the output of a matched-filter receiver is

$$f_Y(y) = \sqrt{\frac{N_0}{4\pi}} e^{-\frac{N_0}{4}\left(y-\frac{1}{N_0}\right)^2}. \tag{9.29}$$

This log-likelihood density fulfills the *symmetry conditions*

$$f_Y(-y) = f_Y(y)e^{-y}, \tag{9.30}$$

$$m_Y = \frac{\sigma_Y^2}{2}, \tag{9.31}$$

where m_Y and σ_Y^2 are the mean and variance of Y, respectively.

The condition (9.31), also called the *consistency condition*, means that the density f_Y is completely determined by the mean m_Y. The consistent Gaussian assumption allows us to consider the evolution of a single parameter.

For variable nodes under probability propagation in the log domain, since the output message is the sum of incoming messages (9.9), and using the now familiar tree assumption which results in the messages being independent and identically distributed, we obtain

$$m_v^{(l)} = m_v^{(0)} + (d_v - 1)m_u^{(l-1)}, \tag{9.32}$$

where $m_v^{(l)}$ is the mean output from the variable node at iteration l, $m_u^{(l-1)}$ is the mean output from a check node at iteration $(l-1)$, and $m_v^{(0)}$ is equal to the mean of $f_Y(y)$ in (9.29), which equals $\frac{1}{N_0}$.

The density evolution through a check node according to (9.8) is more complicated. To find the evolution of the mean, we assume the messages are i.i.d.

and, after moving the $\tanh(\cdot)^{-1}$ to the left, take the expectation on both sides of (9.8) to obtain

$$E\left[\tanh\left(\frac{U}{2}\right)\right] = E\left[\tanh\left(\frac{V_i}{2}\right)\right]^{d_c-1}. \tag{9.33}$$

To simplify our analysis, we define the function ϕ as

$$\phi(m_u) = 1 - E\left[\tanh\left(\frac{U}{2}\right)\right] \tag{9.34}$$

$$= 1 - \frac{1}{\sqrt{4\pi m_u}} \int_{\mathcal{R}} \tanh\left(\frac{u}{2}\right) \exp\left[-\frac{1}{4m_u}(u - m_u)^2\right] du, \tag{9.35}$$

where $m_u > 0$. The motivation for defining $\phi(\cdot)$ is mainly computational. The bulk of computational difficulty occurs in the integral of (9.35). We will see that there are some convenient approximations to $\phi(\cdot)$ which can greatly speed computations with only minor effects on accuracy.

The mean of the check node's output message is therefore

$$m_u^{(l-1)} = \phi^{-1}\left(1 - \left[1 - \phi\left(m_v^{(l-1)}\right)\right]^{d_c-1}\right). \tag{9.36}$$

The results (9.32) and (9.36) are sufficient to compute thresholds for regular ensembles. The density evolution for irregular ensembles is found in close analogy with the derivation of (9.12). We treat node degrees as random variables. The message densities are now assumed to be Gaussian mixtures, with (9.32) and (9.36) as partial solutions. Combining these partial solutions to produce the general density evolution for irregular ensembles, we arrive at

$$m_u^{(l-1)} = \sum_{i=1}^{d_c} \rho_i \, \phi^{-1}\left(1 - \left[1 - \phi\left(m_v^{(l-1)}\right)\right]^{i-1}\right), \tag{9.37}$$

$$m_v^{(l)} = m_v^{(0)} + \sum_{j=1}^{d_v} \lambda_j (j-1) m_u^{(l-1)}. \tag{9.38}$$

We now proceed to determine the thresholds for the AWGN channel, which will inform us about the potential of LDPC codes with message passing. From our assumption that the all-zero codeword is transmitted, correct bits and messages have positive signs. Decoding is error-free if the mean $m_v^{(l)}$ in (9.38) diverges to infinity as the number of iterations becomes large, since an LLR value distributed according to (9.29) becomes unequivocal as $m_Y \to \infty$. In this case, only messages with positive sign can occur, and the probability of an incorrect message goes to zero.

The approximate threshold for the AWGN is therefore the boundary of the set of parameters $m_u^{(0)} = \frac{1}{N_0}$ for which $m_v^{(l)} \to \infty$ as $l \to \infty$. It is numerically more convenient to find solutions for a recursion which converges to zero. Equations (9.37) and (9.38) can be rearranged to produce the alternate expressions[2]

$$h_j(s,\, r) = \phi\left[s + (j-1)\sum_{i=1}^{d_c} \rho_i \cdot \phi^{-1}\left(1 - (1-r)^{i-1}\right)\right], \tag{9.39}$$

$$h(s,\, r) = \sum_{j=1}^{d_v} \lambda_j h_j(s,\, r). \tag{9.40}$$

Setting $r_0 = \phi(s)$ and $s = m_u^{(0)} = \frac{1}{N_0}$, decoding is error-free if the recursion $r_l = h(s,\, r_{l-1})$ converges to zero as $l \to \infty$. This condition is satisfied if and only if $r > h(s,\, r)$ for all $r \in (0,\, \phi(s))$. The threshold is therefore identified as

$$s^* = \inf\left\{s \in \mathcal{R}^+ :\, h(s,\, r) - r < 0,\, \forall r \in (0,\, \phi(s))\right\}. \tag{9.41}$$

Using (9.41), the threshold can be found numerically using a simple search algorithm. It is common to express the threshold in terms of the standard deviation of the channel noise, $\sigma^* = \sqrt{\frac{2}{s^*}}$.

The function $h(s,\, r)$ can also be studied graphically as shown in Figure 9.11 to obtain a visual impression of the rate of convergence. If $h(s,\, r) - r$ is close to zero, then a "slow region" or "bottleneck" occurs. If the channel parameter places r_0 in or near the bottleneck region, then many iterations may be required to achieve good performance. The bottleneck tends to be more severe when s is very close to the channel's threshold.

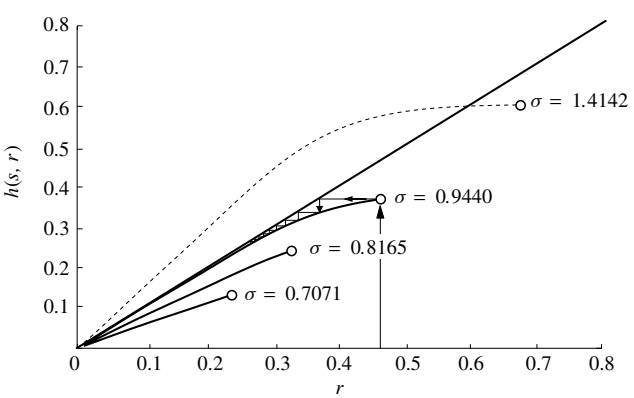


Figure 9.11 Iterative behavior of the function $h(s, r)$, illustrating the convergence of LDPC codes on AWGN channels.

[2]The inverse function ϕ^{-1} is estimated numerically. A simple analytical solution for this inverse is not known.

The iterated behavior of $h(s, r)$ is illustrated in Figure 9.11. The curves shown are for the $d_v = 15$ degree distribution reported in Table 9.2. The curve for $\sigma = .9440$ is at the threshold for this degree distribution. We see that $h(s, r) \to 0$ as the number of iterations goes to infinity. The "channel" that exists between $h(s, r)$ and r is very narrow, especially for small r. This means that many iterations are required to achieve the limiting performance. As σ gets smaller, the channel becomes more open and fewer iterations are required for convergence.

Table 9.2 shows degree distributions and thresholds for rate $R = 1/2$ irregular LDPC codes for various maximal variable node degrees. The channel capacity limit at this rate occurs for $\sigma = 0.9787$, illustrating the excellent performance of these codes. The thresholds are computed according to both the exact [19] and the Gaussian approximative density evolution [5].

The exact computation of $\phi(\cdot)$ and $\phi^{-1}(\cdot)$ can be computationally expensive. The computation speed can be greatly improved using a few approximations. For small x, perhaps $x < 10$, a workable approximation for ϕ is [5]:

$$\phi(x) \approx e^{\alpha x^{\gamma} + \beta}, \tag{9.42}$$

where $\alpha = -.4527$, $\beta = .0218$, and $\gamma = .86$. For larger x, the following upper and lower bounds become tight, so that their average can be used as a good approximation to ϕ:

$$\sqrt{\frac{\pi}{x}} e^{-\frac{x}{4}} \left(1 - \frac{3}{x}\right) < \phi(x) < \sqrt{\frac{\pi}{x}} e^{-\frac{x}{4}} \left(1 - \frac{1}{7x}\right). \tag{9.43}$$

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 is worse when the maximum variable-node degree is increased [5]. It is believed that the approximation is more accurate for regular codes and that irregular codes with maximal variable-node degree d_v less than about 10 are "regular enough" for an accurate approximation.

Other results from [5] are useful for optimizing LDPC codes, and will be summarized here. One important result is the *concentration theorem* for check-node degree distributions. A concentrated degree distribution satisfies

$$\rho(x) = \rho_k x^{k-1} + (1 - \rho_k) x^k. \tag{9.44}$$

A distribution of this form maximizes the rate of convergence under decoding iterations.

Based on the concentration theorem, Chung et al. [4] 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 distribution was obtained via linear optimization using constraint rules. The degree distributions of some of the optimized codes are given in Table 9.3.

TABLE 9.2 Exact (σ^*) and Approximate (σ^*_{GA}) Thresholds for Various Rate $R = \frac{1}{2}$ Degree Distributions on the AWGN Channel

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
σ^*	.9114	.9497	.9540	.9558	.9572	.9580	.9622	.9649	.9690	.9718
σ^*_{GA}	.9072	.9379	.9431	.9426	.9427	.9447	.9440	.9460	.9481	.9523

[a] The capacity limit is .9787.

TABLE 9.3 Parameters of Near-Capacity Optimized LDPC Codes

d_v	100	200	8000
λ_2	.170031	.153425	.096294
λ_3	.160460	.147526	.095393
λ_6	.112837	.041539	.033599
λ_7	.047489	.147551	.091918
λ_{10}	.011481		
λ_{11}	.091537		
λ_{15}			.031642
λ_{18}		.047938	
λ_{19}		.119555	
λ_{20}			.086563
λ_{26}	.152978		
λ_{27}	.036131		
λ_{50}			.093896
λ_{55}		.036379	
λ_{56}		.126714	
λ_{70}			.006035
λ_{100}	.217056		.018375
λ_{150}			.086919
λ_{200}		.179373	
λ_{400}			.089018
λ_{900}			.057176
λ_{2000}			.085816
λ_{3000}			.006163
λ_{6000}			.003028
λ_{8000}			.118165
$\rho_{\text{av}} = k - 1 + \rho$	10.9375	12	18.5
σ^*	0.97592	0.97794	0.9781869

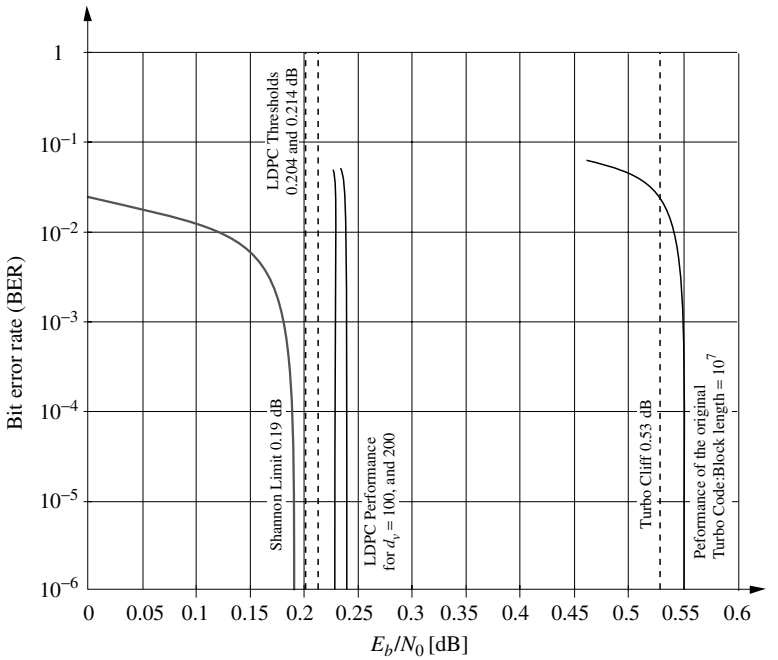


Figure 9.12 Simulations of large-size LDPC codes and the original turbo code.

The check-node distribution is given as $\rho_{\text{av}} = (1-\rho_k)(k-1)+\rho_k k = k-1+\rho_k$ as a single number. Given that the Shannon capacity threshold is at $\sigma = 0.97869$, in terms of signal-to-noise ratio, these codes come to within 0.0247, 0.0147, and 0.0045 dB of the Shannon capacity, respectively.

Chung et al. [4] also provide simulation results for a block length of 10^7, which are reproduced in Figure 9.12, illustrating the tight predictions of the density evolution method.

9.8 LDPC ENCODING

The sparseness of an LDPC parity-check matrix translates into the efficient graph-based decoding algorithms described above. Encoding of LDPC codes, however, has quadratic complexity with respect to the code length. This has prompted several researchers to explore efficient encoding algorithms for LDPC codes. Some of these approaches require specialized code construction.

In general, a linear block code may be encoded by finding a generator matrix, defined by the relation $\boldsymbol{G}\boldsymbol{H}^T = 0$ (see Chapter 5). $\boldsymbol{G}$ provides a basis for the code $\mathcal{C}$, and it also acts as encoder for an information sequence $\boldsymbol{u}$ by the multiplication $\boldsymbol{u}\boldsymbol{G}$. If we assume the code is systematic, then we may also encode directly using

the parity-check matrix. Suppose a codeword has the structure $\boldsymbol{x} = \left[\boldsymbol{x}_u\, \boldsymbol{x}_p\right]$, where $\boldsymbol{x}_u$ is the information sequence and $\boldsymbol{x}_p$ is the parity sequence.

We can then split $\boldsymbol{H}$ into $\left[\boldsymbol{H}_u \mid \boldsymbol{H}_p\right]$, giving the equation

$$\boldsymbol{H}_p \boldsymbol{x}_p^T = \boldsymbol{H}_u \boldsymbol{x}_u^T; \tag{9.45}$$

$$\Rightarrow\ \boldsymbol{x}_p^T = \boldsymbol{H}_p^{-1} \boldsymbol{H}_u \boldsymbol{x}_u^T. \tag{9.46}$$

Note also that $\boldsymbol{H}_u$ and $\boldsymbol{H}_p$ are square matrices. There are a few methods available to make this computation efficient. The authors of [20] suggest placing the matrix in "approximate lower triangular" form, in which the upper right corner is populated with only 0s as shown in Figure 9.13.

A given parity-check matrix $\boldsymbol{H}$ must 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 $\boldsymbol{H}$ to a true lower-triangular matrix. We then split $\boldsymbol{x}_p$ into $\left[\boldsymbol{p}_1,\ \boldsymbol{p}_2\right]$ and multiply $\boldsymbol{H}$ from the left by

$$\begin{bmatrix} \boldsymbol{I} & \boldsymbol{0} \\ -\boldsymbol{E}\boldsymbol{T}^{-1} & \boldsymbol{I}, \end{bmatrix}, \tag{9.47}$$

giving

$$\begin{bmatrix} \boldsymbol{A} & \boldsymbol{B} & \boldsymbol{T} \\ \boldsymbol{C} - \boldsymbol{E}\boldsymbol{T}^{-1}\boldsymbol{A} & \boldsymbol{D} - \boldsymbol{E}\boldsymbol{T}^{-1}\boldsymbol{B} & \boldsymbol{0} \end{bmatrix}. \tag{9.48}$$

The parity-check rule then gives the following equations:

$$\begin{aligned} \boldsymbol{A}\boldsymbol{x}_u^T + \boldsymbol{B}\boldsymbol{p}_1^T + \boldsymbol{T}\boldsymbol{p}_2^T &= \boldsymbol{0}, \\ (\boldsymbol{C} - \boldsymbol{E}\boldsymbol{T}^{-1}\boldsymbol{A})\boldsymbol{x}_u^T + (\boldsymbol{D} - \boldsymbol{E}\boldsymbol{T}^{-1}\boldsymbol{B})\boldsymbol{p}_1^T &= \boldsymbol{0}. \end{aligned} \tag{9.49}$$

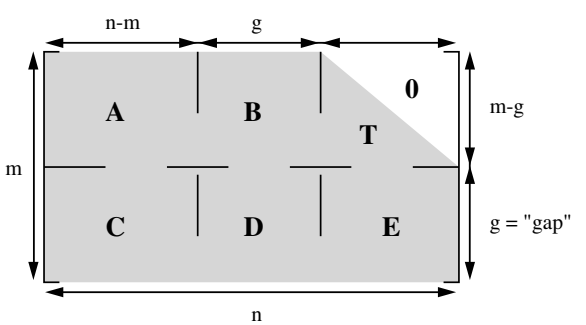


Figure 9.13 Conversion of the parity-check matrix of an LDPC code into approximate lower triangular form.

Now define $\boldsymbol{\phi} = \boldsymbol{D} - \boldsymbol{E}\boldsymbol{T}^{-1}\boldsymbol{B}$, and assume $\boldsymbol{\phi}$ is nonsingular. We then arrive at

$$\begin{aligned} \boldsymbol{p}_1^T &= \boldsymbol{\phi}^{-1}\left(\boldsymbol{C} - \boldsymbol{E}\boldsymbol{T}^{-1}\boldsymbol{A}\right)\boldsymbol{x}_u^T, \\ \boldsymbol{p}_2^T &= -\boldsymbol{T}^{-1}\left(\boldsymbol{A}\boldsymbol{x}_u^T + \boldsymbol{B}\boldsymbol{p}_1^T\right). \end{aligned} \tag{9.50}$$

A judicious arrangement of these computations gives a sequence of complexity $O(n)$. Only the computation $-\boldsymbol{\phi}^{-1}\left(\boldsymbol{C}\boldsymbol{x}_u^T - \boldsymbol{E}\boldsymbol{T}^{-1}\boldsymbol{A}\boldsymbol{x}_u^T\right)$ has complexity $O(g^2)$, because $\boldsymbol{\phi}^{-1}$ is a dense $g \times g$ matrix. The authors of ref. 20 also show that, for sufficiently large n, $g \leq O(\sqrt{n})$ with high probability for a randomly chosen LDPC code. This algorithm therefore provides encoding with linear complexity in the length n.

9.9 ENCODING VIA MESSAGE-PASSING

If H_p is truly lower triangular then the BEC decoding algorithm can be used to perform iterative encoding in at most m steps. This can be seen through an example:

$$\boldsymbol{H} = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 1 \end{bmatrix}. \tag{9.51}$$

The constraint graph for this matrix is shown in Figure 9.14.

The information bits have been placed at the top of the graph. They are represented by filled circles to indicate that they are known. The parity bits at the bottom are unknown. For encoding, we initialize the information-bit nodes with ± 1 and the parity nodes with 0. When the BEC decoding algorithm is

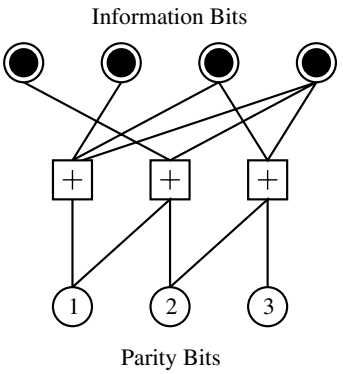


Figure 9.14 Constraint graph for the matrix in (9.51).

applied, the first round of message-passing will cause parity bit 1 to become known. In the second round, a nonzero message from parity bit 1 allows parity bit 2 to become known. Similarly, the parity bit 3 becomes known in the third round. Encoding therefore takes place in $m = 3$ iterations.

This algorithm is very convenient, and provides the following possible advantages:

- Encoding complexity is linear with m.
- The encoder circuit can share the same silicon area with a decoder. This can be valuable in transceiver systems.
- A suitably designed decoder may substitute as an encoder. This may be useful in half-duplex systems that either transmit or receive information (but not both) during any transaction.

The drawback is the more strict restriction on code structure which can result in limits on decoder performance. If the truly lower-triangular matrix is obtained by linear combinations from some original parity-check matrix, then it is also possible to encode using the lower-triangular matrix graph while decoding with the original matrix graph.

The authors of ref. 10 found that encoding can also be performed by means of iterated approximation. We wish to solve $\boldsymbol{H}_p \boldsymbol{x}_p^T = \boldsymbol{H}_u \boldsymbol{x}_u^T$ for $\boldsymbol{x}_p^T$. To do so, we first select an initial guess for $\boldsymbol{x}_p$, which we'll denote by $\boldsymbol{x}_0$. We will then use an iterative method to obtain subsequent approximations, denoted $\boldsymbol{x}_k$. These approximations will converge to $\boldsymbol{x}_p$ using the following iteration:

$$\boldsymbol{x}_{k+1}^T = \left(\boldsymbol{H}_p + I\right) \boldsymbol{x}_k^T + \boldsymbol{H}_u \boldsymbol{x}_u^T, \tag{9.52}$$

provided that $\boldsymbol{H}_p$ is nonsingular. Correct encoding results after $k' \geq k$ iterations if and only if $(\boldsymbol{H}_p + \boldsymbol{I})^k = 0$. Let $\boldsymbol{b} = \boldsymbol{H}_u \boldsymbol{x}_u^T$. To apply the algorithm, we can split the code's graph into two parts as shown in Figure 9.15: an information-bit section and a parity-bit section, as below. The arrows in the graph below indicate which edges will be enabled to transmit edges from check nodes in the encoding algorithm.

The top half of the graph computes $\boldsymbol{b}$. The bottom part only represents the constraint in $\boldsymbol{H}_p$. $\boldsymbol{H}_p$ 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 graphically using 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 $\mu_{ij} = v_i$ to all $j \in V_{i \setminus i}$.

Step 3: Check nodes answer with $\beta_{jj} = \prod_{l \in C_{j \setminus j}} \mu_{lj}$, 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.

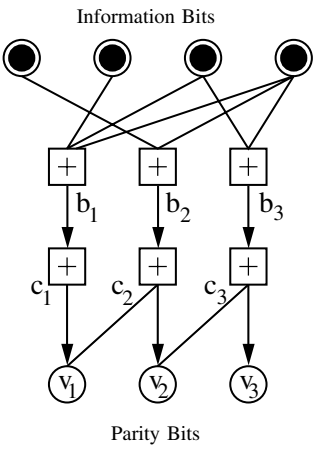


Figure 9.15 Splitting the code graph into two parts to apply iterative encoding.

This result allows us to construct a code which is guaranteed to be graphically encodable in a fixed number of iterations.

In ref. 10 a method is presented for constructing "reversible" LDPC codes, which can be encoded in two iterations. It is first noted that $\left(\boldsymbol{H}_p \oplus I\right)^2 = \boldsymbol{I} \rightarrow \boldsymbol{H}_p^2 = \boldsymbol{I}$. A reversible code may then be constructed by selecting a 2×2 matrix $\boldsymbol{A}_0$ for which $\boldsymbol{A}_0^2 = \boldsymbol{I}$. $\boldsymbol{H}_p$ can be obtained by recursive application of one of the following rules:

$$\boldsymbol{A}_k = \begin{bmatrix} \boldsymbol{A}_{k-1} & \boldsymbol{I} \\ \boldsymbol{0} & \boldsymbol{A}_{k-1} \end{bmatrix} \quad \text{or} \quad \boldsymbol{A}_k = \begin{bmatrix} \boldsymbol{A}_{k-1} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{A}_{k-1} \end{bmatrix}. \tag{9.53}$$

As an example, we can construct the matrix

$$\boldsymbol{A}_0 = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \Rightarrow \boldsymbol{H}_p = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{9.54}$$

and concatenate it with

$$\boldsymbol{H}_u = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 \end{bmatrix}. \tag{9.55}$$

The resulting code is an [8,4] extended Hamming code, whose graph is shown in Figure 9.16.

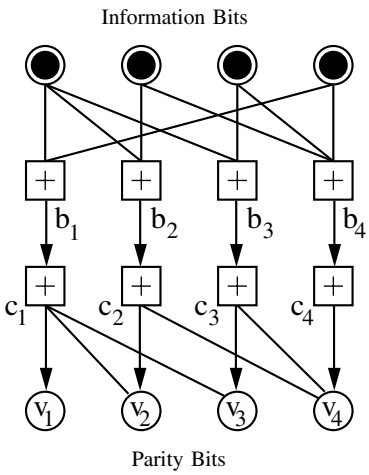


Figure 9.16 Graph of the [8, 4] extended Hamming code.

9.10 REPEAT ACCUMULATE CODES ON GRAPHS

The near-capacity performance of low-density parity-check codes and the elegance of the graphical description of the codes and the decoding algorithm have led to the investigation of many graph-based coding schemes. In this section, we consider a particularly simple alternative to LDPC codes known as repeat accumulate (RA) codes [9]. RA codes are serially concatenated codes (and are covered from that perspective in Chapter 11) that have a convenient graphical representation that enables them to benefit from the notion of irregularity and to be analyzed using density evolution.

