SECOND EDITION

Fundamentals of Convolutional Coding

ROLF JOHANNESSON
KAMIL Sh. ZIGANGIROV

IEEE PRESS

IEEE SERIES ON DIGITAL & MOBILE COMMUNICATION

John B. Anderson, *Series Editor*

WILEY

FUNDAMENTALS OF CONVOLUTIONAL CODING

IEEE Press
445 Hoes Lane
Piscataway, NJ 08854

Technical Reviewer
John Proakis
Northeastern University

FUNDAMENTALS OF CONVOLUTIONAL CODING

Second Edition

ROLF JOHANNESSON
KAMIL Sh. ZIGANGIROV

IEEE Press

WILEY

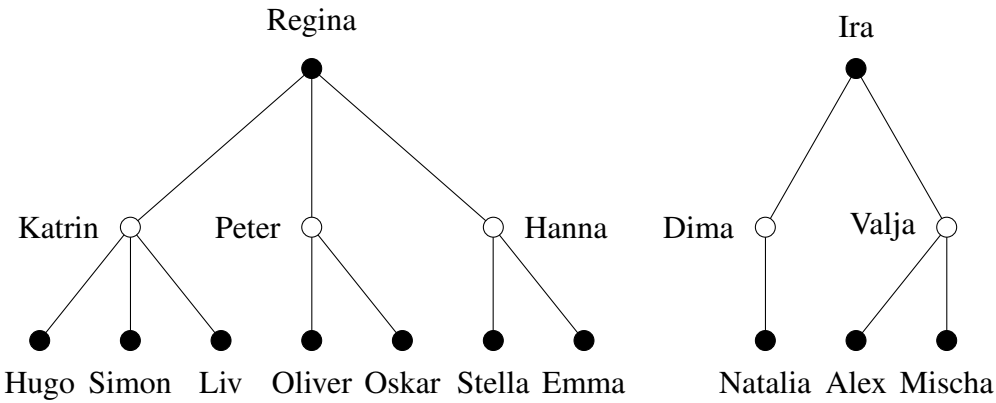
Library of Congress Cataloging-in-Publication is available.

ISBN 978-0-470-27683-9

Printed in the United States of America.

To Jim Massey, 1934 - 2013
our friend and mentor

To our clans[1]



[1] *See* Figure 8.2.

CONTENTS

PREFACE

Our goal with this book is to present a comprehensive treatment of convolutional codes, their construction, their properties, and their performance. The purpose is that the book could serve as a graduate-level textbook, be a resource for researchers in academia, and be of interest to industry researchers and designers.

This book project started in 1989 and the first edition was published in 1999. The work on the second edition began in 2009. By now the material presented here has been maturing in our minds for more than 40 years, which is close to our entire academic lives. We believed that the appearance of some of David Forney's important structural results on convolutional encoders in a textbook was long overdue. For us, that and similar thoughts on other areas generated new research problems. Such interplays between research and teaching were delightful experiences. This second edition is the final result of those experiences.

Chapter 1 provides an overview of the essentials of coding theory. Capacity limits and potential coding gains, classical block codes, convolutional codes, Viterbi decoding, and codes on graphs are introduced. In Chapter 2, we give formal definitions of convolutional codes and convolutional encoders. Various concepts of minimality are discussed in-depth using illuminative examples. Chapter 3 is devoted to a flurry of distances of convolutional codes. Time-varying convolutional codes are introduced

and upper and lower distance bounds are derived. An in-depth treatment of Viterbi decoding is given in Chapter 4, including both error bounds and tighter error bounds for time-invariant convolutional codes as well as a closed-form expression for the exact bit error probability. Both the BCJR (Bahl-Cocke-Jelinek-Raviv) and the one-way algorithms for *a posteriori* probability decoding are discussed. A simple upper bound on the bit error probability for extremely noisy channels explains why it is important that the constituent convolutional encoders are systematic when iterative decoding is considered. The chapter is concluded by a treatment of tailbiting codes, including their BEAST (Bidirectional Efficient Algorithm for Searching Trees) decoding. In Chapter 5, we derive various random ensemble bounds for the decoding error probability. As an application we consider quantization of channel outputs. Chapter 6 is devoted to list decoding of convolutional codes, which is thoroughly analyzed. Once again we make the important conclusion that there are situations when it is important that a convolutional encoder is systematic. In Chapter 7, we discuss a subject that is close to our hearts, namely sequential decoding. Both our theses were on that subject. We describe and analyze the stack algorithm, the Fano algorithm, and Creeper. James Massey regarded the Fano algorithm as being the most clever algorithm among all algorithms! Chapters 8 and 9 rectify the lack of a proper treatment of low-density parity-check (LDPC) codes and turbo codes in the first edition, where these important areas got a too modest section. These codes revolutionized the world of coding theory at the end of the previous millennium. In Chapter 8, the LDPC block codes, which were invented by Robert Gallager and appeared in his thesis, are discussed. Then they are generalized to LDPC convolutional codes, which were invented by the second author and his graduate student Alberto Jiménez-Feltström. They are discussed in-depth together with bounds on their distances. Iterative decoding is introduced and iterative limits and thresholds are derived. The chapter is concluded by the introduction of the related braided block codes. Turbo codes are treated in Chapter 9 together with bounds on their distances and iterative decoding. Moreover, the braided block codes are generalized to their convolutional counterpart. In Chapter 10, we introduce two efficient algorithms, FAST (Fast Algorithm for Searching Trees) and BEAST, for determining code distances. FAST was designed to determine the Viterbi spectrum for convolutional encoders while using BEAST we can determine the spectral components for block codes as well. Extensive lists are given of the best known code generators with respect to free distance, numbers of nearest neighbors and of information bit errors, Viterbi spectrum, distance profile, and minimum distance. These lists contain both nonsystematic and systematic generators. In Appendix A we demonstrate how to minimize two examples of convolutional encoders and in Appendix B we present Wald's identity and related results that are necessary for our analyses in Chapters 3–7.

For simplicity's sake, we restricted ourselves to binary convolutional codes. In most of our derivations of the various bounds we only considered the binary symmetric channel (BSC). Although inferior from a practical communications point of view, we believe that its pedagogical advantages outweigh that disadvantage.

Each chapter ends with some comments, mainly historical in nature, and sets of problems that are highly recommended. Many of those were used by us as exam questions. Note that sections marked with asterisk ($*$) can be skipped at the first reading without loss of continuity.

There are various ways to organize the material into an introductory course on convolutional coding. Chapter 1 should always be read first. Then one possibility is to cover the following sections, skipping most of the proofs found there: 2.1–2.7, 2.10, 2.13, 3.1, 3.5, 3.10, 3.11, 4.1–4.2, 4.5, 4.7–4.11, 5.6, 6.1–6.2, 7.1–7.3, 7.5, 7.10, 8.1–8.6, 8.8, 9.1–9.4, 9.6, and perhaps also 10.1, 10.2, and 10.7. With our younger students (seniors), we emphasize explanations, and discussions of algorithms and assign a good deal of the problems found at the end of the chapters. With the graduate students we stress proving theorems, because a good understanding of the proofs is an asset in advanced engineering work.

Finally, we do hope that some of our passion for convolutional coding has worked its way into these pages.

Rolf Johannesson

Kamil Sh. Zigangirov

ACKNOWLEDGEMENT

Rolf is particularly grateful to Göran Einarsson, who more than 40 years ago not only suggested convolutional codes as Rolf's thesis topic but also recommended that he spend a year of his graduate studies with James Massey at the University of Notre Dame. This year was the beginning of a lifelong devotion to convolutional codes and a lifelong genuine friendship which lasted until Jim passed away in 2013. The influence of Jim cannot be overestimated. Rolf would also like to acknowledge David Forney's outstanding contributions to the field of convolutional codes; without his work convolutional codes would have been much less exciting.

Kamil would like to thank his colleagues at the Institute for Problems of Information Transmission in Moscow. In particular, he would like to mention the "daytime" seminars organized by the late Roland Dobrushin and the late Mark Pinsker and the "evening" seminars organized by Leonid Bassalygo. During these seminars, Kamil had the opportunity to use the participants as guinea pigs when he wanted to test many of the fundamental ideas presented in this book. Special thanks go to Mark Pinsker and Victor Zyablov. Mark introduced Kamil to convolutional codes, and he inspired Kamil for more than 30 years. Over the years, Kamil has also benefited from numerous discussions with Victor.

In our research we have benefited considerably from our cooperation—sometimes long lasting, sometimes a single joint paper—with colleagues John Anderson, Irina Bocharova, Daniel Costello, David Forney, Boris Kudryashov, James Massey, Zhe-Xian Wan, and Victor Zyablov.

Needless to say, we would not have reached our view of convolutional codes without our work with former graduate students: Gunilla Bratt, Mats Cedervall, Karin Engdahl, Marc Handlery, Florian Hug, Stefan Höst, Alberto Jiménez-Feltström, Michael Lentmaier, Per Ljungberg, Maja Lončar, Johan Nyström, Harro Osthoff, Victor Pavlushkov, Joakim Persson, Ben Smeets, Per Ståhl, Dmitry Trukhachev, Kristian Wahlgren, Ola Winzell, and Emma Wittenmark, who have all contributed in many ways. They inspired us, produced research results, did simulations, provided us with various figures, suggested improvements, saved us from many blunders, and much more. We are grateful to Florian Hug, Alberto Jiménez-Feltström, Michael Lentmaier, Maja Lončar, Johan Nyström, Harro Osthoff, and Dmitry Trukhachev for permitting us to use pieces of LaTeX source code taken from their theses. We are greatly indebted to Lena Månsson, who with great enthusiasm and outstanding skill typed the first edition and partly the second edition and above all to LaTeX-wizard Florian Hug who volunteered to finalize the second edition. Without Florian's editing and composition expertise this edition would not have been completed. Thanks Florian!

Special thanks go to Mary Hatcher, Danielle LaCourciere, and Brady Chin at Wiley for their support. We are grateful to Beatrice Ruberto who did an outstanding job and saved us from making embarrassing blunders. In particular, we appreciate that Mary turned out to have the patience of an angel when Rolf numerous times failed to meet deadlines.

CHAPTER 1

INTRODUCTION

1.1 WHY ERROR CONTROL?

The fundamental idea of *information theory* is that all communication is essentially digital—it is equivalent to generating, transmitting, and receiving randomly chosen *bi* nary digi*ts*, *bits*. When these bits are transmitted over a communication channel—or stored in a memory—it is likely that some of them will be corrupted by noise. In his 1948 landmark paper "A Mathematical Theory of Communication" [Sha48] Claude E. Shannon recognized that randomly chosen binary digits could (and should) be used for *measuring* the generation, transmission, and reception of *information*. Moreover, he showed that the problem of communicating information from a source over a channel to a destination can always be separated—without sacrificing optimality—into the following two subproblems: representing the source output efficiently as a sequence of binary digits (*source coding*) and transmitting binary, random, independent digits over the channel (*channel coding*). In Fig. 1.1 we show a general digital communication system. We use Shannon's *separation principle* and split the encoder and decoder into two parts each as shown in Fig. 1.2. The channel coding parts can be designed independently of the source coding parts, which simplifies the use of the same communication channel for different sources.

Fundamentals of Convolutional Coding, 2. Edition. By Rolf Johannesson, Kamil Sh. Zigangirov

To a computer specialist, "bit" and "binary digit" are entirely synonymous. In information theory, however, "bit" is Shannon's unit of information [Sha48, Mas82]. For Shannon, *information* is what we receive when uncertainty is reduced. We get exactly 1 bit of information from a binary digit when it is drawn in an experiment in which successive outcomes are independent of each other and both possible values, 0 and 1, are equiprobable; otherwise, the information is less than 1. In the sequel, the intended meaning of "bit" should be clear from the context.

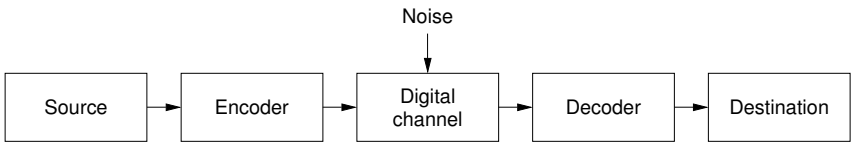


Figure 1.1 Overview of a digital communication system.

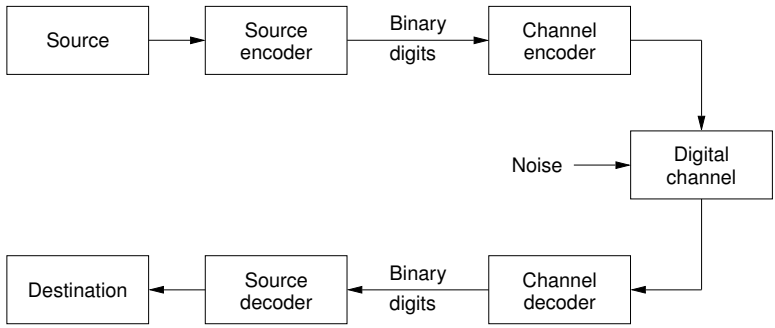


Figure 1.2 A digital communication system with separate source and channel coding.

Shannon's celebrated channel coding theorem states that every communication channel is characterized by a single parameter C_t, the *channel capacity*, such that R_t randomly chosen bits per second can be transmitted arbitrarily reliably over the channel if and only if $R_t \leq C_t$. We call R_t the *data transmission rate*. Both C_t and R_t are measured in bits per second. Shannon showed that the specific value of the *signal-to-noise ratio* is not significant as long as it is large enough, that is, so large that $R_t \leq C_t$ holds; what matters is how the information bits are encoded. The information should not be transmitted one information bit at a time, but long information sequences should be *encoded* such that each information bit has some influence on many of the bits transmitted over the channel. This radically new idea gave birth to the subject of *coding theory*.

Error control coding should protect digital data against errors that occur during transmission over a noisy communication channel or during storage in an unreliable memory. The last decades have been characterized not only by an exceptional increase in data transmission and storage but also by a rapid development in micro-electronics,

providing us with both a need for and the possibility of implementing sophisticated algorithms for error control.

Before we study the advantages of coding, we shall consider the digital communication channel in more detail. At a fundamental level, a channel is often an analog channel that transfers waveforms (Fig. 1.3). Digital data $u_0u_1u_2\ldots$, where $u_i \in \{0, 1\}$, must be *modulated* into waveforms to be sent over the channel.

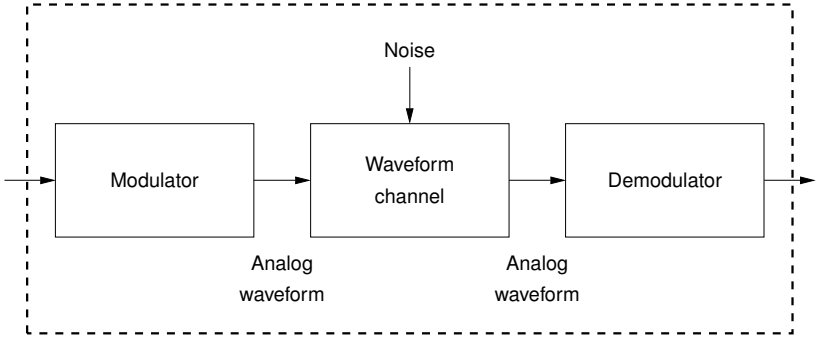


Figure 1.3 A decomposition of a digital communication channel.

In communication systems where carrier phase tracking is possible (coherent demodulation), *phase-shift keying* (PSK) is often used. Although many other modulation systems are in use, PSK systems are very common and we will use one of them to illustrate how modulations generally behave. In binary PSK (BPSK), the modulator generates the waveform

$$s_1(t) = \begin{cases} \sqrt{\frac{2E_s}{T}} \cos \omega t, & 0 \le t < T \\ 0, & \text{otherwise} \end{cases} \tag{1.1}$$

for the input 1 and $s_0(t) = -s_1(t)$ for the input 0. This is an example of *antipodal signaling*. Each symbol has duration T seconds and energy $E_s = ST$, where S is the power and $\omega = \frac{2\pi}{T}$. The transmitted waveform is

$$v(t) = \sum_{i=0}^{\infty} s_{u_i}(t - iT) \tag{1.2}$$

Assume that we have a waveform channel such that additive white Gaussian noise (AWGN) $n(t)$ with zero mean and two-sided power spectral density $N_0/2$ is added to the transmitted waveform $v(t)$, that is, the received waveform $r(t)$ is given by

$$r(t) = v(t) + n(t) \tag{1.3}$$

where

$$E[n(t)] = 0 \tag{1.4}$$

and

$$E[n(t+\tau)n(t)] = \frac{N_0}{2}\delta(\tau) \tag{1.5}$$

where $E[\cdot]$ and $\delta(\cdot)$ denote the mathematical expectation and the delta function, respectively.

Based on the received waveform during a signaling interval, the demodulator produces an estimate of the transmitted symbol. The optimum receiver is a *matched filter* with impulse response

$$h(t) = \begin{cases} \sqrt{2/T}\cos\omega t, & 0 \le t < T \\ 0, & \text{else} \end{cases} \tag{1.6}$$

which is sampled each T seconds (Fig. 1.4). The matched filter output Z_i at the sample time iT,

$$Z_i = \int_{(i-1)T}^{iT} r(\tau)h(iT-\tau)d\tau \tag{1.7}$$

is a Gaussian random variable $N(\mu, \sigma^2)$ with mean

$$\mu = \pm\int_0^T \left(\sqrt{\frac{2E_s}{T}}\cos\omega\tau\right)\left(\sqrt{\frac{2}{T}}\cos\omega(T-\tau)\right)d\tau = \pm\sqrt{E_s} \tag{1.8}$$

where the sign is $+$ or $-$ according to whether the modulator input was 1 or 0, respectively, and variance

$$\sigma^2 = \frac{N_0}{2}\int_0^T \left(\sqrt{\frac{2}{T}}\cos\omega\tau\right)^2 d\tau = \frac{N_0}{2} \tag{1.9}$$

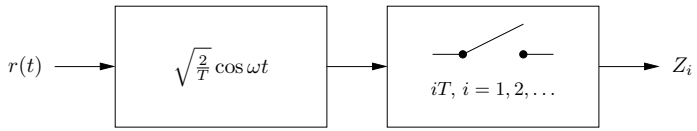


Figure 1.4 Matched filter receiver.

After the sampler we can make a *hard decision*, that is, a binary quantization with threshold zero, of the random variable Z_i. Then we obtain the simplest and most important binary-input and binary-output channel model, the *binary symmetric channel* (BSC) with *crossover probability* ϵ (Fig. 1.5). The crossover probability is of course closely related to the signal-to-noise ratio E_s/N_0. Since the channel output for a given signaling interval depends only on the transmitted waveform and noise during that interval and not on other intervals, the channel is said to be *memoryless*.

Because of symmetry, we can without loss of generality assume that a 0, that is, $-\sqrt{2E_s/T}\cos\omega t$, is transmitted over the channel. Then we have a channel "error"

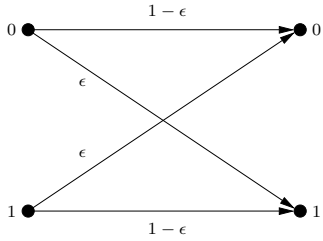


Figure 1.5 Binary symmetric channel.

if and only if the matched filter output at the sample time iT is positive. Thus, the probability that $Z_i > 0$ given that a 0 is transmitted is

$$\epsilon = P\left(Z_i > 0 \mid 0 \text{ sent}\right) \tag{1.10}$$

where Z_i is a Gaussian random variable, $Z_i \in N(-\sqrt{E_\mathrm{s}}, \sqrt{N_0/2})$, and E_s is the *energy per symbol*. Since the probability density function of Z_i is

$$f_{Z_i}(z) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(z-\mu)^2}{2\sigma^2}} \tag{1.11}$$

we have

$$\begin{aligned} \epsilon &= \frac{1}{\sqrt{\pi N_0}} \int_0^\infty e^{-\frac{(z+\sqrt{E_\mathrm{s}})^2}{N_0}} dz \\ &= \frac{1}{\sqrt{2\pi}} \int_{\sqrt{2E_\mathrm{s}/N_0}}^\infty e^{-y^2/2} dy = Q\left(\sqrt{2E_\mathrm{s}/N_0}\right) \end{aligned} \tag{1.12}$$

where

$$Q(x) = \frac{1}{\sqrt{2\pi}} \int_x^\infty e^{-y^2/2} dy \tag{1.13}$$

is the *complementary error function* of Gaussian statistics (often called the Q-function).

When coding is used, we prefer measuring the energy per *information* bit, E_b, rather than per symbol. For uncoded BPSK, we have $E_\mathrm{b} = E_\mathrm{s}$. Letting P_b denote the *bit error probability* (or *bit error rate*), that is, the probability that an information bit is erroneously delivered to the destination, we have for uncoded BPSK

$$P_\mathrm{b} = \epsilon = Q\left(\sqrt{2E_\mathrm{b}/N_0}\right) \tag{1.14}$$

How much better can we do with coding?

It is clear that when we use coding, it is a waste of information to make hard decisions. Since the influence of each information bit will be spread over several channel symbols, the decoder can benefit from using the *value* of Z_i (hard decisions use only the *sign* of Z_i) as an indication of how reliable the received symbol is. The demodulator can give the analog value of Z_i as its output, but it is often more

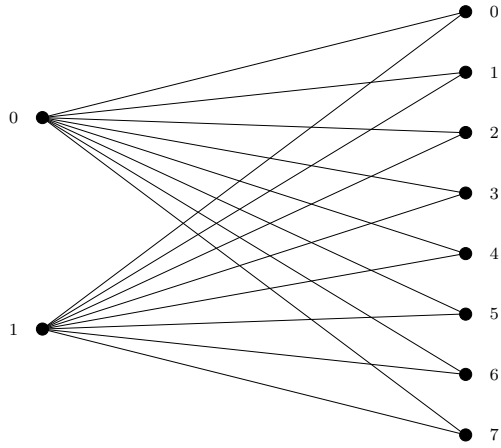


Figure 1.6 Binary input, 8-ary output, DMC.

practical to use, for example, a three-bit quantization—a *soft decision*. By introducing seven thresholds, the values of Z_i are divided into eight intervals and we obtain an eight-level soft-quantized *discrete memoryless channel* (DMC) as shown in Fig. 1.6.

Shannon [Sha48] showed that the capacity of the bandlimited AWGN channel with bandwidth W is[2]

$$C_t^W = W \log\left(1 + \frac{S}{N_0 W}\right) \text{ bits/s} \tag{1.15}$$

where $N_0/2$ and S denote the two-sided noise spectral density and the signaling power, respectively. If the bandwidth W goes to infinity, we have

$$\begin{aligned} C_t^\infty &\stackrel{\text{def}}{=} \lim_{W\to\infty} W \log\left(1 + \frac{S}{N_0 W}\right) \\ &= \frac{S}{N_0 \ln 2} \text{ bits/s} \end{aligned} \tag{1.16}$$

If we transmit K information bits during τ seconds, where τ is a multiple of bit duration T, we have

$$E_\text{b} = \frac{S\tau}{K} \tag{1.17}$$

Since the data transmission rate is $R_t = K/\tau$ bits/s, the energy per bit can be written

$$E_\text{b} = \frac{S}{R_t} \tag{1.18}$$

Combining (1.16) and (1.18) gives

$$\frac{C_t^\infty}{R_t} = \frac{E_\text{b}}{N_0 \ln 2} \tag{1.19}$$

[2]Here and hereafter we write log for $\log_2$.

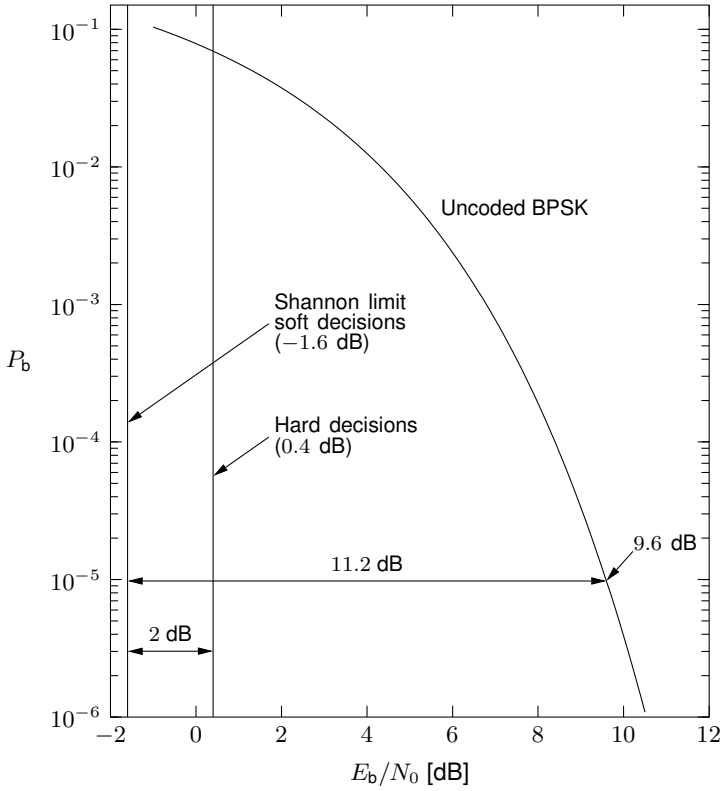


Figure 1.7 Capacity limits and regions of potential coding gain.

From Shannon's celebrated *channel coding theorem* [Sha48] it follows that for reliable communication we must have $R_t \leq C_t^{\infty}$. Hence, from this inequality and (1.19) we have

$$\frac{E_b}{N_0} \geq \ln 2 = 0.69 = -1.6 \text{ dB} \tag{1.20}$$

which is the fundamental *Shannon limit*.

In any system that provides reliable communication in the presence of additive white Gaussian noise the signal-to-noise ratio E_b/N_0 cannot be less than the Shannon limit, -1.6 dB!

On the other hand, as long as E_b/N_0 exceeds the Shannon limit, -1.6 dB, Shannon's channel coding theorem guarantees the existence of a system—perhaps very complex—for reliable communication over the channel.

In Fig. 1.7, we have plotted the fundamental limit of (1.20) together with the bit error rate for uncoded BPSK, that is, equation (1.14). At a bit error rate of 10^{-5}, the infinite-bandwidth additive white Gaussian noise channel requires an E_b/N_0 of at least 9.6 dB. Thus, at this bit error rate we have a potential *coding gain* of 11.2 dB!

For the bandlimited AWGN channel with BPSK and hard decisions, that is, a BSC with crossover probability ϵ (Fig. 1.5) Shannon [Sha48] showed that the capacity is

$$C_t^{\text{BSC}} = 2W(1 - h(\epsilon)) \text{ bits/s} \tag{1.21}$$

where

$$h(\epsilon) = -\epsilon \log \epsilon - (1 - \epsilon) \log(1 - \epsilon) \tag{1.22}$$

is the *binary entropy function*. If we restrict ourself to hard decisions, we can use (1.21) and show (Problem 1.2) that for reliable communication we must have

$$\frac{E_{\text{b}}}{N_0} \geq \frac{\pi}{2} \ln 2 = 1.09 = 0.4 \text{ dB} \tag{1.23}$$

In terms of capacity, soft decisions are about 2 dB more efficient than hard decisions.

Although it is practically impossible to obtain the entire theoretically promised 11.2 dB coding gain, communication systems that pick up 2–8 dB are routinely in use. During the last decade iterative decoding has been used to design communication systems that operate only tenths of a dB from the Shannon limit.

We conclude this section, which should have provided some motivation for the use of coding, with an adage from R. E. Blahut [Bla92]: "To build a communication channel as good as we can is a waste of money"—use coding instead!

1.2 BLOCK CODES—A PRIMER

For simplicity, we will deal only with *binary* block codes. We consider the entire sequence of information bits to be divided into *blocks* of K bits each. These blocks are called *messages* and denoted $\boldsymbol{u} = u_0\, u_1\, \ldots\, u_{K-1}$. In block coding, we let $\boldsymbol{u}$ denote a message rather than the entire information sequence as is the case in convolutional coding to be considered later.

A binary (N, K) *block code* $\mathcal{B}$ is a set of $M = 2^K$ binary N-tuples (or row vectors of length N) $\boldsymbol{v} = v_0\, v_1\, \ldots\, v_{N-1}$ called *codewords*. N is called the *block length* and the ratio

$$R = \frac{\log M}{N} = \frac{K}{N} \tag{1.24}$$

is called the *code rate* and is measured in bits per (channel) use. The data transmission rate in bits/s is obtained by multiplying the code rate (1.24) by the number of transmitted channel symbols per second:

$$R_t = R/T \tag{1.25}$$

If we measure the channel capacity for the BSC in bits/channel use (bits/c.u.), then the capacity of the BSC equals

$$C = 1 - h(\epsilon) \quad \text{(bits/c.u.)} \tag{1.26}$$

According to Shannon's channel coding theorem, for reliable communication, we must have $R \leq C$ and the block length N should be chosen sufficiently large.