A block diagram of a nonsystematic repeat accumulate encoder is shown in Figure 9.17. 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 an interleaver or permuter, and the overall code rate is $R = 1/q$. It is worth noting at this point that an obvious advantage of this encoder is its simplicity. Indeed, the encoding complexity of most LDPC codes is quadratic in block length, whereas the encoding complexity of an RA code is linear in block length. The simplicity of these codes is deceiving. For large block lengths and rates $R = 1/q \leq 1/3$,

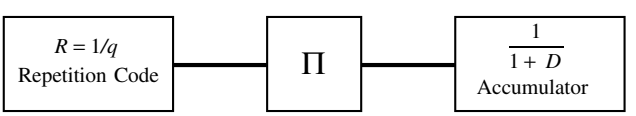


Figure 9.17 Block diagram of a repeat accumulate encoder.

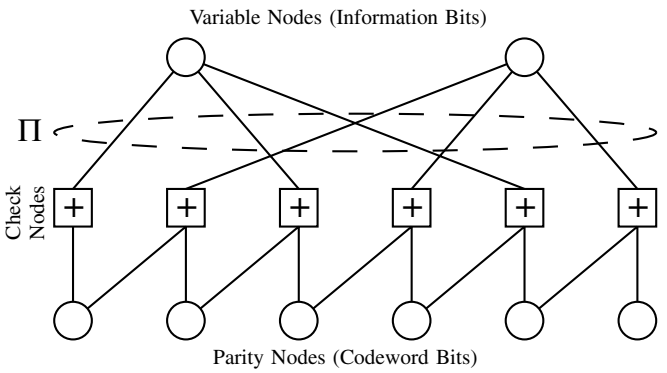


Figure 9.18 Tanner graph of an RA code with $q = 3$ and $k = 2$.

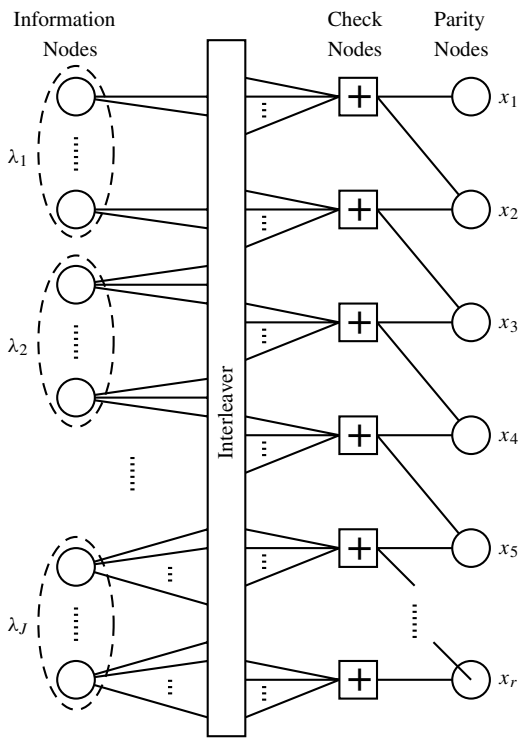


Figure 9.19 Tanner graph of an irregular RA code.

TABLE 9.4 Degree Profiles for Optimized Systematic Irregular Repeat Accumulate Codes [11]

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
σ^*	1.1981	1.2607	1.2780
σ_{GA}	1.1840	1.2415	1.2615
Capacity (dB)	−0.4953	−0.4958	−0.4958

these codes perform within roughly 1 dB of the Shannon limit on the AWGN channel [9].

The graphical representation of the RA code of Figure 9.17 is straightforward and best illustrated by an example. Let $q = 3$ and the information block length $k = 2$, then the graph of the resulting code is shown in Figure 9.18, where the interleaver is represented by the dashed ellipse below the variable nodes. From this graph, it is clear that each variable node has degree $q = 3$ and that each parity node has degree 2 due to the accumulator. Thus, the graph of an RA code is a regular bipartite graph with degree q information variable nodes, degree-1 check nodes, and degree-2 parity nodes.

In light of the success of irregular LDPC codes, it is natural to ask if there exist irregular repeat accumulate codes and, if so, how do they perform. It was suggested in ref. 11 that in order to get an irregular RA code, the degree of the information variable nodes should be varied. That is, the number of times that an information bit is repeated is varied. The resulting graph is depicted in Figure 9.19 and is parameterized by the degree distribution $(\lambda_1, \lambda_2, \ldots, \lambda_J)$ of the information variable nodes and the left degree a of the intermediate check nodes. Note that in the graph of a regular RA code, such as that depicted in Figure 9.18, the value of $a = 1$.

In ref. 11, the authors considered the subclass of systematic irregular RA codes with a fixed check node left degree of a. In a systematic IRA code, the graph is used to compute the parity bits and the codeword is $\boldsymbol{x} = (u_1, u_2, \ldots, u_k, x_1, x_2, \ldots, x_r)$ and the overall code rate is $R = a/(a + \sum_i i\lambda_i$. 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 9.4. These codes come within approximately 0.1 dB of the capacity limit.

BIBLIOGRAPHY

1. S. M. Aji and R. J. McEliece, "The generalized distributive law," *IEEE Trans. Inform. Theory*, pp. 325–343, March 2000.
2. 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.
3. 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*, in press.
4. 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. COMM-5, no. 2, pp. 58–60, Feb. 2001.
5. 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.
6. 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.
7. G. D. Forney, "Codes on graphs: Normal realizations," *IEEE Trans. Inform. Theory*, pp. 520–548, Feb. 2001.
8. R. G. Gallager, "Low-density parity-check codes," *IRE Trans. Inform. Theory*, pp. 21–28, vol. 8, no. 1, Jan. 1962.
9. R. G. Gallager, *Low-Density Parity-Check Codes*, MIT Press, Cambridge, MA, 1963.
10. D. Haley, A. Grant, and J. Buetefuer, "Iterative encoding of low-density parity-check codes," *Proceedings, IEEE Globecom 2002*, Oct. 2002.
11. 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.
12. F. R. Kschischang, B. J. Frey, and H.-A. Loeliger, "Factor graphs and the sum-product algorithm," *IEEE Trans. Inform. Theory*, Feb. 2001.
13. 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.
14. 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.

15. M. Luby, M. Mitzenmacher, A. Shokrollahi, D. Spielman, and V. Stemann, "Practical loss-resilient codes," *Proceedings, 29th Annual ACM Symposium on Theory of Computing*, pp. 150–159, 1997.
16. 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.
17. 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.
18. 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.
19. 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.
20. T. J. Richardson and R. Urbanke, "Efficient encoding of low-density parity-check codes," *IEEE Trans. Inform. Theory*, pp. 638–656, Feb. 2001.
21. R. M. Tanner, "A recursive approach to low complexity codes," *IEEE Trans. Inform. Theory*, vol. 74, no. 2, pp. 533–547, Sept. 1981.
22. N. Wiberg, H.-A. Loeliger, and R. Kötter, "Codes and iterative decoding on general graphs," *Eur. Trans. Telecommun.*, pp. 513–525," Sept./Oct. 1995.

CHAPTER 10

Parallel Concatenation (Turbo Codes)

10.1 INTRODUCTION

The discovery of turbo codes was a monumental event in the error control coding discipline, one that fundamentally changed the way it will be thought of and practiced. The presentation, at the International Conference on Communications in 1993, of implementable error control codes that achieve Shannon's bound to within 0.5 dB of signal-to-noise ratio was met with strong disbelief; so widespread was the conviction that in order to achieve near-Shannon performance, nearly infinite complexity had to be spent at the decoder.

The near-capacity performance of turbo codes and their novel iterative decoding algorithm has stimulated an explosion of research efforts to understand this new coding scheme [1–34]. From this emerged two fundamental questions regarding turbo codes. First, assuming optimum or near-optimum decoding can be performed, why do the turbo codes perform so well? Second, how does the iterative decoder work, why does it perform as well as it does, and does it converge to the optimum solution [14, 30, 44]?

In practical systems, coding gains have always been limited by the technology available for implementation. That is, while strong codes at the time of the earliest applications of FEC coding were available, it was not feasible to implement them. 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 decodes rate $1/n$ convolutional codes with 2^{14} states [37], and is the most complex decoder ever realized in hardware. 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 it appeared that a law of diminishing returns was manifesting itself.

The significance of turbo codes becomes clear upon observing their performance. Figure 10.1 shows simulation results of the original rate $R = 1/2$ turbo code presented in ref. 12, along with simulation results of a maximum free distance (MFD) $R = 1/2$, memory $\nu = 14$ (2, 1, 14) convolutional code with Viterbi

Trellis and Turbo Coding, by Christian B. Schlegel and Lance C. Pérez
ISBN 0-471-22755-2

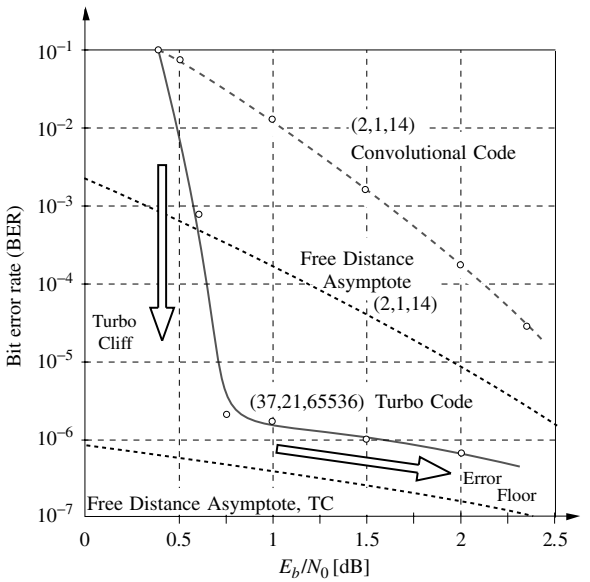


Figure 10.1 Simulated performance of the original turbo code and the code's free distance asymptote and the simulated performance of the (2, 1, 14) MFD convolutional code and its free distance asymptote.

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 much smaller than that of the BVD. The comparison of these simulation results raises two issues regarding the performance of turbo codes. First, what is it that allows turbo codes to achieve a BER of 10^{-5} at a signal-to-noise ratio (SNR) of merely 0.7 dB? Second, what causes the "error floor"—that is, the flattening of the performance curve—for moderate to high SNR's?

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 detailed performance and structural comparison of the MFD (2, 1, 14) code with a specific turbo code based on the original Berrou code. This comparison leads to the conclusion that the distance spectrum, and in particular the free distance, discussed in Chapter 3, is the appropriate tool to understand the behavior of turbo codes in the error floor region [31]. This effort results in an interpretation that not only applies 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 refs. 7, 10, and 19. This analysis leads to the fundamental rules that govern the design of turbo codes.

Having addressed the performance in the error floor region, we then proceed to 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 this 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 transfer* (EXIT) charts discussed later in this chapter in Section 10.8 [40–42]. The chapter concludes with a discussion of the application of turbo codes to the channel coding in various new standards.

10.2 PARALLEL CONCATENATED CONVOLUTIONAL CODES

An encoder for a classical turbo code consists of the *parallel concatenation* of two or more, usually identical, rate $R = 1/2$ encoders, realized in recursive systematic or systematic feedback form[1], and a pseudorandom interleaver as illustrated in Figure 10.2. 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. For the remainder of the chapter, we consider only turbo encoders with two identical component convolutional encoders, though this encoding method can easily be extended to the case where different encoders or three or more encoders are used [18]. However, the major conclusions and performance results are achievable with only two component encoders.

The interleaver 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 $v_r^{(1)}$ and $v_r^{(2)}$. 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

[1]The recursive systematic form is a special case of the controller canonical form while the systematic feedback form is the observer canonical form. See Sections 4.2 and 4.4

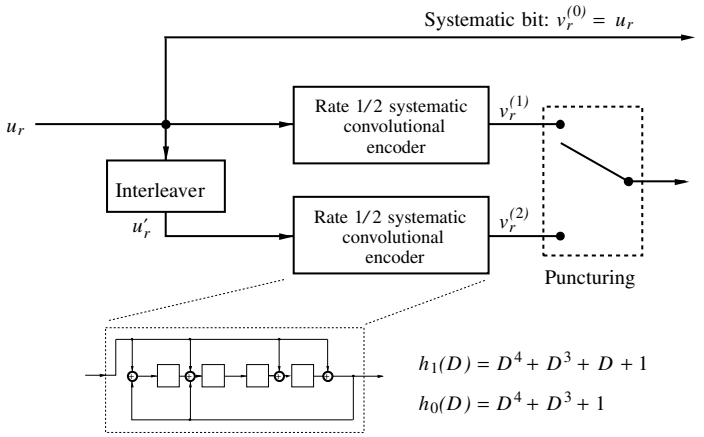


Figure 10.2 Block diagram of a turbo encoder with two component encoders and an optional puncturer.

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 order to illustrate the basic principles of parallel concatenation, consider the encoder shown in Figure 10.2 with (2, 1, 2) component encoders with parity-check polynomials $h_0(D)$ = reverse polynomial $D^2 + 1 = 5$ and $h_1(D) = D = 2$. For purposes of illustration, assume a small pseudorandom interleaver of size $N = 16$ bits which generates a (5, 2, 16) turbo code. The interleaver is realized as a 4 x 4 matrix which is filled sequentially, row by row, with the input bits u_r. Once the interleaver 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 $\{u_0 \ldots u_{15}\} = \{1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0\}$ is the input sequence, and the interleaver is represented by the permutation Π_{16}, then the sequence $\boldsymbol{u}' = \Pi_{16}(\{u_0 \ldots u_{15}\}) = \{0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0\}$ is the input to the second encoder. The trellis diagrams for both constituent encoders with these inputs are shown in Figure 10.3. 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 are {0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1} for the second encoder. The resulting codeword has Hamming weight $d = w(\boldsymbol{u}) + w(\boldsymbol{v}^{(1)}) + w(\boldsymbol{v}^{(2)}) = 4 + 3 + 3 = 10$ without puncturing. If the code is punctured 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 $4 + 0 + 1 = 5$.

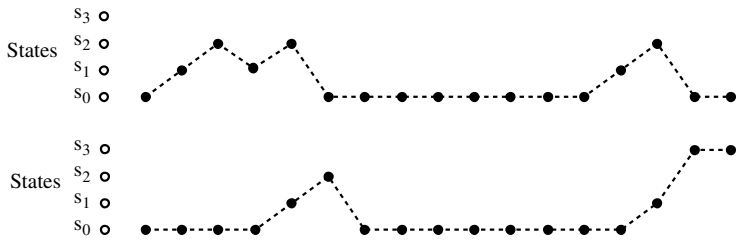


Figure 10.3 Examples of detours in the constituent encoders.

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 $\boldsymbol{u}$ and $\boldsymbol{u}'$ 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 10.3. Since the constituent encoders are realized in recursive systematic form, a nonzero 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 one for departure and one 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 $\boldsymbol{u}$ are used to force the first encoder back to the all-zero state.

If neither encoder is forced to the all-zero state—that is, 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 $\boldsymbol{u}$ to the first encoder. For some interleavers, this sequence will be permuted to itself and $\boldsymbol{u}' = \boldsymbol{u}$. 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, 33]. Special interleaver structures that result in both encoders returning to the all-zero state are discussed in refs. 2, 3, and 25.

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 sequence[2] $Du(D)$ 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 10.3, if $Du(D)$ is the input sequence, then $\Pi_{16}(Du(D)) = \{1, 0, 1, 0, 0,$

[2]We will use the D-transform notation for sequences whenever more convenient, such as to describe shifting of sequences.

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.

10.3 DISTANCE SPECTRUM ANALYSIS OF TURBO CODES

In Chapter 6, the bounds on the performance of trellis codes were developed assuming an infinite-length trellis. The result is the bound of Equation (6.10) 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, where N corresponds to the size of the interleaver in the turbo encoder. 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 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

$$P_b \leq \sum_{i=1}^{2^N} \frac{w_i}{N} Q\left(\sqrt{d_i \frac{2RE_b}{N_0}}\right), \tag{10.1}$$

where w_i and d_i are the information weight and total Hamming weight, respectively, of the ith 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 $d_{\text{free}} \le d \le 2d_{\text{free}} - 1$, we have

$$\lim_{N\to\infty} \frac{N_d}{N} = N_d^0$$

and

$$\lim_{N\to\infty} \tilde{w}_d = \lim_{N\to\infty} \frac{w_d}{N_d} = \frac{w_d^0}{N_d^0} \triangleq \tilde{w}_d^0,$$

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 $d \ge 2d_{\text{free}}$, the situation is somewhat more complicated due to the fact that codewords can consist of multiple error events. That is, as $N \to \infty$, $N_d = N \cdot N_d^0 + N_d(m)$, 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

$$\begin{aligned} P_b &\le \sum_{d=d_{\text{free}}}^{2\cdot d_{\text{free}}-1} N_d^0 \tilde{w}_d^0 Q\left(\sqrt{d\frac{2RE_b}{N_0}}\right) + \sum_{d=2\cdot d_{\text{free}}}^{\infty} \frac{N_d \tilde{w}_d}{N} Q\left(\sqrt{d\frac{2RE_b}{N_0}}\right) \\ &= \sum_{d=d_{\text{free}}}^{2\cdot d_{\text{free}}-1} w_d^0 Q\left(\sqrt{d\frac{2RE_b}{N_0}}\right) + \sum_{d=2\cdot d_{\text{free}}}^{\infty} \frac{N_d \tilde{w}_d}{N} Q\left(\sqrt{d\frac{2RE_b}{N_0}}\right), \end{aligned} \quad (10.2)$$

which is similar to the standard union bound for ML decoding as developed in Chapter 6. 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 (10.1). Collecting codewords of the same total Hamming weight, the bound on the BER for turbo codes also becomes

$$P_b \le \sum_{d=d_{\text{free}}}^{\infty} \frac{N_d \tilde{w}_d}{N} Q\left(\sqrt{d\frac{2RE_b}{N_0}}\right). \quad (10.3)$$

However, in the turbo encoder the pseudorandom interleaver maps the input sequence $u(D)$ to $u'(D)$ and the input sequence $Du(D)$ to a sequence $u''(D)$ that is different from $Du'(D)$ with very high probability. Thus, unlike convolutional codes, the input sequences $u(D)$ and $Du(D)$ produce different 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

$$\frac{\tilde{w}_d N_d}{N} \ll 1,$$

where

$$\frac{N_d}{N} \tag{10.4}$$

is called the *effective multiplicity* of codewords of weight d.

10.4 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

$$P_b \approx \frac{N_{\text{free}} \tilde{w}_{\text{free}}}{N} Q\left(\sqrt{d_{\text{free}} \frac{2RE_b}{N_0}}\right), \tag{10.5}$$

where N_{free} is the error coefficient and $\tilde{w}_{\text{free}}$ is the average weight of the information sequences causing free distance codewords. The expression on the right-hand side of Equation (10.5) and its associated graph is called the *free distance asymptote*, of a code.

As the error floor in Figure 10.1 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 [34] and [22]. These algorithms were applied to a turbo code with the same component encoders, puncturing pattern, and interleaver size N as in [12] 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 $(37, 21, 65536)$ 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 $N = 65536$.

For this particular turbo code, the free distance asymptote is given by

$$P_{\text{free}} = \frac{3 \cdot 2}{65536} Q\left(\sqrt{6 \frac{2 \cdot 0.5 \cdot E_b}{N_0}}\right),$$

where the rate loss due to the addition of a 4-bit tail to terminate the first encoder is ignored and

$$\frac{N_{\text{free}}}{N} = \frac{3}{65536}$$

is the effective multiplicity. The free distance asymptote is shown plotted in Figure 10.1 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 (10.5) 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 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 10.4. Due to the steepness of the performance curve

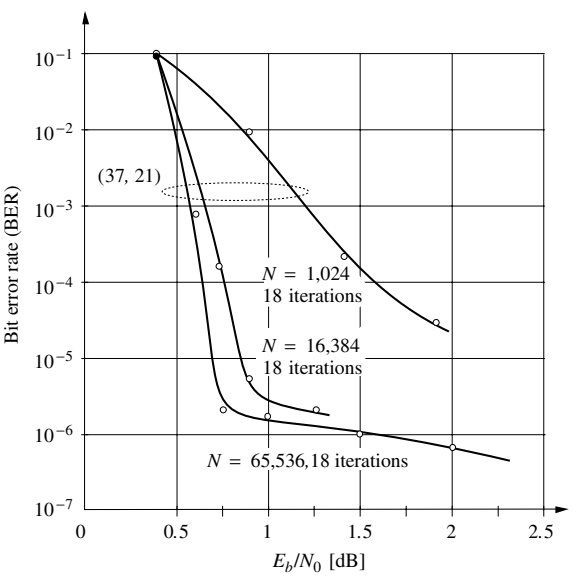


Figure 10.4 Simulations illustrating the effect of the interleaver size on the performance of a turbo code.

in turbo cliff region, very small changes in the SNR can result in the onset of the error floor.

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 10.2 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 error coefficient. This has the effect of changing the slope of the free distance asymptote. That is, increasing the free distance increases the slope of the asymptote, and decreasing the free distance decreases the slope of the asymptote. 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.

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 (10.2). The maximum free distance (2, 1, 14) code whose performance is shown in Figure 10.1 has $d_{\text{free}} = 18$, $N_{\text{free}}^0 = 18$, and $w_{\text{free}}^0 = 137$ [15]. 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 10.1.

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 (10.4), 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 better than the (37, 21, 65536) turbo code, the turbo code is better for the error rates 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 rectangular interleaver. The same component encoders and puncturing pattern as in ref. 12 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 $\boldsymbol{u}$ is written row by row. The input sequence to the second encoder $\boldsymbol{u}'$ 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, 14400) turbo code.

Using the algorithm described in ref. 34, 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 10.5 along with simulation results using the iterative decoding algorithm of ref. 12 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, 65536) turbo code with a pseudorandom interleaver even though it has a much larger free distance. The relatively poor performance of the (37, 21, 14400) 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 (37, 21, 65536) 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.

The free distance paths in the turbo code with the rectangular interleaver are due to four basic information sequences of weight 4. These information

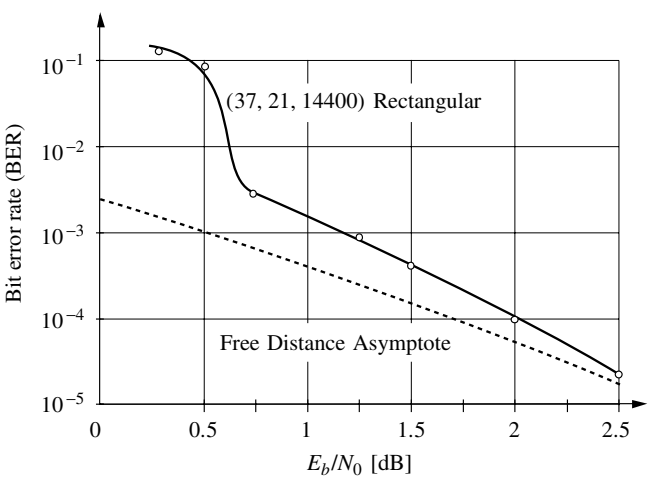


Figure 10.5 Comparison of the simulated performance with the free distance asymptote for a turbo code with a rectangular interleaver.

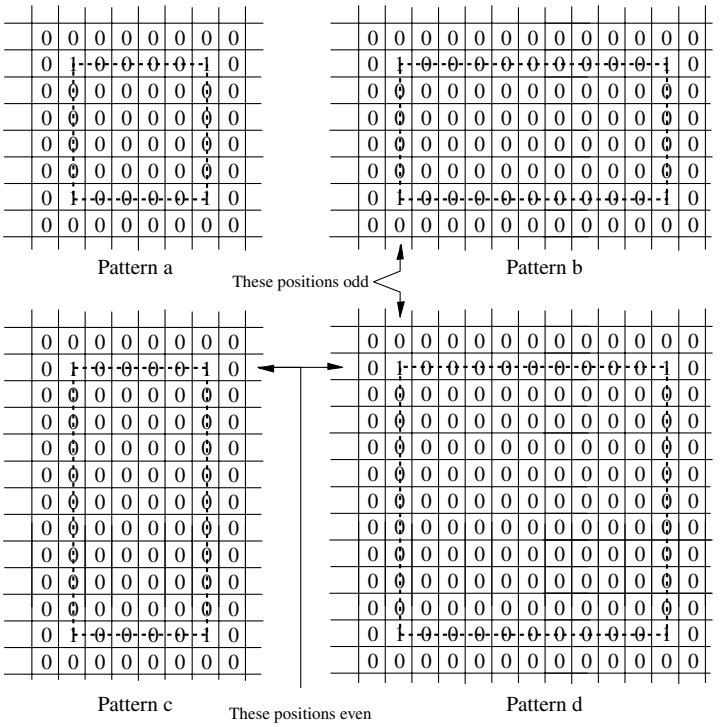


Figure 10.6 Information sequences causing d_{free} codewords in a turbo code with a rectangular interleaver.

sequences are depicted in Figure 10.6 as they would appear in the rectangular interleaver. The "square" sequence in Figure 10.6a depicts the sequence $\boldsymbol{u} = 1, 0, 0, 0, 0, 1, 0_{594}, 1, 0, 0, 0, 0, 1, 0_{\infty}$, 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 $\boldsymbol{u}$ to itself and therefore $\boldsymbol{u}' = \boldsymbol{u}$. The sequence $\boldsymbol{u}$ results in a parity sequence $\boldsymbol{v}^{(1)}$ from the first constituent encoder which, after puncturing, has weight 4. Similarly, the input sequence $\boldsymbol{u}' = \boldsymbol{u}$ results in a parity sequence $\boldsymbol{v}^{(2)}$ 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 10.6a can appear in $(\sqrt{N} - 5) \times (\sqrt{N} - 5) = 13225$ distinct positions in the rectangular interleaver, and in each case $\boldsymbol{u}' = \boldsymbol{u}$ 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 10.6b and 10.6c 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 10.6b and Figure 10.6c each result in $0.5(\sqrt{N} - 10) \times (\sqrt{N} - 5) = 6325$ free distance codewords. For the sequence in Figure 10.6d, 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 $0.25(\sqrt{N} - 10) \times (\sqrt{N} - 10) = 3025$ free distance codewords. Summing the contributions of each type of sequence results in a total of $N_{\text{free}} = 28{,}900$ code words 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 10.6 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 significantly. 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 for moderate BERs. 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.

10.5 THE DISTANCE SPECTRUM OF A TURBO CODE

In the previous section, it was shown that the error floor observed in the performance of turbo codes is due to their relatively low free distance. It is now demonstrated that the outstanding performance of turbo codes at low SNRs is a manifestation of the sparse distance spectrum that results when a pseudorandom interleaver is used in a parallel concatenation scheme. To illustrate this the partial distance spectrum of the $(37, 21, 65536)$ turbo code with the same pseudorandom interleaver used previously is found and its relationship to the performance of the code is discussed. The distance spectrum of the turbo code is then compared to the distance spectrum of the $(2, 1, 14)$ code.

Using the algorithm described in ref. 22, the $(37, 21, 65536)$ turbo code was found to have the following distance spectrum:

d	N_d	w_d
6	3	6
8	19	56
10	37	78
12	161	351

where w_d is the total information weight of the input sequences generating code words of weight d. The distance spectrum information for a particular distance d is referred to as a spectral line. These data can be used in conjunction with the bound of equation (10.3) to estimate the performance of the code. In addition, by plotting each term of equation (10.3) the contribution of each spectral line to the overall performance of the code can be estimated.

The bound on the performance of the turbo code with the above distance spectrum is given in Figure 10.7 along with curves showing the contribution of each spectral line. This clearly shows that the contribution to the code's BER by the higher distance spectral lines is less than the contribution of the free distance term for SNRs greater than 0.75 dB. Thus, the free distance asymptote dominates the performance of the code not only for moderate and high SNRs, but also for low SNRs. We call distance spectra for which this is true *sparse* or *spectrally thin*.

The ramifications of a sparse distance spectrum are made evident by examining the distance spectrum of convolutional codes. The (2, 1, 14) convolutional code introduced in Section 10.3 has the following distance spectrum as reported in ref. 15.

d	N_d^0	w_d^0
18	33	187
20	136	1034
22	835	7857
24	4787	53994
26	27941	361762
28	162513	2374453
30	945570	15452996
32	5523544	99659236

When comparing the distance spectrum of a convolutional code and a turbo code, it is important to remember that for a convolutional code $N_d \approx N \times N_d^0$ for the codewords with weight less than $2 \cdot d_{\text{free}}$. Figure 10.8 shows the estimated performance of this code using the bound of equation (10.2) and the contribution of each spectral line.

In this case, the contribution of the higher distance spectral lines to the overall BER is greater than the contribution of the free distance term for SNRs less than 2.7 dB, which corresponds to BERs of less than 10^{-6}! The large SNR required for the free distance asymptote to dominate the performance of the (2, 1, 14) code is due to the rapid increase in the multiplicity for increasing d. We call distance spectra for which this is true *spectrally dense*. The dense distance spectrum of convolutional codes also accounts for the discrepancy between the real coding gain at a particular SNR and the asymptotic coding gain calculated using just the free distance [29].

Thus, it can be concluded that the outstanding performance of turbo codes at low signal-to-noise ratios is partially a result of the dominance of the free distance

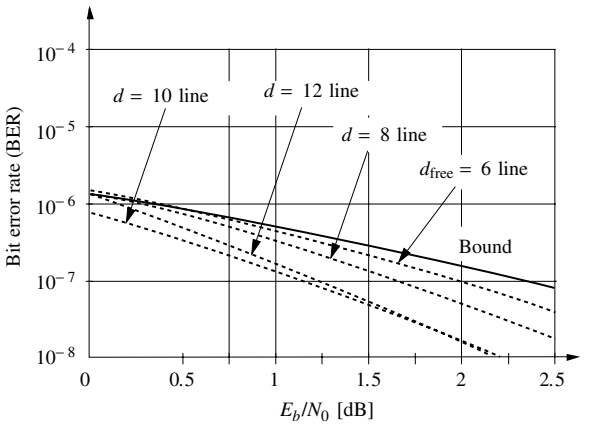


Figure 10.7 Performance bound and the influence of the different spectral lines for a turbo code.

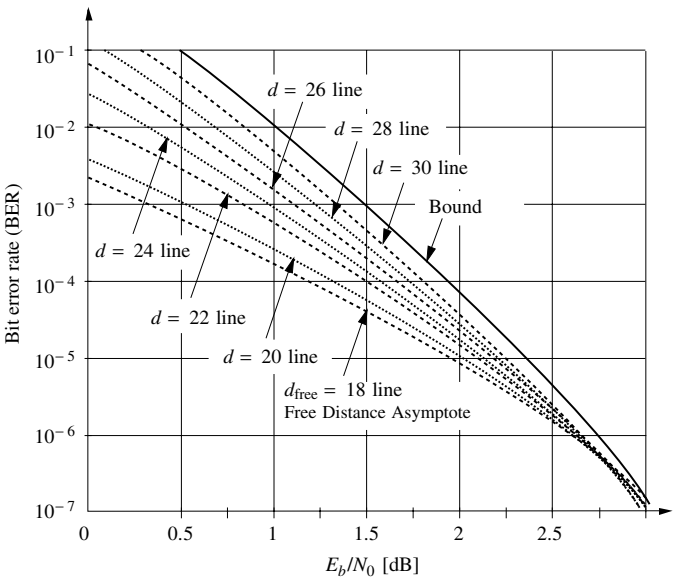


Figure 10.8 Performance bound and the influence of the different spectral lines for the MFD (2, 1, 14) convolutional code.

asymptote, which in turn is a consequence of the sparse distance spectrum of turbo codes, as opposed to spectrally dense convolutional codes. Finally, the sparse distance spectrum of turbo codes is due to the structure of the codewords in a parallel concatenation and the use of pseudorandom interleaving.

10.6 WEIGHT ENUMERATOR ANALYSIS OF TURBO CODES

In the previous sections, the performance of turbo codes was studied using distance spectrum data for a particular turbo code that was 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 and spectral thinning of turbo codes 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 [34]. As with convolutional codes and trellis codes, the primary result of this analysis will be insight into the design of the codes rather than easily computable bounds on the performance.

The bound of (10.3) 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):

$$A(W, X) = \sum_{d=d_{\text{free}}}^{\infty} \sum_{w} A_{w,d} W^w X^d,$$

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 6. Unfortunately, finding the exact transfer function of a turbo code is infeasible for all but trivial turbo codes due to the pairwise state complexity introduced by the interleaver.

In order to get a more manageable technique, we will compute an approximate WEF based on the notion of a probabilistic uniform interleaver. Random interleaving was introduced in refs. 7 and 9 to develop bounds on the average performance of turbo codes, and our presentation closely follows these. 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 $\boldsymbol{u}$ of weight w to an information sequence $\boldsymbol{u}'$, 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 $\boldsymbol{u}$ and $\boldsymbol{u}'$ occurs is

$$\frac{w!(N-w)!}{N!} = \frac{1}{\binom{N}{w}}. \tag{10.6}$$

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, emphasizing the similarity between these approaches. The uniform assumption is defendable here, since with this each interleaver is given equal weight, which is reasonable for the random selection discussed so far. The derivations here, however, do not constitute an existence argument as in the random coding bounds, since it is not excluded that different interleavers may be required for different weight input sequences.

This probabilistic interleaving argument can now be used to connect the codewords of the first component encoder in the turbo encoder to the codewords in the second component encoder. As (10.6) is a function of the weight of the *information* sequence, and the parallel concatenation demands this, some additional weight enumeration functions are necessary.

Following ref. 10, define the input redundancy weight enumerating function (IRWEF) of a systematic component encoder as

$$A^C(W, Z) = \sum_w \sum_z A^C_{w,z} W^w Z^z,$$

where $A^C_{w,z}$ 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)

$$A^C_w(Z) = \sum_z A^C_{w,z} Z^z.$$

$A^C_w(Z)$, 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 easily found as

$$A^C(W, Z) = \sum_w W^w A^C_w(Z).$$

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

$$P_b \leq \sum_{w=w_{\min}}^{N} \frac{w}{N} W^w \; A_w(Z)|_{W=Z=e^{-RE_b/No}}, \tag{10.7}$$

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 expressed 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 6 on the performance of a *specific* code. However, if we seek insight into turbo code design and other more general results, additional manipulation is necessary.

This additional manipulation is primarily a consequence of the differences between viewing a convolutional code as a having an infinite trellis and viewing a terminated convolutional code as a block code. In the former, codewords are frequently interpreted as error events—that is, a path that diverges from the all-zero state at a specific time and reemerges for the first time some number of branches later. As discussed in Section 10.2, the latter viewpoint results in codewords that are the concatenation of several error events. 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 error events with total information weight w and total parity weight z. Unlike in Section 10.2, we further constrain concatenation to not allow strings of zeros between the error events. Finally, assuming that the length of the error events 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 error events due to a weight w information sequence that can be positioned in an interleaver of length N.

The overall CWEF of the turbo code with uniform interleaving can now be approximated 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 with $\frac{N^n}{n!}$ and by approximating each term in the summations by the maximum values of n_1 and n_2, respectively. Doing this yields

$$A_w(Z) \approx \frac{w!}{(n_{\max_w}!)^2} N^{2n_{\max_w}-w} \left[A_w^{n_{\max_w}}\right]^2.$$

Finally, using this approximation in the bound of (10.7) results in

$$P_b \approx \sum_{w=w_{\min}}^{N} W^w \cdot w \cdot \frac{w!}{(n_{\max_w}!)^2} N^{2n_{\max_w}-w-1} \left[A_w^{n_{\max_w}}(Z)\right]^2 \Big|_{W=Z=e^{-RE_b/N_o}} \tag{10.8}$$

The approximation of (10.8) is now in a form that can be used to develop some design rules for turbo codes.

We begin by considering the effect of the choice of component encoder realization on the performance of the overall code. If the component encoders are realized in nonsystematic 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 (10.8) yields [10]

$$P_b \approx \sum_{w=1}^{N} W^w \frac{1}{(w-1)!} N^{w-1} \left[A_1^{(1)}(Z)\right]^{2w} \Big|_{W=Z=e^{-RE_b/N_o}}.$$

This expression reveals that for nonrecursive component encoders, the exponent of the interleaver length N is always nonnegative. That is, the effective multiplicity loses the factor of $1/N$ just as conventional convolutional codes. 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 10.2, 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 ref. 10, the cases for w odd and even are treated separately. For $w = 2j + 1$ odd, it is straightforward to show that the summand in (10.8) 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 N^{-2} factor, and we will no longer consider odd weight sequences. For $w = 2j$ even, the summand becomes

$$(2j) \cdot \binom{2j}{j} N^{-1} \cdot W^{(2j)} [A_{2j}^{(j)}(Z)]^2,$$

which has an interleaver 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 design rule for turbo codes.

Rule 10.1: *Choose component encoders with feedback.*

In order to develop further design rules, attention is now focuses only on those information sequences of even weight since they have a smaller interleaver gain. For even weight information sequences, the approximation of (10.8) becomes

$$P_b \approx \sum_{j=1}^{\lfloor N/2 \rfloor} (2j) \cdot \binom{2j}{j} N^{-1} \cdot W^{2j} \left[A_{2j}^{(j)}(Z)\right]^2 \Big|_{W=Z=e^{-RE_b/N_0}} . \tag{10.9}$$

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 consideration of code words associated with information sequences of weight 2. It is now necessary to briefly digress and explore the structure of code words generated by weight two information sequences in recursive encoder realizations.

In recursive encoders, codewords generated by weight two information sequences must correspond to error events with the first '1' required to leave the all-zero state and the 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 two information sequence is of the form

$$k \cdot z_{\text{cycle}} + t, \tag{10.10}$$

where k is an integer and t is parity weight achieved on the diverging and remerging transitions caused by the '1s'. Clearly, the minimum value of (10.10) is

$$z_{\min} = z_{\text{cycle}} + t,$$

which occurs when $k = 1$.

Returning to our analysis, we can now write

$$\begin{aligned} A_2^{(1)} &\approx Z^{z_{\min}} + Z^{2z_{\min}-t} + Z^{3z_{\min}-2t} \ldots \\ &= \frac{Z^{z_{\min}}}{1 - Z^{z_{\min}-t}} \end{aligned}$$

and substituting this expression into (10.9) yields

$$P_b \approx \sum_{j=1}^{\lfloor N/2 \rfloor} (2j) \cdot \binom{2j}{j} N^{-1} \cdot W^{2j} \left[A_2^{(1)}(Z) \right]^{2j} \Big|_{W=Z=e^{-RE_b/N_o}}$$

$$= \sum_{j=1}^{\lfloor N/2 \rfloor} (2j) \cdot \binom{2j}{j} N^{-1} \cdot W^{2j} \left[\frac{Z^{z_{\min}}}{1 - Z^{z_{\min}-t}} \right]^{2j} \Big|_{W=Z=e^{-RE_b/N_o}}.$$

Finally, letting $W = Z = H$ results in

$$P_b \approx \sum_{j=1}^{\lfloor N/2 \rfloor} (2j) \cdot \binom{2j}{j} N^{-1} \frac{(H^{2+2z_{\min}})^j}{(1 - H^{z_{\min}-t})^{2j}} \Big|_{H=e^{-RE_b/N_o}}, \tag{10.11}$$

where the quantity

$$d_{\text{eff}} = 2 + 2z_{\min}$$

has been termed the *effective free distance* of the turbo code [10].

The expression (10.11) demonstrates again that for large interleavers and moderate to high SNRs, the effective free distance is an important parameter affecting performance 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 that will be stated without proof and illustrated with examples.

Rule 10.2: *Choose the feedback polynomial to be a primitive polynomial.*

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 choosing the feedback polynomial to be a primitive polynomial of degree ν results in the maximum cycle length of $2^\nu - 1$. A formal proof of this rule may be found in ref. 10.

The third design rule is based on the observation that the effective free distance occurs when a weight two 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 two sequence that generates a complete codeword in the component encoder. This situation can be avoided by designing the interleaver such that two ones entering the first encoder separated by less than some number (say, S) of zeroes are interleaved such that they are separated by more than, say, $T > S$ zeroes. More precisely, let n_1 and n_2 be the indices of two '1s' in the original information sequence and $\pi(n_1)$ 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, than it can ensure that the resulting effective free distance is greater than the minimum.

Rule 10.3: *Choose an interleaver with appropriate spreading factors.*

This design rule presents somewhat of a dilemma. As demonstrated in Section 10.3, 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 low effective free distance. The solution may be found in the so-called $S-$random interleavers described in [20] which are algorithmically constructed using a computer. 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 Section 10.9.

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 [12] which has a nonprimitive feedback polynomial. 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 ref. 20. 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 10.9. 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. That is, at some BER the $(23, 33, 5000)$ code with a pseudorandom interleaver will also have an error floor.

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 order to understand the performance of turbo codes at low SNRs, in the so-called turbo cliff region, a different technique is required. This technique, known as EXIT analysis, is explicitly based on the iterative

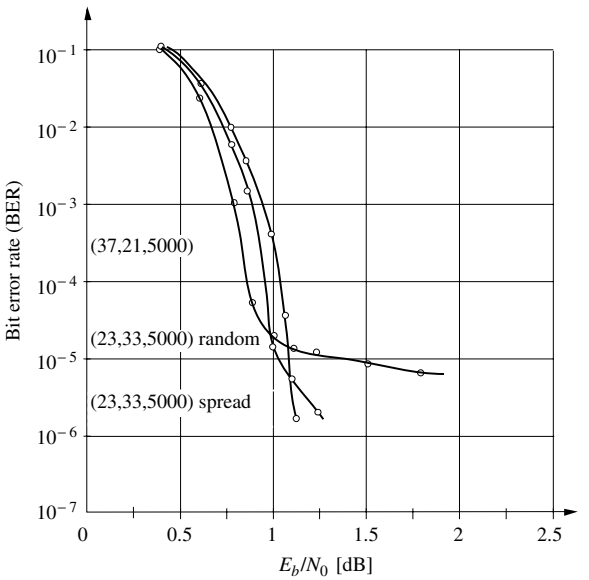


Figure 10.9 Performance of three different $R = 1/2$ turbo codes with an interleaver-length of $N = 5000$ and 20 decoder iterations.

decoding algorithm that makes turbo codes possible. This decoding algorithm is the subject of the next section.

10.7 ITERATIVE DECODING OF TURBO CODES

Optimal decoding of turbo codes is quickly realized to be too complex, and the discoverers of turbo codes proposed a novel iterative decoder based on the a posteriori probability (APP) decoding algorithm of the component codes described in Chapter 7. Empirical evidence suggests that this decoding algorithm performs remarkably well in the sense that it almost always finds the optimum decoding solution. Many of the convergence properties of this iterative algorithm, however, remain open research questions and are discussed in the literature. For the current discussion, a description of the algorithm is sufficient.

Even though the optimal decision rule is infeasible to implement, it can be used to motivate an iterative decoding procedure. Let us start with the maximum a posteriori rule, which would select

$$\hat{u}_r = \max_{u=\{0,1\}} P[u_r = u | \boldsymbol{y}^{(0)}, \boldsymbol{y}^{(1)}, \boldsymbol{y}^{(2)}],$$

where $\boldsymbol{y}^{(0)}$ is the received systematic bit sequence and $\boldsymbol{y}^{(m)}$, $m = 1, 2$, are the received parity sequences corresponding to the mth constituent encoder. (We will see later that the number of component encoders can be extended to $m > 2$.)

This formula can be developed as follows:

$$\begin{aligned} P[u_r = u|\boldsymbol{y}^{(0)}, \boldsymbol{y}^{(1)}, \boldsymbol{y}^{(2)}] &\equiv P(\boldsymbol{y}^{(0)}, \boldsymbol{y}^{(1)}, \boldsymbol{y}^{(2)}|u_r = u)P[u_r] \\ &= \sum_{\substack{\boldsymbol{u} \\ (u_r = u)}} P(\boldsymbol{y}^{(0)}, \boldsymbol{y}^{(1)}, \boldsymbol{y}^{(2)}|\boldsymbol{u})P[\boldsymbol{u}] \\ &= \sum_{\substack{\boldsymbol{u} \\ (u_r = u)}} P(\boldsymbol{y}^{(2)}|\boldsymbol{u})P(\boldsymbol{y}^{(0)}, \boldsymbol{y}^{(1)}|\boldsymbol{u})P[\boldsymbol{u}]. \quad (10.12) \end{aligned}$$

The conditioning in the second term in the summation of (10.12) can be reversed using Bayes' Rule to obtain

$$P(\boldsymbol{y}^{(0)}, \boldsymbol{y}^{(1)}|\boldsymbol{u})P[\boldsymbol{u}] = P(\boldsymbol{y}^{(0)}, \boldsymbol{y}^{(1)}, \boldsymbol{u}) \equiv P[\boldsymbol{u}|\boldsymbol{y}^{(0)}, \boldsymbol{y}^{(1)}]. \quad (10.13)$$

Clearly, exact evaluation of (10.13) is far too complex, and we simply use the *distribution separation*

$$P[\boldsymbol{u}|\boldsymbol{y}^{(0)}, \boldsymbol{y}^{(1)}] \approx \prod_{r=1}^{L} P[u_r|\boldsymbol{y}^{(0)}, \boldsymbol{y}^{(1)}],$$

where the values

$$P[u_r|\boldsymbol{y}^{(0)}, \boldsymbol{y}^{(1)}]$$

are seen to be soft information produced by the first decoder which has access only to $\boldsymbol{y}^{(0)}$ and $\boldsymbol{y}^{(1)}$. These are now used as *a priori probability information* in (10.12) delivered by decoder #1, to which decoder #2 adds the part concerning received sequence $\boldsymbol{y}^{(2)}$. The APP calculation of (10.12) can now be approximated by

$$\begin{aligned} P[u_r = u|\boldsymbol{y}^{(0)}, \boldsymbol{y}^{(1)}, \boldsymbol{y}^{(2)}] &\approx \sum_{\substack{\boldsymbol{u} \\ (u_r = u)}} P(\boldsymbol{y}^{(2)}|\boldsymbol{u}) \prod_{l=1}^{L} P[u_l|\boldsymbol{y}^{(0)}, \boldsymbol{y}^{(1)}] \\ &= \sum_{\substack{\boldsymbol{u} \\ (u_r = u)}} P[\boldsymbol{u}|\boldsymbol{y}^{(2)}] \prod_{l=1}^{L} P[u_l|\boldsymbol{y}^{(0)}, \boldsymbol{y}^{(1)}]P[u_l]. \end{aligned}$$

In Section 7.7, it was shown that this is precisely the quantity that the APP decoder for code #2 computes, using $P[u_l|\boldsymbol{y}^{(0)}, \boldsymbol{y}^{(1)}]P[u_l] \rightarrow P[u_l]$ as the 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

$$\Lambda(u_l) = \log \frac{P[u_l = 1|\boldsymbol{y}^{(0)}, \boldsymbol{y}^{(1)}]}{P[u_l = 0|\boldsymbol{y}^{(0)}, \boldsymbol{y}^{(1)}]}; \qquad \Lambda_s(u_l) = \log \frac{P[u_l = 1]}{P[u_l = 0]}.$$

Furthermore, according to (7.27)

$$L(u_r) = \log \frac{P[u_r = 1|\boldsymbol{y}^{(0)}, \boldsymbol{y}^{(1)}, \boldsymbol{y}^{(2)}]}{P[u_r = 0|\boldsymbol{y}^{(0)}, \boldsymbol{y}^{(1)}, \boldsymbol{y}^{(2)}]}$$

can be calculated as

$$L(u_r) = \log \frac{\sum_{i,j \in A(u_r=1)} \gamma_r(j,i)\alpha_{r-1}(i)\beta_r(j)}{\sum_{i,j \in A(u_r=0)} \gamma_r(j,i)\alpha_{r-1}(i)\beta_r(j)}, \tag{10.14}$$

where the transition probability $\gamma_r(i, j)$ is given by (6.32) as

$$\gamma_r(i, j) = P[u_r|\boldsymbol{y}^{(0)}, \boldsymbol{y}^{(1)}]P(y_r^{(2)}|x_r^{(2)}), \tag{10.15}$$

and where u_r is the information bit causing the transition from $i \to j$. Substituting (10.15) into (10.14) and factoring yields

$$\begin{aligned} L(u_r) &= \log \frac{\sum_{i,j \in A(u_r=1)} P(y_r^{(2)}|x_r^{(2)})\alpha_{r-1}(i)\beta_r(j)}{\sum_{i,j \in A(u_r=0)} P(y_r^{(2)}|x_r^{(2)})\alpha_{r-1}(i)\beta_r(j)} + \log \frac{P[u_r = 1|\boldsymbol{y}^{(0)}, \boldsymbol{y}^{(1)}]}{P[u_r = 0|\boldsymbol{y}^{(0)}, \boldsymbol{y}^{(1)}]} \\ &= \Lambda_{e,r}^{(2)} + \Lambda_r, \end{aligned}$$

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

$$L(u_r) = \Lambda_{e,r}^{(2)} + \Lambda_{e,r}^{(1)} + \Lambda_s, \tag{10.16}$$

where

$$\Lambda_s = \log \frac{P(\boldsymbol{y}_r^{(0)}|u_r = 1)}{P(\boldsymbol{y}_r^{(0)}|u_r = 0)}$$

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 10.10, where each APP decoder corresponds to a constituent encoder 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 the sequences at each decoder are properly aligned.

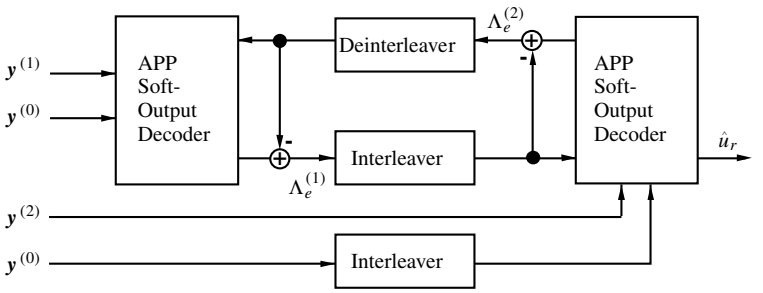


Figure 10.10 Block diagram of a turbo decoder with two constituent decoders.

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 (10.16) would lead to an accumulation of "old" extrinsic information by calculating

$$L_r^{(1)\prime} = \Lambda_{e,r}^{(1)\prime} + \underbrace{\Lambda_{e,r}^{(2)} + \Lambda_{e,r}^{(1)} + \Lambda_s}_{L_r^{(1)}}.$$

Therefore the decoders are constrained to exchange extrinsic information only, which is accomplished by subtracting the input values to the APP decoders from the output values as indicated in Figure 10.10.