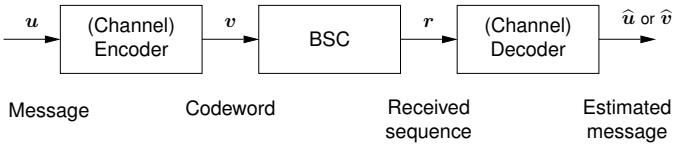


Figure 1.8 A binary symmetric channel (BSC) with (channel) encoder and decoder.

■ **EXAMPLE 1.1**

The set $\mathcal{B} = \{000, 011, 101, 110\}$ is a $(3, 2)$ code with four codewords and rate $R = 2/3$.

An *encoder* for an (N, K) block code $\mathcal{B}$ is a one-to-one mapping from the set of $M = 2^K$ binary messages to the set of codewords $\mathcal{B}$.

■ **EXAMPLE 1.2**

$u_0\, u_1$	$v_0\, v_1\, v_2$
00	000
01	011
10	101
11	110

and

$u_0\, u_1$	$v_0\, v_1\, v_2$
00	101
01	011
10	110
11	000

are two different encoders for the code $\mathcal{B}$ given in the previous example.

The rate $R = K/N$ is the fraction of the digits in the codeword that are necessary to represent the information; the remaining fraction, $1-R = (N-K)/N$, represents the *redundancy* that can be used to detect or correct errors.

Suppose that a codeword $\boldsymbol{v}$ corresponding to message $\boldsymbol{u}$ is sent over a BSC (see Fig. 1.8). The channel output $\boldsymbol{r} = r_0\, r_1\, \ldots\, r_{N-1}$ is called the *received sequence*. The decoder transforms the received N-tuple $\boldsymbol{r}$, which is possibly corrupted by noise, into the K-tuple $\widehat{\boldsymbol{u}}$, called the estimated *message* $\boldsymbol{u}$. Ideally, $\widehat{\boldsymbol{u}}$ will be a replica of the message $\boldsymbol{u}$, but the noise may cause some *decoding errors*. Since there is a one-to-one correspondence between the message $\boldsymbol{u}$ and the codeword $\boldsymbol{v}$, we can, equivalently, consider the corresponding N-tuple $\widehat{\boldsymbol{v}}$ as the decoder output. If the codeword $\boldsymbol{v}$ was transmitted, a decoding error occurs if and only if $\widehat{\boldsymbol{v}} \neq \boldsymbol{v}$.

Let P_{E} denote the *block* (or *word*) *error probability*, that is, the probability that the decision $\widehat{\boldsymbol{v}}$ for the codeword differs from the transmitted codeword $\boldsymbol{v}$. Then we have

$$P_{\mathrm{E}} = \sum_{\boldsymbol{r}} P\left(\widehat{\boldsymbol{v}} \neq \boldsymbol{v} \mid \boldsymbol{r}\right) P(\boldsymbol{r}) \tag{1.27}$$

where the probability that we receive $\boldsymbol{r}$, $P(\boldsymbol{r})$, is independent of the decoding rule and $P\left(\widehat{\boldsymbol{v}} \neq \boldsymbol{v} \mid \boldsymbol{r}\right)$ is the conditional probability of decoding error given the received sequence $\boldsymbol{r}$. Hence, in order to minimize P_{E}, we should specify the decoder such

that $P(\widehat{\boldsymbol{v}} \neq \boldsymbol{v} \mid \boldsymbol{r})$ is minimized for a given $\boldsymbol{r}$ or, equivalently, such that $P(\boldsymbol{v} \mid \boldsymbol{r}) \stackrel{\text{def}}{=} P(\widehat{\boldsymbol{v}} = \boldsymbol{v} \mid \boldsymbol{r})$ is maximized for a given $\boldsymbol{r}$. Thus the block error probability P_{E} is minimized by the decoder, which as its output chooses $\widehat{\boldsymbol{u}}$ such that the corresponding $\widehat{\boldsymbol{v}} = \boldsymbol{v}$ maximizes $P(\boldsymbol{v} \mid \boldsymbol{r})$. That is, $\boldsymbol{v}$ is chosen as the most likely codeword given that $\boldsymbol{r}$ is received. This decoder is called a *maximum a posteriori probability* (MAP) *decoder*.

Using Bayes' rule we can write

$$P(\boldsymbol{v} \mid \boldsymbol{r}) = \frac{P(\boldsymbol{r} \mid \boldsymbol{v})\, P(\boldsymbol{v})}{P(\boldsymbol{r})} \tag{1.28}$$

The code carries the most information possible with a given number of codewords when the codewords are equally likely. It is reasonable to assume that a decoder that is designed for this case also works satisfactorily—although not optimally—when the codewords are not equally likely, that is, when less information is transmitted. When the codewords are equally likely, maximizing $P(\boldsymbol{v} \mid \boldsymbol{r})$ is equivalent to maximizing $P(\boldsymbol{r} \mid \boldsymbol{v})$. The decoder that makes its decision $\widehat{\boldsymbol{v}} = \boldsymbol{v}$ such that $P(\boldsymbol{r} \mid \boldsymbol{v})$ is maximized is called a *maximum-likelihood* (ML) *decoder*.

Notice that in an erroneous decision for the codeword some of the information digits may nevertheless be correct. The bit error probability, which we introduced in the previous section, is a better measure of quality in most applications. However, it is in general more difficult to calculate. The bit error probability depends not only on the code and on the channel, like the block error probability, but also on the encoder and on the information symbols!

The use of block error probability as a measure of quality is justified by the inequality

$$P_{\text{b}} \leq P_{\text{E}} \tag{1.29}$$

When P_E can be made very small, inequality (1.29) implies that P_{b} can also be made very small.

The *Hamming distance* between the two N-tuples $\boldsymbol{r}$ and $\boldsymbol{v}$, denoted $d_{\text{H}}(\boldsymbol{r}, \boldsymbol{v})$, is the number of positions in which their components differ.

■ **EXAMPLE 1.3**

Consider the 5-tuples 10011 and 11000. The Hamming distance between them is 3.

The Hamming distance, which is an important concept in coding theory, is a *metric*; that is,

(i) $d_{\text{H}}(\boldsymbol{x}, \boldsymbol{y}) \geq 0$, with equality if and only if $\boldsymbol{x} = \boldsymbol{y}$ (positive definiteness)

(ii) $d_{\text{H}}(\boldsymbol{x}, \boldsymbol{y}) = d_{\text{H}}(\boldsymbol{y}, \boldsymbol{x})$ (symmetry)

(iii) $d_{\text{H}}(\boldsymbol{x}, \boldsymbol{y}) \leq d_{\text{H}}(\boldsymbol{x}, \boldsymbol{z}) + d_{\text{H}}(\boldsymbol{z}, \boldsymbol{y})$, all $\boldsymbol{z}$ (triangle inequality)

The *Hamming weight* of an N-tuple $\boldsymbol{x} = x_0 x_1 \ldots x_{N-1}$, denoted $w_{\text{H}}(\boldsymbol{x})$, is defined as the number of nonzero components in $\boldsymbol{x}$.

For the BSC, a transmitted symbol is erroneously received with probability ϵ where ϵ is the channel crossover probability. Thus, assuming ML decoding, we must make our decision $\widehat{\boldsymbol{v}}$ for the codeword $\boldsymbol{v}$ to maximize $P(\boldsymbol{r} \mid \boldsymbol{v})$; that is,

$$\widehat{\boldsymbol{v}} = \arg\max_{\boldsymbol{v}} \{P(\boldsymbol{r} \mid \boldsymbol{v})\} \tag{1.30}$$

where

$$P(\boldsymbol{r} \mid \boldsymbol{v}) = \epsilon^{d_{\mathrm{H}}(\boldsymbol{r},\boldsymbol{v})}(1-\epsilon)^{N-d_{\mathrm{H}}(\boldsymbol{r},\boldsymbol{v})} = (1-\epsilon)^N \left(\frac{\epsilon}{1-\epsilon}\right)^{d_{\mathrm{H}}(\boldsymbol{r},\boldsymbol{v})} \tag{1.31}$$

Since $0 < \epsilon < 1/2$ for the BSC, we have

$$0 < \frac{\epsilon}{1-\epsilon} < 1 \tag{1.32}$$

and, hence, maximizing $P(\boldsymbol{r} \mid \boldsymbol{v})$ is equivalent to minimizing $d_{\mathrm{H}}(\boldsymbol{r}, \boldsymbol{v})$. Clearly, ML decoding is equivalent to *minimum (Hamming) distance* (MD) *decoding*, that is, choosing as the decoder output the message $\widehat{\boldsymbol{u}}$ whose corresponding codeword $\widehat{\boldsymbol{v}}$ is (one of) the closest codeword(s) to the received sequence $\boldsymbol{r}$.

In general, the decoder must compare the received sequence $\boldsymbol{r}$ with all $M = 2^K = 2^{RN}$ codewords. The *complexity* of ML or MD decoding grows exponentially with the block length N. Thus it is infeasible to decode block codes with large block lengths. But to obtain low decoding error probability we have to use codes with relatively large block lengths. One solution of this problem is to use codes with algebraic properties that can be exploited by the decoder. Other solutions are to use codes on graphs (see Section 1.3) or convolutional codes (see Section 1.4).

In order to develop the theory further, we must introduce an algebraic structure.

A *field* is an algebraic system in which we can perform addition, subtraction, multiplication, and division (by nonzero numbers) according to the same associative, commutative, and distributive laws as we use with real numbers. Furthermore, a field is called *finite* if the set of numbers is finite. Here we will limit the discussion to block codes whose codewords have components in the simplest, but from a practical point of view also the most important, finite field, the *binary field*, $\mathbb{F}_2$, for which the rules for addition and multiplication are those of *modulo-two* arithmetic, namely

$+$	0	1
0	0	1
1	1	0

$\cdot$	0	1
0	0	0
1	0	1

We notice that addition and subtraction coincide in $\mathbb{F}_2$!

The set of binary N-tuples are the *vectors* in an N-dimensional *vector space*, denoted $\mathbb{F}_2^N$, over the field $\mathbb{F}_2$. *Vector addition* is component-by-component addition in $\mathbb{F}_2$. The *scalars* are the elements in $\mathbb{F}_2$. *Scalar multiplication* of scalar $a \in \mathbb{F}_2$ and vector $x_0 x_1 \ldots x_{N-1} \in \mathbb{F}_2^N$ is carried out according to the rule

$$a(x_0 x_1 \ldots x_{N-1}) = ax_0 ax_1 \ldots ax_{N-1} \tag{1.33}$$

Since a is either 0 or 1, scalar multiplication is trivial in $\mathbb{F}_2^N$.

Hamming weight and distance are clearly related:

$$d_{\mathrm{H}}(\boldsymbol{x}, \boldsymbol{y}) = w_{\mathrm{H}}(\boldsymbol{x} - \boldsymbol{y}) = w_{\mathrm{H}}(\boldsymbol{x} + \boldsymbol{y}) \tag{1.34}$$

where the arithmetic is in the vector space $\mathbb{F}_2^N$ and where the last equality follows from the fact that subtraction and addition coincide in $\mathbb{F}_2$.

The *minimum distance*, $d_{\min}$, of a code $\mathcal{B}$ is defined as the minimum value of $d_{\mathrm{H}}(\boldsymbol{v}, \boldsymbol{v}')$ over all $\boldsymbol{v}$ and $\boldsymbol{v}'$ in $\mathcal{B}$ such that $\boldsymbol{v} \neq \boldsymbol{v}'$.

■ **EXAMPLE 1.4**

The code $\mathcal{B}$ in Example 1.1 has $d_{\min} = 2$. It is a *single-error-detecting code* (see Problem 1.3).

Let $\boldsymbol{v}$ be the actual codeword and $\boldsymbol{r}$ the possibly erroneously received version of it. The *error pattern* $\boldsymbol{e} = e_0\, e_1\, \ldots\, e_{N-1}$ is the N-tuple that satisfies

$$\boldsymbol{r} = \boldsymbol{v} + \boldsymbol{e} \tag{1.35}$$

The *number of errors* is

$$w_{\mathrm{H}}(\boldsymbol{e}) = d_{\mathrm{H}}(\boldsymbol{r}, \boldsymbol{v}) \tag{1.36}$$

Let $\mathcal{E}_t$ denote the set of all error patterns with t or fewer errors, that is,

$$\mathcal{E}_t = \{\boldsymbol{e} \mid w_{\mathrm{H}}(\boldsymbol{e}) \leq t\} \tag{1.37}$$

We will say that a code $\mathcal{B}$ *corrects* the error pattern $\boldsymbol{e}$ if for all $\boldsymbol{v}$ the decoder maps $\boldsymbol{r} = \boldsymbol{v} + \boldsymbol{e}$ into $\widehat{\boldsymbol{v}} = \boldsymbol{v}$. If $\mathcal{B}$ corrects all error patterns in $\mathcal{E}_t$ and there is at least one error pattern in $\mathcal{E}_{t+1}$ which the code cannot correct, then t is called the *error-correcting capability* of $\mathcal{B}$.

Theorem 1.1 The code $\mathcal{B}$ has error-correcting capability t if and only if $d_{\min} > 2t$.

Proof: Suppose that $d_{\min} > 2t$. Consider the decoder which chooses $\widehat{\boldsymbol{v}}$ as (one of) the codeword(s) closest to $\boldsymbol{r}$ in Hamming distance (MD decoding). If $\boldsymbol{r} = \boldsymbol{v} + \boldsymbol{e}$ and $\boldsymbol{e} \in \mathcal{E}_t$, then $d_{\mathrm{H}}(\boldsymbol{r}, \boldsymbol{v}) \leq t$. The decoder output $\widehat{\boldsymbol{v}}$ must also satisfy $d_{\mathrm{H}}(\boldsymbol{r}, \widehat{\boldsymbol{v}}) \leq t$ since $\widehat{\boldsymbol{v}}$ must be at least as close to $\boldsymbol{r}$ as $\boldsymbol{v}$ is. Thus,

$$d_{\mathrm{H}}(\boldsymbol{v}, \widehat{\boldsymbol{v}}) \leq d_{\mathrm{H}}(\boldsymbol{v}, \boldsymbol{r}) + d_{\mathrm{H}}(\boldsymbol{r}, \widehat{\boldsymbol{v}}) \leq 2t < d_{\min} \tag{1.38}$$

which implies that $\widehat{\boldsymbol{v}} = \boldsymbol{v}$ and thus the decoding is correct.

Conversely, suppose that $d_{\min} \leq 2t$. Let $\boldsymbol{v}$ and $\boldsymbol{v}'$ be two codewords such that $d_{\mathrm{H}}(\boldsymbol{v}, \boldsymbol{v}') = d_{\min}$, and let the components of $\boldsymbol{r}$ be specified as

$$r_i = \begin{cases} v_i = v_i', & \text{all } i \text{ such that } v_i = v_i' \\ v_i', & \text{the first } t \text{ positions with } v_i \neq v_i' \text{ (if } t \leq d_{\min}) \text{ or} \\ & \text{all positions with } v_i \neq v_i' \text{ (otherwise)} \\ v_i, & \text{the remaining } d_{\min} - t \text{ positions (if any)} \end{cases} \tag{1.39}$$

Thus, $d_{\mathrm{H}}(\boldsymbol{v}, \boldsymbol{r}) = t$ and $d_{\mathrm{H}}(\boldsymbol{v}', \boldsymbol{r}) = d_{\min} - t \leq t$ (if $t \leq d_{\min}$) and $d_{\mathrm{H}}(\boldsymbol{v}, \boldsymbol{r}) = d_{\min}$ and $d_{\mathrm{H}}(\boldsymbol{v}', \boldsymbol{r}) = 0$ (otherwise). Next we observe that both error patterns $\boldsymbol{e}$ and $\boldsymbol{e}'$ satisfying

$$\boldsymbol{r} = \boldsymbol{v} + \boldsymbol{e} = \boldsymbol{v}' + \boldsymbol{e}' \tag{1.40}$$

are in $\mathcal{E}_t$, but the decoder cannot make the correct decision for both situations, and the proof is complete. ∎

To make codes easier to analyze and to simplify the implementation of their encoders and decoders, we impose a *linear structure* on the codes.

A binary, *linear block code* $\mathcal{B}$ of rate $R = K/N$ is a K-dimensional subspace of the vector space $\mathbb{F}_2^N$; that is, each codeword can be written as a linear combination of linearly independent vectors $\boldsymbol{g}_1, \boldsymbol{g}_2, \ldots, \boldsymbol{g}_K$, where $\boldsymbol{g}_i \in \mathbb{F}_2^N$, called the *basis* for the linear code $\mathcal{B}$. Then we call the $K \times N$ matrix G having $\boldsymbol{g}_1, \boldsymbol{g}_2, \ldots, \boldsymbol{g}_K$ as rows a *generator matrix* for $\mathcal{B}$. Clearly, since the vectors $\boldsymbol{g}_1, \boldsymbol{g}_2, \ldots, \boldsymbol{g}_K$ are linearly independent, the matrix G has full rank. The *row space* of G is $\mathcal{B}$ itself.

■ **EXAMPLE 1.5**

For the code in Example 1.1, which is linear, the codewords 011 and 101 form a basis. This basis determines the generator matrix

$$G = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \end{pmatrix} \tag{1.41}$$

The generator matrix offers a *linear encoding rule* for the code $\mathcal{B}$:

$$\boldsymbol{v} = \boldsymbol{u}G \tag{1.42}$$

where

$$G = \begin{pmatrix} g_{11} & g_{12} & \cdots & g_{1N} \\ g_{21} & g_{22} & \cdots & g_{2N} \\ \cdots & \cdots & \cdots & \cdots \\ g_{K1} & g_{K2} & \cdots & g_{KN} \end{pmatrix} \tag{1.43}$$

and the *information symbols* $\boldsymbol{u} = u_0\, u_1\, \ldots\, u_{K-1}$ are encoded into the codeword $\boldsymbol{v} = v_0\, v_1\, \ldots\, v_{N-1}$.

A generator matrix is often called an *encoding matrix* and is any matrix whose rows are a basis for $\mathcal{B}$. It is called *systematic* whenever the information digits appear unchanged in the first K components of the codeword; that is, G is systematic if and only if it can be written as

$$G = (I_K\ P) \tag{1.44}$$

where I_K is the $K \times K$ identity matrix.

■ **EXAMPLE 1.6**

The generator matrix

$$G = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix} \tag{1.45}$$

is a systematic encoding matrix for the code in Example 1.1.

Two codes $\mathcal{B}$ and $\mathcal{B}'$ are said to be *equivalent* if the order of the digits in the codewords $\boldsymbol{v} \in \mathcal{B}$ are simply a rearrangement of that in the codewords $\boldsymbol{v}' \in \mathcal{B}'$.

Theorem 1.2 Either a linear code $\mathcal{B}$ has a systematic encoding matrix or there exists an equivalent linear code $\mathcal{B}'$ which has a systematic encoding matrix.

Proof: See Problem 1.5. ■

Let G be an encoding matrix of the (N, K) code $\mathcal{B}$. Then G is a $K \times N$ matrix of rank K. By the theory of linear equations, the solutions of the system of linear homogeneous equations

$$G\boldsymbol{x}^{\mathrm{T}} = \mathbf{0} \tag{1.46}$$

where $\boldsymbol{x} = x_1\, x_2 \ldots x_N$, form an $(N-K)$-dimensional subspace of $\mathbb{F}_2^N$. Therefore, there exists an $(N-K) \times N$ matrix H of rank $N-K$ such that

$$GH^{\mathrm{T}} = \mathbf{0} \tag{1.47}$$

We are now ready to prove a fundamental result.

Theorem 1.3 An N-tuple $\boldsymbol{v}$ is a codeword in the linear code $\mathcal{B}$ with encoding matrix G if and only if

$$\boldsymbol{v}H^{\mathrm{T}} = \mathbf{0} \tag{1.48}$$

where H is an $(N-K) \times N$ matrix of full rank which satisfies

$$GH^{\mathrm{T}} = \mathbf{0} \tag{1.49}$$

Proof: Assume that the N-tuple $\boldsymbol{v} \in \mathcal{B}$, then $\boldsymbol{v} = \boldsymbol{u}G$ for some $\boldsymbol{u} \in \mathbb{F}_2^K$. Thus,

$$\boldsymbol{v}H^{\mathrm{T}} = \boldsymbol{u}GH^{\mathrm{T}} = \mathbf{0} \tag{1.50}$$

Conversely, suppose that $\boldsymbol{v}H^{\mathrm{T}} = \mathbf{0}$. Since $GH^{\mathrm{T}} = \mathbf{0}$ and both H and G have full rank, the rows of G form a basis of the solution space of $\boldsymbol{x}H^{\mathrm{T}} = \mathbf{0}$. Therefore, $\boldsymbol{v} = \boldsymbol{u}G$ for some $\boldsymbol{u} \in \mathbb{F}_2^K$, that is, $\boldsymbol{v} \in \mathcal{B}$. ■

From (1.49) it follows that each row vector of H is orthogonal to every codeword; the rows of H are *parity checks* on the codewords and we call H a *parity-check*

matrix[3] of the linear code $\mathcal{B}$. Equation (1.48) simply says that certain coordinates in each codeword must sum to zero.

It is easily verified that an (N, K) binary linear code with systematic encoding matrix $G = (I_K \ P)$ has

$$H = \left(P^{\mathrm{T}} \ I_{N-K}\right) \tag{1.51}$$

as a parity-check matrix.

■ **EXAMPLE 1.7**

The code in Example 1.1 with an encoding matrix given in Example 1.6 has

$$H = \begin{pmatrix} 1 & 1 & 1 \end{pmatrix} \tag{1.52}$$

as a parity-check matrix.

Next, we will consider a member of a much celebrated class of *single-error-correcting* codes due to Hamming [Ham50].

■ **EXAMPLE 1.8**

The binary $(7, 4)$ Hamming code with encoding matrix

$$G = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix} \tag{1.53}$$

has

$$H = \begin{pmatrix} 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{pmatrix} \tag{1.54}$$

as a parity-check matrix.

Note that $\boldsymbol{v}H^{\mathrm{T}} = \mathbf{0}$ can be written as

$$\begin{array}{ccccccccc} v_1 & + & v_2 & + & v_3 & + & v_4 & = & 0 \\ v_0 & + & v_2 & + & v_3 & + & v_5 & = & 0 \\ v_0 & + & v_1 & + & v_3 & + & v_6 & = & 0 \end{array} \tag{1.55}$$

so that each row in H determines one of the three *parity-check symbols* v_4, v_5, and v_6. The remaining four code symbols, v_0, v_1, v_2, and v_3, are the information symbols.

[3]When we consider LDPC codes in Section 1.3 and Chapter 8, we omit the full-rank requirement in the definition of parity-check matrices.

We notice that if we start with the 7-tuple 1011000, take all cyclic shifts, and form all linear combinations, we obtain a $(7, 4)$ Hamming code with a parity-check matrix

$$H = \begin{pmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{pmatrix} \tag{1.56}$$

which is *equivalent* to the one given in Example 1.8, that is, the two codes differ only by the ordering of the components in their codewords (permute columns 1–4). The $(7, 4)$ Hamming codes are members of a large class of important linear code families called *cyclic codes*—every cyclic shift of a codeword is a codeword. The class of cyclic codes includes, besides the Hamming codes, such famous codes as the Bose-Chaudhuri-Hocquenhem (BCH) and Reed-Solomon (RS) codes.

The cyclic behavior of these codes makes it possible to exploit a much richer algebraic structure that has resulted in the development of very efficient decoding algorithms suitable for hardware implementations.

Since the $N - K$ rows of the parity-check matrix H are linearly independent, we can use H as an encoding matrix of an $(N, N - K)$ linear code $\mathcal{B}^\perp$, which we call the *dual* or *orthogonal code* of $\mathcal{B}$.

Let $\boldsymbol{v}^\perp \in \mathcal{B}^\perp$ and assume that $\boldsymbol{v}^\perp = \boldsymbol{u}^\perp H$, where $\boldsymbol{u}^\perp \in \mathbb{F}_2^{N-K}$. Then from (1.49) it follows that

$$\boldsymbol{v}^\perp G^{\mathrm{T}} = \boldsymbol{u}^\perp H G^{\mathrm{T}} = \boldsymbol{u}^\perp \left(G H^{\mathrm{T}}\right)^{\mathrm{T}} = \mathbf{0} \tag{1.57}$$

Conversely, assume that $\boldsymbol{v}^\perp G^{\mathrm{T}} = \mathbf{0}$ for $\boldsymbol{v}^\perp \in \mathbb{F}_2^N$. Since $HG^{\mathrm{T}} = \mathbf{0}$ and both H and G have full rank, $\boldsymbol{v}^\perp$ is a linear combination of the rows of H, that is, $\boldsymbol{v}^\perp \in \mathcal{B}^\perp$. Hence, G is a $K \times N$ parity-check matrix of the dual code $\mathcal{B}^\perp$ and we have proved the following

Theorem 1.4 An $(N - K) \times N$ parity-check matrix for the linear code $\mathcal{B}$ is an $(N - K) \times N$ encoding matrix for the dual code $\mathcal{B}^\perp$, and conversely.

From (1.48) and (1.57) it follows that every codeword of $\mathcal{B}$ is orthogonal to that of $\mathcal{B}^\perp$ and conversely.

■ **EXAMPLE 1.9**

The code $\mathcal{B}$ in Example 1.1 has the dual code $\mathcal{B}^\perp = \{000, 111\}$.

If $\mathcal{B} = \mathcal{B}^\perp$, we call $\mathcal{B}$ *self-dual*.

■ **EXAMPLE 1.10**

The $(2, 1)$ *repetition code* $\{00, 11\}$ is self-dual.

The *minimum weight*, $w_{\min}$, of a linear code $\mathcal{B}$ is the smallest Hamming weight of its nonzero codewords.

Theorem 1.5 For a linear code,

$$d_{\min} = w_{\min} \tag{1.58}$$

Proof: The theorem follows from (1.34) and the facts that for a linear code the sum of two codewords is a codeword. ■

For the class of linear codes, the study of distance properties reduces to the study of weight properties that concern only single codewords! A most convenient consequence of this is the following.

Theorem 1.6 If H is any parity-check matrix for a linear code $\mathcal{B}$, then $d_{\min} = w_{\min}$ equals the smallest number of columns of H that form a linearly dependent set.

Proof: Follows immediately from $\boldsymbol{v}H^{\mathrm{T}} = \mathbf{0}$ for $\boldsymbol{v} \in \mathcal{B}$. ■

■ **EXAMPLE 1.11**

Consider the (7,4) Hamming code with parity-check matrix (1.54),

$$H = \begin{pmatrix} 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{pmatrix} \tag{1.59}$$

All pairs of columns are linearly independent. Many sets of three columns are linearly dependent, for example, columns 1, 6, and 7. It follows from the previous theorem that $d_{\min} = 3$. All single errors (that is, all error patterns of weight 1) can be corrected. This is a single-error-correcting code. This code can also detect double errors but not simultaneously with correcting single errors (see Problem 1.4).

The following theorem establishes an upper bound for the minimum distance.

Theorem 1.7 (Singleton bound) If $\mathcal{B}$ is an (N, K) linear code with minimum distance $d_{\min}$, then the number of parity-check digits is lower-bounded by

$$N - K \geq d_{\min} - 1 \tag{1.60}$$

Proof: A codeword with only one nonzero information digit has weight at most $1 + N - K$. Then, from Theorem 1.5 follows

$$w_{\min} = d_{\min} \leq N - K + 1 \tag{1.61}$$

■

An (N, K) linear code that meets the Singleton bound with equality is called a *maximum-distance-separable* (MDS) *code*.

The only *binary* MDS codes are the trivial ones: the (N, N) code $\mathcal{B} = \mathbb{F}_2^N$, the $(N, 1)$ repetition code $\mathcal{B} = \{00\ldots0, 11\ldots1\}$, and the $(N, N-1)$ code consisting of all even-weight N-tuples. (For a nontrivial code, $2 \leq K \leq N - 1$.)

The most celebrated examples of *nonbinary* MDS codes are the Reed-Solomon codes.

To find (nontrivial) lower bounds on the minimum distance is a much harder problem. It has been proved [Gil52, Var57] that there exists a sequence of binary (N, K) linear block codes with increasing block length N having minimum distances lower-bounded by the inequality

$$d_{\min} > \rho_{\text{GV}} N \tag{1.62}$$

where ρ_{GV} is the so-called *Gilbert-Varshamov parameter*. For a given rate $R = K/N$, ρ_{GV} is equal to the smallest root $\rho < 1/2$ of the equation

$$R = 1 - h(\rho) \tag{1.63}$$

where $h(\cdot)$ is the binary entropy function (1.22).

Let $\mathcal{B}$ be an (N, K) linear code. For any binary N-tuple $\boldsymbol{a}$, the set

$$\boldsymbol{a} + \mathcal{B} \stackrel{\text{def}}{=} \{\boldsymbol{a} + \boldsymbol{v} \mid \boldsymbol{v} \in \mathcal{B}\} \tag{1.64}$$

is called a *coset* of $\mathcal{B}$. Every $\boldsymbol{b} \in \mathbb{F}_2^N$ is in some coset; for example, $\boldsymbol{b} + \mathcal{B}$ contains $\boldsymbol{b}$. Two binary N-tuples $\boldsymbol{a}$ and $\boldsymbol{b}$ are in the same coset if and only if their difference (sum in $\mathbb{F}_2^N$) is a codeword, or, equivalently, $(\boldsymbol{a} + \boldsymbol{b}) \in \mathcal{B}$. Every coset of $\mathcal{B}$ contains the same number of elements, 2^K, as $\mathcal{B}$ does.

Theorem 1.8 Any two cosets are either disjoint or identical.

Proof: Suppose that $\boldsymbol{c}$ belongs to both $\boldsymbol{a} + \mathcal{B}$ and $\boldsymbol{b} + \mathcal{B}$. Then $\boldsymbol{c} = \boldsymbol{a} + \boldsymbol{v} = \boldsymbol{b} + \boldsymbol{v}'$, where $\boldsymbol{v}, \boldsymbol{v}' \in \mathcal{B}$. Thus, $\boldsymbol{a} = \boldsymbol{b} + \boldsymbol{v} + \boldsymbol{v}' \in \boldsymbol{b} + \mathcal{B}$, and so $\boldsymbol{a} + \mathcal{B} \subseteq \boldsymbol{b} + \mathcal{B}$. Similarly $\boldsymbol{b} + \mathcal{B} \subseteq \boldsymbol{a} + \mathcal{B}$. Hence, $\boldsymbol{a} + \mathcal{B} = \boldsymbol{b} + \mathcal{B}$. ∎

From Theorem 1.8 two corollaries follow immediately:

Corollary 1.9 $\mathbb{F}_2^N$ is the union of all the cosets of $\mathcal{B}$.

Corollary 1.10 A binary (N, K) code $\mathcal{B}$ has 2^{N-K} cosets.

Suppose that the binary N-tuple $\boldsymbol{r}$ is received over the BSC. Then

$$\boldsymbol{r} = \boldsymbol{v} + \boldsymbol{e} \tag{1.65}$$

where $\boldsymbol{v} \in \mathcal{B}$ is a codeword and $\boldsymbol{e}$ is an error pattern. Clearly $\boldsymbol{r}$ is in the coset $\boldsymbol{r} + \mathcal{B}$. From (1.65) it follows that *the coset $\boldsymbol{r} + \mathcal{B}$ contains exactly the possible error patterns*! The N-tuple of smallest weight in a coset is called a *coset leader*. (If there is more than one N-tuple with smallest weight, any one of them can be chosen as coset leader.)

An MD decoder will select as its output the error pattern, $\widehat{\boldsymbol{e}}$ say, which is a coset leader of the coset containing $\boldsymbol{r}$, subtract (or, equivalently in $\mathbb{F}_2^N$, add) $\widehat{\boldsymbol{e}}$ from (to) $\boldsymbol{r}$, and, finally, obtain its maximum-likelihood decision $\widehat{\boldsymbol{v}}$.

We illustrate what the decoder does by showing the *standard array*. The first row consists of the code $\mathcal{B}$ with the allzero codeword on the left. The following rows are the cosets $\boldsymbol{e}_i + \mathcal{B}$ arranged in the same order with the coset leader on the left:

$\mathbf{0}$	$\boldsymbol{v}_1$	$\ldots$	$\boldsymbol{v}_{2^K-1}$
$\boldsymbol{e}_1$	$\boldsymbol{v}_1 + \boldsymbol{e}_1$	$\ldots$	$\boldsymbol{v}_{2^K-1} + \boldsymbol{e}_1$
$\vdots$	$\vdots$		$\vdots$
$\boldsymbol{e}_{2^{N-K}-1}$	$\boldsymbol{v}_1 + \boldsymbol{e}_{2^{N-K}-1}$	$\ldots$	$\boldsymbol{v}_{2^K-1} + \boldsymbol{e}_{2^{N-K}-1}$

The MD decoder decodes $\boldsymbol{r}$ to the codeword $\widehat{\boldsymbol{v}}$ at the top of the column that contains $\boldsymbol{r}$.

■ **EXAMPLE 1.12**

The $(4, 2)$ code $\mathcal{B}$ with encoding matrix

$$G = \begin{pmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix} \tag{1.66}$$

has the following standard array:

0000	1011	0110	1101
1000	0011	1110	0101
0100	1111	0010	1001
0001	1010	0111	1100

Suppose that $\boldsymbol{r} = 1001$ is received. An MD decoder outputs $\widehat{\boldsymbol{v}} = 1101$.

Theorem 1.11 An (N, K) binary linear code $\mathcal{B}$ can correct all error patterns in a set $\mathcal{E}$ if and only if these error patterns all lie in different cosets of $\mathbb{F}_2^N$ relative to $\mathcal{B}$.

Proof: Suppose that $\boldsymbol{e}$ and $\boldsymbol{e}'$ are distinct error patterns in the same coset. Then there is a $\boldsymbol{v} \in \mathcal{B}$ such that $\boldsymbol{v} + \boldsymbol{e} = \boldsymbol{e}'$. No decoder can correct both $\boldsymbol{e}$ and $\boldsymbol{e}'$.

Conversely, suppose that all error patterns in $\mathcal{E}$ lie in different cosets. If $\boldsymbol{v}$ is the actual transmitted codeword and $\boldsymbol{e}$ the actual error pattern, then $\boldsymbol{r} = \boldsymbol{v} + \boldsymbol{e}$ lies in $\boldsymbol{e} + \mathcal{B}$. Thus, all error patterns in $\mathcal{E}$ can be corrected by a decoder that maps $\boldsymbol{r}$ into the error pattern $\widehat{\boldsymbol{e}} \in \mathcal{E}$ (if any) that lies in the same coset $\boldsymbol{e} + \mathcal{B}$ as $\boldsymbol{r}$ does. ■

The *syndrome* of the received N-tuple $\boldsymbol{r}$, relative to the parity-check matrix H, is defined as

$$\boldsymbol{s} \stackrel{\text{def}}{=} \boldsymbol{r}H^{\mathrm{T}} \tag{1.67}$$

Assume that the transmitted codeword is $\boldsymbol{v}$ and that $\boldsymbol{r} = \boldsymbol{v} + \boldsymbol{e}$, where $\boldsymbol{e}$ is the error pattern. Both $\boldsymbol{r}$ and H are known to the receiver, which exploits (1.48) and forms

$$\begin{aligned} \boldsymbol{s} &= \boldsymbol{r}H^{\mathrm{T}} = (\boldsymbol{v} + \boldsymbol{e})H^{\mathrm{T}} \\ &= \boldsymbol{v}H^{\mathrm{T}} + \boldsymbol{e}H^{\mathrm{T}} = \mathbf{0} + \boldsymbol{e}H^{\mathrm{T}} \end{aligned} \tag{1.68}$$

so that

$$\boldsymbol{s} = \boldsymbol{e}H^{\mathrm{T}} \tag{1.69}$$

The syndrome depends only on the error pattern and not on the codeword!

In medical terminology, a syndrome is a pattern of symptoms. Here the disease is the error pattern, and a symptom is a parity-check failure.

Equation (1.69) gives $N-K$ linearly independent equations for the N components of the error pattern $\boldsymbol{e}$. Hence, there are exactly 2^K error patterns satisfying (1.69). These are precisely all the error patterns that are differences between the received N-tuple $\boldsymbol{r}$ and all 2^K different codewords $\boldsymbol{v}$. For a given syndrome, these 2^K error patterns belong to the same coset. Furthermore, if two error patterns lie in the same coset, then their difference is a codeword and it follows from (1.68) that they have the same syndrome. Hence, we have the following theorem:

Theorem 1.12 Two error patterns lie in the same coset if and only if they have the same syndrome.

From the two previous theorems a corollary follows:

Corollary 1.13 An (N, K) binary linear code $\mathcal{B}$ can correct all error patterns in a set $\mathcal{E}$ if and only if the syndromes of these error patterns are all different.

No information about the error pattern is lost by calculating the syndrome!

In Fig. 1.9 we show the structure of a general *syndrome decoder*. The syndrome former, H^{T}, is linear, but the error pattern estimator is always nonlinear in a useful decoder. Clearly, a syndrome decoder is an MD or, equivalently, an ML decoder.

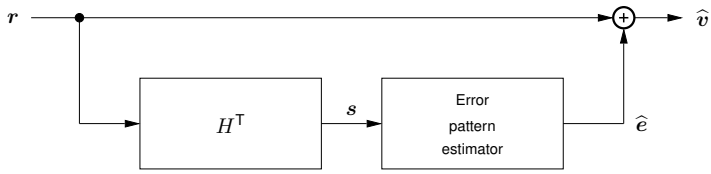


Figure 1.9 Syndrome decoder for a linear block code.

■ **EXAMPLE 1.13**

Consider the $(7, 4)$ Hamming code whose parity-check matrix is given in Example 1.11. Let $\boldsymbol{r} = 0010001$ be the received 7-tuple. The syndrome is

$$\boldsymbol{s} = (0010001)\begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} = 111 \tag{1.70}$$

Since $\boldsymbol{s} \neq \mathbf{0}$, $\boldsymbol{r}$ contains at least one erroneous component. For the Hamming codes there is a one-to-one correspondence between the single errors and the nonzero syndromes. Among all $2^K = 16$ possible error patterns, the MD decoder chooses the one with least Hamming weight, that is, the single error pattern corresponding to the given syndrome $\boldsymbol{s} = 111$. Since the fourth row in H^{T} is the triple 111, the MD decoder gives as its output $\widehat{\boldsymbol{e}} = 0001000$ (a single 1 in the *fourth* position). It immediately follows that

$$\begin{aligned}\widehat{\boldsymbol{v}} &= \boldsymbol{r} + \widehat{\boldsymbol{e}} \\ &= 0010001 + 0001000 \\ &= 0011001 \end{aligned} \tag{1.71}$$

If $\boldsymbol{v} = 0011001$ was sent, we have corrected the transmission error. However, if $\boldsymbol{v} = 0000000$ was sent and $\boldsymbol{e} = 0010001$, the Hamming code is not able to correct the error pattern. The syndrome decoder will in this case give as its output $\widehat{\boldsymbol{v}} = 0011001$!

Suppose that the $(7, 4)$ Hamming code is used to communicate over a BSC with channel error probability ϵ. The ML decoder can correctly identify the transmitted codeword if and only if the channel causes at most one error. The block (or word) error probability P_{E}, that is, the probability that the decision for the codeword differs from the actual codeword, is

$$\begin{aligned}P_{\mathrm{E}} &= \sum_{i=2}^{7} \binom{7}{i} \epsilon^i (1-\epsilon)^{7-i} \\ &= 21\epsilon^2 - 70\epsilon^3 + \cdots \end{aligned} \tag{1.72}$$

For the $(7, 4)$ Hamming code, it can be shown (Problem 1.21) that for all digits

$$\begin{aligned}P_{\mathrm{b}} &= 9\epsilon^2(1-\epsilon)^5 + 19\epsilon^3(1-\epsilon)^4 + 16\epsilon^4(1-\epsilon)^3 \\ &\quad +12\epsilon^5(1-\epsilon)^2 + 7\epsilon^6(1-\epsilon) + \epsilon^7 \\ &= 9\epsilon^2 - 26\epsilon^3 + \cdots \end{aligned} \tag{1.73}$$

Maximum-likelihood (ML) decoding that we discussed in this section is an important decoding method in the sense that (assuming equiprobable messages) it minimizes the block error probability. In Chapter 5 we shall show that if we use ML decoding there exist codes for which the block and bit error probabilities decrease exponentially with block length N.

A drawback with ML decoding is that its decoding complexity increases very fast—exponentially fast—with increasing block length. There is a need for low-complexity suboptimum decoding algorithms.

1.3 CODES ON GRAPHS

In this section, we introduce an important class of block codes, *codes on graphs*, which are very powerful in combination with *iterative decoding*. A well-known class

of such codes is the *low-density parity-check (LDPC)* codes invented by Gallager [Gal62, Gal63]. When these codes are decoded iteratively, they have better bit error probability/decoding complexity tradeoff than general block codes.

Tanner [Tan81] used *bipartite graphs* to describe the structure of linear block codes. These so-called *Tanner graphs* consist of two sets of nodes, the set of *symbol* (or *variable*) nodes which correspond to the symbols of the codewords and the set of *constraint* (or *factor*) nodes which correspond to the parity-check equations defining the code. In a Tanner graph, each symbol node v_n is connected by edges only with constraint nodes c_l and, similarly, each constraint node is connected only with symbol nodes. The number of edges connected to a node is called the *degree* of that node.

Each column in the parity-check matrix corresponds to a symbol node and each row corresponds to a constraint node. A 1 in the (l, n)th position of the parity-check matrix H corresponds to an edge between the symbol node v_n and the constraint node c_l.

■ **EXAMPLE 1.14**

Consider the following parity-check matrix (1.56) of a $(7, 4)$ Hamming code,

$$H = \begin{pmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{pmatrix} \tag{1.74}$$

This code has seven code symbols interrelated by three parity-check equations (*constraints*),

$$\begin{array}{ccccccccc} v_0 & + & v_1 & + & v_2 & + & v_4 & = & 0 \\ v_1 & + & v_2 & + & v_3 & + & v_5 & = & 0 \\ v_0 & + & v_1 & + & v_3 & + & v_6 & = & 0 \end{array} \tag{1.75}$$

The Tanner graph of the Hamming $(7, 4)$ code (Fig. 1.10) has seven symbol nodes and three constraint nodes. All constraint nodes have degree 4, but the degrees of symbol nodes varies from 1 to 3.

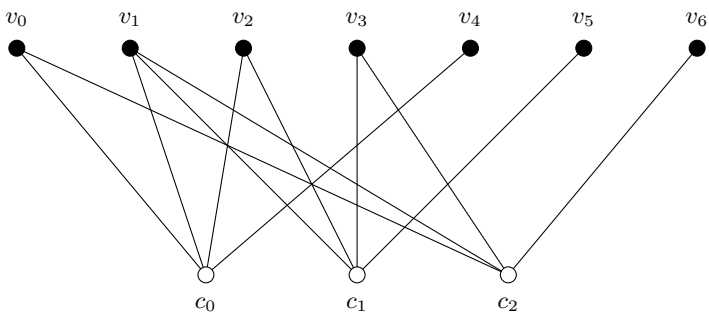


Figure 1.10 Tanner graph of a Hamming $(7, 4)$ code defined by parity-check matrix (1.74).

Note that the set of four symbols included in each of the linear equations (1.75) forms a $(4, 3)$ single-error-detecting code. Totally we have three such *constituent codes*.

An LDPC code can be defined either by a parity-check matrix or by a Tanner graph. In the first case it is determined by an $L \times N$ sparse parity-check matrix H, "containing mostly 0s and relatively few 1s" as it was formulated by Gallager [Gal62, Gal63]. A *regular* (N, J, K) LDPC code of block length N has a parity-check matrix with a fixed number K 1s in each row[4] and a fixed number J 1s in each column, where $K/J = N/L$.

The Tanner graph of a regular (N, J, K) LDPC code has symbol node degree equal to J and constraint node degree equal to K. The rows of the parity-check matrix H of a regular (N, J, K) LDPC code can be linearly dependent, then the *design rate* $R_d = 1 - J/K$ can be less than the actual code rate R.

Remark: In the sequel we will for simplicity often use short block codes as illustrations and call them LDPC codes even if the requrement "containing mostly 0s and relatively few 1s" is not fulfilled.

■ **EXAMPLE 1.15**

Consider the following parity-check matrix of the rate $R = 6/15 = 2/5$ regular $(N, J, K) = (15, 3, 5)$ LDPC block code:

$$H = \begin{pmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \tag{1.76}$$

This code was found by computer search and has relatively good minimum distance, $d_{\min} = 6$, for an LDPC code of this size. Its Tanner graph is shown in Fig. 1.11.

An *irregular* LDPC code has in general a Tanner graph whose symbol node degrees as well as constraint node degrees are different.

For an LDPC code, the rows of the $L \times N$ parity-check matrix H can in general be linearly dependent and, equivalently, this matrix has not full rank. If we delete linearly dependent rows such that the remaining rows are linearly independent, then we obtain a parity-check matrix for a rate $R > R_d = 1 - L/N$ LDPC code.

[4]With a slight abuse of notations we use K to denote both the number of 1s in each row of and the number of information symbols in an (N, K) block code.

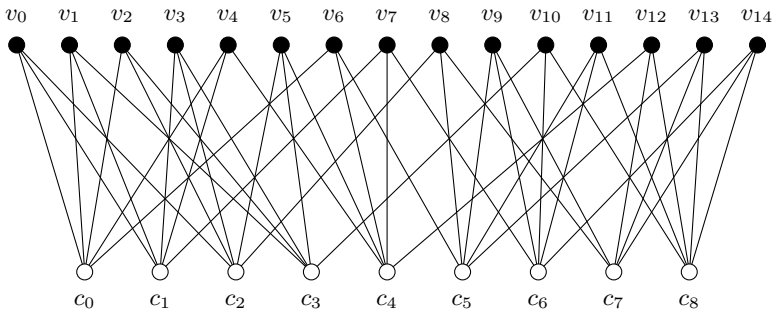


Figure 1.11 Tanner graph of the code defined by parity-check matrix (1.76).

Gallager introduced a special class of regular (N, J, K) LDPC codes. The $L \times N$ parity-check matrices H of these codes, where $N = KM$, $L = JM$, and $M > 0$ is an integer, can be represented as a composition of J submatrices H_j, $j = 1, 2, \ldots, J$,

$$H = \begin{pmatrix} H_1 \\ H_2 \\ \vdots \\ H_J \end{pmatrix} \tag{1.77}$$

Here the $M \times N$ submatrices H_j have one 1 in each column and K 1s in each row. The other entries of the submatrices are 0s. If $J \geq 2$, then the rank of the matrix (1.77) is always strictly less than L (see Problem 1.24).

Using the ensemble of regular (N, J, K) LDPC codes with random parity-check matrices given by (1.77) Gallager[5] proved that, if $N \to \infty$, then the minimum distance $d_{\min}$ of almost all codes from this ensemble is lower-bounded by the inequality

$$d_{\min} > \rho_{J,K} N \tag{1.78}$$

where the parameter $\rho_{J,K}$ is a positive constant (if $J > 2$). In other words, the minimum distance of typical (N, J, K) codes from this ensemble, as well as of typical randomly chosen block codes, is growing linearly with N.

In Table 1.1 we show the numerical values of $\rho_{J,K}$ for some J and K. For comparison also the value of the Gilbert-Varshamov parameter ρ_{GV} is given for the corresponding design rates $R_d = 1 - J/K$. In Chapter 8 we will prove the interesting fact that if $J = 2$ then $d_{\min}$ grows only logarithmically with N.

The main advantage of LDPC codes is that these codes can be decoded *iteratively* and that the complexity of iterative decoding grows slower than exponentially with the block length N. We shall conclude this introductory section by describing a simple iterative decoding algorithm for LDPC codes.

[5]Gallager used the slightly different ensemble with the first matrix $H^{(1)}$ fixed. This has, however, no effect on the technique of his proof.

Table 1.1 Gallager's parameter $\rho_{J,K}$ compared with the Gilbert-Varshamov parameter ρ_{GV}.

J	K	R_c	$\rho_{J,K}$	ρ_{GV}	$\rho_{\text{GV}}/\rho_{J,K}$
3	4	0.25	0.112	0.215	1.920
	5	0.4	0.045	0.146	3.244
	6	0.5	0.023	0.11	4.783
4	5	0.2	0.211	0.243	1.152
	6	0.333	0.129	0.174	1.349
5	6	0.167	0.255	0.264	1.035

Gallager introduced a simple hard-decision iterative decoding algorithm for LDPC codes called the *bit-flipping (BF) algorithm* [Gal62, Gal63]. It was later modified by Zyablov and Pinsker [ZyP75]. We will describe a version of Gallager's algorithm for regular (N, J, K) LDPC codes which uses an adaptive threshold.

Suppose that the binary N-tuple (codeword) $\boldsymbol{v}$ is sent over a BSC and that the binary N-tuple $\boldsymbol{r}$ is received. On the receiver side, a bit-flipping decoder calculates the tentative syndrome

$$\boldsymbol{s}' = (s'_0\, s'_1\, \ldots\, s'_{L-1}) = \boldsymbol{r}' H^{\mathrm{T}} \tag{1.79}$$

for the tentative sequence $\boldsymbol{r}' = r'_0\, r'_1\, \ldots\, r'_{N-1}$ which initially is equal to the received sequence $\boldsymbol{r}$.

Since the code is regular, each tentative symbol r'_n, $n = 0, 1, \ldots, N-1$, is included in J parity-check equations. Let δ_n, $n = 0, 1, \ldots, N-1$, denote the number of unsatisfied parity-check equations for the tentative symbol r'_n. We introduce a threshold T which is initially set to J. The decoder searches for any r'_n such that $\delta_n = T$. If such an r'_n does not exist, we decrease the threshold by 1, that is, $T \leftarrow T - 1$, and repeat the "search/decrease threshold" procedure until we have one of the following three situations:

(i) We find a tentative symbol r'_n with $\delta_n = T > \lfloor J/2 \rfloor$.

(ii) All $\delta_n = 0$, $n = 0, 1, \ldots, N-1$, that is, $\boldsymbol{s}' = \mathbf{0}$.

(iii) $\boldsymbol{s}' \neq \mathbf{0}$ and for all symbols r'_n we have $\delta_n \leq \lfloor J/2 \rfloor$.

If we find a tentative symbol r'_n such that $\delta_n = T > \lfloor J/2 \rfloor$, then we *flip* (change) r'_n and obtain a new tentative sequence $\boldsymbol{r}'$ which yields a new tentative syndrome $\boldsymbol{s}'$ with reduced Hamming weight. Whenever such a reduction of this Hamming weight occurs, we reset the threshold to $T = J$. (This is necessary since the flipping of r'_n could result in a $\delta_{n'}$ for $n' \neq n$ being as large as J.) Then we repeat the search procedure.

If the tentative syndrome $\boldsymbol{s}' = \mathbf{0}$, then the corresponding tentative sequence $\boldsymbol{r}'$ is a codeword and $\widehat{\boldsymbol{v}} = \boldsymbol{r}'$ is the decision for $\boldsymbol{v}$. The decoding is considered successful.

If $\boldsymbol{s}' \neq \mathbf{0}$ and $\delta_n \leq \lfloor J/2 \rfloor$ for $n = 0, 1, \ldots, N-1$, then the decoding is declared as a failure. The corresponding tentative sequence $\boldsymbol{r}'$ is called a *trapping set*.

Gallager's bit-flipping algorithm is characterized by its error-correcting capability and its decoding complexity. Its error-correcting capability $t^{(\mathrm{BF})}$ is the largest integer such that all error patterns of Hamming weight $t^{(\mathrm{BF})}$ or less are correctly decoded by the algorithm. For Gallager's bit-flipping algorithm, we can show that asymptotically, that is, when $N \to \infty$, the error-correcting capability is lower-bounded by a linear function of N. For maximum-likelihood decoding we have, as shown in the previous section, the error-correcting capability $t^{(\mathrm{ML})} = \lfloor \frac{d_{\min}-1}{2} \rfloor$, where $d_{\min}$ is the minimum distance of the code.

Since the bit-flipping algorithm is essentially less powerful than an ML algorithm, we have in general $t^{(\mathrm{BF})} \leq t^{(\mathrm{ML})}$.

What can we say about the decoding complexity of the bit-flipping algorithm? We will show that the total number of operations for Gallager's bit-flipping algorithm is upper-bounded by[6] $O(N^2)$. During each iteration the decoder checks the symbols r'_n, $n = 0, 1, \ldots, N-1$, of the tentative sequence $\boldsymbol{r}'$ up to the moment when it finds a symbol which is included in δ unsatisfied parity-check equations. Then it flips this symbol which decreases the tentative syndrome weight by at least one. Since the initial syndrome weight does not exceed N, the total number of iterations is upper-bounded by $O(N)$. In each iteration, the decoder checks at most N symbols. Assuming that checking one symbol requires one computational operation, the total number of operations for one iteration is upper-bounded by $O(N)$. Then the decoding complexity of Gallager's bit-flipping algorithm is upper-bounded by $O(N^2)$.

We can also describe the bit-flipping algorithm using the Tanner graph. In this case the decoder checks how many constraint nodes connected to a given tentative symbol node correspond to unsatisfied parity-check equations. If this number equals the threshold T, then the decoder flips the corresponding tentative symbol and goes to the next phase of the decoding.

Remark: We gave one possible description of Gallager's bit-flipping algorithm, with an adaptive threshold. In principle, the decoder can flip a tentative symbol whenever the number of unsatisfied parity-check equations for this symbol $\delta > \lfloor J/2 \rfloor$. We can say in this case that the decoder uses the lowest possible threshold $\lfloor J/2 \rfloor + 1$. The error-correcting capability when $N \to \infty$ of the algorithm with lowest possible threshold is still $O(N)$, but for finite N it is less than that for the algorithm with an adaptive threshold. In Problem 1.25 we study an example when the algorithm with an adaptive threshold has error-correcting capability $t^{(\mathrm{BF})} = 1$ but the algorithm with lowest possible threshold has zero error-correcting capability.

■ EXAMPLE 1.16

Consider the regular $(15, 3, 5)$ LDPC code given in Example 1.15. Suppose that we use the bit-flipping algorithm with an adaptive threshold for decoding. We will

[6]Here and hereafter we write $f(x) = O(g(x))$ if $|f(x)| \leq Ag(x)$ for x sufficiently near a given limit, A is a positive constant independent of x and $g(x) > 0$. We have, e.g., $f(x) = \log x$, $x > 0$, can be written as $f(x) = O(x)$ when $x \to \infty$.

show that this algorithm corrects all single errors and that there are double errors which the algorithm does not correct.

Let the transmitted sequence be the allzero code sequence $\boldsymbol{v} = \mathbf{0} = 00\ldots0$ sent over the BSC and assume that all symbols of the received sequence $\boldsymbol{r}$ except r_7 are correctly received, that is, $\boldsymbol{r} = 000000010000000$. The decoder calculates the syndrome (cf. Fig. 1.11)

$$\boldsymbol{s} = \boldsymbol{r}H^{\mathrm{T}} = 010010100 \tag{1.80}$$

It has Hamming weight 3. The initial value of the threshold is $T = J = 3$. Note that flipping, for example, the symbol r_4 decreases the syndrome weight by 1, but the only symbol included in $\delta = 3$ unsatisfied parity-check equations is the symbol r_7. Flipping this symbol decreases the syndrome weight by 3 and results in successful decoding.

It is easily shown that the decoder corrects all single errors. It does not, however, correct all double errors. Suppose that all symbols of the received sequence $\boldsymbol{r}$ except r_4 and r_7 are correctly received, that is, $\boldsymbol{r} = 000010010000000$. The syndrome

$$\boldsymbol{s} = \boldsymbol{r}H^{\mathrm{T}} = 100000100 \tag{1.81}$$

has Hamming weight 2 and there are no symbols such that flipping causes a decrease of the syndrome weight. The decoding has failed and the sequence $\boldsymbol{r} = 000010010000000$ is a trapping set.

Next we consider another pattern of double errors, namely, all symbols of the received sequence $\boldsymbol{r}$ except r_0 and r_{14} are correctly received, that is, $\boldsymbol{r} = 100000000000001$. The syndrome

$$\boldsymbol{s} = \boldsymbol{r}H^{\mathrm{T}} = 111000111 \tag{1.82}$$

has Hamming weight 6. Flipping the symbol r_0 yields the tentative syndrome

$$\boldsymbol{s}' = \boldsymbol{r}'H^{\mathrm{T}} = 000000111 \tag{1.83}$$

with Hamming weight 3. Then, flipping r_{14} yields the tentative syndrome $\boldsymbol{s}' = \mathbf{0}$ and results in successful decoding of this particular double-error pattern.

Since the minimum distance $d_{\min} = 6$, we would correct all double errors if we were using a maximum-likelihood decoder.

We have described a variant of Gallager's original bit-flipping algorithm. The Zyablov-Pinsker iterative decoding algorithm finds in each step all bits of the received sequence which are included in more than $\lfloor J/2 \rfloor$ unsatisfied parity-check equations and then flips simultaneously all these bits. A theoretical analysis of the Zyablov-Pinsker algorithm [ZPZ08] gives the following asymptotical lower bound for the error-correcting capability when $N \to \infty$:

$$t^{(\mathrm{ZP})} > \rho_{J,K}^{(\mathrm{ZP})} N \tag{1.84}$$

where the coefficient $\rho_{J,K}^{(\mathrm{ZP})}$ is much smaller than both the corresponding Gilbert-Varshamov parameter ρ_{GV} and the Gallager parameter $\rho_{J,K}$. For example, for $J = 9$ and $K = 10$ we have $\rho_{J,K}^{(\mathrm{ZP})} = 1.29 \times 10^{-3}$.

It can be shown that the decoding complexity of the Zyablov-Pinsker algorithm is upper-bounded by $O(N \log N)$.

In Chapter 8 we will consider a more powerful iterative decoding algorithm for LDPC codes called the *belief propagation* (BP) algorithm. This algorithm was also invented by Gallager [Gal62, Gal63] and provides, at the cost of higher decoding complexity, better error correction than the bit-flipping algorithm.