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 10.11, which shows the performance of the original (37, 21, 65536) turbo code as a function of the decoder iterations. It is impressive that the performance of the code with iterative decoding continues to improve up to 18 iterations (and possibly beyond).

10.8 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 better understood by a statistical analysis, called the *extrinsic information transfer* (EXIT) *analysis*, first presented by ten Brink [40–42]. The EXIT analysis is related to similar methods that are based on the transfer of variances [21].

The basic philosophy of both methods is to view the component decoders of the turbo decoder (Figure 10.10) as a statistical processor that transforms an input

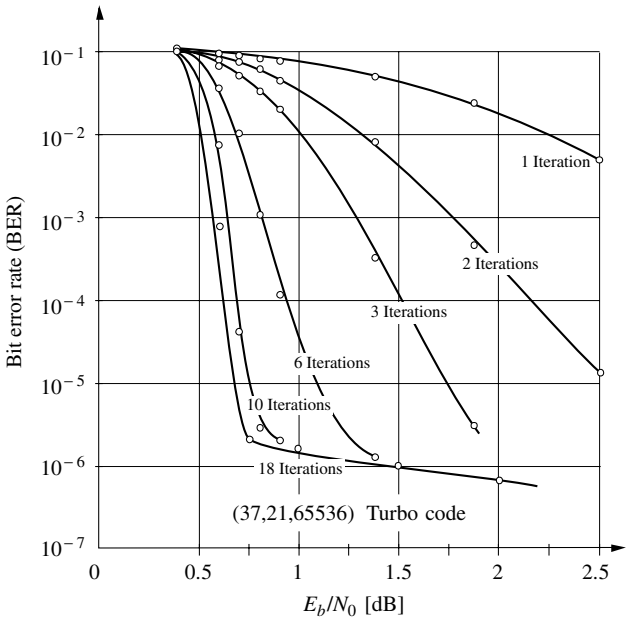


Figure 10.11 Performance of the (37, 21, 65536) turbo code as function of the number of decoder iterations.

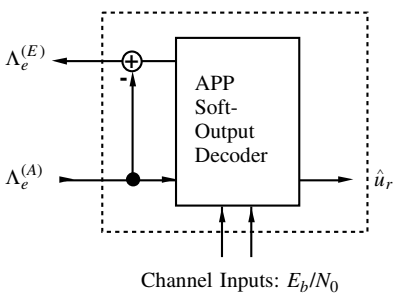


Figure 10.12 Component decoder seen as extrinsic LLR transformer.

value—that is, 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 transformer, as indicated in Figure 10.12. 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 will lead nowhere. The question is then how to measure the

quality of the extrinsic LLR value. Measurements of $\Lambda_e^{(m)}$ show that it has an approximately Gaussian distribution, that is,

$$\Lambda_e^{(m)} = \mu_\lambda u + n_\lambda; \qquad n_\lambda \sim \mathcal{N}(0, \sigma_\lambda^2), \tag{10.17}$$

where $u \in \{-1, 1\}$ is the information bit whose LLR is expressed by $\Lambda_e^{(m)}$, μ_λ is the mean, 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, if $\Lambda_e^{(m)}$ has variance σ_λ^2, then $\sigma_\lambda^2 \to 0$.
- The lower bound indicates low reliability, and $\sigma_\lambda^2 \to \infty$.
- The measure is monotonically increasing.

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 (10.17) with independent realizations. This independence models the effect of the interleaver, which, ideally, destroys any dependence between successive LLR values.

In the case of Gaussian modeled input LLR values, the mutual information measure, in bits, can be calculated as

$$I_A = I(u; \Lambda_e^{(A)}) = \frac{1}{\sqrt{2\pi}\sigma_\lambda} \int_{-\infty}^{\infty} \exp\left(-\frac{(\lambda - \mu_\lambda)^2}{2\sigma_\lambda^2}\right) \left(1 - \log_2\left[1 + \exp(-\lambda)\right]\right) d\lambda.$$

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 modeled input LLR with mutual extrinsic information value I_A is generated and

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 10.13 shows the EXIT function $T(I_A, E_b/N_0)$ for a memory 4, rate $R = 1/2$ component convolutional code with $h_0(D) = D^4 + D^3 + 1$ and $h_1(D) = D^4 + D^3 + D^2 + D + 1$ (see Section 4.3). The numbers on the different trajectories are the E_b/N_0 values in dB.

Figure 10.14 shows the EXIT function $T(I_A, E_b/N_0)$ for several rate 1/2 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 stronger 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

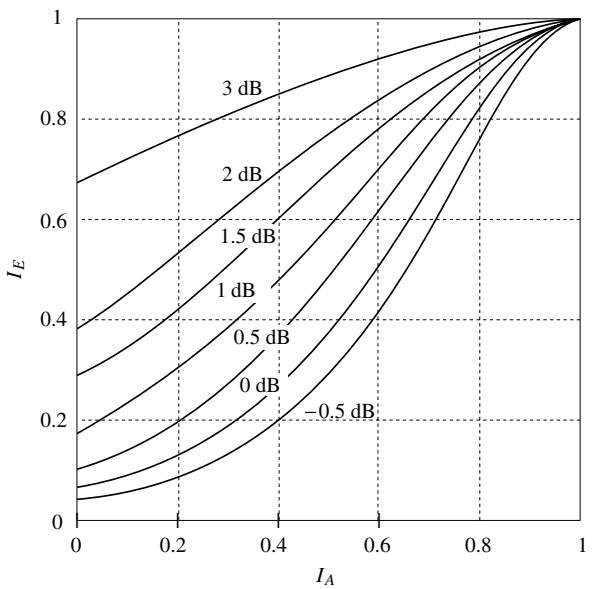


Figure 10.13 EXIT function of a rate 1/2, 16-state convolutional code.

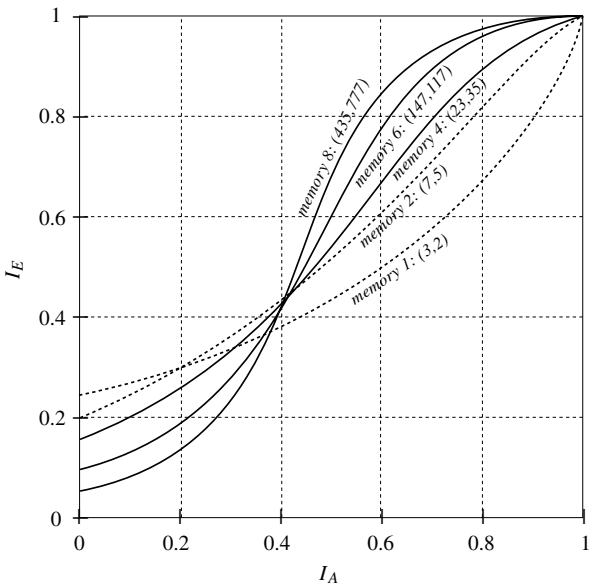


Figure 10.14 EXIT function of various rate 1/2 convolutional encoders.

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 this extrinsic information is now exchanged between the 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,

$$0 \xrightarrow{T_1} I_E^{(1)} = I_A^{(2)} \xrightarrow{T_2} I_E^{(2)} = I_A^{(2)} \xrightarrow{T_1} I_E^{(1)} = I_A^{(2)} \xrightarrow{T_2} \cdots ,$$

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 the input mutual information, and I_A is the output mutual information. As shown in Figure 10.15, 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—that is, with zero mutual information at point (0, 0) in the chart—the iterations progress by bouncing from one curve to the other as illustrated in Figure 10.15. 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), and so on, until

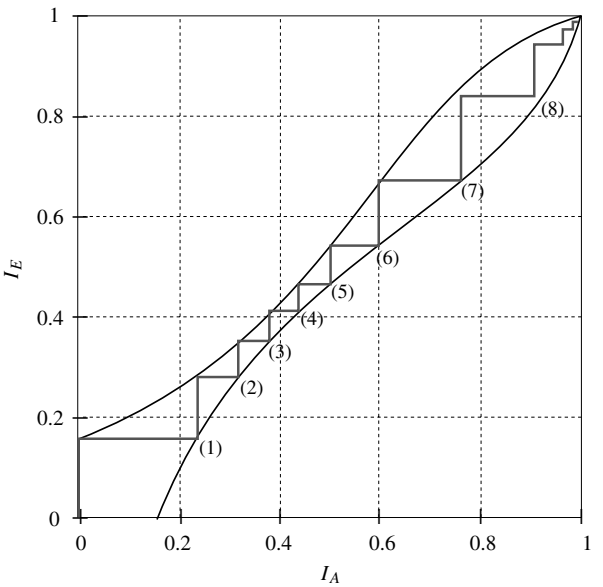


Figure 10.15 EXIT chart combining the EXIT functions of two 16-state decoders involved in a turbo decoder for $E_b/N_0 = 0.8$ dB.

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 10.11).

The trajectory plotted in Figure 10.15 is a measured characteristic of a single decoding 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 $N = 60{,}000$ [40]. The larger the interleaver chosen, the more accurate the EXIT chart becomes as prediction tool.

It is worth noting that most of these iterations occur at very high bit error rates, that is, $P_b > 10^{-2}$, 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. 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, and so on, but delivers no information about the error floor performance of a code and, thus, its ultimate operational performance.

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 in Figure 10.15

until the open channel closes at $E_b/N_0 = 0.53$ dB. This represents the *pinch-off* signal-to-noise ratio of, the original turbo code combination of encoders. 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 10.16 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 [40] 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 10.14. He presented the following Table 10.1 as a table of pinch-off values for all combinations of component codes.

Table 10.1 illustrates some very interesting facts. First, it clearly demonstrates that using larger codes, $\nu > 4$ does not give stronger turbo codes since the pinch-off is not improved; 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

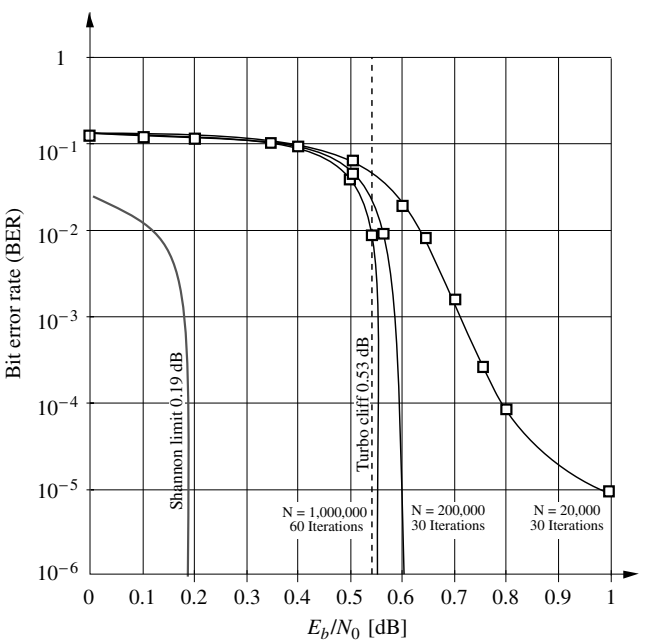


Figure 10.16 Simulations of the turbo code using 16-state component decoders and different sized interleavers related to theoretical prediction of the turbo cliff.

TABLE 10.1 Table of Pinch-off Signal-to-Noise Ratios for Various Combinations of Component Codes

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

ratio of 0.53 dB (see Figure 10.16). Much searching [40] 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. Further research into asymmetric code combinations is underway; see, for example, refs. 11 and 50.

10.9 INTERLEAVERS

As discussed earlier, an interleaver Π is a mapping of indices given by a permutation, and it represents an element of the permutation group of N elements. Permutation groups can be decomposed in a unique way, up to their order of arrangement, into cycles. This cycle representation can be useful, as seen later. The interleaver $\Pi_{16} = \{15, 10, 1, 12, 2, 0, 13, 9, 5, 3, 8, 11, 7, 4, 14, 6\}$ from Section 10.2, for example, can be written in the disjoint cycle decomposition with two cycles

$$\Pi_{16} = \{(0, 15, 6, 13, 4, 2, 1, 10, 8, 5), (3, 12, 7, 9)\}, \tag{10.18}$$

which simply means that the symbols map as follows: $0 \to 15 \to 6 \to 13 \to 4 \to 2 \to 1 \to 10 \to 8 \to 5 \to 0 \to 15$ into a cycle of length 8, and a second cycle of length 4. Alternately, the interleaver can be expressed by its index mapping function

$$d_I(i) = j, \qquad 0 \le i, j < N. \tag{10.19}$$

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

$$d_{I_B}(i) = ni + \left\lfloor \frac{i}{m} \right\rfloor \mod N, \tag{10.20}$$

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 ref. 39 with indexing function

$$d_{I_N}(i) = ki + u \mod N, \tag{10.21}$$

where k and u are fixed integers, and k is relatively prime to N, that is, the greatest common divisor of k and N, $\gcd(k, N) = 1$. The index k is called the *angular coefficient* of the linear interleaver. 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: Assume that two indices map into the same interleaved index, that is,

$$\begin{aligned} ki_1 + u &= ki_2 + qN, \\ k(i_1 - i_2) &= qN, \end{aligned} \tag{10.22}$$

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 is of weight 2 with separation t, the output sequence is

$$j + \tau = ki + kt + u = k(i + t) + u \tag{10.23}$$

and is always separated by τ. For any given τ there exists a unique value of t. Now, if the input sequence is

$$u(D) = \left((1 + D^t) + (1 + D^t)D^\tau\right) D^q = \left((1 + D^\tau) + (1 + D^\tau)D^t\right) D^q, \tag{10.24}$$

the corresponding output sequence is

$$u'(D) = \left((1 + D^\tau) + (1 + D^\tau)D^{t'}\right) D^{q'}. \tag{10.25}$$

Choosing $\tau = z_{\min}$ it becomes evident that, while linear interleavers can successfully break up weight-2 input sequences, they fail to do so for weight-4 sequences, which can cause low-weight parity sequences in both encoders. Thus, block interleavers may achieve good values of d_{free}, but fail at generating a thin spectrum. Angular coefficients of approximately $\sqrt{N}$ seem to work best, resulting in good scattering of the index points [39].

The next step toward improving performance of turbo codes in the error floor region was made by introducing S-random interleavers [20], which explicitly avoid mapping close indices at the input to close indices at the output. Generation of spread interleavers as defined on page 306 is straightforward. They are generated analogously to random interleavers, selecting each index $d_I(i)$ randomly and testing if $|d_I(i) - d_I(l)| \geq T$ for all $i - S < l < i$, that is, by checking all the previous choices. If an index choice does not fulfill this condition, it is discarded and a 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 even complete successfully. However,

according to ref. 20, 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 low free distance as discussed earlier.

In an effort to construct deterministic interleavers that have the statistical properties of random interleavers, Takeshita and Costello [39] designed what they called *quadratic interleavers*, which are defined by the quadratic indexing function

$$d_{I_N}(i) = \frac{ki(i+1)}{2} \mod N, \tag{10.26}$$

where k is an odd constant. Statistical comparisons with "ideal" uniform interleavers were conducted for weight-1 and weight-2 sequences in ref. 39. 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 ideal interleaver. Nonetheless, quadratic interleavers perform well and close to random interleavers, including in the error floor region. Their main advantage is therefore the rapid algorithmic construction, which may have to be performed at the receiver on the fly, as code parameters may change in adaptive communication systems.

Figure 10.17 shows these different interleavers for $N = 512$ as two-dimensional scatter plots; that is, the horizontal axis is the original index i and the vertical axis is the permuted index $d_I(i)$. The scatter plots give a easily comprehensible visual representation of a given interleaver. The leftmost plot in Figure 10.17 is that of a linear block interleaver, and the middle plot is that of a quadratic and the rightmost plot shows an S-random interleaver. A truly random interleaver has a scatter plot that is visually indistinguishable from that of a quadratic interleaver. These scatter plots offer yet another explanation of the functioning of the interleavers. It becomes evident, for example, that the linear interleaver gives a good spread, and hence a large d_{free}, but its regular pattern allows for large multiplicities as illustrated earlier. The quadratic interleaver offers random-like behavior, but has poor spreading, leading its turbo code to

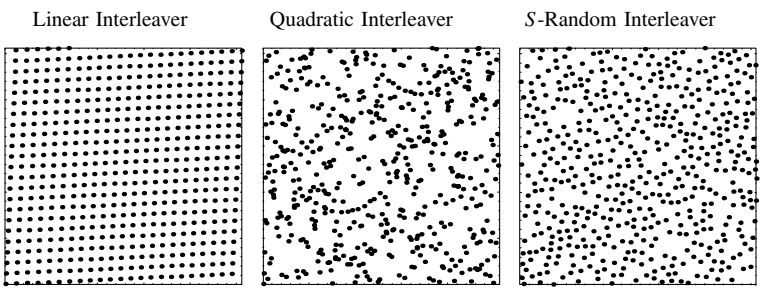


Figure 10.17 Two-dimensional scatter plots of a linear, a quadratic, and an S-random interleaver for a block size of $N = 512$.

suffer from a high error floor. The S-random interleaver, in contrast, combines a good spreading property with a random-like quality of the index points which avoids the accumulation of large multiplicities.

10.10 TURBO CODES IN TELECOMMUNICATION STANDARDS

Given their outstanding 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 we will discuss in this section.

10.10.1 The Space Data System Standard

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 new standard 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 [36]. The CCSDS Telemetry Channel Coding standard [45] uses a turbo code with two component codes with selectable rates and block lengths.

The encoder for this code is shown in Figure 10.18. Two rate $R = 1/4$ recursive convolutional encoders are used to generate $R = 1/2, 1/3, 1/4$, and

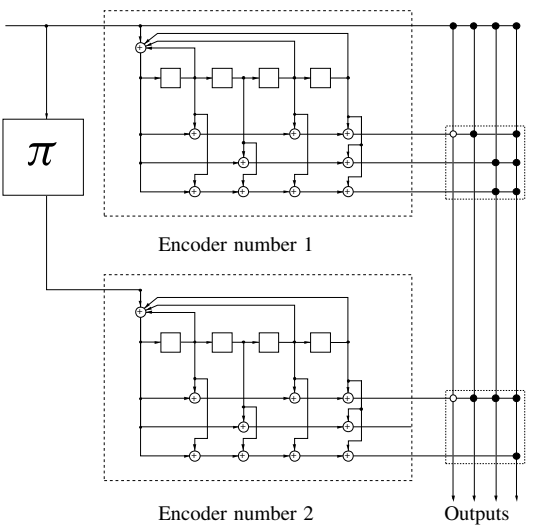


Figure 10.18 The CCSDS turbo encoder. A solid circle means every symbol is taken, and an open circle means every other symbol is taken.

$R = 1/6$ 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 feedforward 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-hand side of Figure 10.18. 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 [46]. The CCSDS standard allows five interleaver lengths $N_1 = 1784$, $N_2 = 3568$, $N_3 = 7136$, $N_4 = 8920$, 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 $N = 8920$ and various rates is shown in Figure 10.19.

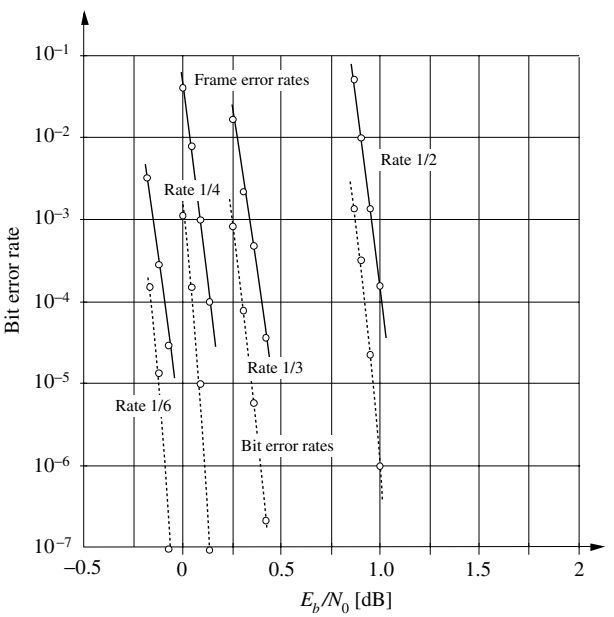


Figure 10.19 Frame and bit error rate performance of the CCSDS turbo code with $N = 8920$.

10.10.2 3G Wireless Standards

The third-generation (3G) mobile telecommunication standard [32] 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 are two main bodies undertaking work for the IMT-2000 standardization: the Third-Generation Partnership Project (3GPP) and 3GPP2. 3GPP addresses a wide-band CDMA (W-CDMA) air interface, and 3GPP2 addresses the cdma2000 air interface, which is an evolution from the IS-95 standard. Both specify the use of turbo codes.

Besides specifying constraint-length 9 convolutional codes, 3GPP also specifies the rate $R = 1/3$ turbo code shown in Figure 10.20 using 8-state constituent component codes.

The cdma2000 standard specifies three types of convolutional codes with rate $R = 1/4, 1/3$, and $R = 1/2$, all with constraint length 9. The convolutional codes are used for various signaling channels. The turbo codes specified use code rates of $R = 1/2, 1/3, 1/4$, or 1/5 and also use 8-state constituent component codes. It is shown in Figure 10.21. The different rates are achieved by puncturing.

For multimedia communications a wide range of possible block lengths are required, and therefore the 3GPP standard specifies interleavers for block sizes ranging from 40 to 5114. Due to this, interleavers with an efficient generation method are preferred. Prime interleavers were proposed [35] and later adopted by 3GPP [47]. The interleavers adopted by 3GPP2 have similar structures [48].

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 [35, 47], 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

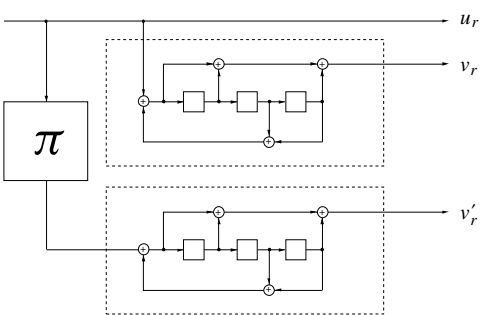


Figure 10.20 The 3GPP turbo encoder uses 8-state component codes.

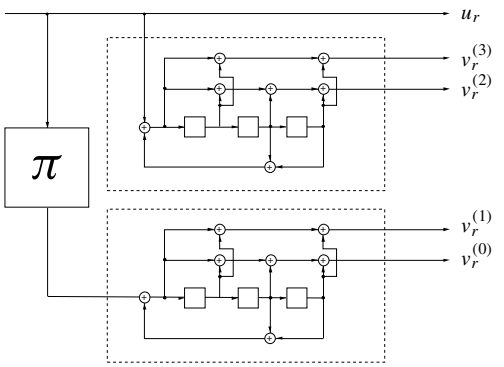


Figure 10.21 The 3GPP turbo encoder uses 8-state component codes.

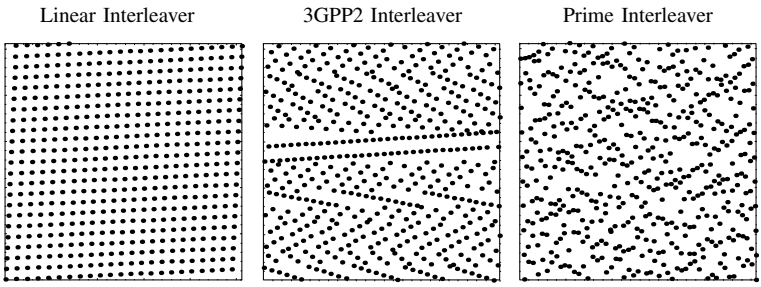


Figure 10.22 Two-dimensional scatter plots of a linear and the two 3GPP interleavers for a block size of $N = 512$.

angular coefficients. The net effect of both strategies is to make the interleaver less regular with respect to the original mother interleaver in order to avoid the high multiplicities of the latter. 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 10.22 give an impression of their randomness.

10.10.3 Digital Video Broadcast Standards

Finally, turbo codes have also 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 kbits/s and 2 Mbits/s to complement ADSL and cable modems [49]. 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

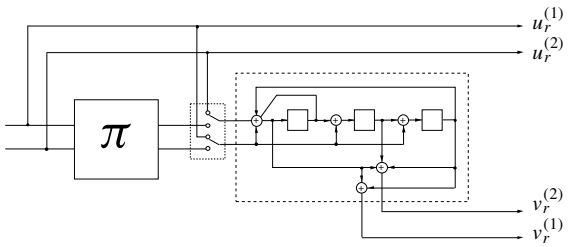


Figure 10.23 Self-concatenated turbo encoder used in the digital video broadcast standard.

block diagram is shown in Figure 10.23. 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 6/7 through various puncturing schemes. Interleaver sizes between $N = 48$ and $N = 752$ are specified using simple interleaving schemes [49].

A standard using product codes with iterative decoding has also been formulated for IEEE 802.16. It will be discussed in Chapter 12 where the product codes are introduced.

10.11 EPILOG

Turbo codes are an anomaly in the 50-year search for good error control codes since the publication of Shannon's mathematical theory of communication. Their introduction in 1993 has shaken a community that had come to believe that near-Shannon performance was "nearly" impossible, and now turbo codes with relatively simple iterative decoders allow decoding at signal-to-noise ratios arbitrarily close to the Shannon limit [16, 41], at least for low-rate binary codes with $R < 1$. (An example of a serially concatenated code with astonishingly simple component codes and with a pinch-off only 0.1 dB from the Shannon limit will be presented in the next chapter.)