1.4 A FIRST ENCOUNTER WITH CONVOLUTIONAL CODES

Convolutional codes are often thought of as nonblock linear codes over a finite field, but it can be an advantage to treat them as block codes over certain infinite fields. We will postpone the precise definitions until Chapter 2 and instead begin by studying a simple example of a binary convolutional encoder (Fig. 1.12).

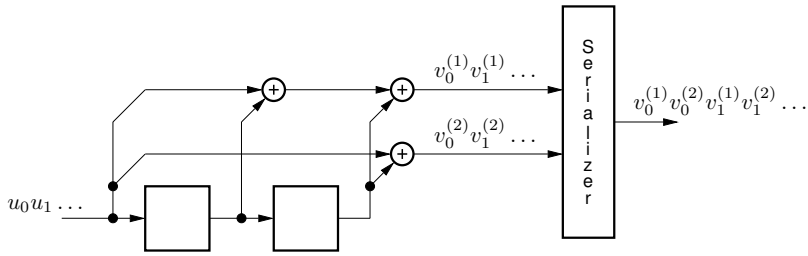


Figure 1.12 An encoder for a binary rate $R = 1/2$ convolutional code.

The information digits $\boldsymbol{u} = u_0 u_1 \ldots$ are not as in the previous section separated into blocks. Instead they form an infinite sequence that is shifted into a register, in our example, of length or *memory* $m = 2$. The encoder has two linear output functions. The two output sequences $\boldsymbol{v}^{(1)} = v_0^{(1)} v_1^{(1)} \ldots$ and $\boldsymbol{v}^{(2)} = v_0^{(2)} v_1^{(2)} \ldots$ are interleaved by a serializer to form a single-output sequence $v_0^{(1)} v_0^{(2)} v_1^{(1)} v_1^{(2)} \ldots$ that is transmitted over the channel. For each information digit that enters the encoder, two channel digits are emitted. Thus, the code rate of this encoder is $R = 1/2$ bits/channel use.

Assuming that the content of the register is zero at time $t = 0$, we notice that the two output sequences can be viewed as a convolution of the input sequence $\boldsymbol{u}$ and the two sequences $11100 \ldots$ and $10100 \ldots$, respectively. These latter sequences specify the linear output functions; that is, they specify the encoder. The fact that the output sequences can be described by convolutions is why such codes are called convolutional codes.

In a general rate $R = b/c$, where $b \leq c$, binary *convolutional encoder* (without feedback) the causal, that is, zero for time $t < 0$, information sequence

$$\boldsymbol{u} = \boldsymbol{u}_0\boldsymbol{u}_1 \ldots = u_0^{(1)}u_0^{(2)} \ldots u_0^{(b)}u_1^{(1)}u_1^{(2)} \ldots u_1^{(b)} \ldots \tag{1.85}$$

is encoded as the causal code sequence

$$\boldsymbol{v} = \boldsymbol{v}_0\boldsymbol{v}_1 \ldots = v_0^{(1)}v_0^{(2)} \ldots v_0^{(c)}v_1^{(1)}v_1^{(2)} \ldots v_1^{(c)} \ldots \tag{1.86}$$

where

$$\boldsymbol{v}_t = f\left((\boldsymbol{u}_t, \boldsymbol{u}_{t-1}, \ldots, \boldsymbol{u}_{t-m}\right) \tag{1.87}$$

The parameter m is called the encoder *memory*. The function f is required to be a *linear* function from $\mathbb{F}_2^{(m+1)b}$ to $\mathbb{F}_2^c$. It is often convenient to write such a function in matrix form:

$$\boldsymbol{v}_t = \boldsymbol{u}_t G_0 + \boldsymbol{u}_{t-1}G_1 + \cdots + \boldsymbol{u}_{t-m}G_m \tag{1.88}$$

where $G_i, 0 \leq i \leq m$, is a binary $b \times c$ matrix.

Using (1.88), we can rewrite the expression for the code sequence as

$$\boldsymbol{v}_0\boldsymbol{v}_1 \ldots = (\boldsymbol{u}_0\boldsymbol{u}_1 \ldots)\boldsymbol{G} \tag{1.89}$$

or, in shorter notation, as

$$\boldsymbol{v} = \boldsymbol{u}\boldsymbol{G} \tag{1.90}$$

where

$$\boldsymbol{G} = \begin{pmatrix} G_0 & G_1 & \ldots & G_m & & \\ & G_0 & G_1 & \ldots & G_m & \\ & & \ddots & \ddots & & \ddots \end{pmatrix} \tag{1.91}$$

and where here and hereafter the parts of matrices left blank are assumed to be filled in with zeros. We call $\boldsymbol{G}$ the *generator matrix* and $G_i, 0 \leq i \leq m$, the *generator submatrices*.

In Fig. 1.13, we illustrate a general convolutional encoder (without feedack).

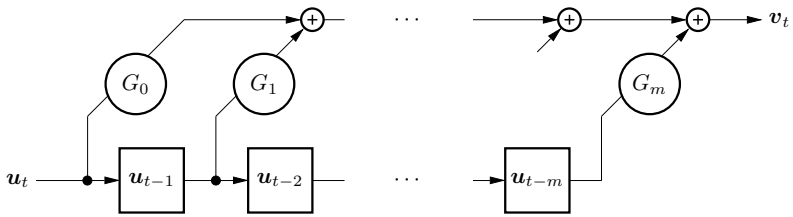


Figure 1.13 A general convolutional encoder (without feedback).

■ **EXAMPLE 1.17**

The rate $R = 1/2$ convolutional encoder shown in Fig. 1.12 has the following generator submatrices:

$$G_0 = (11) \tag{1.92}$$
$$G_1 = (10) \tag{1.93}$$
$$G_2 = (11) \tag{1.94}$$

The generator matrix is

$$\boldsymbol{G} = \begin{pmatrix} 11 & 10 & 11 & & \\ & 11 & 10 & 11 & \\ & & \ddots & \ddots & \ddots \end{pmatrix} \tag{1.95}$$

■ **EXAMPLE 1.18**

The rate $R = 2/3$ convolutional encoder shown in Fig. 1.14 has generator submatrices

$$\begin{aligned} G_0 &= \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix} \\ G_1 &= \begin{pmatrix} 1 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \\ G_2 &= \begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & 1 \end{pmatrix} \end{aligned} \tag{1.96}$$

The generator matrix is

$$\boldsymbol{G} = \begin{pmatrix} 101 & 110 & 000 & & \\ 011 & 001 & 101 & & \\ & 101 & 110 & 000 & \\ & 011 & 001 & 101 & \\ & & \ddots & \ddots & \ddots \end{pmatrix} \tag{1.97}$$

It is often convenient to represent the codewords of a convolutional code as paths through a *code tree*. A convolutional code is sometimes called a (linear) *tree code*. The code tree for the convolutional code generated by the encoder in Fig. 1.12 is shown in Fig. 1.15. The leftmost node is called the *root*. Since the encoder has one binary input, there are, starting at the root, two branches stemming from each node. The upper branch leaving each node corresponds to the input digit 0, and the lower branch corresponds to the input digit 1. On each branch we have two binary code digits, the two outputs from the encoder. The information sequence $1011\ldots$ is clearly seen from the tree to be encoded as the code sequence $11\,10\,00\,01\ldots$.

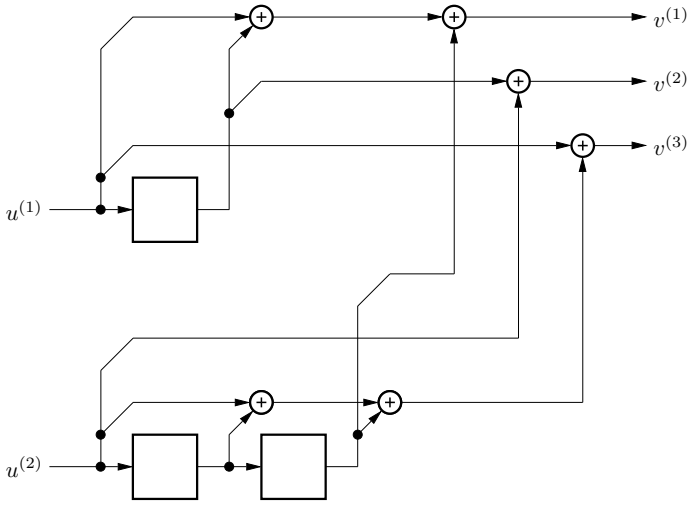


Figure 1.14 A rate $R = 2/3$ convolutional encoder.

The *state* of a system is a description of its past history which, together with a specification of the present and future inputs, suffices to determine the present and future outputs. For the encoder in Fig. 1.12, we can choose the encoder state $\boldsymbol{\sigma}$ to be the contents of its memory elements; that is, at time t we have

$$\boldsymbol{\sigma}_t = u_{t-1}u_{t-2} \tag{1.98}$$

Thus, our encoder has only four different encoder states, and two consecutive input digits are enough to drive the encoder to any specified encoder state.

For convolutional encoders, it is sometimes useful to draw the *state-transition diagram*. If we ignore the labeling, the state-transition diagram is a *de Bruijn graph* [Gol67]. In Fig. 1.16, we show the state-transition diagram for our convolutional encoder.

Let us return to the tree code in Fig. 1.15. As an example, the two input sequences 010 (node A) and 110 (node B) both drive the encoder to the same encoder state, $\boldsymbol{\sigma} = 01$. Thus, the two subtrees stemming from these two nodes are identical! Why treat them separately? We can replace them with one node corresponding to state 01 at time 3. For each time or *depth* in the tree, we can similarly replace all equivalent nodes with only one—we obtain the *trellis*-like structure shown in Fig. 1.17, where the upper and lower branches leaving the encoder states correspond to information symbols 0 and 1, respectively.

The information sequence $1011\ldots$ corresponds in the trellis to the same code sequence as in the tree, $11\,10\,00\,01\ldots$. The trellis is just a more convenient representation of the same set of encoded sequences as is specified by the tree, and it is easily constructed from the state-transition diagram. A convolutional code is often called a (linear) *trellis code*.

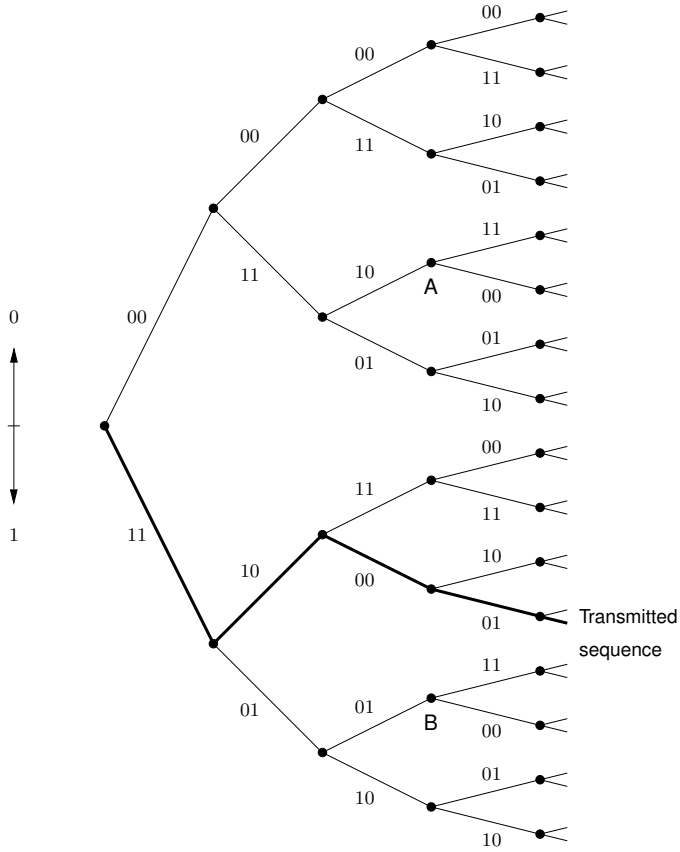


Figure 1.15 A binary rate $R = 1/2$ tree code.

We will often consider sequences of finite length; therefore, it is convenient to introduce the notations

$$\boldsymbol{x}_{[0,n)} = \boldsymbol{x}_0\boldsymbol{x}_1 \ldots \boldsymbol{x}_{n-1} \tag{1.99}$$

and

$$\boldsymbol{x}_{[0,n]} = \boldsymbol{x}_0\boldsymbol{x}_1 \ldots \boldsymbol{x}_n \tag{1.100}$$

Suppose that our trellis code in Fig. 1.17 is used to communicate over a BSC with crossover probability ϵ, where $0 < \epsilon < 1/2$. We start the encoder in encoder state $\boldsymbol{\sigma} = 00$, and feed it with the finite information sequence $\boldsymbol{u}_{[0,n)}$ followed by $m = 2$ dummy zeros in order to drive the encoder back to encoder state $\boldsymbol{\sigma} = 00$. The convolutional code is terminated and, thus, converted into a block code. The corresponding encoded sequence is the codeword $\boldsymbol{v}_{[0,n+m)}$. The received sequence is denoted $\boldsymbol{r}_{[0,n+m)}$.

To simplify the notations in the following discussion, we simply write $\boldsymbol{u}$, $\boldsymbol{v}$, and $\boldsymbol{r}$ instead of $\boldsymbol{u}_{[0,n)}$, $\boldsymbol{v}_{[0,n+m)}$, and $\boldsymbol{r}_{[0,n+m)}$.

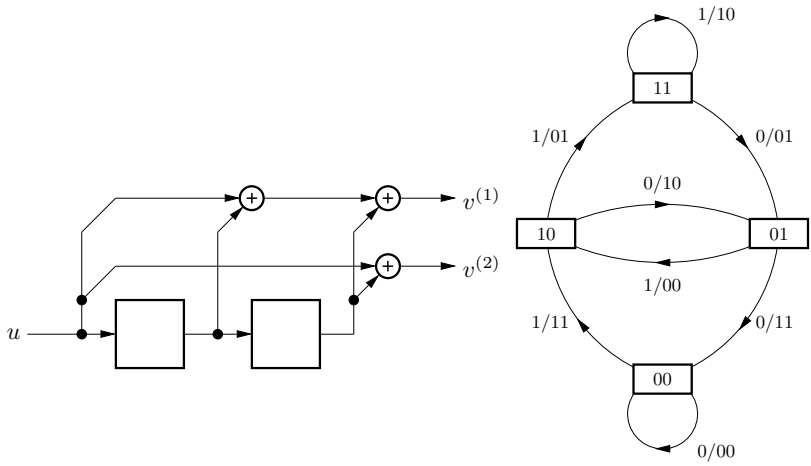


Figure 1.16 A rate $R = 1/2$ convolutional encoder and its state-transition diagram.

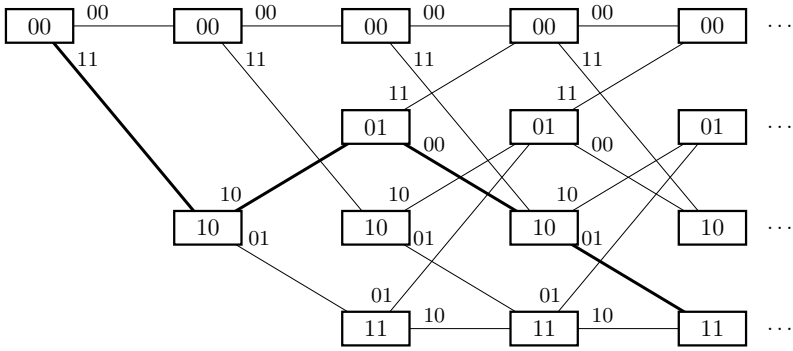


Figure 1.17 A binary rate $R = 1/2$ trellis code.

We shall now, by an example, show how the structure of the trellis can be exploited to perform maximum-likelihood (ML) decoding in a very efficient way. The memory $m = 2$ encoder in Fig. 1.12 is used to encode three information digits together with $m = 2$ dummy zeros; the trellis is terminated and our convolutional code has become a block code. A codeword consisting of 10 code digits is transmitted over a BSC. Suppose that $\boldsymbol{r} = 11\,00\,11\,00\,10$ is received. The corresponding trellis is shown in Fig. 1.18. (In practice, typically a few thousand information bits are encoded before the encoder is forced back to the allzero state by encoding m dummy zeros.)

As shown by the discussion following (1.31), the ML decoder (and the MD decoder) chooses as its decision $\widehat{\boldsymbol{v}}$ for the codeword $\boldsymbol{v}$ that minimizes the Hamming distance $d_{\mathrm{H}}(\boldsymbol{r}, \boldsymbol{v})$ between $\boldsymbol{r}$ and $\boldsymbol{v}$. That is, it minimizes the number of positions in which the codeword and the received sequence differ. In order to find the codeword that is closest to the received sequence, we move through the trellis from left to

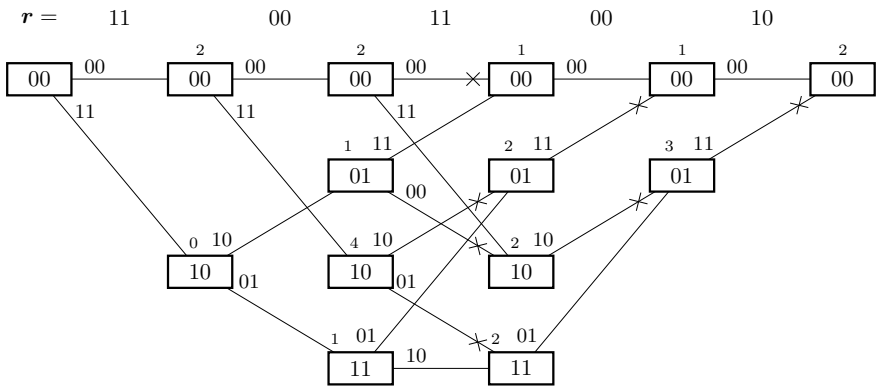


Figure 1.18 An example of Viterbi decoding for the received sequence $\boldsymbol{r} = 11\,00\,11\,00\,10$.

right, discarding *all* subpaths that could not turn out to be the prefix of the best path through the trellis. When we reach depth $m = 2$, we have four subpaths—one for each encoder state. At the next depth, however, there are eight subpaths—two per encoder state. For each encoder state at this depth, we keep only one subpath—the one that is closest to the corresponding prefix of the received sequence. We simply discard the poorer subpath into each encoder state since this poorer subpath could not possibly be the prefix of the best path through the trellis. We proceed in this manner until we reach encoder state 00 at depth 5. Because only one path through the trellis has survived, we have then found the best path through the trellis. In Fig. 1.18, the Hamming distance between the prefix of the received sequence and the best subpath leading to each encoder state is shown above the encoder state. The discarded poorer subpath is marked with the symbol $\times$ on the branch that enters the encoder state.

Two subpaths leading to an encoder state may both have the same Hamming distance to the prefix of the received sequence. In fact, this happened at state 01, depth 4. Both subpaths have distance 3 to the prefix of the received sequence! Both are equally likely to be the prefix of the best path—we can discard either subpath without eliminating all "best paths" through the trellis, in case there are more than one best path. We arbitrarily chose to discard the upper of the two subpaths entering encoder state 01 at depth 4.

The best codeword through the trellis was found to be $\widehat{\boldsymbol{v}} = 11\,10\,11\,00\,00$, which corresponds to the information sequence $\widehat{\boldsymbol{u}} = 100$. If the decision $\widehat{\boldsymbol{v}} = 11\,10\,11\,00\,00$ happened to be the transmitted codeword, we have corrected two transmission errors.

How many errors can we correct?

The most likely error event is that the transmitted codeword is changed by the BSC so that it is decoded as its closest (in Hamming distance) neighbor. It is readily seen from Fig. 1.18 that the smallest Hamming distance between any two different codewords is 5, for example, $d_{\mathrm{H}}\,(00\,00\,00\,00\,00,\ 11\,10\,11\,00\,00) = 5$. This minimum distance is called the *free distance* of the convolutional code and is denoted d_{free}. It is the single most important parameter for determining the error-correcting

capability of the code. (The free distance and several other distance measures will be discussed in detail in Chapter 3.) Since $d_{\text{free}} = 5$, we can correct all patterns of two errors.

The ML decoding algorithm described above is usually called the *Viterbi algorithm* in honor of its inventor [Vit67]. It is as simple as it is ingenious, and it is easily implementable. Viterbi decoders for memory $m = 6$ (64 states) and longer are often used in practice.

In Chapter 4, we will study Viterbi decoding in more detail and obtain tight upper bounds on the decoded bit error probability.

1.5 BLOCK CODES VERSUS CONVOLUTIONAL CODES

The system designer's choice between block and convolutional codes should depend on the application. The diehard block code supporters always advocate in favor of block codes, while their counterparts on the other side claim that in almost all situations convolutional codes outperform block codes. As always, the "truth" is not only somewhere in between, but it also depends on the application.

The theory of block codes is much richer than the theory of convolutional codes. Many sophisticated finite field concepts have been used to design block codes with a beautiful mathematical structure that has simplified the development of efficient error-correcting decoding algorithms. From a practical point of view, the Reed-Solomon (RS) codes constitute the most important family of block codes. They are extremely well suited for digital implementation. Berlekamp's bit-serial RS encoders [Ber82] have been adopted as a NASA standard for deep-space communication. The RS codes are particularly powerful when the channel errors occur in clusters—*burst errors*—which is the case in secondary memories such as magnetic tapes and disks. All compact disc players use RS codes with table-look-up decoding.

Assuming that a decoded bit error rate of 10^{-5} is satisfactory, which is the case, for example, for digitized voice, convolutional codes in combination with Viterbi decoding appear to be an extremely good combination for communication when the noise is white and Gaussian. For example, Qualcomm Inc. has on a single chip implemented a memory $m = 6$ Viterbi decoder for rates $R = 1/3, 1/2, 3/4, 7/8$. The rate $R = 1/2$ coding gain is 5.2 dB at $P_{\text{b}} = 10^{-5}$. This very powerful error-correcting system operates either with hard decisions or with eight-level quantized soft decisions.

The major drawback of RS codes is the difficulty of making full use of soft-decision information. As we will see in Chapter 4, the Viterbi algorithm can easily exploit the full soft-decision information provided at the decoder input and thus easily pick up the 2 dB gain over hard-decision. Furthermore, code synchronization is in general much simpler for convolutional codes than for block codes.

If a combination of a high level of data integrity, $P_{\text{b}} = 10^{-10}$ say, and a larger coding gain than a Viterbi decoder can provide is required, then we could use either an RS code or a large memory, $m = 25$ say, convolutional encoder. The complexity of the Viterbi decoder, which is essentially 2^m, will be prohibitively large in the latter

case. Instead we could use *sequential decoding* (Chapter 7) of the convolutional code whose complexity is essentially independent of the memory of the encoder.

In many applications where the noise is predominantly Gaussian, the best solution is obtained when block and convolutional codes join forces and are used in series. In Fig. 1.19 we show a *concatenated coding* system, where we use a convolutional code as the *inner code* to clean up the channel. The Viterbi decoder will correct most channel errors but will occasionally output a burst of errors. This output then becomes the input to the outer decoder. Since an RS code is very well suited to cope with bursts of errors, we use an RS code as the *outer code*. Such a concatenated coding system combines a very high level of data integrity with large coding gain and low complexity. Often a *permutor* is used between the outer and inner encoders and a corresponding *inverse permutor* between the inner and outer decoders. Then the output error burst from the inner decoder will be smeared out by the inverse permutor before the outer decoder has to cope with it.

An alternative method of decreasing the decoding complexity without decreasing the code reliability is using low-density parity-check (LDPC) codes, a class of codes on graphs, or so-called turbo codes. In Chapter 8 we discuss LDPC convolutional codes and in Chapter 9 we introduce turbo codes.

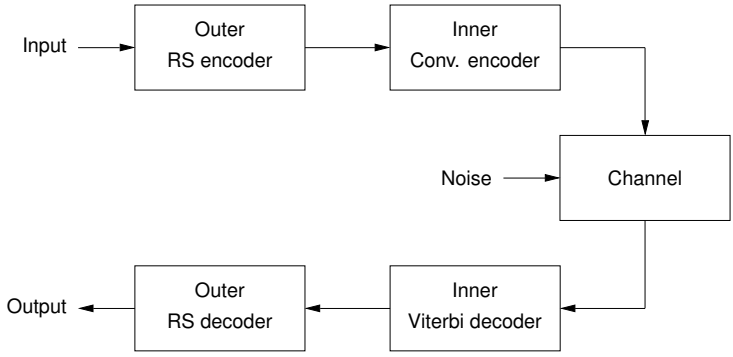


Figure 1.19 Concatenated coding system.

1.6 CAPACITY LIMITS AND POTENTIAL CODING GAIN REVISITED

We will now return to the problem of determining the regions of potential coding gain which we first encountered in Section 1.1.

Consider Shannon's formula for the capacity of the bandlimited Gaussian channel (1.15),

$$C_t^W = W \log \left(1 + \frac{S}{N_0 W}\right) \text{ bits/s}$$

where W as before denotes the bandwidth. Assume that we are transmitting at the so-called *Nyquist rate*, (that is, at a rate of $2W$ samples per second) and that we use

a rate $R = K/N$ block code. If we transmit K information bits during $\mathcal{T}$ seconds, we have

$$N = 2W\mathcal{T} \text{ samples per codeword} \tag{1.101}$$

Hence,

$$R_t = K/\mathcal{T} = 2WK/N = 2WR \text{ bits/s} \tag{1.102}$$

(Assuming a constant transmission rate R_t, the required bandwidth W is inversely proportional to the code rate R.)

By combining (1.18) and (1.102), we obtain

$$\frac{S}{WN_0} = \frac{2RE_\mathrm{b}}{N_0} \tag{1.103}$$

For reliable communication, we must have $R_t \leq C_t^W$, that is,

$$R_t = 2WR \leq W \log\left(1 + \frac{2RE_\mathrm{b}}{N_0}\right) \tag{1.104}$$

or, equivalently,

$$\frac{E_\mathrm{b}}{N_0} \geq \frac{2^{2R} - 1}{2R} \tag{1.105}$$

Letting $R \to 0$, we obtain (1.20). Since the right hand side of inequality (1.105) is increasing with R, we notice that in order to communicate close to the Shannon limit, -1.6 dB, we have to use both an information rate R_t and a code rate R close to zero. Furthermore, if we use a rate $R = 1/2$ code, it follows from (1.105) that the required signal-to-noise ratio is

$$E_\mathrm{b}/N_0 \geq 1 = 0 \text{ dB} \tag{1.106}$$

In Fig. 1.20 we show the coding limits according to (1.105) and the regions of potential coding gain for various rates R.

When we derived the coding limits determined by inequality (1.105), we assumed a required error probability P_b arbitrarily close to zero. If we are willing to tolerate a certain given value of the error probability P_b, we can of course obtain a larger coding gain. It follows from Shannon's *rate distortion theory* [McE77] that if we are transmitting the output of a binary symmetric source and can tolerate an average distortion of KP_b for a block of K information symbols, then we can represent R_t bits of information per second with only $R_t(1 - h(P_\mathrm{b}))$ bits per second, where $h(\cdot)$ is the binary entropy function (1.22).

These $R_t(1 - h(P_\mathrm{b}))$ bits per second should now be transmitted over the channel with an error probability arbitrarily close to zero. Hence, instead of (1.104) we have now the inequality

$$R_t(1 - h(P_\mathrm{b})) = 2WR(1 - h(P_\mathrm{b})) \leq W \log\left(1 + \frac{2RE_\mathrm{b}}{N_0}\right) \tag{1.107}$$

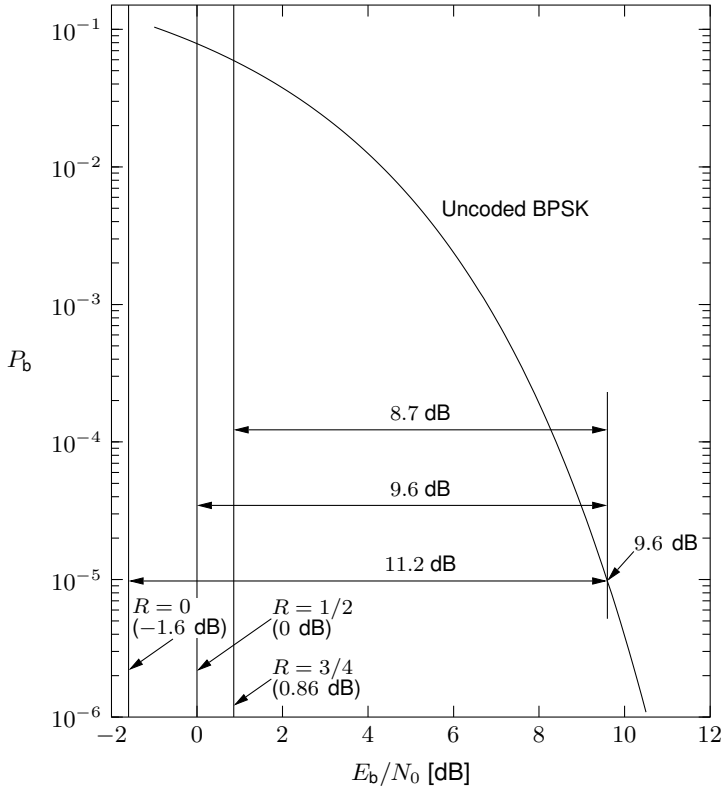


Figure 1.20 Coding limits and regions of potential coding gains for various rates R.

or, equivalently,

$$\frac{E_b}{N_0} \geq \frac{2^{2R(1-h(P_b))} - 1}{2R} \tag{1.108}$$

In Fig. 1.21 we show the coding limits according to (1.108) and the regions of potential coding gains for various rates R when we can tolerate the bit error probability P_b. We also show a comparison between these coding limits and Qualcomm's Viterbi decoder performance and that of a rate $R = 3/4$ $(256, 192)$ RS decoder.

In order to achieve the rate distortion bound we need a nonlinear source encoder. Hence, we have *not* shown that the coding limit (1.108) can be reached with *linear* codes.

Remark: In order to achieve the capacity C_t^W promised by (1.15), we have to use nonquantized inputs to the channel. If we restrict ourselves to the binary input Gaussian channel, then the formula for C_t^W, (1.15), must be replaced by a more complicated expression and the coding limits shown in Fig. 1.21 should be shifted to the right by a small fraction of a dB [BMc74].

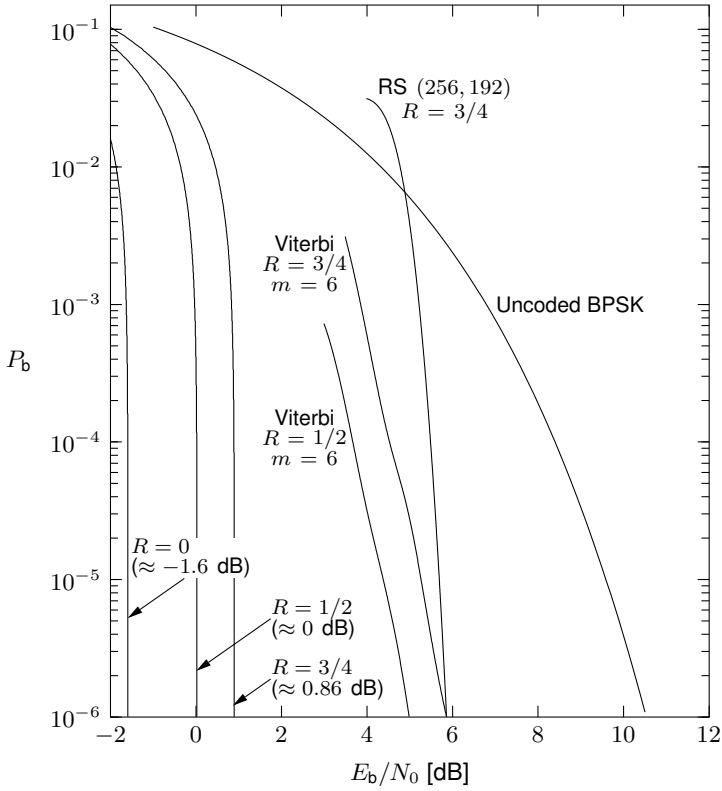


Figure 1.21 Regions of potential coding gains for various rates R when we can tolerate bit error probability P_b and a comparison with the performance of two convolutional codes and a block code.

1.7 COMMENTS

Back in 1947 when Hamming had access to a computer only on weekends, he was very frustrated over its behavior: "Damn it, if the machine can detect an error, why can't it locate the position of the error and correct it?" [Tho83]. That question inspired the development of error-correcting codes. Hamming's famous single-error-correcting $(7, 4)$ block code is mentioned by Shannon in "A Mathematical Theory of Communication" [Sha48], but Hamming's paper was not published until two years later [Ham50]. The first paper published in coding theory was that of Golay [Gol49], which in less than one page gave the generalization of Hamming codes to nonbinary fields, gave the only two multi-error-correcting perfect codes aside from the trivial binary repetition codes of odd length, and introduced the parity-check matrix (see also Problem 1.19).

Elias introduced convolutional codes in 1955 [Eli55]. The first decoding method for these codes was sequential decoding suggested by Wozencraft in 1957 [Woz57]

and further developed by Fano, who in 1963 presented a most ingenious decoding algorithm [Fan63]. The conceptually simplest algorithm for sequential decoding is the stack algorithm introduced by Zigangirov in 1966 [Zig66] and Jelinek in 1969 [Jel69]. In the meantime, Massey had suggested threshold decoding of convolutional codes [Mas63]. In Viterbi's famous paper from 1967 [Vit67], the Viterbi algorithm was invented as a proof technique and presented as "a new probabilistic nonsequential decoding algorithm". Forney [For67] was the first to draw a trellis and it was he who coined the name "trellis," which made understanding of the Viterbi algorithm easy and its maximum-likelihood nature obvious. Forney realized that the Viterbi algorithm was optimum, but it was Heller who realized that it was practical [For94]. Later, Omura [Omu69] observed that the Viterbi algorithm can be viewed as the application of dynamic programming to the problem of decoding a convolutional code.

The most important contributions promoting the use of convolutional codes were made by Jacobs and Viterbi when they founded Linkabit Corporation in 1968 and Qualcomm Inc. in 1985, completing the path "from a proof to a product" [Vit90].

LDPC block codes were invented by Gallager [Gal62, Gal63] in the early 1960s. Unfortunately, Gallager's remarkable discovery was to a large extent ignored by the coding community during almost 20 years. Two important papers by Zyablov and Pinsker [ZyP74, ZyP75] published in the middle of the 1970s were overlooked by many coding theoretists. In the beginning of the 1980s Tanner [Tan81] and Margulis [Mar82] published two important papers concerning LDPC codes. Tanner's work provided a new interpretation of LDPC codes from a graph theoretical point of view. Margulis gave explicit graph constructions of the codes. These works were also essentially ignored by the coding specialists for more than 10 years, until the beginning 1990s when Berrou, Glavieux, and Thitimajshima [BGT93] introduced the so-called *turbo codes* which inspired many coding researchers to investigate codes on graphs and iterative decoding. It has been shown that long LDPC codes with iterative decoding based on belief propagation almost achieve the Shannon limit. This rediscovery makes the LDPC codes strong competitors with other codes for error control in many communication and digital storage systems when high reliability is required.

PROBLEMS

1.1 The channel capacity for the ideal bandlimited AWGN channel of bandwidth W with two-sided noise power spectral density $N_0/2$ is given by (1.15). The signal power can be written $S = E_\mathrm{b} R_t$.

Define the *spectral bit rate* r by

$$r = R_t/W \text{ (bits/s)/Hz}$$

and show that

$$\frac{E_\mathrm{b}}{N_0} \geq \frac{2^r - 1}{r}$$

for rates R_t less than capacity. Sketch r as a function of E_b/N_0 expressed in dB.

1.2 Consider an ideal bandlimited AWGN channel with BPSK and with hard decisions. Based on transmitting $R_t = 2WR$ bits/s, where R is the code rate, the capacity is

$$C_t = 2W(1 + \epsilon \log \epsilon + (1-\epsilon)\log(1-\epsilon)) \text{ bits/s}$$

where $\epsilon = Q\left(\sqrt{rE_\mathrm{b}/N_0}\right)$ and r is the spectral bit rate $r = R_t/W$.

Show that

$$\frac{E_\mathrm{b}}{N_0} \geq \frac{\pi}{2} \ln 2$$

for reliable communication.

Hint: The Taylor series expansion of C_t is

$$C_t = 2W\left(\frac{2^2}{1\cdot 2}\left(\tfrac{1}{2}-\epsilon\right)^2 + \frac{2^4}{3\cdot 4}\left(\tfrac{1}{2}-\epsilon\right)^4 + \frac{2^6}{5\cdot 6}\left(\tfrac{1}{2}-\epsilon\right)^6 + \cdots\right)\log e$$

and

$$\epsilon = Q\left(\sqrt{rE_\mathrm{b}/N_0}\right) \geq \frac{1}{2} - \frac{1}{\sqrt{2\pi}}\sqrt{rE_\mathrm{b}/N_0}$$

1.3 Show that a block code $\mathcal{B}$ can detect all patterns of s or fewer errors if and only if $d_\mathrm{min} > s$.

1.4 Show that a block code $\mathcal{B}$ can correct all patterns of t or fewer errors and simultaneously detect all patterns of $t+1, t+2, \ldots, t+s$ errors if and only if $d_\mathrm{min} > 2t + s$.

1.5 Prove Theorem 1.2.

1.6 Consider a block code $\mathcal{B}$ with encoding matrix

$$G = \begin{pmatrix} 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 \end{pmatrix}$$

a) List all codewords.

b) Find a systematic encoding matrix and its parity-check matrix.

c) Determine $d_{\min}$.

1.7 Consider the following two block codes.

$\mathcal{B}_1 = \{110011, 101010, 010101, 011001, 100110, 111111, 001100, 000000\}$

$\mathcal{B}_2 = \{010101, 101010, 001100, 110110, 111111, 011001, 110011, 100110\}$

a) Are the two codes linear?

b) Determine $w_{\min}$ for each of the codes.

c) Determine $d_{\min}$ for each of the codes.

d) Determine the rate $R = K/N$.

1.8 Consider the block code $\mathcal{B} = \{000000, 110110, 011011, 101101\}$.

a) Is $\mathcal{B}$ linear?

b) Find the rate $R = K/N$.

c) Find, if it exists, a linear encoder.

d) Find, if it exists, a nonlinear encoder.

e) Determine $d_{\min}$.

1.9 Show that if $\mathcal{B}$ is a linear code and $\boldsymbol{a} \notin \mathcal{B}$, then $\mathcal{B} \cup (\boldsymbol{a} + \mathcal{B})$ is also a linear code.

1.10 Consider the binary $(6, K)$ *even-weight* code. (All codewords have even weight.)

a) Find K.

b) Give the encoding and parity-check matrices.

1.11 Consider the binary (4,3) even-weight code.

a) Construct a standard array.

b) For each coset give its syndrome.

c) How many errors can it correct?

d) Determine $d_{\min}$.

1.12 Show that a binary code can correct all single errors if and only if any parity-check matrix has distinct nonzero columns.

1.13 Consider a binary code with encoding matrix

$$G = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 \end{pmatrix}$$

a) Find a parity-check matrix.

b) Construct a standard array.

c) List all codewords.

d) Determine from the standard array how many errors it can correct.

e) Determine $d_{\min}$.

f) For each coset give its syndrome.

g) Suppose that $\boldsymbol{r} = 110000$ is received over a BSC with $0 < \epsilon < 1/2$. Find the maximum-likelihood decision $\widehat{\boldsymbol{u}}$ for the information sequence.

1.14 Consider a block code $\mathcal{B}$ with encoding matrix

$$G = \begin{pmatrix} 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 \end{pmatrix}$$

a) Find a parity-check matrix.
b) List all codewords.
c) Determine $d_{\min}$.
d) Suppose that $\boldsymbol{r} = 000111$ is received over a BSC with $0 < \epsilon < 1/2$. Find the maximum-likelihood decision $\widehat{\boldsymbol{u}}$ for the information sequence.

1.15 Consider a binary (N, K) code $\mathcal{B}$ with parity-check matrix H and minimum distance d. Assume that some of its codewords have odd weight. Form a code $\widehat{\mathcal{B}}$ by concatenating a 0 at the end of every codeword of even weight and a 1 at the end of every codeword of odd weight. This technique is called *extending* a code.
a) Determine $d_{\min}$ for $\widehat{\mathcal{B}}$.
b) Give a parity-check matrix for the extended code $\widehat{\mathcal{B}}$.

1.16 Consider the (8,4) extended Hamming code.
a) Give a parity-check matrix.
b) Determine $d_{\min}$.
c) Find an encoding matrix.
d) Show how a decoder can detect that an odd number of errors has occurred.

1.17 The *Hamming sphere* of radius t with center at the N-tuple $\boldsymbol{x}$ is the set of all $\boldsymbol{y}$ in $\mathbb{F}_2^N$ such that $d_{\mathrm{H}}(\boldsymbol{x}, \boldsymbol{y}) \leq t$. Thus, this Hamming sphere contains exactly

$$V_t = \sum_{i=0}^{t} \binom{N}{i}$$

distinct N-tuples. Prove the *Hamming bound* for binary codes, that is,

$$V_{\left\lfloor \frac{d_{\min}-1}{2} \right\rfloor} \leq 2^{N(1-R)}$$

which is an implicit upper bound on $d_{\min}$ in terms of the block length N and rate R.

1.18 The systematic parity-check matrices for the binary Hamming codes can be written recursively as

$$H_2 = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}$$

and

$$H_m = \begin{pmatrix} H_{m-1} & H_{m-1} & \mathbf{0} \\ 1 \ldots 1 & 0 \ldots 0 & 1 \end{pmatrix}, \quad m \geq 3$$

Find the parameters N, K, and $d_{\min}$ for the mth Hamming code.

1.19 A code for which the Hamming bound (see Problem 1.17) holds with equality is called a *perfect code*.

a) Show that the *repetition code*, that is, the rate $R = 1/N$ binary linear code with generator matrix $G = (1\,1\,\ldots\,1)$, is a perfect code if and only if N is odd.

b) Show that the Hamming codes of Problem 1.18 are perfect codes.

c) Show that the Hamming bound admits the possibility that an $N = 23$ perfect binary code with $d_{\rm dim} = 7$ might exist. What must K be?

Remark: The perfect code suggested in Problem 1.19(c) was found by Golay in 1949 [Gol49]. There exist no perfect binary codes other than those mentioned in this problem.

1.20 Suppose that the block code $\mathcal{B}$ with parity-check matrix

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

is used for communication over a BSC with $0 < \epsilon < 1/2$.

a) Find $d_{\rm min}$.

b) How many errors can the code correct?

c) How many errors can the code detect?

d) For each syndrome give the error pattern $\widehat{\boldsymbol{e}}$ that corresponds to the error-correcting capability of the code.

e) For $\boldsymbol{r} = 0111011$ find $\widehat{\boldsymbol{v}}$, the maximum-likelihood decision.

f) For $\boldsymbol{r} = 0110111$ find $\widehat{\boldsymbol{v}}$, the maximum-likelihood decision.

1.21 Verify formula (1.73).

Hint: The $(7, 4)$ Hamming code has one codeword of weight 0, seven codewords of weight 3, seven codewords of weight 4, and one codeword of weight 7. The error probability is the same for all bits.

1.22 Given a Hamming code $\mathcal{B}$ with parity-check matrix H.

a) Construct an *extended* code $\mathcal{B}_{\rm ext}$ with parity-check matrix

$$H_{\rm ext} = \begin{pmatrix} 0 & & & \\ \vdots & & H & \\ 0 & & & \\ 1 & 1 & \cdots & 1 \end{pmatrix}$$

b) Determine $d_{\rm min}$ for $\mathcal{B}_{\rm ext}$.

c) Construct an *expurgated* code $\mathcal{B}_{\rm exp}$ with parity-check matrix

$$H_{\rm exp} = \begin{pmatrix} & H & \\ & & \\ 1 & \cdots & 1 \end{pmatrix}$$

d) Determine $d_{\rm min}$ for $\mathcal{B}_{\rm exp}$.

e) What is characteristic for the weights of the codewords of $\mathcal{B}_{\text{exp}}$?

1.23 Draw the Tanner graph for the extended $(8, 4)$ Hamming code defined by the parity-check matrix

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

1.24 Show that the parity-check matrix (1.77) has rank less than L.

1.25 Consider the code given in Example 1.15. Show that the bit-flipping algorithm with lowest possible threshold does not correct all single errors.

1.26 Consider the extended $(8, 4)$ Hamming code defined in Problem 1.23 and suppose that it is used to communicate over the BSC. Show that the bit-flipping algorithm with adaptive threshold corrects all single errors and that there exists a double error which the algorithm does not correct.

1.27 Consider the binary input, ternary output *binary erasure channel* (BEC) given in Fig. 1.22, where Δ denotes an erasure and δ is the probability of an erasure. Assume that we use this channel for communication together with maximum-likelihood (ML) decoding. Show that the ML decoding algorithm corrects all erasure patterns whose Hamming weights are less than the minimum distance $d_{\min}$.

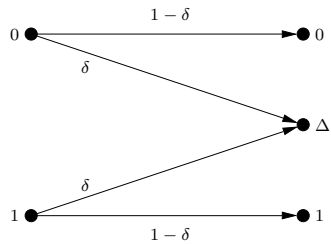


Figure 1.22 BEC used in Problem 1.27.

1.28 Consider the trellis given in Fig. 1.18.

a) List all codewords.

b) Find the ML estimate of the information sequence for the received sequence $\boldsymbol{r} = 01\,10\,01\,10\,11$ on a BSC with $0 < \epsilon < 1/2$.

1.29 Consider the convolutional encoder shown in Fig. 1.23.

a) Draw the trellis corresponding to four information digits and $m = 1$ dummy zero.

b) Find the number of codewords M represented by the trellis in Problem 1.29(a).

c) Use the Viterbi algorithm to decode when the sequence $\boldsymbol{r} = 11\,01\,10\,10\,01$ is received over a BSC with $0 < \epsilon < 1/2$.

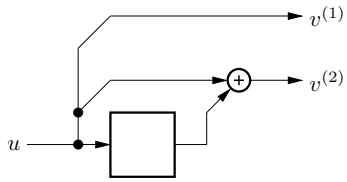


Figure 1.23 Convolutional encoder used in Problem 1.29.

1.30 Consider the convolutional encoder with generator matrix

$$\boldsymbol{G} = \begin{pmatrix} 11 & 10 & 01 & 11 & & \\ & 11 & 10 & 01 & 11 & \\ & & \ddots & \ddots & \ddots & \ddots \end{pmatrix}$$

a) Find the rate and the memory.
b) Draw the encoder.
c) Find the codeword $\boldsymbol{v}$ that corresponds to the information sequence $\boldsymbol{u} = 1100100\ldots$.

1.31 Consider the code $\mathcal{C}$ with the encoding rule

$$\boldsymbol{v} = \boldsymbol{u}\boldsymbol{G} + (11\,01\,11\,10\,11\ \ldots)$$

where

$$\boldsymbol{G} = \begin{pmatrix} 11 & 10 & 01 & 11 & & \\ & 11 & 10 & 01 & 11 & \\ & & \ddots & \ddots & \ddots & \ddots \end{pmatrix}$$

a) Is the code $\mathcal{C}$ linear?
b) Is the encoding rule linear?

1.32 Consider the rate $R = 2/3$, memory $m = 2$ convolutional encoder illustrated in Fig. 1.14.

a) Draw the trellis diagram.
b) Find the encoder matrix $\boldsymbol{G}$.
c) Let $\boldsymbol{u} = 10\,11\,01\,10\,00\,00\ldots$ be the information sequence. Find the corresponding codeword $\boldsymbol{v}$.

1.33 Plot in Fig. 1.21 the bit error probability for the $(7, 4)$ Hamming code when used to communicate over the Gaussian channel with hard decisions.
Hint: From formula (1.12), that is, $\epsilon = Q\left(\sqrt{2E_{\mathrm{s}}/N_0}\right)$, where $E_{\mathrm{s}} = RE_{\mathrm{b}}$, we obtain the following table:

E_s/N_0 [dB]	ϵ
0	$0.79 \cdot 10^{-1}$
2	$0.38 \cdot 10^{-1}$
4	$0.12 \cdot 10^{-1}$
6	$0.24 \cdot 10^{-2}$
8	$0.19 \cdot 10^{-3}$
10	$0.39 \cdot 10^{-5}$
12	$0.90 \cdot 10^{-8}$

CHAPTER 2

CONVOLUTIONAL ENCODERS—STRUCTURAL PROPERTIES

After defining convolutional codes and convolutional encoders, we show that a given convolutional code can be encoded by many different encoders. We carefully distinguish code properties from encoder properties. The Smith form of a polynomial matrix is used to obtain important structural results for convolutional encoders. We give several equivalent conditions for an encoding matrix to be minimal— that is, to be realizable by as few memory elements as any encoder for the code, though not necessarily in controller or observer canonical forms. We also show that a systematic encoding matrix is always minimal.

2.1 CONVOLUTIONAL CODES AND THEIR ENCODERS

In general, the rate $R = b/c$, $b \leq c$, convolutional encoder input (information) sequence $\boldsymbol{u} = \ldots \boldsymbol{u}_{-1}\boldsymbol{u}_0\boldsymbol{u}_1\boldsymbol{u}_2 \ldots$, where $\boldsymbol{u}_i = (u_i^{(1)}u_i^{(2)} \ldots u_i^{(b)})$, and output (code) sequence $\boldsymbol{v} = \ldots \boldsymbol{v}_{-1}\boldsymbol{v}_0\boldsymbol{v}_1\boldsymbol{v}_2 \ldots$, where $\boldsymbol{v}_j = (v_j^{(1)}v_j^{(2)} \ldots v_j^{(c)})$, must start at some finite time (positive or negative) and may or may not end. It is often convenient to express them in terms of the delay operator D (D-transforms):

$$\boldsymbol{u}(D) = \cdots + \boldsymbol{u}_{-1}D^{-1} + \boldsymbol{u}_0 + \boldsymbol{u}_1 D + \boldsymbol{u}_2 D^2 + \cdots \tag{2.1}$$

$$\boldsymbol{v}(D) = \cdots + \boldsymbol{v}_{-1}D^{-1} + \boldsymbol{v}_0 + \boldsymbol{v}_1 D + \boldsymbol{v}_2 D^2 + \cdots \tag{2.2}$$

Fundamentals of Convolutional Coding, 2. Edition. By Rolf Johannesson, Kamil Sh. Zigangirov

In the sequel, we will not distinguish between a sequence and its D-transform.

Let $\mathbb{F}_2((D))$ denote the *field of binary Laurent series*. The element $x(D) = \sum_{i=r}^{\infty} x_i D^i \in \mathbb{F}_2((D)),\ r \in \boldsymbol{Z}$, contains at most finitely many negative powers of D. The *delay* of a Laurent series is the "time index" at which the Laurent series starts. For example,

$$x(D) = D^{-2} + 1 + D^3 + D^7 + D^{12} + \cdots \tag{2.3}$$

is a Laurent series with delay

$$\operatorname{del} x(D) = -2 \tag{2.4}$$

Let $\mathbb{F}_2[[D]]$ denote the *ring of formal power series*. The element $f(D) = \sum_{i=0}^{\infty} f_i D^i \in \mathbb{F}_2[[D]]$ is a Laurent series without negative powers of D. Thus, $\mathbb{F}_2[[D]]$ is a subset of $\mathbb{F}_2((D))$. The element $f(D) = \sum_{i=0}^{\infty} f_i D^i$ with $f_0 = 1$ has delay $\operatorname{del} f(D) = 0$ and is called *delayfree*.

A *polynomial* $p(D) = \sum_{i=0}^{\infty} p_i D^i$ contains no negative and only finitely many positive powers of D. If $p_0 = 1$, we have a delayfree polynomial, for example,

$$p(D) = 1 + D^2 + D^3 + D^5 \tag{2.5}$$

is a binary delayfree polynomial of degree 5.

The set of binary polynomials $\mathbb{F}_2[D]$ is a subset of $\mathbb{F}_2[[D]]$ and, hence, a subset of $\mathbb{F}_2((D))$. Multiplication of two polynomials is ordinary polynomial multiplication with coefficient operations performed modulo 2. Since 1 is the only polynomial with a polynomial as its multiplicative inverse, that is, 1 itself, $\mathbb{F}_2[D]$ cannot be a field. It is a ring—the *ring of binary polynomials*.

Given any pair of polynomials $x(D), y(D) \in \mathbb{F}_2[D]$, with $y(D) \neq 0$, we can obtain the element $x(D)/y(D) \in \mathbb{F}_2((D))$ by long division. Since sequences must start at some finite time, we must identify, for instance, $(1 + D)/D^2(1 + D + D^2)$ with the series $D^{-2} + 1 + D + D^3 + \cdots$ instead of the alternative series $D^{-3} + D^{-5} + D^{-6} + \cdots$ that can also be obtained by long division but that is not a Laurent series. Obviously, all nonzero ratios $x(D)/y(D)$ are invertible, so they form the *field of binary rational functions* $\mathbb{F}_2(D)$, which is a subfield of the field of Laurent series $\mathbb{F}_2((D))$.

As an element in $\mathbb{F}_2((D))$, a rational function either has finitely many terms or is ultimately periodic, but a Laurent series can be aperiodic! Finite rational functions are also called *Laurent polynomials*. The *degree* of a Laurent polynomial is the "time index" at which the Laurent polynomial ends. For example,

$$x(D) = D^{-2} + 1 + D^3 + D^7 \tag{2.6}$$

is a Laurent polynomial with degree

$$\deg x(D) = 7 \tag{2.7}$$

We can consider n-tuples of elements from $\mathbb{F}_2[D], \mathbb{F}_2[[D]], \mathbb{F}_2(D)$, or $\mathbb{F}_2((D))$. For example, an n-tuple $\boldsymbol{x}(D) = (x^{(1)}(D)\, x^{(2)}(D) \ldots x^{(n)}(D))$, where $x^{(1)}(D)$,

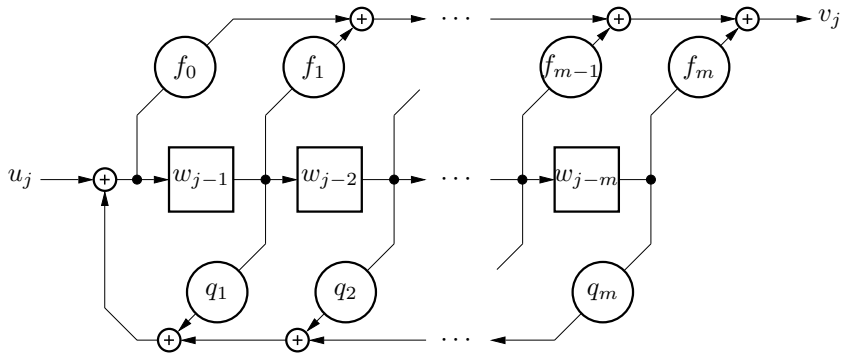


Figure 2.1 The controller canonical form of a rational transfer function.

$x^{(2)}(D), \ldots, x^{(n)}(D) \in \mathbb{F}_2((D))$, can be expressed as $\boldsymbol{x}(D) = \sum_{i=r}^{\infty}(x_i^{(1)}\, x_i^{(2)} \ldots x_i^{(n)})D^i$, $r \in \boldsymbol{Z}$, if $x^{(j)}(D) = \sum_{i=r}^{\infty} x_i^{(j)} D^i$, $1 \leq j \leq n$. So we denote the set of n-tuples of elements from $\mathbb{F}_2((D))$ by $\mathbb{F}_2^n((D))$, which is the n-dimensional vector space over the field of binary Laurent series $\mathbb{F}_2((D))$. Relative to $\mathbb{F}_2^n((D))$ the elements in the field $\mathbb{F}_2((D))$ are usually called *scalars*. Similarly, we have $\mathbb{F}_2^n[D], \mathbb{F}_2^n[[D]]$, and $\mathbb{F}_2^n(D)$.

If $\boldsymbol{x}(D) \in \mathbb{F}_2^n[D]$, we say that $\boldsymbol{x}(D)$ is polynomial in D. The degree of the element $\boldsymbol{x}(D) = \sum_{i=0}^{m}(x_i^{(1)}\, x_i^{(2)} \ldots x_i^{(n)})D^i$ is defined to be m, provided $(x_m^{(1)}\, x_m^{(2)} \ldots x_m^{(n)}) \neq (0\ 0 \ldots 0)$. For simplicity we call the elements in $\mathbb{F}_2^n[[D]]$ a formal power series also when $n > 1$.

For our rate $R = b/c$ encoder, we have the input sequences $\boldsymbol{u}(D) \in \mathbb{F}_2^b((D))$ and the output sequences $\boldsymbol{v}(D) \in \mathbb{F}_2^c((D))$.

We next consider the realization of linear systems. Consider for simplicity the *controller canonical form* of a single-input, single-output linear system as shown in Fig. 2.1. The delay elements form a shift register, the output is a linear function of the input and the shift register contents, and the input to the shift register is a linear function of the input and the shift register contents.

The output at time j,

$$v_j = \sum_{i=0}^{m} f_i w_{j-i} \tag{2.8}$$

has the D-transform

$$\begin{aligned} v(D) &= \sum_{j=-\infty}^{\infty} v_j D^j = \sum_{j=-\infty}^{\infty} \sum_{i=0}^{m} f_i w_{j-i} D^j \\ &= \sum_{k=-\infty}^{\infty} \left(\sum_{i=0}^{m} f_i D^i \right) w_k D^k = f(D)w(D) \end{aligned} \tag{2.9}$$

where we have replaced $j - i$ by k and where

$$f(D) = f_0 + f_1 D + \cdots + f_m D^m \tag{2.10}$$

and

$$w(D) = \sum_{k=-\infty}^{\infty} w_k D^k \tag{2.11}$$

From Fig. 2.1 it follows that

$$w_j = u_j + \sum_{i=1}^{m} q_i w_{j-i} \tag{2.12}$$

Upon defining $q_0 = 1$, (2.12) can be rewritten as

$$u_j = \sum_{i=0}^{m} q_i w_{j-i} \tag{2.13}$$

or, by repeating the steps in (2.9), as

$$u(D) = q(D)w(D) \tag{2.14}$$

where

$$u(D) = \sum_{j=-\infty}^{\infty} u_j D^j \tag{2.15}$$

and

$$q(D) = 1 + q_1 D + \cdots + q_m D^m \tag{2.16}$$

Combining (2.9) and (2.14) we have

$$v(D) = u(D)\frac{f(D)}{q(D)} = u(D)\frac{f_0 + f_1 D + \cdots + f_m D^m}{1 + q_1 D + \cdots + q_m D^m} \tag{2.17}$$

Let $g(D) = f(D)/q(D)$, then $v(D) = u(D)g(D)$, and we say that $g(D)$ is a *rational transfer function* that transfers the input $u(D)$ into the output $v(D)$. From (2.17), it follows that every rational function with a constant term 1 in the denominator polynomial $q(D)$ (or, equivalently, with $q(0) = 1$ or, again equivalently, with $q(D)$ delayfree) is a rational transfer function that can be realized in the canonical form shown in Fig. 2.1. Every rational function $g(D) = f(D)/q(D)$, where $q(D)$ is delayfree, is called a *realizable function*.