While it has taken the scientific community a few years to come to terms with the new paradigm, nowadays iterative decoding is seen as the key to achieving ultimate performance limits, and applications have already 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 paradigm teaches pinch-off signal-to-noise ratios and turbo cliff behaviors. A communications link works either perfectly or not at all. It is a binary on–off situation, which will certainly have far-reaching impacts on other parts of complex communications systems, such as communications protocols.

BIBLIOGRAPHY

1. D. Arnold and G. Meyerhans, *The Realization of the Turbo-Coding System*, Swiss Federal Institute of Technology, Zurich, Switzerland, July 1995.
2. A. S. Barbulescu and S. S. Pietrobon, "Interleaver design for turbo codes," *Electron. Lett.*, vol. 30, no. 25, p. 2107, Dec. 8, 1994.
3. 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, January 5, 1995.
4. A. S. Barbulescu and S. S. Pietrobon, "Rate compatible turbo codes," *Electronics Letters*, vol. 31, no. 7, pp. 535–536, March 30, 1995.
5. 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.
6. G. Battail, "On random-like codes," unpublished manuscript.
7. S. Benedetto and G. Montorsi, "Average performance of parallel concatenated block codes," *Electron. Lett.*, vol. 31, no. 3, pp. 156–158, Feb. 2, 1995.
8. S. Benedetto and G. Montorsi, "Performance evaluation of TURBO-codes," *Electron. Lett.*, vol. 31, no. 3, pp. 163–165, Feb. 2, 1995.
9. 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.
10. S. Benedetto and G. Montorsi, "Design of parallel concatenated convolutional codes," *IEEE Trans. Commun.*, vol. COM-44, no. 5, pp. 591–600, May 1996.
11. 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, Sept. 1998.
12. 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.
13. 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, Sept. 1993.
14. 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, Sept. 1995.
15. M. Cedervall and R. Johannesson, "A fast algorithm for computing distance spectrum of convolutional codes," *IEEE Trans. Inform. Theory*, vol. IT-35, pp. 1146–1159, Nov. 1989.
16. 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. 16, no. 2, pp. 58–60, Feb. 2001.
17. D. Divsalar and F. Pollara, "Turbo codes for PCS applications," *Proceedings of the 1995 IEEE International Conference on Communications*, Seattle, Washington, June 1995.

18. D. Divsalar and F. Pollara, "Turbo codes for deep-space communications," *JPL TDA Progress Report 42-120*, Feb. 1995.

19. 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.

20. S. Dolinar and D. Divsalar, "Weight distributions for turbo codes using random and nonrandom permutations," *JPL TDA Progress Report 42-122*, pp. 56–65, Aug. 1995.

21. D. Divsalar, S. Dolinar, and F. Pollara, "Iterative Turbo decoder analysis based on density evolution," TMO Progress Report 42-144, Feb. 15, 2001.

22. 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.

23. J. Hagenauer, E. Offer, and L. Papke, "Iterative decoding of binary block and convolutional codes," *IEEE Trans. Inform. Theory*, Vol. IT-42, no. 2, pp. 429–445, March 1996.

24. C. Heegard and S. Wicker, *Turbo Coding*, Kluwer Academic Publishers, Boston, 1999.

25. O. Joerssen and H. Meyr, "Terminating the trellis of turbo-codes," *Electron. Lett.*, vol. 30, no. 16, pp. 1285–1286, Aug. 4, 1994.

26. 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. 20, 1994.

27. 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. 17, 1994.

28. P. Jung, "Novel low complexity decoder for turbo-codes," *Electron. Lett.*, vol. 31, no. 2, pp. 86–87, Jan. 19, 1995.

29. S. Lin and D.J. Costello, Jr., *Error Control Coding: Fundamentals and Applications*, Prentice-Hall, Englewood Cliffs, NJ, 1983.

30. 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, Illinois, Oct. 1995.

31. 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.

32. R. Prasad, W. Mohr, and W. Konhauser, *Third Generation Mobile Communication System*, Artech House, Norwood, MA, 2000.

33. P. Robertson, "Illuminating the Structure of Parallel Concatenated Recursive Systematic (TURBO) Codes," *Proceedings, GLOBECOM '94*, vol. 3, pp. 1298–1303, San Francisco, Nov. 1994.

34. 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.

35. A. Shibutani, H. Suda, and F. Adachi, "Complexity reduction of turbo coding," Proceedings, IEEE Vehicular Technology Conference VTC'99 Fall, Sep. 1999.

36. M. R. Soleymani, Y. Gao, and U. Uilaipornsawai, *Turbo Coding for Satellite and Wireless Communications*, Kluwer Academic Publishers, Boston, 2002.

37. 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.

38. Y. V. Svirid, "Weight distributions and bounds for turbo codes," *Eur. Trans. Telecommun.*, vol. 6, no. 5, pp. 543–556, Sept.–Oct. 1995.

39. 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.

40. S. ten Brink, "Convergence behavior of iteratively decoded parallel concatenated codes," *IEEE Trans. Commun.*, vol. 49, no. 10, Oct., 2001.

41. S. ten Brink, "A rate one-half code for approaching the Shannon limit by 0.1 dB," *IEE Electron. Lett.,* vol. 36, no. 15, pp. 1293–1294, July 2000.

42. S. ten Brink, "Convergence of iterative decoding," *IEE Electron. Lett.* vol. 35, no. 15, pp. 1117–1119, June 1999.

43. 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.

44. N. Wiberg, H.-A. Loeliger, and R. Kötter, "Codes and iterative decoding on general graphs," *Eur. Trans. Telecommun.*, vol. 6, pp. 513–525, Sept./Oct. 1995.

45. *Telemetry Channel Coding*, Consultative Committee on Space Data Systems, CCSDS 101.0-B-5, June 2001.

46. *Recommendations for Space Data System Standards*, Consultative Committee on Space Data Systems, CCSDS 101.0-B-6, Oct. 2002.

47. *Multiplexing and Channel Coding (FDD)*, TSG RAN TS25.212 v5.4.0, March 2003.

48. *Physical Layer Standard for CDMA2000*, Spread Spectrum Systems Release C, C.S0002-C Version 1.0, May 2003.

49. *Digital Video Broadcasting (DVB) Interaction Channel for Satellite Distribution Systems*, ETSI EN 301 790 V1.3.1, Final Draft, Nov. 2002.

50. O. Y. Takeshita, O. M. Collins, P. C. Massey, and O. J. Costello, Jr., "A note an asymmetric turbo-codes", *IEEE Comm. Lett.*, vol. 3, pp. 69–71, Nov. 1999.

CHAPTER 11

Serial Concatenation

11.1 INTRODUCTION

In the previous chapter it was shown that the parallel concatenation of convolutional codes coupled with an iterative decoding algorithm could perform near capacity at virtually all practical error rates. This required careful selection of the component codes and the interleaver. In this chapter we 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. Of course, the decoder becomes serially concatenated as well. We will see that the lessons and techniques learned from parallel concatenation can be applied to serial concatenation with a few minor adaptations.

Serially concatenated codes were first considered in detail by Forney [1] as a means of constructing codes with long block lengths and practical decoding algorithms. 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 technique for mitigating the error floor of parallel concatenation [2, 14].) 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 refs. 1 and 3 for additional information.

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 ref. 10 and came to full fruition in the outpouring of work that followed the discovery of turbo codes [4–8]. The chapter begins 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

Trellis and Turbo Coding, by Christian B. Schlegel and Lance C. Pérez
ISBN 0-471-22755-2

modifications required to the iterative decoding algorithm of Section 10.7 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.

11.2 AN INTRODUCTORY EXAMPLE

Before we begin with the formal analysis of serial concatenated codes, it is worth considering a simple example in order to provide some intuition into the analytical results and the basic differences between SCCCs and other forms of concatenation. The block diagram of a simple SCCC is shown in Figure 11.1. The outer code is a rate $R_o = 1/2$ code with generator matrix

$$G_o(D) = \begin{bmatrix} 1 + D^2 & 1 + D + D^2 \end{bmatrix}, \tag{11.1}$$

which specifies a 4-state code with $d^o_{\text{free}} = 5$. The inner code is a rate $R_i = 2/3$ code with generator matrix

$$G_i(D) = \begin{bmatrix} 1 + D & D & 1 \\ 1 + D & 1 & 1 + D \end{bmatrix} \tag{11.2}$$

which specifies a 4-state code with $d^i_{\text{free}} = 3$. As with the parallel concatenated codes of Chapter 10, the two encoders are separated by an interleaver.

The first observation to make is that without an interleaver, that is, if the interleaver is the identity permutation, the concatenated code simply becomes a convolutional code with overall generator matrix

$$G(D) = G_o(D)G_i(D) = \begin{bmatrix} D + D^2 & 1 + D^2 + D^3 & D^2 + D^3 \end{bmatrix}. \tag{11.3}$$

This generator matrix 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

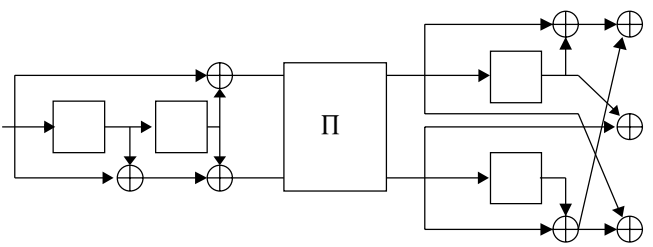


Figure 11.1 Block diagram of a simple serial concatenated convolutional code (SCCC2).

in Chapter 7. Concatenated codes constructed in this manner are referred to as cascaded convolutional codes. These codes were studied before the discovery of turbo codes and had a particularly interesting application on the Galileo mission [13].

Though the current chapter does not address cascaded codes, the cascaded code of (11.3) 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 possible 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 $d^o_{\text{free}} = 5$ 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 practically guarantees that the overall code will have a reasonable large free distance and that free distance will increase as the component code complexity increases. However, cascading by itself does not result in maximal free distance codes; the corresponding maximal free distance code given in Table 4.3 had $d_{\text{free}} = 10$.

As the filtering operation is not affected by the presence of an interleaver, it is reasonable to anticipate that SCCCs will not suffer from the very low free distance and corresponding error floors that hound turbo codes with random interleavers. This does not imply 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 Section 10.1, complex convolutional codes are asymptotically good but are spectrally dense and do not offer near-capacity performance at most practical error rates. In serial concatenation, the interleaver is used for spectral thinning as well as for introducing the randomness necessary for iterative decoding. In order to fully understand the role of the interleaver and the design of SCCCs, we now embark upon a detailed analysis again employing uniform interleaving.

11.3 WEIGHT ENUMERATOR ANALYSIS OF SCCCs

A block diagram of a serial concatenated convolutional code is shown in Figure 11.2. 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 [6].

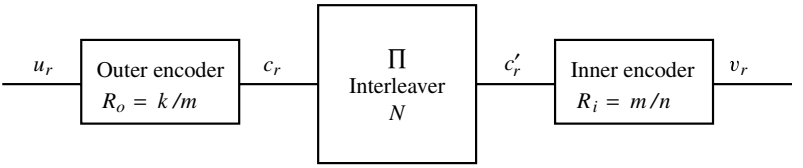


Figure 11.2 Block diagram of a serial concatenated convolutional code.

Recall from Chapter 10 that the input–output weight enumerating function (IOWEF) of a code may be written as

$$A(W, X) = \sum_{d=d_{\text{free}}}^{\infty} \sum_{w} A_{w,d} W^w X^d,$$

where $A_{w,d}$ is 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)

$$A_w(X) = \sum_{d} A_{w,d} X^d.$$

The probability of an information bit error may then be expressed as

$$P_b \leq \sum_{w=w_{\min}}^{NR_o} \frac{w}{NR_o} A_w(X)\Big|_{X=e^{-RE_b/N_o}}, \tag{11.4}$$

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

$$P_b \leq \sum_{d=d_{\text{free}}}^{N/R_i} \sum_{w=w_{\min}}^{NR_o} \frac{w}{NR_o} A_{w,d} e^{-dRE_b/N_o}, \tag{11.5}$$

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,

$$A_{w,d} = \sum_{l=d^o_{\text{free}}}^{N} \frac{A^{(o)}_{w,l} \cdot A^{(i)}_{l,d}}{\binom{N}{l}} \tag{11.6}$$

and

$$A_w(X) = \sum_{l=d^o_{\text{free}}}^{N} \frac{A^{(o)}_{w,l} \cdot A^{(i)}_{l}(X)}{\binom{N}{l}},$$

where $d^{(o)}_{\text{free}}$ 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 (11.4). It now remains to find the CWEFs of the equivalent block codes of the terminated component convolutional codes.

As with parallel concatenated codes, we will find it convenient to derive an approximate expression for these weight enumerators. Let

$$A^{(n)}_w(X) = \sum_d A^{(n)}_{w,d} X^d$$

be the n-event weight enumerating function of the equivalent block code of a terminated convolutional code, where $A^{(n)}_{w,d}$ is the number of codewords of weight d due to information sequences of weight w and which consist of n distinct error events. For large interleaver lengths N, the number of codewords of weight d due to information sequences of weight w may be approximated as

$$A_{w,d} \approx \sum_{n=1}^{n_{\max_w}} \binom{N}{n} A^{(n)}_{w,d}, \tag{11.7}$$

where $n_{\max_w}$ is the maximum number of error events due to information sequences of weight w that can be concatenated in a block of length N.

Applying the approximation of (11.7) to the outer and inner code of the serial concatenated system yields

$$A^{(o)}_{w,l} \approx \sum_{n^o=1}^{n^o_{\max_w}} \binom{N/m}{n^o} A^{(n^o)}_{w,l}$$

and

$$A_{l,d}^{(i)} \approx \sum_{n^i=1}^{n^i_{\max_l}} \binom{N/m}{n^i} A_{l,d}^{(n^i)},$$

respectively. Substituting these two approximations into (11.6) results in

$$A_{w,d} = \sum_{l=d^o_{\text{free}}}^{N} \sum_{n^o=1}^{n^o_{\max_w}} \sum_{n^i=1}^{n^i_{\max_l}} \frac{\binom{N/m}{n^o}\binom{N/m}{n^i}}{\binom{N}{l}} A_{w,l}^{(n^o)} A_{l,d}^{(n^i)}.$$

We will once again make use of the approximation

$$\binom{N}{n} \approx \frac{N^n}{n!}, \tag{11.8}$$

which results in

$$A_{w,d} = \sum_{l=d^o_{\text{free}}}^{N} \sum_{n^o=1}^{n^o_{\max_w}} \sum_{n^i=1}^{n^i_{\max_l}} N^{n^o+n^i-l} \frac{l!}{m^{n^o+n^i} n^o! n^i!} A_{w,l}^{(n^o)} A_{l,d}^{(n^i)}. \tag{11.9}$$

Finally, using (11.9) in the bound of (11.5) results in

$$P_b \le \sum_{d=d_{\text{free}}}^{N/R_i} \sum_{w=w_{\min}}^{N\cdot R_o} \sum_{l=d^o_{\text{free}}}^{N} \sum_{n^o=1}^{n^o_{\max_w}} \sum_{n^i=1}^{n^i_{\max_l}} N^{n^o+n^i-l-1} \frac{l! A_{w,l}^{(n^o)} A_{l,d}^{(n^i)}}{m^{n^o+n^i-1} n^o! n^i!} \frac{w}{k} e^{-dRE_b/N_o}. \tag{11.10}$$

A judicious examination of this bound yields considerable insight into the design of SCCCs for moderate and high SNRs. Indeed, (11.10) is the key to understanding the design rules for SCCCs.

First, notice that the inner four summations in (11.10) 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. Assuming 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 $d_{\text{free}}^{(o)}$, 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 $d_{\text{free}}^{(o)} \geq 2$ the SCCC will exhibit an interleaver gain as the SNR gets large and the outer code should 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 refs. 6 and 8 that the bound of (11.10) becomes loose compared to simulation results at higher and higher SNRs as the interleaver length N increased. 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. As N increases and the code becomes spectrally dense, this SNR moves farther away from the capacity limits discussed in Chapter 1.

This is consistent with observations made in Chapter 10 concerning the performance of maximum free distance convolutional codes with increasing memory. In order to characterize this, the authors in ref. 6 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 it is greater than or less than zero.

Following ref. 6, we first consider the case of inner encoders that do not employ feedback. For these encoder realizations, an error event may be caused by a information sequence of weight 1, and thus a sequence of weight l from the outer encoder may cause l distinct error events in the inner code. When this occurs, $n^i = l$ and the exponent is maximized by $n^o = n_{\max}^o$, which results in

$$\alpha_{\max}(\text{no feedback}) = n_{\max}^o + l - l - 1 = n_{\max}^o - 1 \geq 0, \tag{11.11}$$

and the corresponding multiplicity increases rapidly as N increases. The bound of (11.10) will then diverge from the free distance asymptote at higher SNRs, which suggests that the code will perform poorly at low and even moderate SNRs.

If the outer encoder generates a codeword with weight $d_{\text{free}}^{(o)}$, 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

$$\alpha_{d_{\text{free}}}(\text{nofeedback}) = 1 + d_{\text{free}}^{(o)} - d_{\text{free}}^{(o)} - 1 = 0 \tag{11.12}$$

and the interleaver does not affect the multiplicity. 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 is equal to the product of the free distances. Thus, the interleaver provides a significant improvement in free distance compared to cascaded convolutional codes.

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 $d_{\text{free}}^{(o)}$, 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

$$\alpha_{\max} \leq \max_{l} \left\{ \left\lfloor \frac{l}{d_{\text{free}}^{(o)}} \right\rfloor - \left\lfloor \frac{l+1}{2} \right\rfloor - 1 \right\}, \tag{11.13}$$

which is now a function of only l. In order to perform the maximization in (11.13), we observe that the weight l of a codeword produced by the outer encoder must satisfy

$$i d_{\text{free}}^{(o)} \leq l < (i+1) d_{\text{free}}^{(o)} \tag{11.14}$$

for some integer i. The maximum exponent occurs when $l = d_{\text{free}}^{(o)}$; and for this case

$$\alpha_{\max}(\text{feedback}) = -\left\lfloor \frac{d_{\text{free}}^{(o)} + 1}{2} \right\rfloor, \tag{11.15}$$

which implies that the multiplicities of all codewords see an interleaver gain, provided that 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 11.1: *The inner encoder should be in feedback form.*

Further insight into the design of SCCCs may be gained by considering the minimum weight of the codeword, denoted by d_α, associated with $\alpha_{\max}$. The

derivation of (11.15) required that the number of error events in the inner code be maximized. For $d_{\text{free}}^{(o)}$ even this means that there are $d_{\text{free}}^{(o)}/2$ error events, each due to a weight two input. Let $d_{\text{min}}^{(2)}$ 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

$$d_\alpha = \frac{d_{\text{free}}^{(o)} d_{\text{min}}^{(2)}}{2}. \tag{11.16}$$

If $d_{\text{free}}^{(o)}$ 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

$$d_\alpha = \frac{(d_{\text{free}}^{(o)} - 3) d_{\text{min}}^{(2)}}{2} + d_{\text{min}}^{(3)}, \tag{11.17}$$

where $d_{\text{min}}^{(3)}$ is the minimum weight of codewords in the inner code due to weight three input sequences. Clearly, it is desirable to have an inner code with large $d_{\text{min}}^{(2)}$—that is, a large effective free distance. From Chapter 10, we know that this is accomplished by choosing the feedback polynomial to be primitive, and this becomes the second design rule.

Rule 11.2: *Maximize inner encoder effective free distance by choosing the 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 $d_{\text{free}}^{(o)}$, and thus traditionally good convolutional codes are suitable. It is well known that with traditional convolutional codes, nonrecursive 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 (11.17) which shows that d_α is based on both the effective free distance of the inner code and $d_{\text{min}}^{(3)}$ when $d_{\text{free}}^{(o)}$ is odd. If the inner code can be chosen such that no odd weight input sequences generate finite-length codewords, then $d_{\text{min}}^{(3)} = \infty$ 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 nonprimitive feedback polynomial is a tradeoff between performance at low and high SNRs, respectively [12].

11.3.1 Design Rule Examples

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 nonsystematic feedforward form. The encoder for this code is

$$G_o(D) = \begin{bmatrix} 1 + D^2 & 1 + D + D^2 \end{bmatrix}, \tag{11.18}$$

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

$$G_1(D) = \begin{bmatrix} 1 & 0 & \dfrac{D^2 + 1}{D^2 + D + 1} \\ 0 & 1 & \dfrac{D + 1}{D^2 + D + 1} \end{bmatrix}$$

and is realized in systematic feedback form[1]. Note that the feedback polynomial is primitive. Simulation results for the resulting rate $R = 1/3$ concatenated code, denoted SCCC1 and shown in Figure 11.3, are shown in Figure 11.4 for several interleaver lengths. This code has $d_{\text{free}}^{(o)} = 5$ and from (11.15) the maximum exponent is given by

$$\alpha_{\max} = -\left\lfloor \frac{d_{\text{free}}^{(o)} + 1}{2} \right\rfloor = -\left\lfloor \frac{5 + 1}{2} \right\rfloor = -3 \tag{11.19}$$

and we would expect an interleaver gain of N^{-3}. The simulation results do indeed manifest this gain with increasing N.

The second serial concatenated convolutional code uses a rate $R_i = 2/3$ inner code with total encoder memory $\nu = 2$ with generator matrix

$$G_2(D) = \begin{bmatrix} 1 + D & D & 1 \\ 1 + D & 1 & 1 + D \end{bmatrix}$$

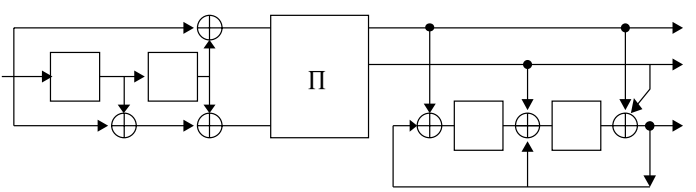


Figure 11.3 Block diagram and encoders for SCCC1.

[1]Note 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.

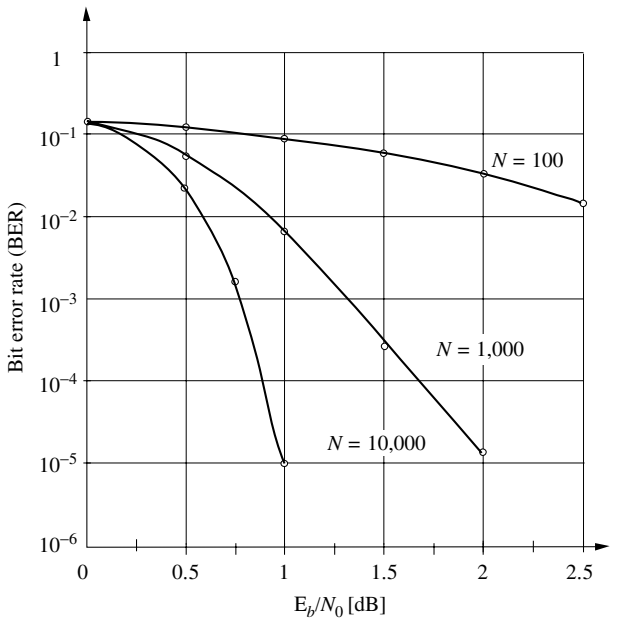


Figure 11.4 Simulation results for the rate $R = 1/3$ code SCCC1 (inner encoder with feedback) as a function of interleaver length.

when realized in nonsystematic feed-forward form. Simulation results for the resulting rate $R = 1/3$ concatenated code, denoted SCCC2 and shown in Figure 11.1, are given in Figure 11.5 for several interleaver lengths. This code clearly performs much worse than SCCC1.

For increasing N this code exhibits an error floor despite its relatively large free distance. In fact, simulations results suggest that the performance curves for $N = 1000$ and $N = 10000$ 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 (11.12) there is no interleaver gain associated with the free distance multiplicity. Thus, the free distance asymptotes of these two codes are essentially identical. For large N, however, the pseudorandom interleaver does have a spectral thinning effect on some higher-weight codewords which results in improved performance at moderate SNRs.

In order to illustrate the second design rule, we construct a third code, SCCC3, using a nonprimitive feedback inner encoder. SCCC3 uses a rate $R_i = 2/3$ inner code with total encoder memory $\nu = 2$ and generator matrix

$$G_3(D) = \begin{bmatrix} 1 & 0 & \dfrac{D^2}{D^2+1} \\ 0 & 1 & \dfrac{D^2+D+1}{D^2+1} \end{bmatrix}$$

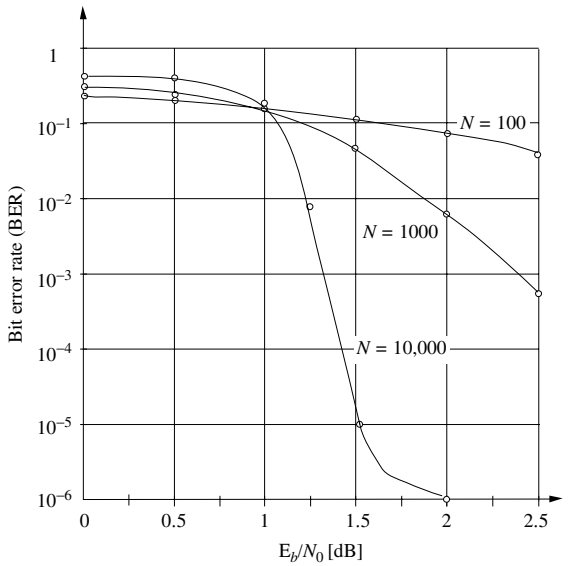


Figure 11.5 Simulation results for the $R = 1/3$ code SCCC2 (inner encoder without feedback) as a function of interleaver length.