In general, a matrix $G(D)$ whose entries are rational functions is called a *rational transfer function matrix*. A rational transfer function matrix $G(D)$ for a linear system with many inputs or many outputs whose entries are realizable functions is called *realizable*.

In practice, given a rational transfer function matrix we have to realize it by linear sequential circuits. It can be realized in many different ways. For instance, the realizable function

$$g(D) = \frac{f_0 + f_1 D + \cdots + f_m D^m}{1 + q_1 D + \cdots + q_m D^m} \tag{2.18}$$

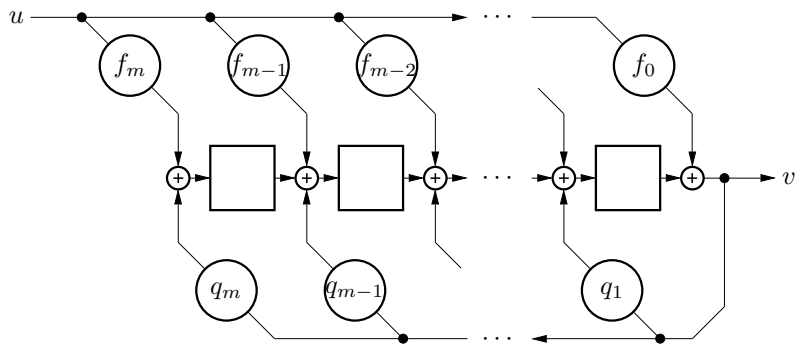


Figure 2.2 The observer canonical form of a rational transfer function.

has the controller canonical form illustrated in Fig. 2.1. On the other hand, since the circuit in Fig. 2.2 is linear, we have

$$\begin{aligned} v(D) = u(D)(f_0 + f_1 D + \cdots + f_m D^m) \\ +v(D)(q_1 D + \cdots + q_m D^m) \end{aligned} \tag{2.19}$$

which is the same as (2.17). Thus, Fig. 2.2 is also a realization of (2.18). In this realization, the delay elements do not in general form a shift register as these delay elements are separated by adders. This is the so-called *observer canonical form* of the rational function (2.18). The controller and observer canonical forms in Figs. 2.1 and 2.2, respectively, are two different realizations of the same rational transfer function.

We are now prepared to give a formal definition of a convolutional transducer.

Definition A rate $R = b/c$ (binary) *convolutional transducer* over the field of rational functions $\mathbb{F}_2(D)$ is a linear mapping

$$\begin{aligned} \tau : \ \mathbb{F}_2^b((D)) &\to \mathbb{F}_2^c((D)) \\ \boldsymbol{u}(D) &\mapsto \boldsymbol{v}(D) \end{aligned}$$

which can be represented as

$$\boldsymbol{v}(D) = \boldsymbol{u}(D)G(D) \tag{2.20}$$

where $G(D)$ is a $b \times c$ *transfer function matrix* of rank b with entries in $\mathbb{F}_2(D)$ and $\boldsymbol{v}(D)$ is called a *code sequence* arising from the *information sequence* $\boldsymbol{u}(D)$.

Obviously, we must be able to reconstruct the information sequence $\boldsymbol{u}(D)$ from the code sequence $\boldsymbol{v}(D)$ when there is no noise on the channel. Otherwise the convolutional transducer would be useless. Therefore, we require that the transducer map is injective; that is, the transfer function matrix $G(D)$ has rank b over the field $\mathbb{F}_2(D)$.

We are now well prepared for the following:

Definition A rate $R = b/c$ *convolutional code* $\mathcal{C}$ over $\mathbb{F}_2$ is the image set of a rate $R = b/c$ convolutional transducer with $G(D)$ of rank b over $\mathbb{F}_2(D)$ as its transfer function matrix.

It follows immediately from the definition that a rate $R = b/c$ convolutional code $\mathcal{C}$ over $\mathbb{F}_2$ with the $b \times c$ matrix $G(D)$ of rank b over $\mathbb{F}_2(D)$ as a transfer function matrix can be regarded as the $\mathbb{F}_2((D))$ row space of $G(D)$. Hence, it can also be regarded as the rate $R = b/c$ block code over the infinite field of Laurent series encoded by $G(D)$.

In the sequel we will only consider realizable transfer function matrices and, hence, we have the following:

Definition A transfer function matrix (of a convolutional code) is called a *generator matrix* if it (has full rank and) is realizable.

Definition A rate $R = b/c$ *convolutional encoder* of a convolutional code with generator matrix $G(D)$ over $\mathbb{F}_2(D)$ is a realization by a linear sequential circuit of a rate $R = b/c$ convolutional transducer whose transfer function matrix $G(D)$ (has full rank and) is realizable.

We call a realizable transfer function matrix $G(D)$ *delayfree* if at least one of its entries $f(D)/q(D)$ has $f(0) \neq 0$. If $G(D)$ is not delayfree, it can be written as

$$G(D) = D^i G_d(D) \tag{2.21}$$

where $i \geq 1$ and $G_d(D)$ is delayfree.

Theorem 2.1 Every convolutional code $\mathcal{C}$ has a generator matrix that is delayfree.

Proof: Let $G(D)$ be any generator matrix for $\mathcal{C}$. The nonzero entries of $G(D)$ can be written

$$g_{ij}(D) = D^{s_{ij}} f_{ij}(D)/q_{ij}(D) \tag{2.22}$$

where s_{ij} is an integer such that $f_{ij}(0) = q_{ij}(0) = 1$, $1 \leq i \leq b$, $1 \leq j \leq c$. The number s_{ij} is the delay of the sequence

$$g_{i_j}(D) = D^{s_{ij}} f_{ij}(D)/q_{ij}(D) = D^{s_{ij}} + g_{s_{ij}+1} D^{s_{ij}+1} + \cdots \tag{2.23}$$

Let $s = \min_{i,j} \{s_{ij}\}$. Clearly,

$$G'(D) = D^{-s} G(D) \tag{2.24}$$

is both delayfree and realizable. Since D^{-s} is a scalar in $\mathbb{F}_2((D))$, both $G(D)$ and $G'(D)$ generate the same convolutional code. Therefore, $G'(D)$ is a delayfree generator matrix for the convolutional code $\mathcal{C}$. ∎

A given convolutional code can be encoded by many essentially different encoders.

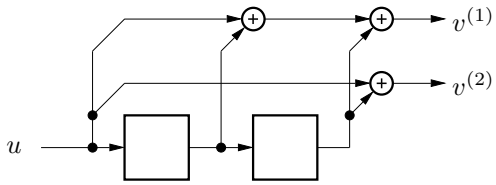


Figure 2.3 A rate $R = 1/2$ convolutional encoder with generator matrix $G_0(D)$.

■ **EXAMPLE 2.1**

Consider the rate $R = 1/2$, binary convolutional code with the basis vector $\boldsymbol{v}_0(D) = (1 + D + D^2 \;\; 1 + D^2)$. The simplest encoder for this code has the generator matrix

$$G_0(D) = (1 + D + D^2 \;\; 1 + D^2) \tag{2.25}$$

Its controller canonical form is shown in Fig. 2.3.

Theorem 2.2 Every convolutional code $\mathcal{C}$ has a polynomial delayfree generator matrix.

Proof: Let $G(D)$ be any (realizable and) delayfree generator matrix for $\mathcal{C}$, and let $q(D)$ be the least common multiple of all the denominators in (2.22). Since $q(D)$ is a delayfree polynomial,

$$G'(D) = q(D)G(D) \tag{2.26}$$

is a polynomial delayfree generator matrix for $\mathcal{C}$. ■

An encoder which realizes a polynomial generator matrix is called a *polynomial encoder*.

■ **EXAMPLE 2.1** ***(Cont'd)***

If we choose the basis to be $\boldsymbol{v}_1(D) = a_1(D)\boldsymbol{v}_0(D)$, where the scalar $a_1(D)$ is the rational function $a_1(D) = 1/(1 + D + D^2)$, we obtain the generator matrix

$$G_1(D) = \begin{pmatrix} 1 & \dfrac{1 + D^2}{1 + D + D^2} \end{pmatrix} \tag{2.27}$$

for the same code. The output sequence $\boldsymbol{v}(D) = (v^{(1)}(D) \;\; v^{(2)}(D))$ of the encoder with generator matrix $G_1(D)$ shown in Fig. 2.4 can be written as

$$\begin{aligned} v^{(1)}(D) &= u(D) \\ v^{(2)}(D) &= u(D)\frac{1 + D^2}{1 + D + D^2} \end{aligned} \tag{2.28}$$

The input sequence appears unchanged among the two output sequences.

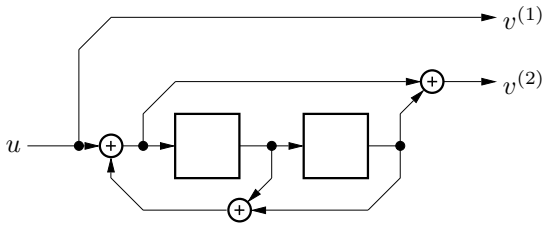


Figure 2.4 A rate $R = 1/2$ systematic convolutional encoder with feedback and generator matrix $G_1(D)$.

Definition A rate $R = b/c$ convolutional encoder whose b information sequences appear unchanged among the c code sequences is called a *systematic encoder*, and its generator matrix is called a *systematic generator matrix*.

If a convolutional code $\mathcal{C}$ is encoded by a systematic generator matrix, we can always permute its columns and obtain a generator matrix for an *equivalent* convolutional code $\mathcal{C}'$ such that the b information sequences appear unchanged *first* among the code sequences. Thus, without loss of generality, a systematic generator matrix can be written as

$$G(D) = (I_b \; R(D)) \tag{2.29}$$

where I_b is a $b \times b$ identity matrix and $R(D)$ a $b \times (c - b)$ matrix whose entries are rational functions of D.

Being systematic is a generator matrix property, not a code property. Every convolutional code has both systematic and *nonsystematic* generator matrices! (Remember that the code is the set of code sequences arising from the set of information sequences; the code does not depend on the mapping.)

■ **EXAMPLE 2.1** ***(Cont'd)***

If we further change the basis to $\boldsymbol{v}_2(D) = a_2(D)\boldsymbol{v}_0(D)$, where $a_2(D) \in \mathbb{F}_2(D)$ is chosen as $a_2(D) = 1 + D$, we obtain a third generator matrix for the same code, viz.,

$$G_2(D) = (1 + D^3 \quad 1 + D + D^2 + D^3) \tag{2.30}$$

Definition A generator matrix for a convolutional code is *catastrophic* if there exists an information sequence $\boldsymbol{u}(D)$ with infinitely many nonzero digits, $w_{\mathrm{H}}(\boldsymbol{u}(D)) = \infty$, that results in a codeword $\boldsymbol{v}(D)$ with only finitely many nonzero digits, that is, $w_{\mathrm{H}}(\boldsymbol{v}(D)) < \infty$.

■ **EXAMPLE 2.2**

The third generator matrix for the convolutional code given above, viz.,

$$G_2(D) = (1 + D^3 \quad 1 + D + D^2 + D^3) \tag{2.31}$$

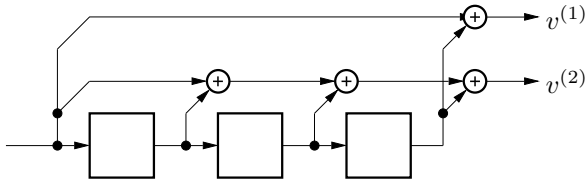


Figure 2.5 A rate $R = 1/2$ catastrophic convolutional encoder with generator matrix $G_2(D)$.

is catastrophic since $\boldsymbol{u}(D) = 1/(1+D) = 1+D+D^2+\cdots$ has $w_{\mathrm{H}}(\boldsymbol{u}(D)) = \infty$ but $\boldsymbol{v}(D) = \boldsymbol{u}(D)G_2(D) = (1+D+D^2 \;\; 1+D^2) = (1\;1)+(1\;0)D+(1\;1)D^2$ has $w_{\mathrm{H}}(\boldsymbol{v}(D)) = 5 < \infty$. In Fig. 2.5 we show its controller canonical form.

When a catastrophic generator matrix is used for encoding, finitely many errors (five in the previous example) in the estimate $\widehat{\boldsymbol{v}}(D)$ of the transmitted codeword $\boldsymbol{v}(D)$ can lead to infinitely many errors in the estimate $\widehat{\boldsymbol{u}}(D)$ of the information sequence $\boldsymbol{u}(D)$—a "catastrophic" situation that must be avoided!

Being catastrophic is a generator matrix property, not a code property. Every convolutional code has both catastrophic and *noncatastrophic* generator matrices.

Clearly, the choice of the generator matrix is of great importance.

■ **EXAMPLE 2.3**

The rate $R = 2/3$ generator matrix

$$G(D) = \begin{pmatrix} \dfrac{1}{1+D+D^2} & \dfrac{D}{1+D^3} & \dfrac{1}{1+D^3} \\ \dfrac{D^2}{1+D^3} & \dfrac{1}{1+D^3} & \dfrac{1}{1+D} \end{pmatrix} \tag{2.32}$$

has the controller and observer canonical forms shown in Figs. 2.6 and 2.7, respectively.

In Chapter 3 we will show that generator matrices $G(D)$ with $G(0)$ of full rank are of particular interest. Hence, we introduce the next definition.

Definition A generator matrix $G(D)$ is called an *encoding matrix* if $G(0)$ has full rank.

We have immediately the following:

Theorem 2.3 An encoding matrix is (realizable and) delayfree.

The polynomial generator matrices $G_0(D)$ (2.25), $G_1(D)$ (2.27), and $G_2(D)$ (2.31) as well as the rational generator $G(D)$ in Example 2.3 are all encoding matrices.

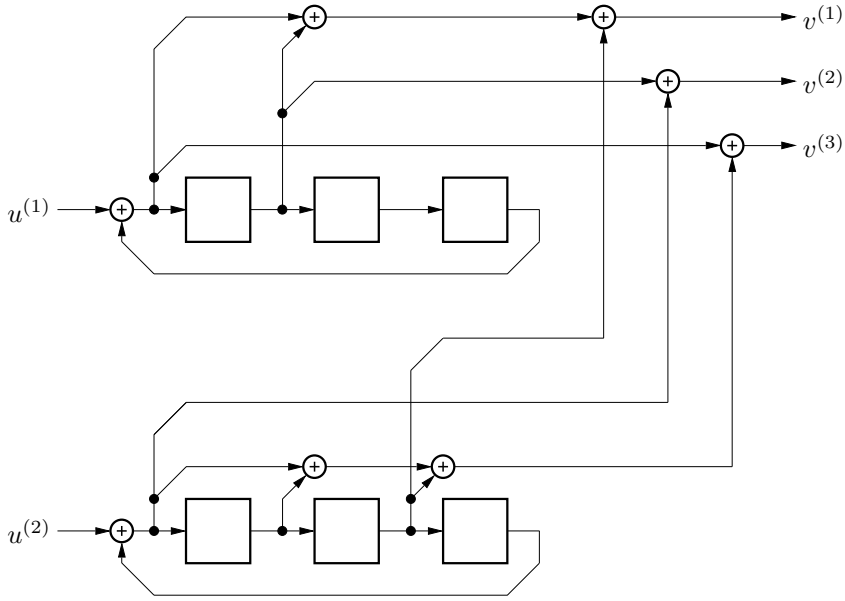


Figure 2.6 The controller canonical form of the generator matrix $G(D)$ in (2.32).

But the polynomial generator matrix

$$G(D) = \begin{pmatrix} 1+D & D & 1 \\ 1 & D+D^2 & 1+D \end{pmatrix} \tag{2.33}$$

is not an encoding matrix since $G(0)$ has rank 1.

In the sequel we will see that *all* generator matrices that are interesting in practice are in fact encoding matrices!

Remark: The generator matrix for the convolutional encoder shown in Fig. 1.13 can be written $G(D) = (g_{ij}(D))_{1\le i\le b,\ 1\le j\le c}$, where $g_{ij}(D) = \sum_{k=0}^{m} g_{ij}^{(k)} D^k$ and where $g_{ij}^{(k)}$ are the entries of the $b \times c$ matrix G_k, $0 \le k \le m$, in (1.91).

2.2 THE SMITH FORM OF POLYNOMIAL CONVOLUTIONAL GENERATOR MATRICES

We now present a useful decomposition of polynomial convolutional generator matrices. This decomposition is based on the following fundamental algebraic result [Jac85]:

Theorem 2.4 (Smith form) Let $G(D)$ be a $b \times c$, $b \le c$, binary polynomial matrix (i.e., $G(D) = (g_{ij}(D))$, where $g_{ij}(D) \in \mathbb{F}_2[D]$, $1 \le i \le b$, $1 \le j \le c$) of rank r.

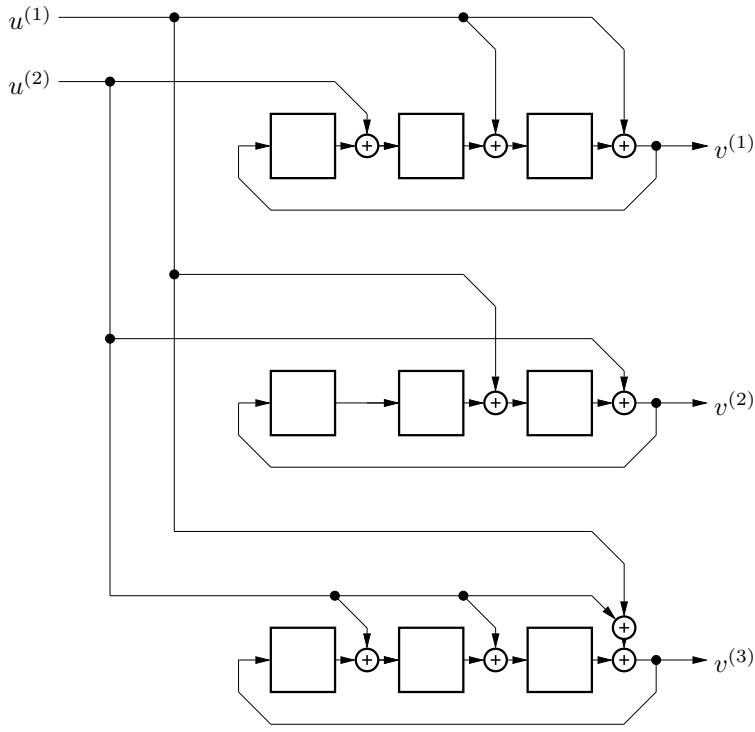


Figure 2.7 The observer canonical form of the generator matrix $G(D)$ in (2.32).

Then $G(D)$ can be written in the following manner:

$$G(D) = A(D)\Gamma(D)B(D) \tag{2.34}$$

where $A(D)$ and $B(D)$ are $b \times b$ and $c \times c$, respectively, binary polynomial matrices with unit determinants and where $\Gamma(D)$ is the $b \times c$ matrix

$$\Gamma(D) = \begin{pmatrix} \gamma_1(D) & & & & & & \\ & \gamma_2(D) & & & & & \\ & & \ddots & & & & \\ & & & \gamma_r(D) & & & \\ & & & & 0 & & \\ & & & & & \ddots & \\ & & & & & & 0 \ \ldots \ 0 \end{pmatrix} \tag{2.35}$$

which is called the *Smith form* of $G(D)$, and whose nonzero elements $\gamma_i(D) \in \mathbb{F}_2[D]$, $1 \leq i \leq r$, called the *invariant factors* of $G(D)$, are unique polynomials that satisfy

$$\gamma_i(D) \mid \gamma_{i+1}(D), \quad i = 1, 2, \ldots, r-1 \tag{2.36}$$

Moreover, if we let $\Delta_i(D) \in \mathbb{F}_2[D]$ be the *determinantal divisor* of $G(D)$, that is, the *greatest common divisor* (gcd) *of the* $i \times i$ *subdeterminants (minors) of* $G(D)$, *then*

$$\gamma_i(D) = \frac{\Delta_i(D)}{\Delta_{i-1}(D)} \tag{2.37}$$

where $\Delta_0(D) = 1$ *by convention and* $i = 1, 2, \ldots, r$.

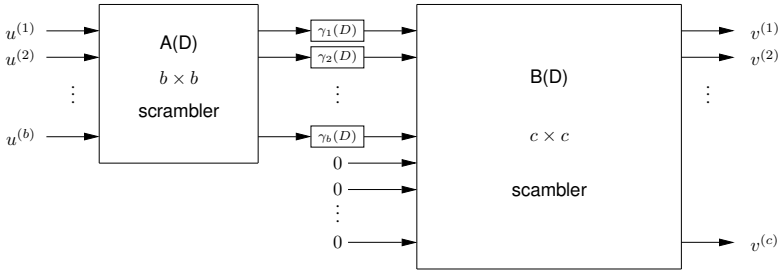


Figure 2.8 Smith form decomposition of a rate $R = b/c$ polynomial convolutional encoder.

By definition a generator matrix has full rank. Its Smith form decomposition is illustrated in Fig. 2.8. We call the matrices $A(D)$ and $B(D)$ *scamblers*. The input sequences are scrambled in the $b \times b$ scrambler $A(D)$. The b outputs of this scrambler are multiplied by the b invariant factors. These b products plus $c - b$ dummy zeros are then scrambled in the $c \times c$ scrambler $B(D)$ to give the output sequences. Since a scrambler is invertible, it follows that its output sequences, when the input sequences range over all possible sequences, are a reordering of all possible sequences.

Before we give a proof of Theorem 2.4, we introduce two types of *elementary operations*.

Type I *The interchange of two rows (or two columns).*

The following nonsingular matrix performs the interchange of rows (or columns) i and j depending on whether it is used as a pre- (or post-) multiplier:

$$P_{ij} = \begin{pmatrix} 1 & & & & & & & & \\ & \ddots & & & & & & & \\ & & 1 & & & & & & \\ & & & 0 & \cdots & \cdots & 1 & \cdots & \cdots \\ & & & & 1 & & & & \\ & & & & & \ddots & & & \\ & & & & & & 1 & & \\ & & & 1 & \cdots & \cdots & 0 & \cdots & \cdots \\ & & & & & & & 1 & \\ & & & & & & & & \ddots \\ & & & & & & & & & 1 \end{pmatrix} \begin{matrix} \\ \\ \\ \text{row } i \\ \\ \\ \\ \text{row } j \\ \\ \\ \end{matrix}$$

It is immediate that $P_{ij}^{-1} = P_{ij}$. Moreover, $\det(P_{ij}) = -1 = 1$ since we are working in $\mathbb{F}_2$.

Type II *The addition to all elements in one row (or column) of the corresponding elements in another row (or column) multiplied by a fixed polynomial in D.*

The following nonsingular matrix performs the addition to the elements in row (or column) i of the elements in row (or column) j multiplied by the polynomial $p(D) \in \mathbb{F}_2[D]$ when used as a pre- (or post-) multiplier.

$$P_{ij} = \begin{pmatrix} 1 & & & & & & \\ & \ddots & & & & & \\ & & 1 & \cdots & p(D) & \cdots & \text{row } i \\ & & & \ddots & \vdots & & \\ & & & & 1 & \cdots & \text{row } j \\ & & & & & \ddots & \\ & & & & & & 1 \end{pmatrix}$$

It is easy to check directly that $R_{ij}^{-1}(p(D)) = R_{ij}(-p(D)) = R_{ij}(p(D))$. One sees trivially that $\det(R_{ij}(p(D))) = 1$.

Premultiplication by any of these elementary matrices results in the associated transformation being performed on the rows whereas postmultiplication does the same for columns.

Proof (Theorem 2.4): If $G(D) = \mathbf{0}$, there is nothing to prove. Therefore, we assume that $G(D) \neq \mathbf{0}$.

First, we show that starting with $G(D)$ we can obtain, by elementary operations only, a matrix whose entry in the upper-left corner is nonzero and has minimal degree of all nonzero entries in all matrices that can be obtained from $G(D)$ by elementary operations. We can bring the polynomial of lowest degree in $G(D)$ to the upper-left corner by elementary operations. Assume now that it is there. Let $\alpha_{ij}(D),\ 1 \leq i \leq b,\ 1 \leq j \leq c$, be the elements in this new matrix. Divide an element in the first row, $\alpha_{1j}(D),\ j > 1$, by $\alpha_{11}(D)$. Then we have $\alpha_{1j}(D) = \alpha_{11}(D)\beta_j(D) + \beta_{1j}(D)$, where $\deg(\beta_{1j}(D)) < \deg(\alpha_{11}(D))$. Now add the first column multiplied by $\beta_j(D)$ to the jth column. This elementary operation replaces $\alpha_{1j}(D)$ by $\beta_{1j}(D)$. Thus, if $\beta_{1j}(D) \neq 0$, we obtain a matrix for which the lowest degree of a nonzero entry has been reduced. Repeat this procedure for the new matrix. Similarly, we can obtain a matrix for which the lowest degree of the nonzero entries in the first column has been reduced. Since the degree is reduced at each step, the procedure is finite. Thus, repeating this process yields a matrix $G'(D) = (\beta_{ij}(D))$ in which $\beta_{11}(D)$ has minimal degree and $\beta_{11}(D)$ divides both $\beta_{1j}(D)$ and $\beta_{i1}(D)$ for all i and j. Except for the upper-left corner, we next clear the first row and first column to zero

by elementary operations. This gives a matrix of the form

$$\begin{pmatrix} \beta_{11}(D) & 0 & \dots & 0 \\ 0 & & & \\ \vdots & & G''(D) & \\ 0 & & & \end{pmatrix}$$

where we let $\beta_{11}(D) = \gamma_1(D)$.

Next we will prove that $\gamma_1(D)$ divides every element in $G''(D) = (\delta_{ij}(D))$. If $\gamma_1(D) \not| \delta_{ij}(D)$, then we add the jth column to the first column and obtain a new first column. Repeating the procedure described above then yields an element in the upper-left corner with a degree less than $\deg(\beta_{11}(D))$, which is a contradiction. Hence, our matrix can be written as

$$\begin{pmatrix} \gamma_1(D) & 0 & \dots & 0 \\ 0 & & & \\ \vdots & & \gamma_1(D)G^*(D) & \\ 0 & & & \end{pmatrix}$$

Repeating the procedure for $G^*(D)$ (induction) proves the first part of the theorem.

Now we will prove that

$$\Delta_i(D) \stackrel{\text{def}}{=} \text{the gcd of all } i \times i \text{ minors of } G(D) \tag{2.38}$$

is unaffected by elementary row and column operations on $G(D)$. Let $A(D) = (a_{ij}(D))$ be a $b \times b$ permutor with entries in $\mathbb{F}_2[D]$. The (i, j) entry of $A(D)G(D)$ is $\sum_k a_{ik}(D)g_{kj}(D)$. This shows that the rows of $A(D)G(D)$ are linear combinations of the rows of $G(D)$. Hence, the $i \times i$ minors of $A(D)G(D)$ are linear combinations of the $i \times i$ minors of $G(D)$. Thus, the gcd of all $i \times i$ minors of $G(D)$ is a divisor of the gcd of all $i \times i$ minors of $A(D)G(D)$. Since $A(D)$ has a unit determinant, $A^{-1}(D)$ is also a $b \times b$ polynomial matrix. By repeating the argument above for the matrix $A^{-1}(D)(A(D)G(D))$ we can show that the gcd of all $i \times i$ minors of $A(D)G(D)$ is a divisor of the gcd of all $i \times i$ minors of $A^{-1}(D)(A(D)G(D)) = G(D)$. Hence, the gcds of all minors of $G(D)$ and $A(D)G(D)$ are the same. Since $A(D)$ can be any product of elementary row operations, we have shown that $\Delta_i(D)$ is unaffected by elementary row operations. Similarly, we can show that $\Delta_i(D)$ is unaffected by elementary column operations.

We have now proved that we can also identify

$$\Delta_i(D) = \text{the gcd of all } i \times i \text{ minors of } \Gamma(D) \tag{2.39}$$

The form of $\Gamma(D)$ then shows that

$$\begin{aligned} \Delta_1(D) &= \gamma_1(D), \\ \Delta_2(D) &= \gamma_1(D)\gamma_2(D), \\ &\vdots \\ \Delta_r(D) &= \gamma_1(D)\gamma_2(D)\dots\gamma_r(D) \end{aligned} \tag{2.40}$$

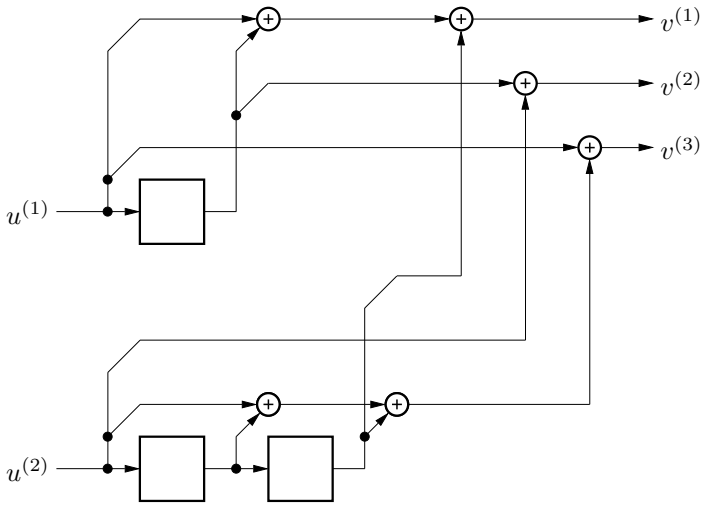


Figure 2.9 A rate $R = 2/3$ convolutional encoder.

or (with $\Delta_0(D) = 1$)

$$\gamma_i(D) = \frac{\Delta_i(D)}{\Delta_{i-1}(D)},\ i = 1, 2, \ldots, r \tag{2.41}$$

Moreover, the uniqueness of $\Delta_i(D),\ i = 1, 2, \ldots, r$, implies that of $\gamma_i(D),\ i = 1, 2, \ldots, r$, which completes the proof. ■

■ **EXAMPLE 2.4**

To obtain the Smith form of the polynomial encoder illustrated in Fig. 2.9 we start with its encoding matrix

$$G(D) = \begin{pmatrix} 1+D & D & 1 \\ D^2 & 1 & 1+D+D^2 \end{pmatrix} \tag{2.42}$$

and interchange columns 1 and 3:

$$\begin{pmatrix} 1+D & D & 1 \\ D^2 & 1 & 1+D+D^2 \end{pmatrix} \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix} = \begin{pmatrix} 1 & D & 1+D \\ 1+D+D^2 & 1 & D^2 \end{pmatrix} \tag{2.43}$$

Now the element in the upper-left corner has minimum degree. To clear the rest of the first row, we can proceed with two type II operations simultaneously:

$$\begin{pmatrix} 1 & D & 1+D \\ 1+D+D^2 & 1 & D^2 \end{pmatrix} \begin{pmatrix} 1 & D & 1+D \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 1+D+D^2 & 1+D+D^2+D^3 & 1+D^2+D^3 \end{pmatrix} \tag{2.44}$$

Next, we clear the rest of the first column:

$$\begin{pmatrix} 1 & 0 \\ 1+D+D^2 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 1+D+D^2 & 1+D+D^2+D^3 & 1+D^2+D^3 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1+D+D^2+D^3 & 1+D^2+D^3 \end{pmatrix} \tag{2.45}$$

Following the technique in the proof, we divide $1+D^2+D^3$ by $1+D+D^2+D^3$:

$$1+D^2+D^3 = (1+D+D^2+D^3)1+D \tag{2.46}$$

Thus, we add column 2 to column 3 and obtain

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1+D+D^2+D^3 & 1+D^2+D^3 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1+D+D^2+D^3 & D \end{pmatrix} \tag{2.47}$$

Now we interchange columns 2 and 3:

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1+D+D^2+D^3 & D \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & D & 1+D+D^2+D^3 \end{pmatrix} \tag{2.48}$$

Repeating the previous step gives

$$1+D+D^2+D^3 = D(1+D+D^2)+1 \tag{2.49}$$

and, hence, we multiply column 2 by $1+D+D^2$, add the product to column 3, and obtain

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & D & 1+D+D^2+D^3 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 1+D+D^2 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & D & 1 \end{pmatrix} \tag{2.50}$$

Again we should interchange columns 2 and 3:

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & D & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & D \end{pmatrix} \tag{2.51}$$

and, finally, by adding D times column 2 to column 3 we obtain the Smith form:

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & D \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & D \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} = \Gamma(D) \tag{2.52}$$

All invariant factors for this encoding matrix are equal to 1.

By tracing these steps backward and multiplying $\Gamma(D)$ with the inverses of the elementary matrices (which are the matrices themselves), we obtain the matrix

$$\begin{aligned} G(D) &= A(D)\Gamma(D)B(D) \\ &= \begin{pmatrix} 1 & 0 \\ 1+D+D^2 & 1 \end{pmatrix} \Gamma(D) \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & D \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 1+D+D^2 \\ 0 & 0 & 1 \end{pmatrix} \\ &\quad \times \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & D & 1+D \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix} \end{aligned} \tag{2.53}$$

and we conclude that

$$A(D) = \begin{pmatrix} 1 & 0 \\ 1+D+D^2 & 1 \end{pmatrix} \tag{2.54}$$

and

$$B(D) = \begin{pmatrix} 1+D & D & 1 \\ 1+D^2+D^3 & 1+D+D^2+D^3 & 0 \\ D+D^2 & 1+D+D^2 & 0 \end{pmatrix} \tag{2.55}$$

Thus, we have the following decomposition of the encoding matrix $G(D)$:

$$\begin{aligned} G(D) &= \begin{pmatrix} 1 & 0 \\ 1+D+D^2 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \\ &\quad \times \begin{pmatrix} 1+D & D & 1 \\ 1+D^2+D^3 & 1+D+D^2+D^3 & 0 \\ D+D^2 & 1+D+D^2 & 0 \end{pmatrix} \end{aligned} \tag{2.56}$$

The extension of the Smith form to matrices whose entries are rational functions is immediate. Let $G(D)$ be a $b \times c$ rational function matrix and let $q(D) \in \mathbb{F}_2[D]$ be the least common multiple (lcm) of all denominators in $G(D)$. Then $q(D)G(D)$ is a polynomial matrix with Smith form decomposition

$$q(D)G(D) = A(D)\Gamma_q(D)B(D) \tag{2.57}$$

Dividing through by $q(D)$, we obtain the so-called *invariant-factor decomposition* of the rational matrix $G(D)$ [KFA69, HaH70]:

$$G(D) = A(D)\Gamma(D)B(D) \tag{2.58}$$

where

$$\Gamma(D) = \Gamma_q(D)/q(D) \tag{2.59}$$

with entries in $\mathbb{F}_2(D)$. Thus

$$\Gamma(D) = \begin{pmatrix} \frac{\gamma_1(D)}{q(D)} & & & & & & \\ & \frac{\gamma_2(D)}{q(D)} & & & & & \\ & & \ddots & & & & \\ & & & \frac{\gamma_r(D)}{q(D)} & & & \\ & & & & 0 & & \\ & & & & & \ddots & \\ & & & & & & 0 \ \ldots \ 0 \end{pmatrix} \tag{2.60}$$

where $\frac{\gamma_1(D)}{q(D)}, \frac{\gamma_2(D)}{q(D)}, \ldots, \frac{\gamma_r(D)}{q(D)}$ are called the *invariant factors* of $G(D)$.

Let

$$\frac{\gamma_i(D)}{q(D)} = \frac{\alpha_i(D)}{\beta_i(D)}, \quad i = 1, 2, \ldots, r \tag{2.61}$$

where the polynomials $\alpha_i(D)$ and $\beta_i(D)$ are relatively prime. Since $\gamma_i(D) \mid \gamma_{i+1}(D)$, $i = 1, 2, \ldots, r-1$, that is,

$$q(D)\frac{\alpha_i(D)}{\beta_i(D)} \,\Big|\, q(D)\frac{\alpha_{i+1}(D)}{\beta_{i+1}(D)} \tag{2.62}$$

we have

$$\alpha_i(D)\beta_{i+1}(D) \mid \alpha_{i+1}(D)\beta_i(D) \tag{2.63}$$

From (2.63) and the fact that $\gcd(\alpha_i(D), \beta_i(D)) = 1$, $i = 1, 2, \ldots, r$, it follows that

$$\alpha_i(D) \mid \alpha_{i+1}(D) \tag{2.64}$$

and

$$\beta_{i+1}(D) \mid \beta_i(D) \tag{2.65}$$

for $i = 1, 2, \ldots, r-1$.

■ **EXAMPLE 2.5**

The rate $R = 2/3$ rational encoding matrix (2.32) in Example 2.3,

$$G(D) = \begin{pmatrix} \dfrac{1}{1+D+D^2} & \dfrac{D}{1+D^3} & \dfrac{1}{1+D^3} \\ \dfrac{D^2}{1+D^3} & \dfrac{1}{1+D^3} & \dfrac{1}{1+D} \end{pmatrix} \tag{2.66}$$

has

$$q(D) = \operatorname{lcm}\left(1+D+D^2, \quad 1+D^3, \quad 1+D\right) = 1+D^3 \tag{2.67}$$

where lcm is the *least common multiple*. Thus we have

$$q(D)G(D) = \begin{pmatrix} 1+D & D & 1 \\ D^2 & 1 & 1+D+D^2 \end{pmatrix} \tag{2.68}$$

which is equal to the encoding matrix in Example 2.4. Hence, from (2.56), (2.59), and (2.67) it follows that

$$G(D) = \begin{pmatrix} 1 & 0 \\ 1+D+D^2 & 1 \end{pmatrix} \begin{pmatrix} \dfrac{1}{1+D^3} & 0 & 0 \\ 0 & \dfrac{1}{1+D^3} & 2 \end{pmatrix} \times \begin{pmatrix} 1+D & D & 1 \\ 1+D^2+D^3 & 1+D+D^2+D^3 & 0 \\ D+D^2 & 1+D+D^2 & 0 \end{pmatrix} \tag{2.69}$$

2.3 ENCODER INVERSES

Let us consider a convolutional encoder in a communication system as shown in Fig. 2.10. From the information sequence $\boldsymbol{u}(D)$ the encoder generates a codeword $\boldsymbol{v}(D)$, which is transmitted over a noisy channel. The decoder operates on the received data to produce an estimate $\widehat{\boldsymbol{u}}(D)$ of the information sequence $\boldsymbol{u}(D)$. As the case was with block codes in Chapter 1, we can split the decoder conceptually into two parts: a *codeword estimator* that from the received data $\boldsymbol{r}(D)$ produces a codeword estimate $\widehat{\boldsymbol{v}}(D)$, followed by an *encoder inverse* τ^{-1} that assigns to the codeword estimate the appropriate information sequence $\widehat{\boldsymbol{u}}(D)$. This communication system is illustrated in Fig. 2.11.

A practical decoder is seldom realized in two parts, although the decoder can do no better estimating the information sequence directly from the received data than by first estimating the codeword and then obtaining the decoded sequence via the inverse map τ^{-1}.

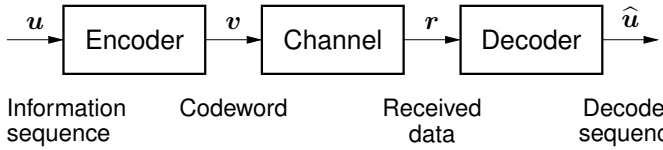


Figure 2.10 A convolutional encoder in a communication situation.

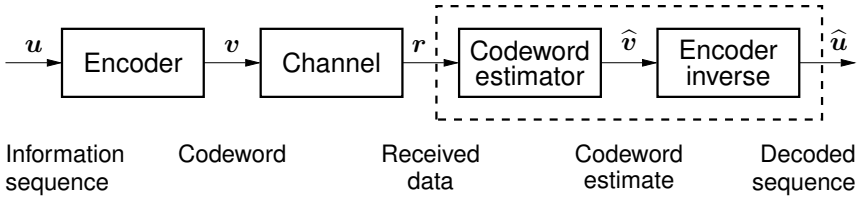


Figure 2.11 A conceptual split of decoder into two parts.

The *inverse* map τ^{-1} is represented by a $c \times b$ *right* inverse matrix $G^{-1}(D)$ of $G(D)$, that is,

$$G(D)G^{-1}(D) = I_b \tag{2.70}$$

where I_b is the $b \times b$ identity matrix.

In general, a right inverse $G^{-1}(D)$ of a generator matrix is not realizable.

Theorem 2.5 A convolutional generator matrix $G(D)$ that has a realizable right inverse is an encoding matrix.

Proof: Let $G^{-1}(D)$ be a realizable right inverse of $G(D)$, that is,

$$G(D)G^{-1}(D) = I_b \tag{2.71}$$

Substituting 0 for D in (2.71), we obtain

$$G(0)G^{-1}(0) = I_b \tag{2.72}$$

Hence, $G(0)$ has full rank. ∎

Theorem 2.6 A rational convolutional generator matrix has a realizable and delayfree inverse if and only if $\alpha_b(D)$ is delayfree.

Proof: For any rational generator matrix $G(D)$ a right inverse can be obtained from the invariant-factor decomposition $G(D) = A(D)\Gamma(D)B(D)$:

$$G^{-1}(D) = B^{-1}(D)\Gamma^{-1}(D)A^{-1}(D) \tag{2.73}$$

where

$$\Gamma^{-1}(D) = q(D)\Gamma_q^{-1}(D) = \begin{pmatrix} \frac{q(D)}{\gamma_1(D)} & & & & \\ & \frac{q(D)}{\gamma_2(D)} & & & \\ & & \ddots & & \\ & & & \frac{q(D)}{\gamma_b(D)} & \\ & & & 0 & \\ & & & \vdots & \\ & & & 0 & \end{pmatrix} \tag{2.74}$$

is a right inverse of the $b \times c$ polynomial matrix $\Gamma(D)$. The matrix $\Gamma_q(D)$ is the Smith form of the polynomial matrix $q(D)G(D)$, where $q(D)$ is the least common multiple of the denominators in $G(D)$ and γ_i, $1 \leq i \leq b$, are the invariant factors of $q(D)G(D)$. We have

$$\Gamma(D)\Gamma^{-1}(D) = I_b \tag{2.75}$$

From (2.61) it follows that $\Gamma^{-1}(D)$ can be written

$$\Gamma^{-1}(D) = \begin{pmatrix} \frac{\beta_1(D)}{\alpha_1(D)} & & & & \\ & \frac{\beta_2(D)}{\alpha_2(D)} & & & \\ & & \ddots & & \\ & & & \frac{\beta_b(D)}{\alpha_b(D)} & \\ & & & 0 & \\ & & & \vdots & \\ & & & 0 & \end{pmatrix} \tag{2.76}$$

Suppose that $\alpha_b(D)$ is delayfree. From (2.64) we deduce that all $\alpha_i(D)$, $1 \leq i \leq b$, are delayfree. Then the right inverse matrix $\Gamma^{-1}(D)$ (2.76) is realizable. Hence, all the entries in $G^{-1}(D) = B^{-1}(D)\Gamma^{-1}(D)A^{-1}(D)$ (2.73) are realizable. From (2.72) it follows that $G^{-1}(0)$ has full rank. Hence, $G^{-1}(D)$ is delayfree.

Conversely, suppose that the rational generator matrix $G(D)$ has a realizable right inverse $G^{-1}(D)$. Since both $G^{-1}(D)$ and $B^{-1}(D)\Gamma^{-1}(D)A^{-1}(D)$ are (possibly different) right inverses of $G(D)$, it follows that

$$G(D)(G^{-1}(D) + B^{-1}(D)\Gamma^{-1}(D)A^{-1}(D)) = \mathbf{0} \tag{2.77}$$

Substituting the invariant-factor decomposition $G(D) = A(D)\Gamma(D)B(D)$ into (2.77), we have

$$A(D)\Gamma(D)B(D)(G^{-1}(D) + B^{-1}(D)\Gamma^{-1}(D)A^{-1}(D)) = \mathbf{0} \tag{2.78}$$

or, equivalently,

$$\Gamma(D)(B(D)G^{-1}(D) + \Gamma^{-1}(D)A^{-1}(D)) = \mathbf{0} \tag{2.79}$$

From (2.79) we conclude that

$$B(D)G^{-1}(D) + \Gamma^{-1}(D)A^{-1}(D) = \begin{pmatrix} \mathbf{0} \\ L_1(D) \end{pmatrix} \tag{2.80}$$

where $L_1(D)$ is a $(c-b) \times b$ matrix with entries in $\mathbb{F}_2(D)$. Thus, we have

$$\begin{aligned} B(D)G^{-1}(D) &= \begin{pmatrix} \mathbf{0} \\ L_1(D) \end{pmatrix} + \Gamma^{-1}(D)A^{-1}(D) \\ &= \begin{pmatrix} \mathbf{0} \\ L_1(D) \end{pmatrix} + \begin{pmatrix} L_2(D) \\ \mathbf{0} \end{pmatrix} \end{aligned} \tag{2.81}$$

where $L_2(D)$ is a $b \times c$ matrix with entries in $\mathbb{F}_2(D)$. Since $G^{-1}(D)$ is realizable, so is $B(D)G^{-1}(D)$ and, hence, $L_2(D)$. Thus, $\Gamma^{-1}(D)A^{-1}(D)$ is realizable. Therefore, $\alpha_b(D)$ is delayfree, and the proof is complete. ■

Corollary 2.7 A polynomial convolutional generator matrix has a realizable and delayfree inverse if and only if $\gamma_b(D)$ is delayfree.

Proof: It follows immediately from (2.61) that $\alpha_b(D) = \gamma_b(D)$ for a polynomial matrix. ■

■ **EXAMPLE 2.6**

Since the encoding matrix $G(D)$ for the encoder in Fig. 2.9 satisfies the condition of Corollary 2.7, it has a realizable and delayfree inverse. A right inverse matrix $G^{-1}(D)$ for $G(D)$ is

$$\begin{aligned} G^{-1}(D) &= B^{-1}(D)\Gamma^{-1}(D)A^{-1}(D) \\ &= \begin{pmatrix} 0 & 1+D+D^2 & 1+D+D^2+D^3 \\ 0 & D+D^2 & 1+D^2+D^3 \\ 1 & 1+D^2 & 1+D+D^3 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 1+D+D^2 & 1 \end{pmatrix} \\ &= \begin{pmatrix} 1+D^2+D^4 & 1+D+D^2 \\ D+D^4 & D+D^2 \\ D+D^3+D^4 & 1+D^2 \end{pmatrix} \end{aligned} \tag{2.82}$$

which happens to be polynomial. In Fig. 2.12 we show the controller canonical form of $G^{-1}(D)$.

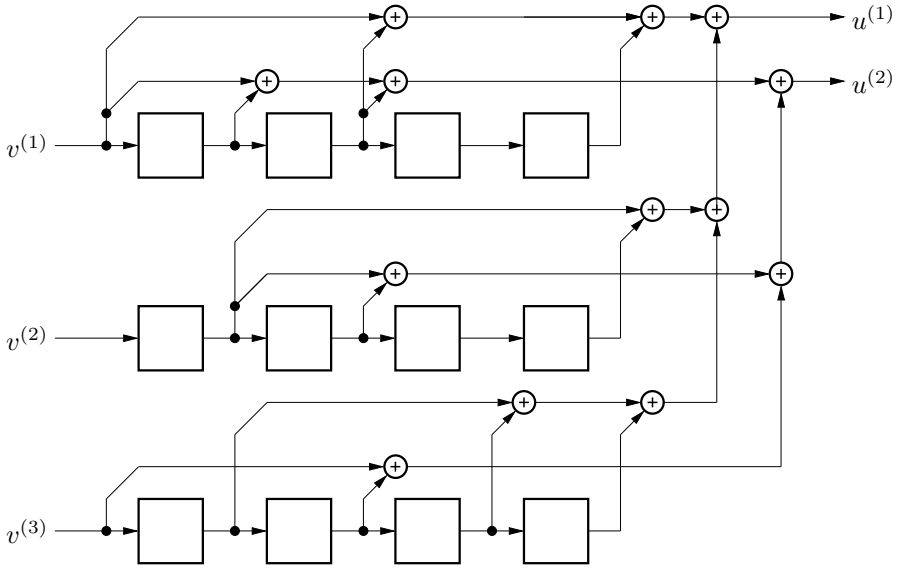


Figure 2.12 The controller canonical form of the encoding right inverse matrix $G^{-1}(D)$ in Example 2.6.

■ EXAMPLE 2.7

The rate $R = 1/2$ convolutional encoder in Fig. 2.3 has encoding matrix

$$G(D) = (1 + D + D^2 \quad 1 + D^2) \tag{2.83}$$

Following the technique in the proof of Theorem 2.4, we divide $g_{12}(D)$ by $g_{11}(D)$:

$$1 + D^2 = (1 + D + D^2)1 + D \tag{2.84}$$

Thus, we add $1 + D + D^2$ to $1 + D^2$ and obtain the sum D:

$$(1 + D + D^2 \quad 1 + D^2) \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} = (1 + D + D^2 \quad D) \tag{2.85}$$

We interchange the columns by an elementary operation:

$$(1 + D + D^2 \quad D) \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} = (D \quad 1 + D + D^2) \tag{2.86}$$

Repeating the first step yields

$$1 + D + D^2 = D(1 + D) + 1 \tag{2.87}$$

and we add $1 + D$ times the first column to the second column:

$$(D \quad 1 + D + D^2) \begin{pmatrix} 1 & 1 + D \\ 0 & 1 \end{pmatrix} = (D \quad 1) \tag{2.88}$$

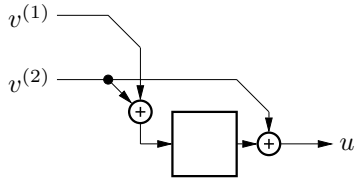


Figure 2.13 The observer canonical form of the encoding right inverse matrix $G^{-1}(D)$ in Example 2.7.

Again we interchange columns:

$$(D \quad 1)\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} = (1 \quad D) \tag{2.89}$$

Finally, adding D times the first column to the second column yields

$$(1 \quad D)\begin{pmatrix} 1 & D \\ 0 & 1 \end{pmatrix} = (1 \quad 0) = \Gamma(D) \tag{2.90}$$

Since we did not perform any elementary operations from the left, the permutor $A(D) = (1)$. Since the inverses of the elementary matrices are the matrices themselves, the permutor $B(D)$ is obtained as

$$\begin{aligned} B(D) &= \begin{pmatrix} 1 & D \\ 0 & 1 \end{pmatrix}\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}\begin{pmatrix} 1 & 1+D \\ 0 & 1 \end{pmatrix}\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}\begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \\ &= \begin{pmatrix} 1+D+D^2 & 1+D^2 \\ 1+D & D \end{pmatrix} \end{aligned} \tag{2.91}$$

Hence, we have the right inverse matrix

$$\begin{aligned} G^{-1}(D) &= B^{-1}(D)\Gamma^{-1}(D)A^{-1}(D) \\ &= \begin{pmatrix} D & 1+D^2 \\ 1+D & 1+D+D^2 \end{pmatrix}\begin{pmatrix} 1 \\ 0 \end{pmatrix}(1) \\ &= \begin{pmatrix} D \\ 1+D \end{pmatrix} \end{aligned} \tag{2.92}$$

Its observer canonical form is shown in Fig. 2.13.

■ **EXAMPLE 2.8**

Consider the rate $R = 2/3$ convolutional generator matrix

$$G(D) = \begin{pmatrix} 1 & 1+D & 1+D+D^2 \\ 1+D+D^3 & 1+D^2+D^3 & 1+D+D^4 \end{pmatrix} \tag{2.93}$$

Since all entries in $G(D)$ are relatively prime, it follows that $\Delta_1(D) = 1$ and, hence, $\gamma_1(D) = 1$.

The determinantal divisor $\Delta_2(D)$ is the gcd of all 2×2 subdeterminants of $G(D)$; that is,

$$\Delta_2(D) = \gcd\left(D^4,\ D + D^5,\ D + D^2 + D^4\right) = D \tag{2.94}$$

Thus, we have

$$\gamma_2(D) = \frac{\Delta_2(D)}{\Delta_1(D)} = D \tag{2.95}$$

which is not delayfree. By Theorem 2.6 none of the right inverses of the generator matrix $G(D)$ is realizable!

Theorem 2.8 A rational convolutional generator matrix has a (delayfree) polynomial right inverse if and only if $\alpha_b(D) = 1$.

Proof: Suppose that $\alpha_b(D) = 1$. From (2.64) it follows that $\alpha_1(D) = \alpha_2(D) = \cdots = \alpha_b(D) = 1$. Then

$$G^{-1}(D) = B^{-1}(D) \begin{pmatrix} \beta_1(D) & & & & \\ & \beta_2(D) & & & \\ & & \ddots & & \\ & & & \beta_b(D) & \\ & & & 0 & \\ & & & \vdots & \\ & & & 0 & \end{pmatrix} A^{-1}(D) \tag{2.96}$$

is a polynomial right inverse of $G(D)$. Since $G(D)$ is realizable, it follows that $G^{-1}(D)$ is delayfree.

Conversely, suppose that $G(D)$ has a polynomial right inverse $G^{-1}(D)$. Since $A(D)$ has unit determinant, there exists a polynomial vector $(x_1(D)\, x_2(D) \ldots x_b(D))$ such that

$$(x_1(D)\, x_2(D) \ldots x_b(D))A(D) = (0\ 0 \ldots 0\ \beta_b(D)) \tag{2.97}$$

Then

$$\begin{aligned} &\big(x_1(D)\, x_2(D) \ldots x_b(D)\big) \\ &\quad = \big(x_1(D)\, x_2(D) \ldots x_b(D)\big)G(D)G^{-1}(D) \\ &\quad = \big(x_1(D)\, x_2(D) \ldots x_b(D)\big)A(D)\Gamma(D)B(D)G^{-1}(D) \\ &\quad = \big(0\ 0 \ldots 0\ \beta_b(D)\big)\Gamma(D)B(D)G^{-1}(D) \\ &\quad = \big(0\ 0 \ldots 0\ \alpha_b(D)\ 0 \ldots 0\big)\ B(D)G^{-1}(D) \\ &\quad = \big(\alpha_b(D)y_1(D)\, \alpha_b(D)y_2(D) \ldots \alpha_b(D)y_b(D)\big) \end{aligned} \tag{2.98}$$

where $(y_1(D)\, y_2(D) \ldots y_b(D))$ is the bth row of $B(D)G^{-1}(D)$. From (2.97) we deduce that $\beta_b(D)$ is the greatest common divisor of the polynomials $x_1(D), x_2(D), \ldots, x_b(D)$. Then it follows from (2.98) that $\beta_b(D)$ is the greatest common divisor of

$\alpha_b(D)y_1(D), \alpha_b(D)y_2(D), \ldots, \alpha_b(D)y_b(D)$. Clearly, $\alpha_b(D)$ is a common divisor of $\alpha_b(D)y_1(D), \alpha_b(D)y_2(D), \ldots, \alpha_b(D)y_b(D)$, which implies that $\alpha_b(D) \mid \beta_b(D)$. Since $\gcd(\alpha_b(D), \beta_b(D)) = 1$, we conclude that $\alpha_b(D) = 1$ and the proof is complete. ■

From Theorem 2.8 the next corollary follows.

Corollary 2.9 A polynomial convolutional generator matrix has a (delayfree) polynomial right inverse if and only if $\gamma_b(D) = 1$.

■ **EXAMPLE 2.9**

Since the encoding matrix $G(D)$ for the encoder in Fig. 2.9 satisfies the condition of Corollary 2.9, it has a polynomial right inverse. See Example 2.6

The catastrophic error situation when finitely many errors in the estimated codeword sequence could cause infinitely many errors in the estimated information sequence is closely related to the existence of a polynomial right inverse matrix $G^{-1}(D)$: if $\boldsymbol{v}(D)$ contains finitely many nonzero digits, then $\boldsymbol{u}(D) = \boldsymbol{v}(D)G^{-1}(D)$ also contains finitely many nonzero digits since $G^{-1}(D)$ is polynomial. No catastrophic error propagation can occur! This result is included in the following:

Theorem 2.10 A rational convolutional generator matrix $G(D)$ is noncatastrophic if and only if $\alpha_b(D) = D^s$ for some integer $s \geq 0$.

Proof: Suppose that $\alpha_b(D) = D^s$. Then from (2.64) it follows that $\alpha_i(D)$ divides D^s for $1 \leq i \leq b$. Thus, the matrix

$$D^s G^{-1}(D) = D^s B^{-1}(D) \begin{pmatrix} \frac{\beta_1(D)}{\alpha_1(D)} & & & & \\ & \frac{\beta_2(D)}{\alpha_2(D)} & & & \\ & & \ddots & & \\ & & & \frac{\beta_b(D)}{\alpha_b(D)} & \\ & & & 0 & \\ & & & \vdots & \\ & & & 0 & \end{pmatrix} A^{-1}(D) \tag{2.99}$$

is polynomial and, hence, $G(D)$ is noncatastrophic.