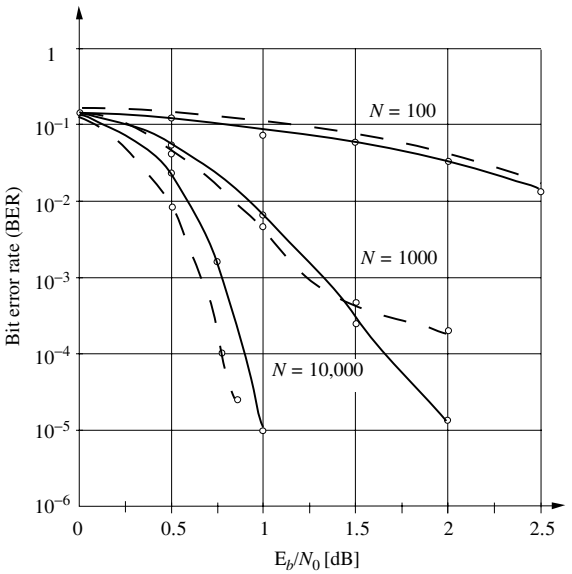


Figure 11.6 Performance of serial concatenated convolutional codes with a primitive inner encoder (SCCC1, solid line) and a nonprimitive inner encoder (SCCC3, dashed line).

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 11.6 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.

11.4 ITERATIVE DECODING AND PERFORMANCE OF SCCCs

Serially 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 11.7. The APP decoders are the same as those described in Chapter 7 and Section 10.7. 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

$$\begin{aligned} L_r^{(1)} &= \log \frac{P[c'_r=1|\boldsymbol{y}]}{P[c'_r=0|\boldsymbol{y}]} \\ &= \log \frac{P[\boldsymbol{y}|c'_r=1]}{P[\boldsymbol{y}|c'_r=0]} + \log \frac{P[c'_r=1]}{P[c'_r=0]} \\ &= \Lambda_{e,r}^{(1)'} + \Lambda_r^{(1)'}, \end{aligned}$$

of which only $\Lambda_{e,r}^{(1)'}$ is passed to the second decoder, again for reasons of avoiding the accumulation of old information. The outer APP decoder must now compute its LLR using the correctly deinterleaved extrinsic information from the first decoder as its only input.

The outer decoder must compute an information bit log-likelihood for final decoding and a codeword bit log-likelihood for iterative processing by the inner

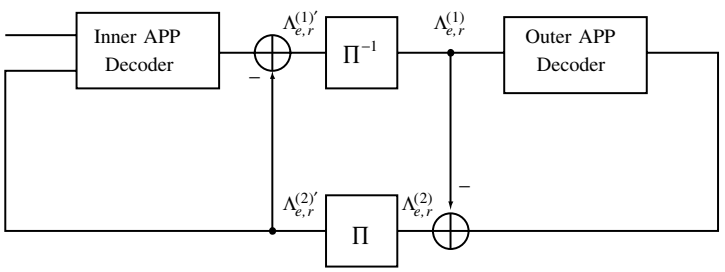


Figure 11.7 Block diagram of the iterative decoder for serial concatenated codes.

decoder. The outer APP thus computes

$$\begin{aligned} L_r^{(2)} &= \log \frac{P[c_r = 1|\mathbf{\Lambda}_r^{(1)}]}{P[c_r = 0|\mathbf{\Lambda}_r^{(1)}]} \\ &= \log \frac{P[\mathbf{\Lambda}_r^{(1)}|c_r = 1]}{P[\mathbf{\Lambda}_r^{(1)}|c_r = 0]} + \log \frac{P[c_r = 1]}{P[c_r = 0]} \\ &= \Lambda_{e,r}^{(2)} + \Lambda_{e,r}^{(1)} \end{aligned}$$

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 information sequence. This is based on the capacity 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 performance of the iterative decoding algorithm as a function of the number of iterations is shown in Figure 11.8 for SCCC1. Notice that only 7 iterations are required for the decoder to converge. This is substantially less than the 18 to 20 required for turbo codes.

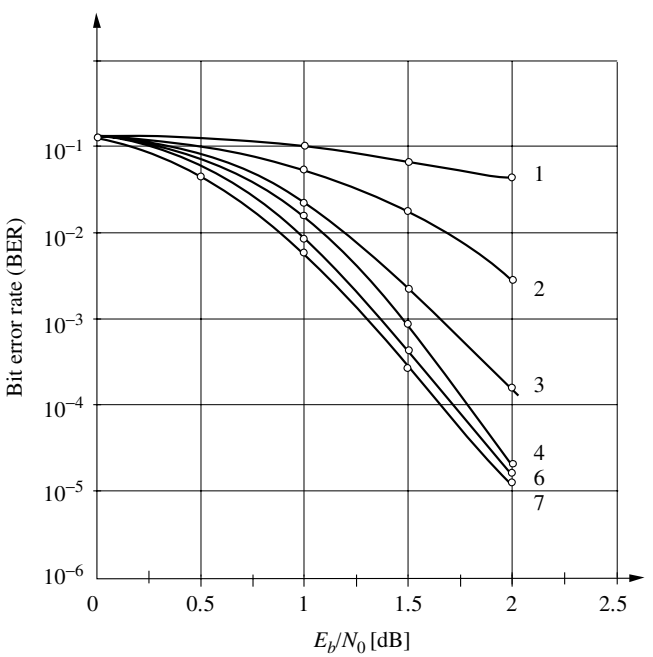


Figure 11.8 Performance of SCCC1 as a function of decoder iterations for $N = 1000$.

11.4.1 Performance of SCCCs and PCCCs

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 10.6 and 11.3 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 (10.11),

$$P_b \approx \sum_{j=1}^{\lfloor N/2 \rfloor} (2j) \cdot \binom{2j}{j} N^{-1} \left. \frac{(H^{2+2z_{\min}})^j}{(1 - H^{z_{\min}-t})^{2j}} \right|_{H=e^{-RE_b/N_o}},$$

which is repeated here for convenience. The quantity

$$d_{\text{eff}} = 2 + 2z_{\min}$$

is the effective free distance of the code and $z_{\min}$ was the minimum parity weight generated by a weight-2 information sequence. This expression yields two insights into the performance of turbo codes. First, the interleaver gain is fixed at N^{-1} and, second, the overall PCCC will on average have a relatively low free distance determined by $z_{\min}$.

For serial concatenated convolutional codes, the situation is markedly different. If we assume that $d_{\text{free}}^{(o)}$ is even, then substituting (11.15) and (11.16) into (11.10) and keeping only the term corresponding to $\alpha_{\max}$ results in

$$P_b \approx K N^{-d_{\text{free}}^{(o)}/2} \exp\left[-\frac{d_{\text{free}}^{(o)} d_{\min}^{(2)}}{2} \frac{RE_b}{N_o} \right], \tag{11.20}$$

where K is a constant proportional to the multiplicity [6]. From this expression, we expect SCCCs to have an interleaver gain of $N^{-d_{\text{free}}^{(o)}/2}$ and the dominant distance to be

$$\frac{d_{\text{free}}^{(o)} d_{\min}^{(2)}}{2},$$

both of which increase with the outer code free distance. The increased interleaver gain and dominant distance suggest that SCCCs should have a distinct advantage over PCCCs with random interleavers for large N and E_b/N_o.

Figure 11.9 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 $h_1(D) = D^2 + 1$ and $h_0(D) = D^2 + D + 1$. Here 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

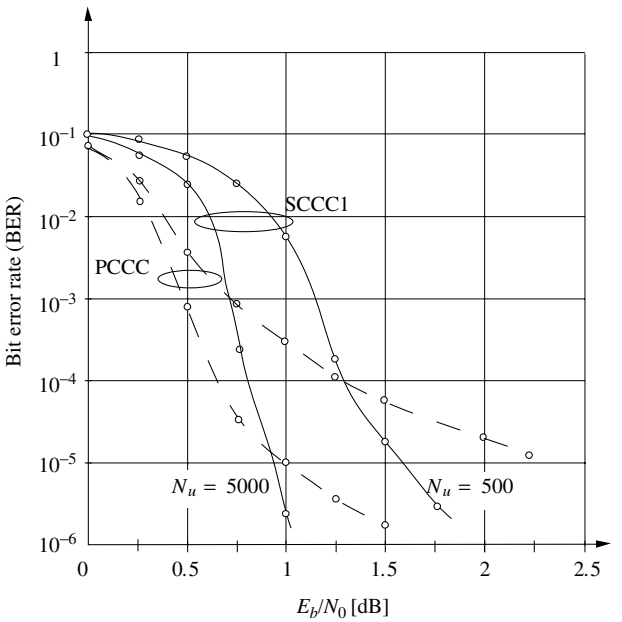


Figure 11.9 Comparison of an SCCC and a PCCC with the same overall block lengths and seven decoder iterations.

iterative decoding algorithm. The results shown in Figure 11.9 clearly show the advantage of the serial concatenated code for large SNR where it does not manifest an error floor due to its larger free distance. 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.

11.5 EXIT ANALYSIS OF SERIALLY CONCATENATED CODES

As was the case 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 10.8. EXIT analysis will once again predict the onset of the turbo cliff for SCCCs and in addition leads to a particularly simple serial scheme with outstanding performance [9, 15].

There are two fundamental differences between the EXIT analysis of parallel concatenated codes and serial concatenated codes [16]. First, for serial concatenated codes the inner decoder passes extrinsic information concerning its code symbols, rather than information symbols as with turbo codes, to the outer decoder. Thus, the EXIT function of the inner decoder,

$$I_E^i = T_i(I_A, E_b/N_0),$$

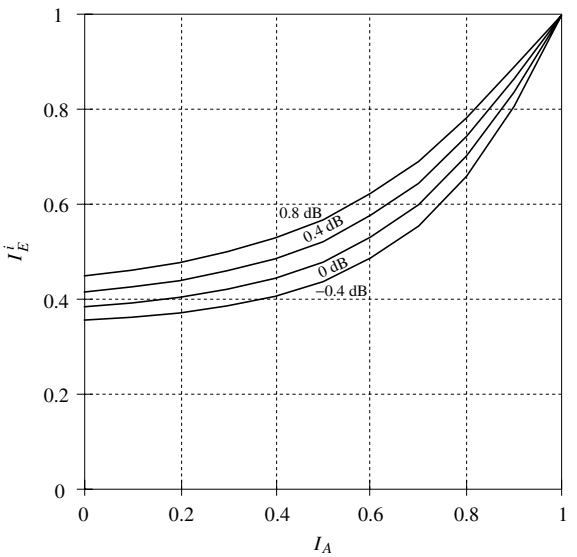


Figure 11.10 EXIT function of the rate $R_i = 2/3$ inner code used in SCCC1.

is slightly different. The EXIT function of the rate $R_i = 2/3$, memory 2 inner code used in SCCC1 is given in Figure 11.10 as a function of E_b/N_0. Notice the relatively large starting value of I_E^i.

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 $R_o = 1/2$ outer codes are given in Figure 11.11 as functions of I_E^i.

As with turbo codes, the EXIT charts of the inner and outer encoders can be combined to form an EXIT exchange chart that predicts the behavior of the iterative decoder. For serial concatenated codes the inner and outer codes are not identical and the EXIT exchange chart of the overall code is not symmetric with respect to the 45° line as in Figure 10.15. The EXIT exchange chart for SCCC1 with $N = 10{,}000$ is shown in Figure 11.12. 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 [11] and *repeat-accumulate* (RA) codes [9]. 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

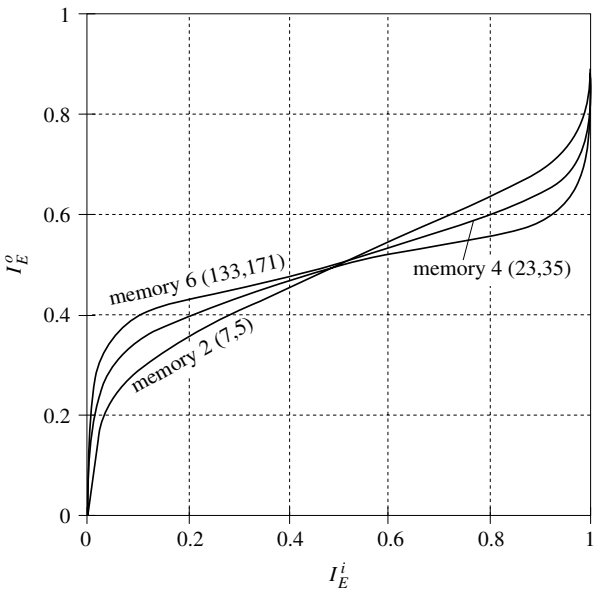


Figure 11.11 EXIT function of several $R_o = 1/2$ outer codes.

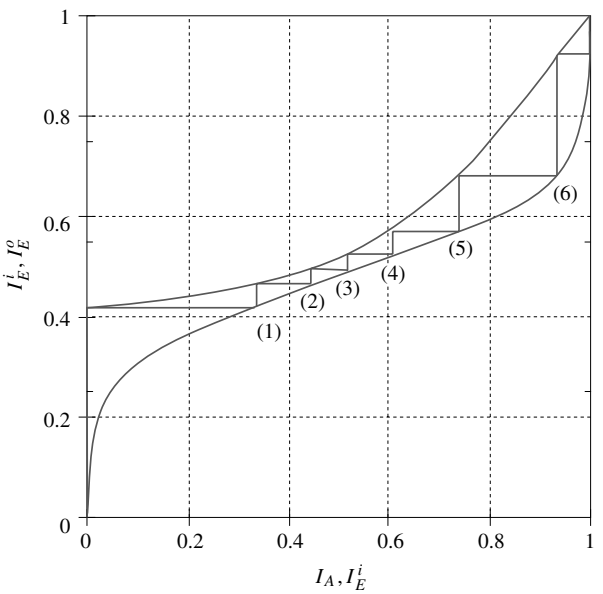


Figure 11.12 EXIT exchange chart for SCCC1 with decoder iterations for $E_b/N_0 = 0$ dB.

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. The overall rate of the code is determined solely by the outer code.

The inner accumulator is simply a rate one-half recursive systematic convolutional encoder that is punctured to rate one. For DRS codes the puncturing pattern is typically such that one out of 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 one recursive encoder with feedback polynomial $h_0(D) = D + 1$ and feedforward polynomial $h_1(D) = 1$.

Both of these serial concatenated codes take advantage of the fact that the EXIT function of a repetition code is a straight line with

$$I_E^o = T_o(I_E^i) = I_E^i,$$

regardless of the rate and 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 [15] revealed that an inner code with feedback polynomial $h_0(D) = D^3 + D^2 + D + 1$ and feedforward polynomial $h_1(D) = D^3 + D^2 + D$ 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 11.13. This figure shows that as the signal-to-noise ratio is decreased,

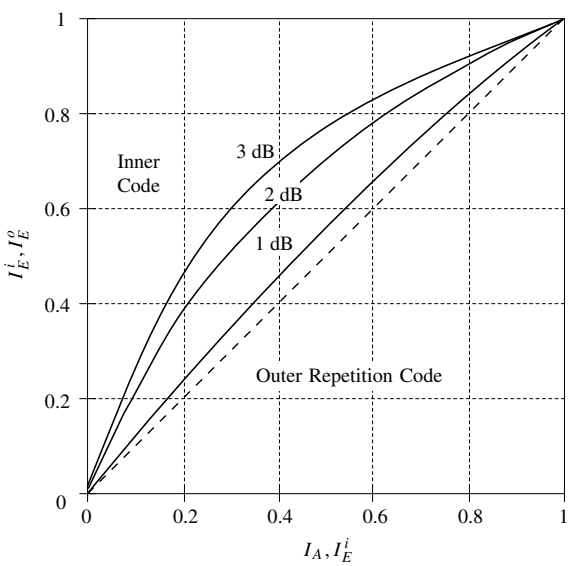


Figure 11.13 EXIT exchange chart for a doped-repeat-scramble code with inner code $(h_0(D), h_1(D)) = (1 + D + D^2 + D^3, D + D^2 + D^3)$ and a doping ratio of 1 in 100.

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.

11.6 CONCLUSION

In this chapter, we have seen that the iterative decoding paradigm enables serial concatenated schemes to also achieve near-capacity performance. Serial concatenated convolutional codes may have some advantages over turbo codes. Because the inner encoder operates on outer encoder codewords rather than information sequences, SCCCs naturally have relatively high free distance and do not usually manifest error floors. In addition, unlike turbo codes, no special interleaver is required to achieve this. Initially, serial concatenated schemes were thought not to be able to push their performance to the capacity limit as did turbo codes. However, the discovery of repeat-accumulator and related codes has demonstrated this is not the case. With low-density parity-check codes and with parallel concatenated and serial concatenated codes, the communications engineer has a variety of error control coding schemes that offer outstanding performance at most error rates.

BIBLIOGRAPHY

1. G. D. Forney, *Concatenated Codes*, MIT Press, Cambridge, MA, 1966.
2. D. Arnold and G. Meyerhans, *The Realization of the Turbo-Coding System*, Swiss Federal Institute of Technology, Zurich, Switzerland, July 1995.
3. V. Bhargava and S. Wicker, *Reed–Solomon Codes and their Applications*, Prentice-Hall, Englewood Cliffs, NJ, 1988.
4. S. Benedetto and G. Montorsi, "Serial concatenation of block and convolutional codes," *IEE Electron. Lett.*, vol. 32, no. 13, pp. 1186–1187, June 20, 1996.
5. S. Benedetto and G. Montorsi, "Iterative decoding of serially concatenated convolutional codes," *IEE Electron. Lett.*, vol. 32, no. 10, pp. 887–888, May 9, 1996.
6. 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.
7. 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.
8. 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. 909–926, May 1998.
9. 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.

10. 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.

11. S. Huettinger, S. ten Brink and J. Huber, "Turbo-code representation of RA-codes and DRS-codes for reduced coding complexity," *Proceedings of the 2001 Conference on Information Science and Systems*, Baltimore, March 21–23, 2001.

12. F. Jiang, *Design and Analysis of Serially Concatenated Convolutional Codes*, Master's thesis, The University of Nebraska, Lincoln, Dec. 2001.

13. F. Pollara and D. Divsalar, "Cascaded Codes," *TDA Progress Report 42-110*, pp. 202–207, Aug. 15, 1992.

14. O. Y. Takeshita, O. M. Collins, P. C. Massey, and D.J. Costello, Jr. "On the frame-error rate of concatenated turbo code," *IEEE Commun. Lett.*, vol. 49, no. 4, pp. 602–608, April 2001.

15. S. ten Brink, "Rate one-half code for approaching the Shannon limit by 0.1 dB," *IEEE Electron. Lett.*, vol. 36, no. 15, pp. 1293–1294, July 20, 2000.

16. S. ten Brink, "Design of serially concatenated codes based on iterative decoding convergence," *Proceedings of the 2nd International Symposium on Turbo Codes*, pp. 100–103, Sept. 2000.

CHAPTER 12

Turbo-Coded Modulation

12.1 INTRODUCTION

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. As we will see, there are very few changes necessary to accommodate higher signaling alphabets, and therefore higher spectral efficiencies. Mainly we are dealing with multilevel symbols now, rather than binary symbols, and the concept of log-likelihood ratios, which served us so well with binary codes, is not applicable any longer.

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 shown in Chapter 2.

The last representatives discussed in this chapter are *block turbo-coded modulation* (BTCM) systems, which are based on traditional product codes, using small block codes as component codes. Originally, BTCM was used with binary signaling, but since they are very easily extended to larger signaling alphabets and higher spectral efficiencies, it seems appropriate to treat them in this chapter. Some of the spectral efficiencies achieved by BTCM were shown in Figure 1.7. Also, to date, the implementation of BTCM is more advanced than that of TTCM systems; in fact there exists standard and complete VLSI circuits that implement a variety of BTCM systems, discussed in Section 12.8.

12.2 TURBO-TRELLIS-CODED MODULATION (TTCM)

Figure 12.1 shows the encoder for a TTCM system, originally proposed by Robertson and Wörz [23–26], and its similarity to a classical turbo encoder (parallel concatenation) is evident. The difference is that blocks of n coded bits are treated as input symbols. The interleaver is symbol-oriented, and the

Trellis and Turbo Coding, by Christian B. Schlegel and Lance C. Pérez
ISBN 0-471-22755-2

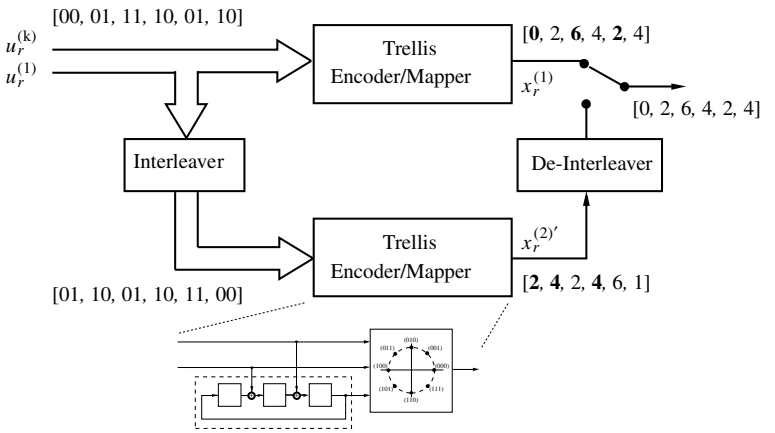


Figure 12.1 Block diagram of a turbo encoder with two constituent encoders and an optional puncturer.

component encoders are trellis encoders—for example, the 8-PSK encoder introduced in Chapter 3 for $n = 3$. The final sequence of transmitted symbols is generated by selecting symbols alternately from the two encoders, that is, $\boldsymbol{x} = [x_0^{(1)}, x_1^{(2)}, x_2^{(1)}, \ldots]$. 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 as an ordered sequence, irrespective of whether the symbol originated from the first or the second component encoder.

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 12.1. 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, 1]. Only the boldfaced 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 12.2. The sequence of received symbols $\boldsymbol{y}$ 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

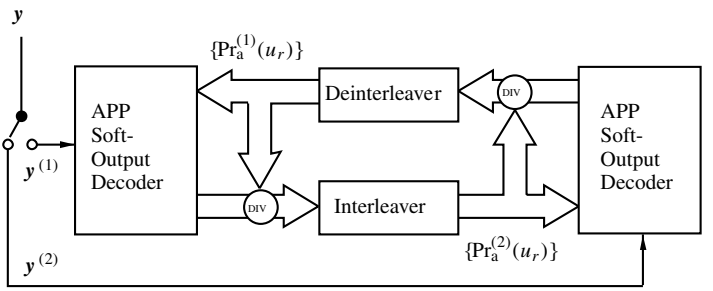


Figure 12.2 Block diagram of a turbo decoder with two constituent decoders.

channel probability $\Pr(y_r|x_r)$. This gives uniform weight to all symbols from the channel input, and it 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. As discussed in Section 7.7, the only place where the received symbol affects the APP decoder is in the calculation of the branch metric

$$\begin{aligned}\gamma_r(j,i) &= \Pr(s_{r+1}=j,\, y_r|s_r=i)\\ &= \underbrace{\Pr[s_{r+1}=j|s_r=i]}_{\text{transition probability}} \begin{cases} 1 & \text{if symbol is punctured}\\ \underbrace{\Pr(y_r|x_r)}_{\text{channel probability}} & \text{otherwise.} \end{cases} \end{aligned} \tag{7.30}$$

While the *transition probability* is simply the a priori probability $\Pr(u_r) = Pr_{\mathrm{a}}^{(i)}(u_r)$ 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 one if that symbol was punctured. 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; that is, they calculate $\Pr(u_r|\boldsymbol{y}^{(i)})$, $i = 1, 2$. From these output a posteriori probabilities, extrinsic probabilities $\Pr_{\mathrm{e}}^{(i)}(u_r)$ are calculated by removing the a priori input probabilities $\Pr_{\mathrm{a}}^{(i)}(u_r)$ by division, that is,

$$\Pr_{\mathrm{e}}^{(i)}(u_r) = \frac{1}{\alpha}\frac{\Pr(u_r|\boldsymbol{y}^{(i)})}{\Pr_{\mathrm{a}}^{(i)}(u_r)}, \quad \forall u_r; \qquad \alpha = \sum_u \frac{\Pr(u|\boldsymbol{y}^{(i)})}{\Pr_{\mathrm{a}}^{(i)}(u)}. \tag{12.1}$$

This is the DIV operation in Figure 12.2. The normalization factor α is used to make the extrinsic probabilities sum up to unity. The operation in (12.1) 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 priori are calculated as

$$\Pr_{\mathrm{a}}^{(1)}(u_r) = \sum_{v_r^{(1)}} p(y_r | \underbrace{u_r, v_r^{(1)}}_{x_r}); \tag{12.2}$$

that is, the unknown parity is simply averaged out.

Figure 12.3 shows some simulated performance curves, taken from refs. 23–25 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 it increases to $E_b/N_0 = 2.9$ dB if the channel symbols are restricted to 8-PSK. Thus the performance of this system comes very close to the Shannon limit even for the relatively small block length of $N = 5000$ symbols.

Other methods of parallel concatenation also exist; ref. 25, for example, discusses a 16-QAM TTCM system where two 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 four 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 in phase axis. The advantage is that no puncturing is required; the disadvantage,

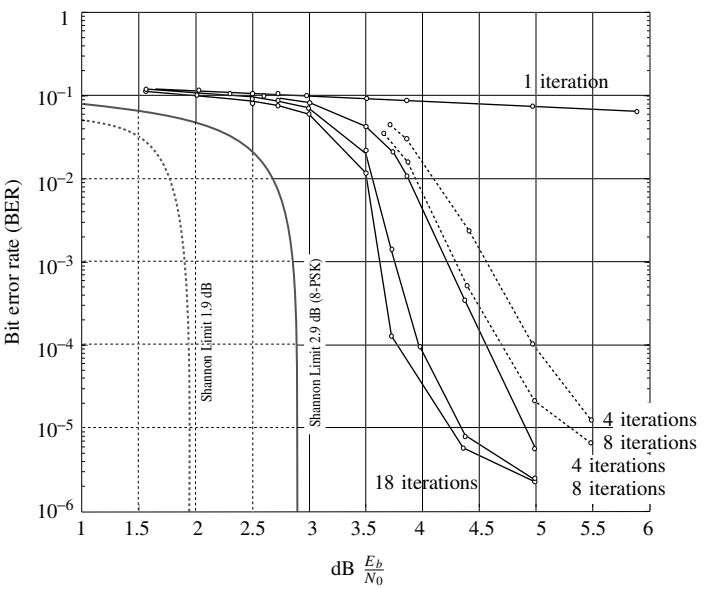


Figure 12.3 Simulations of a turbo-trellis-coded modulation system using two 8-PSK component trellis codes. Solid lines for $L = 5000$ symbols, dashed lines for $L = 1024$ symbols.

however, is the large signal set expansion required. Parallel concatenation has also been applied to space-time coding systems, but it appears that it is generally less elegant than serial concatenation, discussed in the following section.

12.3 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 12.4. A sequence of binary information symbols u_r are encoded in a regular binary encoder, whose outputs $\boldsymbol{v}_r$ 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 12.4 uses a rate $R = 2/3$ 8-PSK trellis encoder as the modulation encoder.

The decoder, too, follows the same principle as for binary serially concatenated codes and is shown in Figure 12.5. The first APP decoder for the modulation code operates as usual with two inputs, namely, the channel values $\boldsymbol{y}$ 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 (12.1).

These a posteriori symbol probabilities are converted to extrinsic probabilities and deinterleaved to line them up with the output symbols $v_r = (v_r^{(n-1)}, \ldots, v_r^{(0)})$ 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, 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. Alternately, the binary APP decoder may use LLR values on the bits v. With bit interleaving, the extrinsic *symbol* probabilities on v_r' from the inner decoder must be marginalized to extrinsic *bit* probabilities on $v_r'^{(l)}$ as

$$\Pr_e(v_r'^{(l)} = v) = \sum_{\substack{v_r'^{(l)}=v \\ (v_r'=\mathcal{V})}} \Pr_e(v_r') \tag{12.3}$$

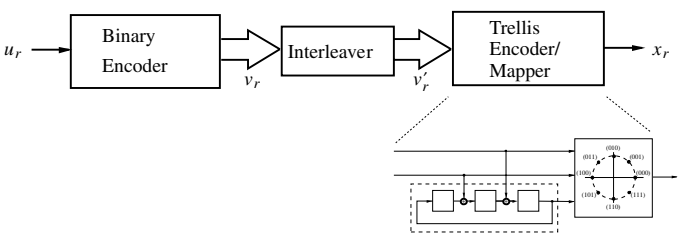


Figure 12.4 Block diagram of a serial turbo encoder for turbo-coded modulation.

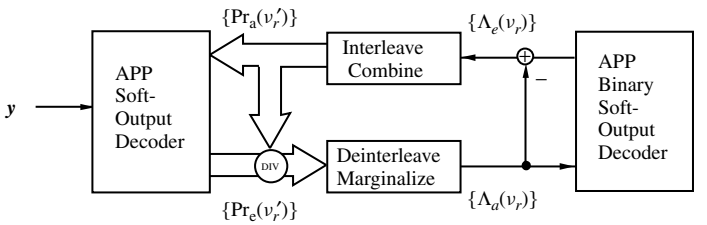


Figure 12.5 Block diagram of a turbo decoder with two constituent decoders.

where $\mathcal{V}$ is the symbol constellation alphabet of v'_r. From (12.3) the extrinsic LLRs $\Lambda_e(v_r^{'(l)})$ from the inner APP decoder are calculated in the usual manner:

$$\Lambda_e(v_r^{'(l)}) = \log\left(\frac{\Pr_{\mathrm{e}}(v_r^{'(l)} = 1)}{\Pr_{\mathrm{e}}(v_r^{'(l)} = 0)}\right). \tag{12.4}$$

The extrinsic LLRs, deinterleaved, become a priori LLRs for the binary APP decoder. Extrinsic LLRs $\Lambda_e(v_r^{(l)}$ are found by subtracting a priori LLRs from output APP LLRs. Interleaved, the extrinsic LLRs must be changed to symbol a priori values for the inner APP decoder's next iteration. First, bit probabilities are found as

$$\Pr_{\mathrm{a}}(v_r^{'(l)} = 1) = \frac{\exp\left(\Lambda_a(v_r^{'(l)})\right)}{1 + \exp\left(\Lambda_a(v_r^{'(l)})\right)}, \qquad \Pr_{\mathrm{a}}(v_r^{'(l)} = 0) = 1 - \Pr_{\mathrm{a}}(v_r^{'(l)} = 1). \tag{12.5}$$

The probability of a symbol is then simply the product of the probabilities of its constituent binary bits, that is,

$$\Pr_{\mathrm{a}}(v'_r) = \prod_{l=1}^{m} \Pr_{\mathrm{a}}(v_r^{'(l)}). \tag{12.6}$$

These symbol probabilities function as a priori information for the next iteration of the inner APP decoder.

12.4 EXIT ANALYSIS

Turbo-coded modulation systems can also be analyzed following the EXIT method presented in Chapters 10 and 11. 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 multilevel 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, namely, the received channel symbols $\boldsymbol{y}$, and a priori symbol probabilities of the transmitted symbols. Under the usual assumption of long random interleavers, the a priori probabilities $\Pr_{\mathrm{a}}(x_r)$ 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 $\boldsymbol{p}_{e,r} = [\Pr_{\mathrm{e}}(x_r = \boldsymbol{s}^{(1)}), \ldots, \Pr_{\mathrm{e}}(x_r = \boldsymbol{s}^{(M)})]$. Likewise, the input probabilities will be delivered to the decoder as an M-valued vector $\boldsymbol{p}_{a,r} = [\Pr_{\mathrm{a}}(x_r = \boldsymbol{s}^{(1)}), \ldots, \Pr_{\mathrm{a}}(x_r = \boldsymbol{s}^{(M)})]$ of input a priori symbol probabilities at each 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 $p_a(\boldsymbol{p}_a)$ and $p_e(\boldsymbol{p}_e)$. Due to the interleaver assumption, these random vectors $\boldsymbol{p}_e$ and $\boldsymbol{p}_a$ are independent of the time 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 signaling alphabet $\mathcal{S} = \{\boldsymbol{s}^{(1)}, \ldots, \boldsymbol{s}^{(M)}\}$ and whose outputs are the (conditional) probability vectors $\boldsymbol{p}_{e|\boldsymbol{s}^{(i)}}$, or $\boldsymbol{p}_{a|\boldsymbol{s}^{(i)}}$. 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 $X = \boldsymbol{s}^{(i)}$ and output $\boldsymbol{p}_{e|\boldsymbol{s}^{(i)}}$, denoted by $I(X, \boldsymbol{p}_e)$, and likewise for the a priori symbols we define the mutual information $I(X, \boldsymbol{p}_a)$ between the actual signals $\boldsymbol{s}^{(i)}$ and the probability distribution $\boldsymbol{p}_{a|\boldsymbol{s}^{(i)}}$. The information measures $I(X, \boldsymbol{p}_a)$ and $I(X, \boldsymbol{p}_e)$ will be used to quantify the reliability of the distributions $\boldsymbol{p}_a$ and $\boldsymbol{p}_e$—that is, their level of uncertainty.

While the a priori distributions $\boldsymbol{p}_{a,r}$ 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 an infinite number of ways to do this, and an intuitively correct way of generating them is by passing the transmitted symbols $\boldsymbol{x}$ 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 12.6. Both inputs, the channel symbols $\boldsymbol{y}$ 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 process.

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 nonlinear operation of the APP decoder, no analytical methods exist to find the multidimensional distribution $\boldsymbol{p}_e$, and also the

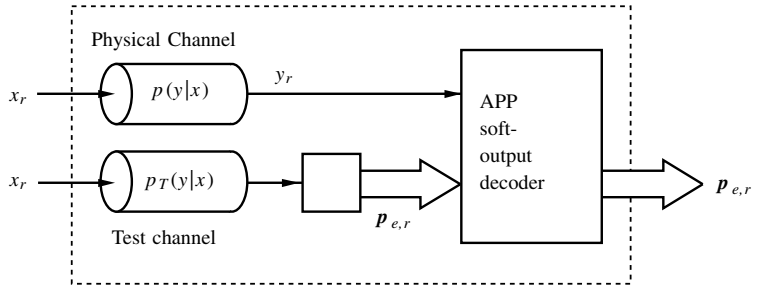


Figure 12.6 The modulation APP decoder as a transformer of probability distributions on the transmitted symbols X.

distribution $\boldsymbol{p}_a$ 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 $I(X, \boldsymbol{p}_a)$ versus $I(X, \boldsymbol{p}_e)$—that is, (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.

12.5 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, that is, $x_r = x_{r-1}v_r$, $v_r, x_r \in \mathcal{X}$. The process is initiated at the beginning of the transmission with a known symbol $x_0 = \boldsymbol{s}^{(i)}$. In the case of M-PSK, the multiplication is traditional complex multiplication, but other forms are also used. In Section 12.6 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

$$\hat{v}_r = \max_{v} \Re\left[v y_r^* y_{r-1}\right]. \tag{12.7}$$

However, we are more interested in using the differential code in conjunction with an outer code in a serially concatenated coding system. Figure 12.7 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 is passed through the interleaver, and it is 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

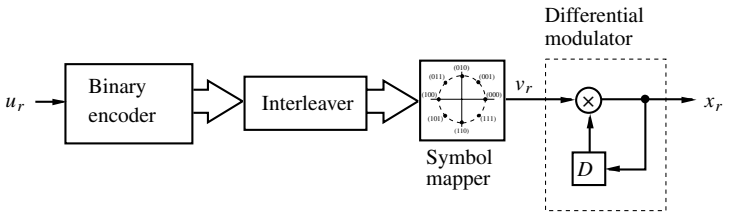


Figure 12.7 Block diagram of a serial turbo encoder for differential turbo-coded modulation.

Figure 12.7 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.

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 [3, 4] for this system can be devised according to the principles discussed in Chapter 11, and the decoder is identical to that in Figure 12.5. Such systems have been studied for the concatenation of convolutional codes with differential BPSK reported [10, 19], leading to very simple encoders and decoders with very good error performance results, rivaling those of more complex turbo codes.

Figure 12.8 shows the performance of a serially concatenated differential 8-PSK modulator with a [3, 2, 2] outer parity code [11, 12]. The bits of the outer parity code are bit interleaved, since symbol interleaving would not generate independent inputs within a parity-check code word. The block size used was 5000 symbols, corresponding to 10,000 information bits per block. As conjectured, the rapid drop-off indicates the threshold behavior typical of turbo-coded systems. Note, however, that this code suffers from a d_{free} problem and exhibits an error floor due to the low free distances of the [3, 2, 2] and the differential 8-PSK codes.

Using the analysis techniques discussed in Section 12.4, we can explain the error performance behavior exhibited in Figure 12.8. 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 12.9. The input mutual information $I_a = I(X, \boldsymbol{p}_a)$ is plotted on the horizontal axis and the output extrinsic mutual information $I(X, \boldsymbol{p}_e)$ 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 $E_b/N_o = 4.3$ dB, exactly where the turbo cliff occurs in the simulations.

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 it can

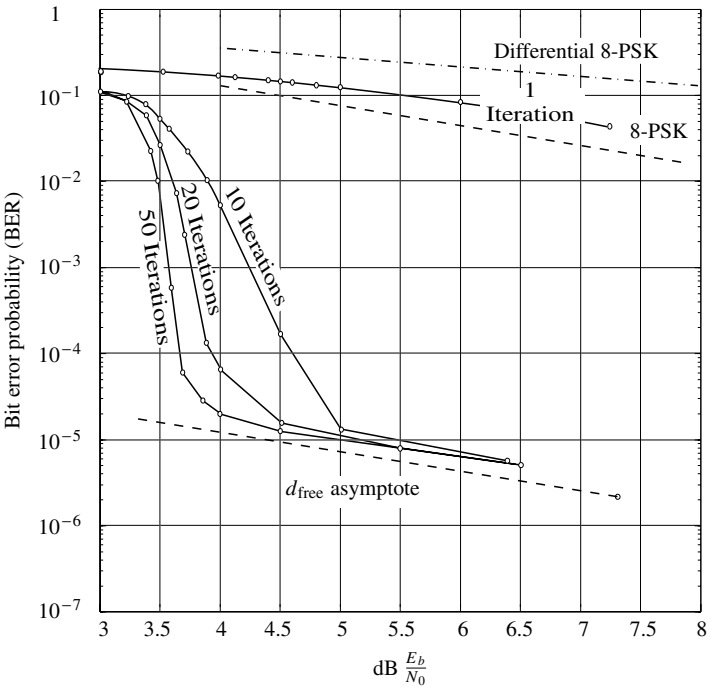


Figure 12.8 Performance of the serially concatentated differential 8-PSK system with an outer parity-check code with random interleavers, compared to regular and differential 8-PSK modulation.

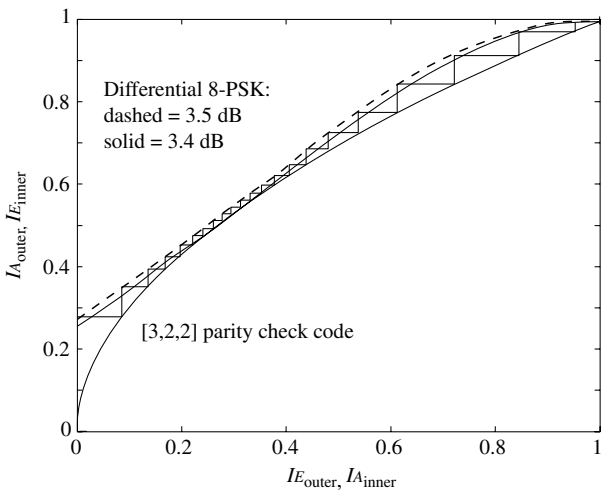


Figure 12.9 Extrinsic information transfer (EXIT) chart for serial concatenation of a [3, 2, 2] parity-check code with a differential 8-PSK code, improved 8-PSK mapping; decoding trajectory shown for SNR = 3.4 dB.

be implemented (for example) in a single ROM look-up table. From the parity-check constraint it is quite straightforward to calculate the output extrinsic bit log-likelihood ratio as

$$\Lambda_E(v_1) = \Lambda_A(v_2) + \log\left(\frac{1+\exp(\Lambda_A(v_3)-\Lambda_A(v_2))}{1+\exp(\Lambda_A(v_3)+\Lambda_A(v_2))}\right), \tag{12.8}$$

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, as detailed in Chapter 10. 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 $d^2_{\text{free}} = 0.586$, 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 12.1 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.

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 $d^2_{\text{min}} = 1.172$ with very high probability, corresponding to two length-2, minimum distance error events. The error floor in Figure 12.8 corresponds to this d^2_{min}.

This situation can be changed dramatically simply by choosing an alternate mapping, which maps triples of bits onto 8-PSK symbols according to the following sequence (000), (111), (001), (100), (010), (011), (110), (101), where

TABLE 12.1 Bit Weights that can be Mapped into Pairs of Diverging and Merging Symbols of the Differential 8-PSK Trellis

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

symbols are labeled successively in a counterclockwise fashion. The weights of length-2 detours are given in Table 12.2.

The only even-weight detour has a large distance, and all of the short-distance detours have odd weight, and thus are not allowed by the outer code. However, the 8PSK mapping is not regular (as shown in Chapter 3) and thus we must consider not only the all-zeros sequence but all sequences, recognizing that parallel branches of the differential trellis carry identical information symbols. If the correct 8PSK bit-mapped sequence is 010-011 with corresponding D8PSK symbols $e^{j\pi}$, $e^{j\pi/4}$ but is decoded as 011-010 with D8PSK symbols $e^{j5\pi/4}$, $e^{j\pi/4}$, the 2-symbol error sequence has Hamming distance 2 which is even and thus

TABLE 12.2 Bit Weights that can be Mapped into Pairs of Diverging and Merging Symbols of the Differential 8-PSK Trellis

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

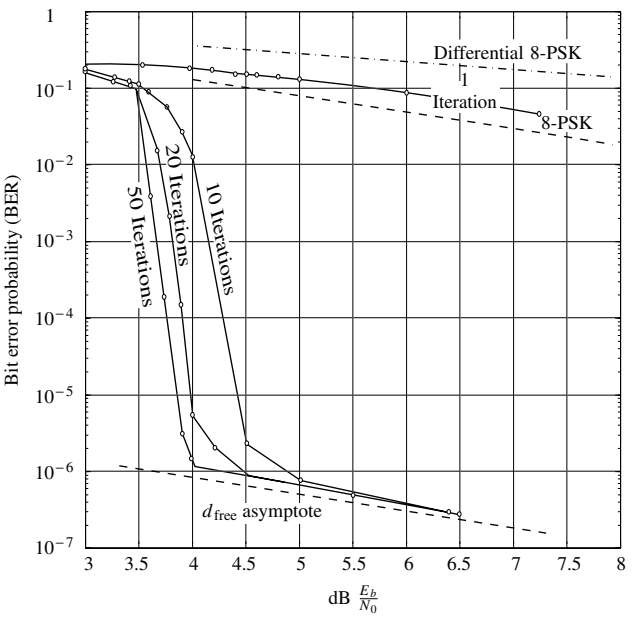


Figure 12.10 Performance of the serially concatentated differential 8-PSK system with alternate improved bit mapping and random interleavers.

allowed by the outer parity code, and $d^2 = 0.586$. The minimum distance is not increased, but the path multiplicities for this mapping are significantly reduced over those of natural mapping, resulting in a lower error floor. The performance of this mapping is shown in Fig. 12.10, which shows the lowered error floor and thus improved performance. This is accomplished at the cost of a small loss of about 0.2 dB in the turbo-cliff onset. For further details about the design and performance of these codes in channel without synchronization, the reader is referred to refs. 11 and 12, where the differential property of the inner code is exploited to iteratively acquire the correct channel phase.

12.6 CONCATENATED SPACE-TIME CODING

Space-time coding is a method to jointly code the signals transmitted from several antennas, and it has recently gained much popularity due to the potential of a large increase in channel capacity [31]. Under certain conditions, the channel capacity of such a multiple antenna channel can be increased by a factor as large as the minimum of the number of transmit and receive antennas used. 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 [6, 30, 31]. 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 the length of the space-time modulation symbol—that is, the number of columns making up the transmit matrix.

Transmission on a multiple antenna channel takes place via N_t transmit antennas and N_r receive antennas, creating $N_t N_r$ subchannels, where, typically, each subchannel h_{ji} from antenna i to antenna j is subject to independent signal fading, as illustrated and discussed in Figure 2.11. At time r, each transmit antenna $i = 1, \ldots, t$ 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 discussed in Chapter 2. Taken as a vector, $\boldsymbol{c}_r = c_{1r}, \ldots, c_{N_t r}$ is referred to as a *space-time symbol*. At each receive antenna $j = 1, \ldots, N_r$, the signal is passed through a filter matched to the pulse waveform and sampled synchronously. These samples are modeled by $y_{jr} = \sqrt{E_s/N_t} \sum_{i=1}^{N_t} h_{ji} c_{ir} + n_{jr}$, 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, $\boldsymbol{C} = [\boldsymbol{c}_1^t, \ldots, \boldsymbol{c}_{N_c}^t]$, where rows correspond to different transmit antennas and columns correspond to different times. Considering a sequence of L such STC transmissions $\boldsymbol{C}_1, \boldsymbol{C}_2, \ldots, \boldsymbol{C}_L$, the channel can be written as

$$\boldsymbol{Y}_r = \sqrt{\frac{E_s}{N_t}} \boldsymbol{H}_r \boldsymbol{C}_r + \boldsymbol{N}_r, \tag{12.9}$$

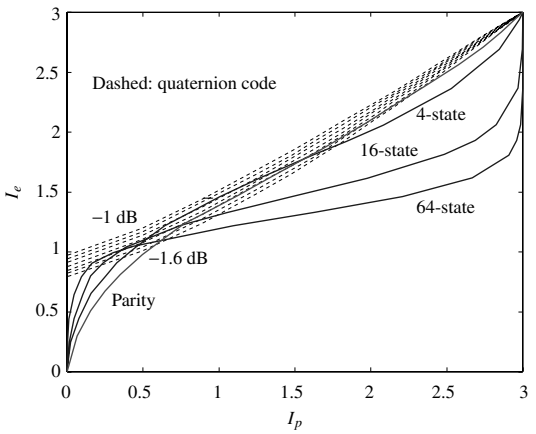


Figure 12.11 Extrinsic information transfer (EXIT) chart for serial concatenation of 4-, 16-, and 64-state convolutional codes and a [3, 2, 2] parity-check code with the differential space-time quaternion code.

where $\boldsymbol{Y}_r$ is the rth received STC, $\boldsymbol{H}_r$ is the $N_r \times N_t$ channel matrix of transmission path gains, and $\boldsymbol{N}_r$ is an $N_r \times N_c$ matrix of complex noise samples.

While we will not study fading channels in any detail (see refs. 14 and 20, or others, for a comprehensive treatment) multiple antenna communication usually is studied for fading channels, where each h_{ij} is modeled as an independent Rayleigh fading process by selecting the elements of $\boldsymbol{H}_r$ 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 $\boldsymbol{H}_r$ 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 $\boldsymbol{H}_r$ 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 [7, 27, 28], uses a differential STC presented by Hughes [13] 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 $\boldsymbol{C} = \boldsymbol{DG}$, where $\boldsymbol{D}$ is an $N_t \times N_c$ matrix and $\boldsymbol{G}$ belongs to a group of unitary $N_c \times N_c$ matrices ($\boldsymbol{GG}^* = \boldsymbol{I}$); that is, the product of any two matrices $\boldsymbol{G}_1\boldsymbol{G}_2$ results in another matrix $\boldsymbol{G}_3$ of the group. The N_c columns of $\boldsymbol{C}$ are transmitted as N_c consecutive space-time symbols.

A basic example of such a code, with $t = n = 2$, is the so-called quaternion code with symbols from the QPSK constellation. There are 8 STC words, given by

$$\mathcal{Q} = \left\{ \pm \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \pm \begin{pmatrix} j & 0 \\ 0 & -j \end{pmatrix}, \pm \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}, \pm \begin{pmatrix} 0 & j \\ j & 0 \end{pmatrix} \right\}, \tag{12.10}$$

$$\boldsymbol{D} = \begin{pmatrix} 1 & -1 \\ 1 & 1 \end{pmatrix}. \tag{12.11}$$

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 the differential PSK. At the start of transmission, the transmitter sends the STC word $\boldsymbol{C}_0 = \boldsymbol{D}$. Thereafter, messages are differentially encoded: To send $\boldsymbol{G}_r \in \mathcal{Q}$ during symbol time r, the transmitter sends

$$\boldsymbol{C}_r = \boldsymbol{C}_{r-1}\boldsymbol{G}_r. \tag{12.12}$$

The group property of the space-time code guarantees that $\boldsymbol{C}_r$ is a codeword if $\boldsymbol{C}_{r-1}$ is a codeword. Like differential PSK, a differential receiver for $\boldsymbol{C}_r$ exists based on the two most recent blocks. It computes [13]

$$\hat{\boldsymbol{G}} = \max_{\boldsymbol{G}\in\mathcal{Q}} \Re[\operatorname{tr} \boldsymbol{G}\boldsymbol{Y}_r^*\boldsymbol{Y}_{r-1}]. \tag{12.13}$$

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 ref. 28 contains a more detailed discussion of the use of these codes in conjunction with unknown channels.

Figure 12.11 shows the EXIT chart for the system discussed using the quaternion differential STC as the inner code ($I_a = I(X, \boldsymbol{p}_a)$ on the horizontal axis and $I(X, \boldsymbol{p}_e)$ 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 $I(X, \boldsymbol{p}_a)$ is plotted on the vertical axis). The [3, 2, 2] outer parity-check code from the last section is also used again. The axes extend from 0 to 3 bits of mutual information, where the maximum of 3 bits corresponds to complete knowledge of $\boldsymbol{s}^{(i)}$ derived from $\boldsymbol{p}_e$. In the case of bit interleaving, input and output bitwise 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.[1]

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 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 12.12 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

[1]The apparent violation of Shannon's absolute limit of $E_b/N_0 > -1.59$ dB stems from the fact that the signal-to-noise ratio in MIMO systems is measured per receive antenna.

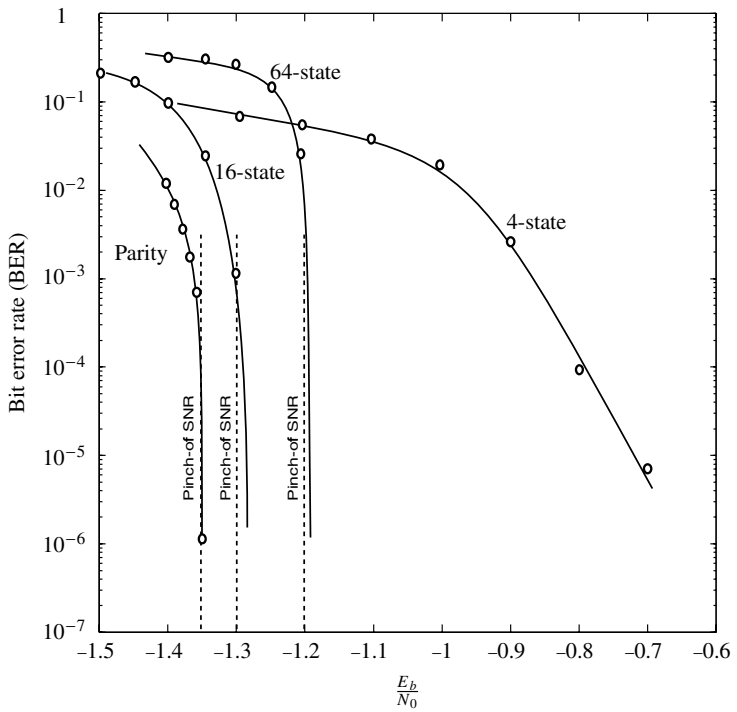


Figure 12.12 Performance of the serially concatentated system with several outer codes for a fast fading channel. (Note that the entire scale is only 1.4 dB.).

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 [31]. 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. This underlines the importance of using bit interleaving for weak outer codes.

It is quite astonishing to see such performance being achieved by the cooperation of two component codes that 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 a weak code, which also by itself will not impress with good performance. It is difficult to imagine that coding systems with lower complexity can be found that perform this well.

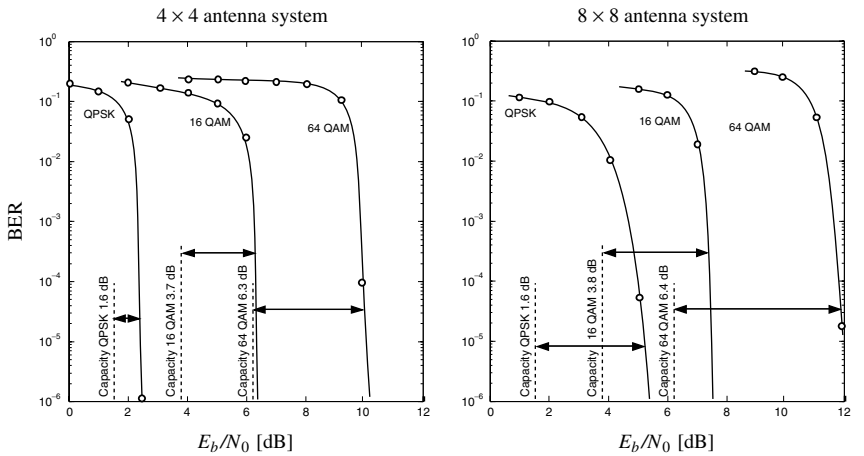


Figure 12.13 Performance of serially concatentated space-time coded systems with outer turbo codes for high spectral efficiencies.

Other combinations of codes have also been used successfully. Hochwald and ten Brink [9] 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 12.5; 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, they have used an approximate decoder instead. Nonetheless, excellent results can be achieved even for high-level modulation alphabets on each antenna. Figure 12.13 shows some of their results.

The simulation results shown in Figure 12.13 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 of $N = 18{,}432$ bits is thus quite large. 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, 16-QAM, and 64-QAM using 4 transmit and receive antennas, and twice these numbers for the 8×8 MIMO channel example.

A general trend becomes evident: It is more and more difficult to approach capacity with larger modulation alphabets. The 64-QAM, 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 using a differential space-time code, both for the error control coding (a complete parallel turbo code) and for the soft demodulator.

12.7 PRODUCT CODES AND BLOCK TURBO DECODING

Product codes represent yet another variant of the general turbo coding principle where two relatively minor 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 same problem. The structure of a product code is shown in Figure 12.14 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 $R = R_1 R_2$, but often the parity checks on parity checks (dashed box in Figure 12.14) 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 flare 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 $d_{\text{free}} = d_{\text{free,row}} d_{\text{free,col}}$. 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.

Product codes were introduced by Elias in 1954 [5] and have been recognized as powerful error control codes. They are well known (see, e.g., [17]), but decoding them to their potential using turbo coding principles had not been realized until the works of Hagenauer et al. [8] and Pyndiah [21], who both used proper probability message passing algorithms. The iterative decoder that achieves the code's full potential is again derived from the turbo decoding principle, and only minor changes are required. It is shown (again) in Figure 12.15.

Note that three types of symbols make up a coded block $\boldsymbol{y} = (\boldsymbol{y}_k, \boldsymbol{y}_{p_1}, \boldsymbol{y}_{p_2})$, where $\boldsymbol{y}_k$ are the received $k_1 k_2$ information bits that were transmitted uncoded,

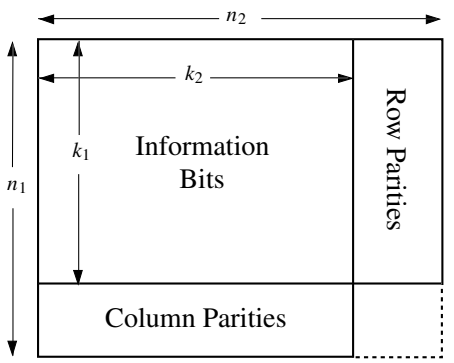


Figure 12.14 Structure of product codes.

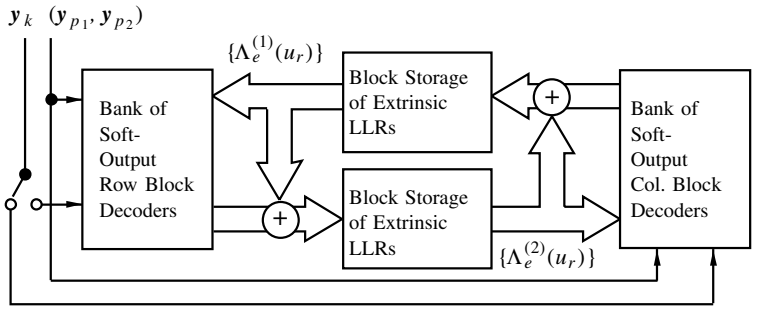


Figure 12.15 Block diagram of a block turbo decoder with banks of constituent soft block component decoders.

analogously to the systematic information bits in parallel concatenation. $\boldsymbol{y}_{p_1}$ is the $k_1(n_2 - k_2)$ block of received parity symbols for the row codes, and $\boldsymbol{y}_{p_2}$ is the $k_2(n_1 - k_1)$ block of parity symbols for the column codes. Each code in each 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 outputs of the decoders are a posteriori extrinsic information, usually in the form of LLRs, which are stored in block storage. 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 7 can certainly be used, but we will see that for several reasons, suboptimal approximate algorithms seem to be more popular due to their complexity advantage.

12.8 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

$$\Lambda_r = \log\left(\frac{\Pr(b_r = 1|\boldsymbol{y})}{\Pr(b_r = -1|\boldsymbol{y})}\right) \tag{12.14}$$

for each information bit b_r or coded binary symbols x_r. Clearly, an exhaustive way of doing this is by calculating the sums

$$\Pr(b_r = 1|\boldsymbol{y}) = \sum_{\boldsymbol{x} \in \mathcal{X}_r^{(+)}} \Pr(\boldsymbol{x}|\boldsymbol{y}), \tag{12.15}$$

$$\Pr(b_r = -1|\boldsymbol{y}) = \sum_{\boldsymbol{x} \in \mathcal{X}_r^{(-)}} \Pr(\boldsymbol{x}|\boldsymbol{y}), \tag{12.16}$$

where $\mathcal{X}_r^{(+)}$ is the set of codewords $\boldsymbol{x}$ corresponding to messages with $b_r = 1$, and $\mathcal{X}_r^{(-)}$ is the set corresponding to messages with $b_r = -1$.

In most cases 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 (12.14) that the decoder is expected to compute, after we substitute the conditional Gaussian probability density function $\Pr(\boldsymbol{y}|\boldsymbol{x}) = \frac{1}{(N_0\pi)^{n/2}} \exp\left(-\frac{|\boldsymbol{y}-\boldsymbol{x}|^2}{N_0}\right)$, giving

$$\Lambda_r = \log\left(\frac{\displaystyle\sum_{\boldsymbol{x}\in\mathcal{X}_r^{(+)}} \exp\left(-\frac{|\boldsymbol{y}-\boldsymbol{x}|^2}{N_0}\right)}{\displaystyle\sum_{\boldsymbol{x}\in\mathcal{X}_r^{(-)}} \exp\left(-\frac{|\boldsymbol{y}-\boldsymbol{x}|^2}{N_0}\right)}\right). \tag{12.17}$$

The problem is that there are no efficient ways of calculating the sum of a large number of exponentials.

If we restrict the sums to only a single term each, $\boldsymbol{x}_r^{(+)}$ and $\boldsymbol{x}_r^{(-)}$, we can approximate (12.17) by the much simpler expression

$$\Lambda_r \approx \frac{1}{N_0}\left(|\boldsymbol{y}-\boldsymbol{x}_r^{(-)}|^2 - |\boldsymbol{y}-\boldsymbol{x}_r^{(+)}|^2\right). \tag{12.18}$$

We need, however, to find the two terms $\boldsymbol{x}_r^{(+)}$ and $\boldsymbol{x}_r^{(-)}$ which generate the best approximation. This is accomplished by choosing $\boldsymbol{x}_r^{(+)}$ as the codeword at minimum distance to $\boldsymbol{y}$ with $b_r = 1$, and $\boldsymbol{x}_r^{(-)}$ as the codeword at minimum distance to $\boldsymbol{y}$ with $b_r = -1$.

Furthermore, (12.18) can rewritten as

$$\Lambda_r \approx \frac{2}{N_0}\left(y_r + \underbrace{\sum_{\substack{l=1\\(l\neq r)}}^{n_1(n_2)} y_l x_l^{(+)} p_l}_{\Lambda_{e,r}:\text{extrinsic information}}\right) \qquad p_l = \begin{cases} 0; & x_l^{(+)} = x_l^{(-)} \\ 1; & x_l^{(+)} \neq x_l^{(-)}. \end{cases} \tag{12.19}$$

An iterative update equation can now be constructed by equating the LLR ratio above with the soft channel input value, since both represent LLR ratios. That is, for following decoding cycles:

$$y_r^{(i)} \leftarrow y_r^{(i-1)} + \alpha_{i-1}\Lambda_{e,r}^{(i-1)}, \qquad y_r^{(0)} = y_r. \tag{12.20}$$

Since the component decoders which are used cannot make direct use of a priori information, the above LLR update equation represents a reasonable alternative.

If an APP decoder which properly uses a priori information is used, the extrinsic update algorithm discussed in Chapters 10 and 11 should be used.

The central question is, of course, how to find the two codewords $\boldsymbol{x}_r^{(-)}$ and $\boldsymbol{x}_r^{(-)}$, and many ad hoc techniques have been explored, and among them we find the *Chase Algorithm*. This algorithm is conceptually very simple, and exploits the fact that the noise probability decays exponentially with the (vector) magnitude of the noise. Succinctly stated, the algorithm restricts the search to a sphere of candidate codewords around the received vector $\boldsymbol{y}$ and draws the two codewords $\boldsymbol{x}_r^{(-)}$ and $\boldsymbol{x}_r^{(-)}$ from this reduced set, which is significantly less complex to search. It does this by performing hard-decision decoding on the received $\boldsymbol{y}$, then a new received vector is generated from $\boldsymbol{y}$ by flipping the least reliable symbol to its opposite sign and repeating the decoding. Alternate flipping of low reliable symbols is continued until a fixed number of vectors have been decoded into candidate codewords $\boldsymbol{x}$, from which the two sets $\mathcal{X}_r^{(+)}$ and $\mathcal{X}_r^{(-)}$ are generated. However, there are a number of difficulties associated with this algorithm:

(a) The algorithm is ad hoc, and it is difficult to evaluate the performance/complexity tradeoff without extensive simulations.

(b) There is a chance that no codeword $\boldsymbol{x}_r^{(-)}$ (or, alternatively, no codeword $\boldsymbol{x}_r^{(+)}$) 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 $\boldsymbol{\alpha} = [\alpha_0, \ldots, \alpha_I]$ 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 12.16 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 ref. 21, and the parameters were chosen as $\boldsymbol{\alpha} = [0, 0.2, 0.3, 0.5, 0.7, 0.9]$, $\boldsymbol{\beta} = [0.2, 0.4, 0.6, 0.8, 1, 1]$. Decoding via Chase algorithm used 16 test patterns, changing the four least reliable bits.

Advanced Hardware Architectures, Inc., Pullman, WA, have built a series of VLSI decoders based on the concept of product codes. Their codes use extended Hamming codes as components and use both two- and three-dimensional arrays. The extension of the decoding algorithm to 3 dimensions is straightforward. Table 12.3 shows the code combinations and rates in their product line [1]. All codes used are extended Hamming codes with the exception of the [4.3] parity code.

Figure 12.17 shows simulation results of a $[\text{Hamming}(64,57,4)]^2$ 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 $R = 0.790$.

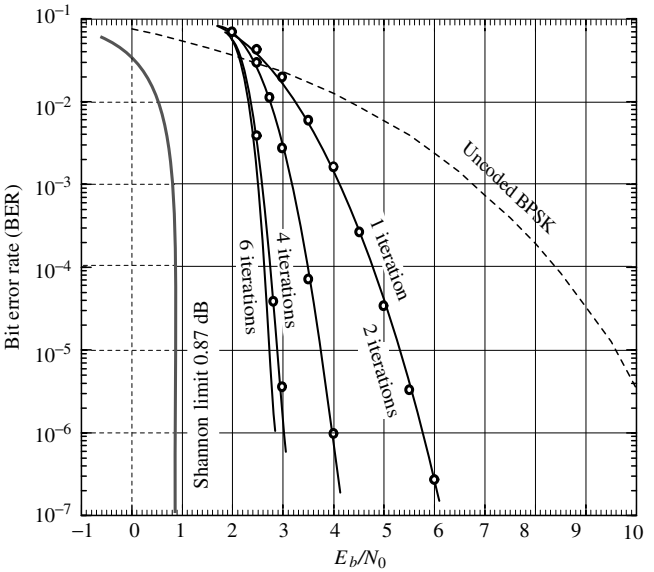


Figure 12.16 Simulations of a $[\text{BCH}(64,51,6)]^2$ product code using iterative decoding with Chase-algorithm based approximations to the APP algorithm.

TABLE 12.3 Table of Code Combinations Offered by AHA Corporation [1]

Code Combinations	N	K	R
$(16, 11) \times (16, 11)$	256	121	0.4727
$(32, 26) \times (16, 11)$	512	286	0.5586
$(16, 11) \times (16, 11) \times (4, 3)$	1024	363	0.3545
$(32, 26) \times (32, 26)$	1024	676	0.6602
$(64, 57) \times (8, 4) \times (4, 3)$	2048	684	0.3340
$(32, 26) \times (16, 11) \times (4, 3)$	2048	858	0.4189
$(32, 26) \times (64, 57)$	2048	1482	0.7236
$(32, 26) \times (16, 11) \times (8, 3)$	4096	1144	0.2793
$(16, 11) \times (16, 11) \times (16, 11)$	4096	1331	0.3250
$(32, 26) \times (32, 26) \times (4, 3)$	4096	2028	0.4951
$(64, 57) \times (64, 57)$	4096	3249	0.7932

12.9 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.

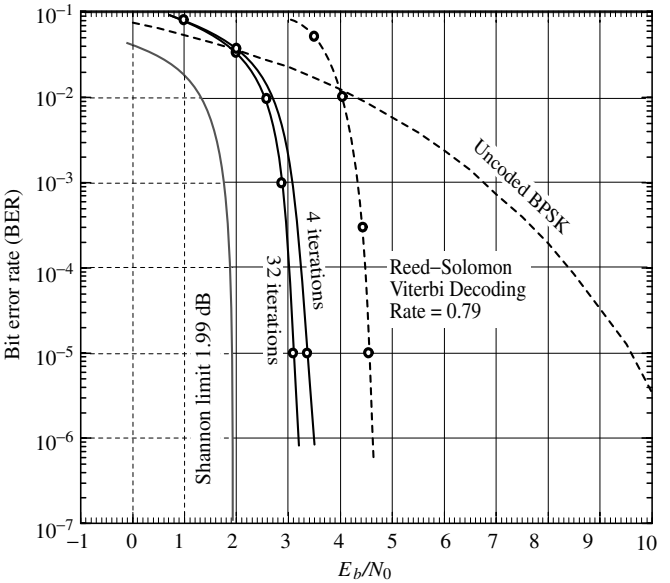


Figure 12.17 Simulations of a $[\text{EHC}(64, 57, 4)]^2$ product code using iterative decoding with AHA's approximation to the APP algorithm. This code has been standardized for the IEEE 802.16 MAN standard.

The only significant change with respect to 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

$$\Pr(b_r = b|y_r) = \sum_{\substack{x_r \in \mathcal{X} \\ (b_r = b)}} \Pr(x_r = x|y_r), \tag{12.21}$$

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

$$\Lambda_r = \log \left(\frac{\displaystyle\sum_{\substack{x_r \in \mathcal{X} \\ (b_r = 1)}} \Pr(x_r = x|y_r)}{\displaystyle\sum_{\substack{x_r \in \mathcal{X} \\ (b_r = 0)}} \Pr(x_r = x|y_r)} \right). \tag{12.22}$$

Note that this marginalization operation has to be performed only once before iterative decoding begins.

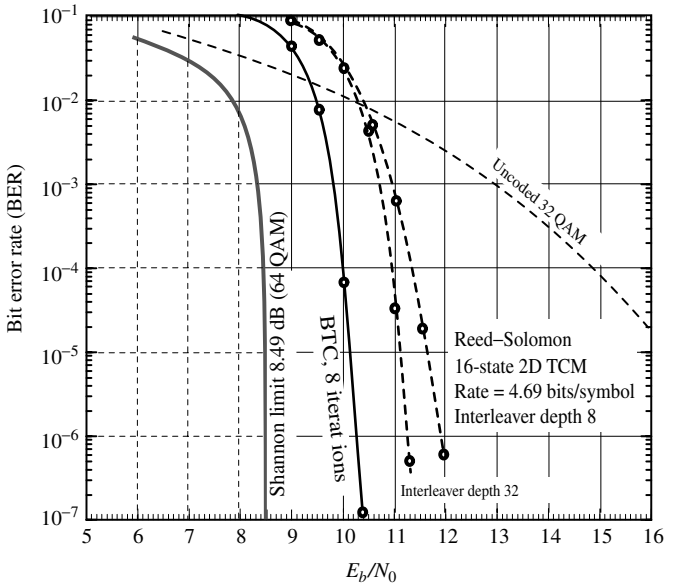


Figure 12.18 Simulations of a $[\text{EHC}(64, 57, 4)]^2$ product code mapped into 64-QAM constellation symbols to increase spectral efficiency. 32 QAM is used as comparison since it provides roughly the same spectral efficiency.

Figure 12.18 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 64-QAM 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.

12.10 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 12.17).

The codes are shortened in a very straightforward way by deleting rows and columns until the required block size is achieved. Deleting rows or column reduces the number of bits that are transmitted. At the decoder these bits are then simply assumed to be zero, and perfectly known. Decoding otherwise continues in the usual fashion. This shortening affects the rate of the code, but has no other major effect; in particular, the minimum distance and 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, and mobile communications applications. Future wire-line applications such as for ADSL are also being investigated.

BIBLIOGRAPHY

1. Advanced Hardware Architecture, "AHA Galaxy core generator product brief," *http://www.aha.com/Publications/pbgalaxy_1099.pdf* .
2. I. Bahceci and T. M. Duman, "Combined turbo coding and unitary space-time modulation," *IEEE Trans. Commun.*, vol. 50, no. 8, Aug. 2002.
3. 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.
4. C. Berrou, A. Glavieux, "Near optimum error correcting coding and decoding: Turbo-codes," *IEEE Trans. Commun.*, vol. 44, no. 10, pp. 1261–1271, Oct. 1996.
5. P. Elias, "Error-free coding," *IEEE Trans. Inform. Theory,* vol. IT-4, pp. 29–37, Sept. 1954.
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. no. 1, 2, pp. 41–59, Aug. 1996.
7. A. Grant and C. Schlegel, "Differential turbo space-time coding," *Proceedings, IEEE Information Theory Workshop, 2001*, pp. 120–122, 2001.
8. 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.
9. B. M. Hochwald and S. ten Brink, "Achieving near-capacity on a multiple-antenna channel," private communication, 2001.
10. 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.
11. 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
12. S. Howard and C. Schlegel, "Differential turbo coded modulation with APP channel estimation," *IEEE Trans. Commun.*, 2003, submitted.

13. B. Hughes, "Differential space-time modulation," *IEEE Trans. Inform. Theory*, vol. 46, no. 7, pp. 2567–2578, Nov. 2000.

14. W. C. Jakes, *Microwave Mobile Communications*, John Wiley & Sons, New York, 1994.

15. H. Nickl, J. Hagenauer, and F. Burkert, "Approaching Shannon's capacity limit by 0.27 dB using simple Hamming codes," *IEEE Commun. Lett.*, vol. 1, no. 5, Sept. 1997.

16. S. LeGoff and F. O. Al-Ayyan, "Design of bit-interleaved turbo-coded modulations," *Electron. Lett.*, vol. 37, no. 16, pp. 1030–1031, Aug. 2001.

17. S. Lin and D. J. Costello, Jr., *Error Control Coding*, Prentice-Hall, Engelwood Cliffs, NJ, 1983.

18. X. Li and J. A. Ritcey, "Bit Interleaved Coded Modulation with Iterative Decoding," *IEEE Commun. Lett.*, vol. 1, no. 6, pp. 169–171, Nov. 1997.

19. 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.

20. J. Proakis, *Digital Communications*, 4th edition, Prentice-Hall, Englewood Cliffs, NJ, 1995.

21. R. M. Pyndiah, "Near-optimum decoding of product codes: Block turbo codes," *IEEE Trans. Commun.,* vol. 46, no. 8, pp. 1003–1010, Aug. 1998.

22. R. M. Pyndiah, and A. Picart, and A. Glavieux, "Performance of block turbo coded 16-QAM and 64-QAM modulations," *IEEE Global Telecommunications Conference GLOBECOM '95,* vol. 2, pp. 1039–1043, 1995.

23. R. Robertson and T. Wörz, "Coded modulation scheme employing turbo codes," *Electron. Lett.*, vol. 31, no. 18, pp. 1546–1547, Aug. 1995.

24. R. Robertson and T. Wörz, "A novel bandwidth efficient coding scheme employing turbo codes," *Proceedings IEEE International Conference on Communication ICC'96*, vol. 2, pp. 962–967, 1996.

25. R. Robertson and T. Wörz, "Extensions of turbo trellis coded modulation to high bandwidth efficiencies," *Proceedings IEEE International Conference on Communication*, vol. 3, pp. 1251–1255, 1997.

26. 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.

27. C. Schlegel and A. Grant, "Concatenated space-time coding," *Proc. Symposium on Personal, Indoor and Mobile Radio Communications*, PIMRC 2001, pp. 139–143, Sept. 30–Oct. 3, San Diego, CA.

28. C. Schlegel and A. Grant, "Differential space-time turbo codes," *IEEE Trans. Inform. Theory.*, vol. 49, no. 9, pp. 2298–2306, Sept. 2003.

29. A. Stefanov and T. M. Duman, "Turbo-coded modulation for systems with transmit and receive antenna diversity over block fading channels: System model, decoding approaches, and practical considerations," *IEEE J. Select. Areas Commun.,* vol. 19, no. 5, pp. 958–968, May 2001.

30. 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.

31. I. E. Telatar, "Capacity of multi-antenna Gaussian channels," *Eur. Trans. Telecom.,* vol. 10, pp. 585–595, Nov. 1999.

32. B. E. Wahlen and C. Y. Mai, "Turbo coding applied to pragmatic trellis-coded modulation," *IEEE Commun. Lett.*, vol. 4, no. 2, pp. 65–67, Feb. 2000.

Index

Trellis and Turbo Coding, by Christian B. Schlegel and Lance C. Pérez
ISBN 0-471-22755-2