Conversely, suppose that $\alpha_b(D)$ is not a power of D. Then $\alpha_b(D)$ has a delayfree factor, and it follows that $w_{\mathrm{H}}(\beta_b(D)/\alpha_b(D)) = \infty$. The input sequence

$$\boldsymbol{u}(D) = (0\ 0 \ldots 0\ \beta_b(D)/\alpha_b(D))A^{-1}(D) \tag{2.100}$$

then also has $w_{\mathrm{H}}(\boldsymbol{u}(D)) = \infty$. But now we see that

$$\begin{aligned}\boldsymbol{v}(D) &= \boldsymbol{u}(D)G(D) \\ &= (0\ 0 \ldots 0\ \beta_b(D)/\alpha_b(D))A^{-1}(D)A(D)\Gamma(D)B(D) \\ &= (0\ 0 \ldots 0\ 1)B(D) \end{aligned} \tag{2.101}$$

which is polynomial and $w_{\mathrm{H}}(\boldsymbol{v}(D)) < \infty$ follows; that is, the generator matrix $G(D)$ is catastrophic. The proof is complete. ■

For the special case when $G(D)$ is polynomial we have the following:

Corollary 2.11 A polynomial convolutional generator matrix is noncatastrophic if and only if $\gamma_b(D) = D^s$ for some integer $s \geq 0$.

From Corollary 2.11, (2.36), and (2.40) the next corollary follows immediately [MaS68].

Corollary 2.12 A polynomial convolutional generator matrix is noncatastrophic if and only if $\Delta_b(D) = D^s$ for some integer $s \geq 0$.

Any $c \times b$ matrix $\widetilde{G}^{-1}(D)$ over $\mathbb{F}_2(D)$ is called a right *pseudoinverse* of the $b \times c$ matrix $G(D)$ if

$$G(D)\widetilde{G}^{-1}(D) = D^s I_b \tag{2.102}$$

for some $s \geq 0$.

Corollary 2.13 A rational convolutional generator matrix $G(D)$ is noncatastrophic if and only if it has a polynomial right pseudoinverse $\widetilde{G}^{-1}(D)$.

Proof: Follows from the proof of Theorem 2.10. ■

■ **EXAMPLE 2.10**

The rate $R = 1/2$ polynomial convolutional encoding matrix $G_2(D) = (1 + D^3 \quad 1 + D + D^2 + D^3)$, whose realization is shown in Fig. 2.5, has the Smith form decomposition (Problem 2.5)

$$G_2(D) = (1)(1 + D \quad 0) \begin{pmatrix} 1 + D + D^2 & 1 + D^2 \\ 1 + D & D \end{pmatrix} \tag{2.103}$$

Since $\gamma_b(D) = 1 + D \neq D^s$, the polynomial encoding matrix $G_2(D)$ is catastrophic.

■ **EXAMPLE 2.11**

From the Smith form of the rate $R = 2/3$ polynomial convolutional encoding matrix in Example 2.4 follows

$$\Gamma(D) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \tag{2.104}$$

Hence, the encoding matrix in Example 2.4 is noncatastrophic.

■ **EXAMPLE 2.12**

The rate $R = 2/3$ rational convolutional encoding matrix

$$G(D) = \begin{pmatrix} \dfrac{1}{1+D^2} & \dfrac{D}{1+D^2} & \dfrac{1}{1+D} \\ \dfrac{D}{1+D^2} & \dfrac{1}{1+D^2} & 1 \end{pmatrix} \tag{2.105}$$

has the invariant-factor decomposition (Problem 2.6)

$$\begin{aligned} G(D) = \frac{1}{1+D^2} \begin{pmatrix} 1 & 0 \\ D & 1 \end{pmatrix} \\ \times \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1+D & 0 \end{pmatrix} \begin{pmatrix} 1 & D & 1+D \\ 0 & 1+D & 1 \\ 0 & 1 & 0 \end{pmatrix} \end{aligned} \tag{2.106}$$

The encoding matrix $G(D)$ is noncatastrophic since $\gamma_2(D) = 1 + D$ divides $q(D) = 1 + D^2$ and, hence, $\alpha_2(D) = 1$.

■ **EXAMPLE 2.13**

The Smith form of the rate $R = 2/3$ polynomial convolutional encoding matrix $G(D)$ given in Example 2.8 is

$$\Gamma(D) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & D & 0 \end{pmatrix} \tag{2.107}$$

The encoding matrix $G(D)$ is noncatastrophic since $\gamma_2(D) = D$, but the right inverse $G^{-1}(D)$ is not realizable ($\gamma_2(D) \neq 1$).

2.4 ENCODER AND CODE EQUIVALENCES

In a communication context, it is natural to say that two encoders are equivalent if they generate the same code $\mathcal{C}$. It is therefore important to look for encoders with the lowest complexity within the class of equivalent encoders. We will study this problem in depth in the following sections.

Definition Two convolutional generator (encoding) matrices $G(D)$ and $G'(D)$ are *equivalent* if they encode the same code. Two convolutional encoders are *equivalent* if their generator matrices are equivalent.

Theorem 2.14 Two rate $R = b/c$ convolutional generator (encoding) matrices $G(D)$ and $G'(D)$ are equivalent if and only if there is a $b \times b$ nonsingular matrix $T(D)$ over $\mathbb{F}_2(D)$ such that

$$G(D) = T(D)G'(D) \tag{2.108}$$

Proof: If (2.108) holds, then $G(D)$ and $G'(D)$ are equivalent.

Conversely, suppose that $G(D)$ and $G'(D)$ are equivalent. Let $\boldsymbol{g}_i(D) \in \mathbb{F}_2^c(D)$ be the ith row of $G(D)$. Then there exists a $\boldsymbol{u}_i(D) \in \mathbb{F}_2^b((D))$ such that

$$\boldsymbol{g}_i(D) = \boldsymbol{u}_i(D)G'(D) \tag{2.109}$$

Let

$$T(D) = \begin{pmatrix} \boldsymbol{u}_1(D) \\ \boldsymbol{u}_2(D) \\ \vdots \\ \boldsymbol{u}_b(D) \end{pmatrix} \tag{2.110}$$

Then

$$G(D) = T(D)G'(D) \tag{2.111}$$

where $T(D)$ is a $b \times b$ matrix over $\mathbb{F}_2((D))$. Let $S'(D)$ be a $b \times b$ nonsingular submatrix of $G'(D)$ and $S(D)$ be the corresponding $b \times b$ submatrix of $G(D)$. Then $S(D) = T(D)S'(D)$. Thus, $T(D) = S(D)S'(D)^{-1}$ and, hence, $T(D)$ is over $\mathbb{F}_2(D)$. Since $G(D)$, being a generator matrix, has rank b, it follows that $T(D)$ also has rank b and, hence, is nonsingular. ■

■ EXAMPLE 2.14

Consider the encoding matrix

$$G'(D) = \begin{pmatrix} 1 & 1 & 1 \\ D & 1 & 0 \end{pmatrix} \tag{2.112}$$

If we multiply $G'(D)$ with

$$T(D) = \begin{pmatrix} 1 & 1 \\ D & 1 \end{pmatrix} \tag{2.113}$$

we obtain the equivalent encoding matrix

$$G(D) = T(D)G'(D) = \begin{pmatrix} 1+D & 1 & 1 \\ D & 1 & 0 \end{pmatrix} \tag{2.114}$$

Next we will carry the equivalence concept further and consider equivalent convolutional codes.

Definition Two convolutional codes $\mathcal{C}$ and $\mathcal{C}'$ are *equivalent* if the codewords $\boldsymbol{v} \in \mathcal{C}$ are permutations of the codewords $\boldsymbol{v'} \in \mathcal{C}'$.

If a rate $R = b/c$ convolutional code $\mathcal{C}$ is encoded by a rational generator matrix $G(D)$, we can always permute the columns of the generator matrix and obtain a generator matrix $G'(D)$. Encoding with $G'(D)$ yields a reordering of the code symbols *within* the c-tuples and, thus, we obtain an equivalent convolutional code $\mathcal{C}'$.

Consider the following two rate $R = 1/2$ convolutional encoding matrices:

$$G(D) = (1 \quad 1+D) \tag{2.115}$$

$$G'(D) = (D \quad 1+D) \tag{2.116}$$

According to Theorem 2.14 they are not equivalent; that is, they encode different convolutional codes. Moreover, $G(D)$ cannot be obtained from $G'(D)$ by a column permutation. However, if we consider the corresponding semi-infinite generator matrices

$$\boldsymbol{G} = \begin{pmatrix} 11 & 01 & & & & \\ & 11 & 01 & & & \\ & & 11 & 01 & & \\ & & & 11 & 01 & \\ & & & & \ddots & \ddots \end{pmatrix} \tag{2.117}$$

and

$$\boldsymbol{G'} = \begin{pmatrix} 01 & 11 & & & & \\ & 01 & 11 & & & \\ & & 01 & 11 & & \\ & & & 01 & 11 & \\ & & & & \ddots & \ddots \end{pmatrix} \tag{2.118}$$

we notice that, if we in $\boldsymbol{G'}$ delete the allzero column and permute the other columns, we can obtain $\boldsymbol{G}$. The two convolutional codes $\mathcal{C}$ and $\mathcal{C}'$ are equivalent and, thus, the encoding matrices $G(D)$ and $G'(D)$ given by (2.115) and (2.116) encode equivalent convolutional codes. This suggests a need for the following notation of equivalence.

Definition Two generator (encoding) matrices $G(D)$ and $G'(D)$ are *weakly equivalent* (WE) if they encode equivalent convolutional codes.

In a *permutation matrix* P we have exactly one 1 in each row and in each column. Next we consider a permutation matrix P and replace all 1s with monomials D^i, where i can be different integers for different matrix entries. Then we obtain a *generalized permutation matrix* $P(D)$. If we multiply from the right the codeword $\boldsymbol{v}(D) \in \mathbb{F}_2^c((D))$ by the $c \times c$ generalized permutation matrix $P(D)$, we obtain the codeword

$$\boldsymbol{v'}(D) = \boldsymbol{v}(D)P(D) \tag{2.119}$$

which is a particular kind of permutation of $\boldsymbol{v}(D) = (\boldsymbol{v}_1(D)\boldsymbol{v}_2(D)\ldots\boldsymbol{v}_c(D))$ such that

$$\boldsymbol{v'_l}(D) = \boldsymbol{v}_k(D)D^{i_{kl}} \tag{2.120}$$

where $D^{i_{kl}}$ is the (k,l)th entry of $P(D)$. If we would like to obtain a general permutation of the codeword $\boldsymbol{v}$ we have to multiply its semi-infinite generator matrix $\boldsymbol{G}$ by a semi-infinite permutation matrix.

From Theorem 2.14 we conclude that all generator matrices

$$G'(D) = T(D)G(D)P(D) \tag{2.121}$$

where $T(D)$ is a nonsingular rational $b \times b$ matrix and $P(D)$ is $c \times c$ generalized permutation matrix, are weakly equivalent to $G(D)$. From (2.121) follows the important observation that if we factor out D^i, $i > 0$, from a column of $G(D)$, we obtain a WE generator matrix. For example, if we factor out D from the first column of $G'(D)$ given in (2.116), we obtain the WE encoding matrix $G(D)$ given in (2.115).

■ **EXAMPLE 2.15**

The two rate $R = 3/5$ encoding matrices

$$G(D) = \begin{pmatrix} 1 & 1 & 1 & 1 & 0 \\ 0 & 1+D & 1+D & 0 & 1 \\ 0 & D & 1 & 1+D & D \end{pmatrix} \tag{2.122}$$

and

$$G'(D) = \begin{pmatrix} 1+D & 1+D & 0 & 1 & 1 \\ D & 1 & 1+D & D & 0 \\ D & D & D & 0 & 1 \end{pmatrix} \tag{2.123}$$

are WE since

$$G'(D) = T(D)G(D)P(D) \tag{2.124}$$

where

$$T(D) = \begin{pmatrix} 0 & D^{-1} & 0 \\ 0 & 0 & D^{-1} \\ 1 & 0 & 0 \end{pmatrix} \tag{2.125}$$

and

$$P(D) = \begin{pmatrix} 0 & 0 & 0 & 0 & 1 \\ D & 0 & 0 & 0 & 0 \\ 0 & D & 0 & 0 & 0 \\ 0 & 0 & D & 0 & 0 \\ 0 & 0 & 0 & D & 0 \end{pmatrix} \tag{2.126}$$

2.5 BASIC ENCODING MATRICES

In this section we will study a class of encoding matrices that were introduced by Forney [For70]. They will play an important role when we discuss minimality of convolutional generator matrices.

Definition A convolutional generator (encoding) matrix is called *basic* if it is polynomial and it has a polynomial right inverse. A convolutional encoder is called *basic* if its generator matrix is basic.

From Theorem 2.5 the next theorem follows immediately.

Theorem 2.15 A basic generator matrix is a basic encoding matrix.

Next we have an important theorem.

Theorem 2.16 Every rational generator matrix is equivalent to a basic encoding matrix.

Proof: By Theorem 2.2 every rational generator matrix has an equivalent polynomial delayfree generator matrix. Let the latter be $G(D)$ with the invariant-factor decomposition $G(D) = A(D)\Gamma(D)B(D)$, where $A(D)$ and $B(D)$ are $b \times b$ and $c \times c$ polynomial matrices, respectively, of determinant 1, and

$$\Gamma(D) = \begin{pmatrix} \gamma_1(D) & & & & & \\ & \gamma_2(D) & & & & \\ & & \ddots & & & \\ & & & \gamma_b(D) & 0 & \ldots & 0 \end{pmatrix} \tag{2.127}$$

Let $G'(D)$ be a generator matrix consisting of the first b rows of $B(D)$. Then

$$G(D) = A(D) \begin{pmatrix} \gamma_1(D) & & & \\ & \gamma_2(D) & & \\ & & \ddots & \\ & & & \gamma_b(D) \end{pmatrix} G'(D) \tag{2.128}$$

Since both $A(D)$ and

$$\begin{pmatrix} \gamma_1(D) & & & \\ & \gamma_2(D) & & \\ & & \ddots & \\ & & & \gamma_b(D) \end{pmatrix}$$

are nonsingular matrices over $\mathbb{F}_2(D)$, it follows from Theorem 2.14 that $G(D)$ and $G'(D)$ are equivalent. But $G'(D)$ is polynomial, and since $B(D)$ has a polynomial inverse, it follows that $G'(D)$ has a polynomial right inverse (consisting of the first b columns of $B^{-1}(D)$). Therefore, $G'(D)$ is a basic generator matrix. Then from Theorem 2.15 follows that $G'(D)$ is a basic encoding matrix. ∎

From Corollary 2.9 the next theorem follows immediately.

Theorem 2.17 A generator matrix is basic if and only if it is polynomial and $\gamma_b(D) = 1$.

Corollary 2.18 A basic encoding matrix $G(D)$ has a Smith form decomposition

$$G(D) = A(D)\Gamma(D)B(D) \tag{2.129}$$

where $A(D)$ is a $b \times b$ polynomial matrix with a unit determinant, $B(D)$ is a $c \times c$ polynomial matrix with a unit determinant, and $\Gamma(D)$ is the $b \times c$ matrix

$$\Gamma(D) = \begin{pmatrix} 1 & & & & & & \\ & 1 & & & & & \\ & & \ddots & & & & \\ & & & 1 & 0 & \dots & 0 \end{pmatrix} \tag{2.130}$$

Corollary 2.19 A basic encoding matrix is noncatastrophic.

Proof: Follows from Corollary 2.11 and Theorem 2.17. ∎

In the sequel, unless explicitly stated otherwise, we shall consider only basic encoders. As long as we do not require that the encoder should be systematic, we have nothing to gain from feedback!

Now we have the following:

Theorem 2.20 Two basic encoding matrices $G(D)$ and $G'(D)$ are equivalent if and only if $G'(D) = T(D)G(D)$, where $T(D)$ is a $b \times b$ polynomial matrix with determinant 1.

Proof: Let $G'(D) = T(D)G(D)$, where $T(D)$ is a polynomial matrix with determinant 1. By Theorem 2.14, $G(D)$ and $G'(D)$ are equivalent.

Conversely, suppose that $G'(D)$ and $G(D)$ are equivalent. By Theorem 2.14 there is a nonsingular $b \times b$ matrix $T(D)$ over $\mathbb{F}_2(D)$ such that $G'(D) = T(D)G(D)$. Since $G(D)$ is basic, it has a polynomial right inverse $G^{-1}(D)$. Then $T(D) = G'(D)G^{-1}(D)$ is polynomial. We can repeat the argument with $G(D)$ and $G'(D)$ reversed to obtain $G(D) = S(D)G'(D)$ for some polynomial matrix $S(D)$. Thus, $G(D) = S(D)T(D)G(D)$. Since $G(D)$ has full rank, we conclude that $S(D)T(D) = I_b$. Finally, since both $T(D)$ and $S(D)$ are polynomial, $T(D)$ must have determinant 1 and the proof is complete. ∎

Corollary 2.21 Let $G(D) = A(D)\Gamma(D)B(D)$ be the Smith form decomposition of a basic encoding matrix $G(D)$, and let $G'(D)$ be the $b \times c$ polynomial matrix that consists of the first b rows of the matrix $B(D)$. Then $G(D)$ and $G'(D)$ are equivalent basic encoding matrices.

Proof: Since $G(D)$ is basic, it follows from Corollary 2.18 that $G(D) = A(D)G'(D)$, where $A(D)$ is a $b \times b$ unimodular (determinant 1) matrix. Applying Theorem 2.20 completes the proof. ∎

■ **EXAMPLE 2.16**

The encoding matrix for the rate $R = 2/3$ convolutional encoder shown in Fig. 2.14,

$$G'(D) = \begin{pmatrix} 1+D & D & 1 \\ 1+D^2+D^3 & 1+D+D^2+D^3 & 0 \end{pmatrix} \tag{2.131}$$

is simply the first two rows of $B(D)$ in the Smith form decomposition of the encoding matrix (2.56) for the encoder in Fig. 2.9, viz.,

$$G(D) = \begin{pmatrix} 1+D & D & 1 \\ D^2 & 1 & 1+D+D^2 \end{pmatrix} \tag{2.132}$$

Thus, $G(D)$ and $G'(D)$ are equivalent, and the encoders in Figs. 2.9 and 2.14 encode the same code.

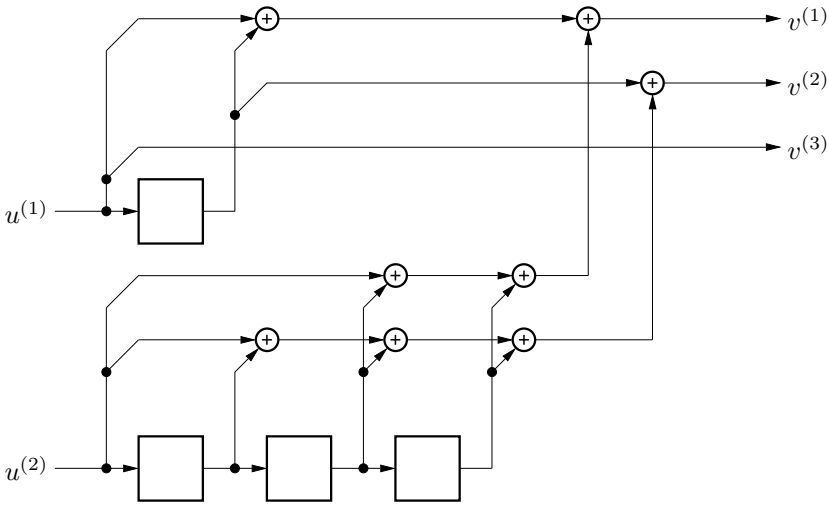


Figure 2.14 The controller canonical form of the encoding matrix $G'(D)$ in Example 2.16.

2.6 MINIMAL-BASIC ENCODING MATRICES

We begin by defining the *constraint length for the ith input* of a polynomial convolutional generator matrix as

$$\nu_i = \max_{1 \le j \le c} \{\deg g_{ij}(D)\} \tag{2.133}$$

the *memory* m of the polynomial generator matrix as the maximum of the constraint lengths, that is,

$$m = \max_{1 \le i \le b} \{\nu_i\} \tag{2.134}$$

and the *overall constraint length* as the sum of the constraint lengths

$$\nu = \sum_{i=1}^{b} \nu_i \tag{2.135}$$

The polynomial generator matrix can be realized in controller canonical form by a linear sequential circuit consisting of b shift registers, the ith of length ν_i, with the outputs formed as modulo 2 sums of the appropriate shift register contents. For example, in Fig. 2.9 we have shown the controllable canonical form of the polynomial encoder given by the encoding matrix

$$G(D) = \begin{pmatrix} 1+D & D & 1 \\ D^2 & 1 & 1+D+D^2 \end{pmatrix} \tag{2.136}$$

whose constraint lengths of the first and second inputs are 1 and 2, respectively, and whose overall constraint length is 3.

The number of memory elements required for the controller canonical form is equal to the overall constraint length.

In Fig. 2.14 we show the controller canonical form of a rate $R = 2/3$ encoder whose constraint lengths of the first and second inputs are 1 and 3, respectively, and the overall constraint length is 4.

We will now proceed and characterize the basic encoding matrix whose controller canonical form requires the least number of memory elements over all equivalent basic encoding matrices.

Definition A *minimal-basic* encoding matrix is a basic encoding matrix whose overall constraint length ν is minimal over all equivalent basic encoding matrices.

In the next section we shall show that a minimal-basic encoding matrix is also minimal in a more general sense.

Let $G(D)$ be a basic encoding matrix. The positions for the row-wise highest order coefficients in $G(D)$ will play a significant role in the sequel. Hence, we let $[G(D)]_h$ be a $(0,1)$-matrix with 1 in the position (i,j) where $\deg g_{ij}(D) = \nu_i$ and 0 otherwise.

Theorem 2.22 Let $G(D)$ be a $b \times c$ basic encoding matrix with overall constraint length ν. Then the following statements are equivalent:

(i) $G(D)$ is a minimal-basic encoding matrix.

(ii) The maximum degree μ among the $b \times b$ subdeterminants of $G(D)$ is equal to the overall constraint length ν.

(iii) $[G(D)]_h$ has full rank.

Proof: Let us write

$$G(D) = G_0(D) + G_1(D) \tag{2.137}$$

where

$$G_1(D) = \begin{pmatrix} D^{\nu_1} & & & \\ & D^{\nu_2} & & \\ & & \ddots & \\ & & & D^{\nu_b} \end{pmatrix} [G(D)]_h \tag{2.138}$$

Then all entries in the ith row of $G_0(D)$ are of degree $< \nu_i$. The maximum degree μ among the $b \times b$ subdeterminants of $G(D)$ is $\leq \nu$.

It follows immediately from (2.138) that (ii) and (iii) are equivalent. Thus we need only prove that (i) and (ii) are equivalent.

(i $\Rightarrow$ ii). Assume that $G(D)$ is minimal-basic.

Suppose that $\mu < \nu$, that is, rank $[G(D)]_h < b$. Denote the rows of $G(D)$ by $\boldsymbol{r}_1, \boldsymbol{r}_2, \ldots, \boldsymbol{r}_b$ and the rows of $[G(D)]_h$ by $[\boldsymbol{r}_1], [\boldsymbol{r}_2], \ldots, [\boldsymbol{r}_b]$. Then there is a linear relation

$$[\boldsymbol{r}_{i_1}] + [\boldsymbol{r}_{i_2}] + \cdots + [\boldsymbol{r}_{i_d}] = \mathbf{0} \tag{2.139}$$

The ith row of $G_1(D)$ is $D^{\nu_i}[\boldsymbol{r}_i]$. Without loss of generality we can assume that $\nu_{i_d} \geq \nu_{i_j}$, $j = 1, 2, \ldots, d-1$. Adding

$$\begin{aligned} D^{\nu_{i_d} - \nu_{i_1}} D^{\nu_{i_1}} [\boldsymbol{r}_{i_1}] + D^{\nu_{i_d} - \nu_{i_2}} D^{\nu_{i_2}} [\boldsymbol{r}_{i_2}] + \cdots + D^{\nu_{i_d} - \nu_{i_{d-1}}} D^{\nu_{i_{d-1}}} [\boldsymbol{r}_{i_{d-1}}] \\ = D^{\nu_{i_d}} ([\boldsymbol{r}_{i_1}] + [\boldsymbol{r}_{i_2}] + \cdots + [\boldsymbol{r}_{i_{d-1}}]) \end{aligned} \tag{2.140}$$

to the i_dth row of $G_1(D)$ reduces it to an allzero row. Similarly, adding

$$D^{\nu_{i_d} - \nu_{i_1}} \boldsymbol{r}_1 + D^{\nu_{i_d} - \nu_{i_2}} \boldsymbol{r}_2 + \cdots + D^{\nu_{i_d} - \nu_{i_{d-1}}} \boldsymbol{r}_{i_{d-1}} \tag{2.141}$$

to the i_dth row of $G(D)$ will reduce the highest degree of the i_dth row of $G(D)$ but leave the other rows of $G(D)$ unchanged. Thus, we obtain a basic encoding matrix equivalent to $G(D)$ with an overall constraint length that is less than that of $G(D)$. This is a contradiction to the assumption that $G(D)$ is minimal-basic, and we conclude that $\mu = \nu$.

(ii $\Rightarrow$ i). Assume that $\mu = \nu$.

Let $G'(D)$ be a basic encoding matrix equivalent to $G(D)$. From Theorem 2.20 it follows that $G'(D) = T(D)G(D)$, where $T(D)$ is a $b \times b$ polynomial matrix with determinant 1. Since $\det T(D) = 1$, the maximum degree among the $b \times b$ subdeterminants of $G'(D)$ is equal to that of $G(D)$. Hence, μ is invariant over all equivalent basic encoding matrices. Since μ is less than or equal to the overall constraint length for all equivalent basic encoding matrices, it follows that $G(D)$ is a minimal-basic encoding matrix. ■

■ EXAMPLE 2.17

The basic encoding matrix for the encoder in Fig. 2.9,

$$G(D) = \begin{pmatrix} 1+D & D & 1 \\ D^2 & 1 & 1+D+D^2 \end{pmatrix} \tag{2.142}$$

has

$$[G(D)]_h = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix} \tag{2.143}$$

with full rank and, hence, is a minimal-basic encoding matrix.

Corollary 2.23 Let $G(D)$ be a $b \times c$ basic encoding matrix with maximum degree μ among its $b \times b$ subdeterminants. Then $G(D)$ has an equivalent minimal-basic encoding matrix whose overall constraint length $\nu = \mu$.

Proof: Follows from the proof of Theorem 2.22 and the fact that μ is invariant over all equivalent basic encoding matrices. ■

■ **EXAMPLE 2.18**

Consider the encoding matrix for the encoder in Fig. 2.14, viz.,

$$G'(D) = \begin{pmatrix} 1+D & D & 1 \\ 1+D^2+D^3 & 1+D+D^2+D^3 & 0 \end{pmatrix} \tag{2.144}$$

The rank of

$$[G'(D)]_h = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \end{pmatrix} \tag{2.145}$$

is 1. Hence, $G'(D)$ cannot be a minimal-basic encoding matrix.

On the other hand, $G'(D)$ has the following three $b \times b$ subdeterminants:

$$1+D+D^3, \quad 1+D^2+D^3, \quad 1+D+D^2+D^3$$

and, thus, $\mu = 3$. Hence, any minimal-basic matrix equivalent to $G'(D)$ has overall constraint length $\nu = 3$.

The equivalent basic encoding matrix for the encoder in Fig. 2.9 has $[G(D)]_h$ of full rank (see Example 2.17) and, hence, is such a minimal-basic encoding matrix.

We can use the technique in the proof of Theorem 2.22 to obtain a minimal-basic encoding matrix equivalent to the basic encoding matrix $G'(D)$ in Example 2.18. We simply multiply the first row of $G'(D)$ by $D^{\nu_2 - \nu_1} = D^2$ and add it to the second row:

$$\begin{pmatrix} 1 & 0 \\ D^2 & 1 \end{pmatrix} \begin{pmatrix} 1+D & D & 1 \\ 1+D^2+D^3 & 1+D+D^2+D^3 & 0 \end{pmatrix} = \begin{pmatrix} 1+D & D & 1 \\ 1 & 1+D+D^2 & D^2 \end{pmatrix} \tag{2.146}$$

It is easily shown that the two minimal-basic encoding matrices (2.142) and (2.146) are equivalent. Thus, a minimal-basic encoding matrix equivalent to a given basic encoding matrix is not necessarily unique.

In general, we have [For70] the following simple algorithm to construct a minimal-basic encoding matrix equivalent to a given basic encoding matrix:

Algorithm MB (Minimal-basic encoding matrix)

MB1. If $[G(D)]_h$ has full rank, then $G(D)$ is a minimal-basic encoding matrix and we STOP; otherwise go to the next step.

MB2. Let $[\boldsymbol{r}_{i_1}], [\boldsymbol{r}_{i_2}], \ldots, [\boldsymbol{r}_{i_d}]$ denote a set of rows of $[G(D)]_h$ such that $\nu_{i_d} \geq \nu_{i_j}$, $1 \leq j < d$, and

$$[\boldsymbol{r}_{i_1}] + [\boldsymbol{r}_{i_2}] + \cdots + [\boldsymbol{r}_{i_d}] = \mathbf{0}$$

Let $\boldsymbol{r}_{i_1}, \boldsymbol{r}_{i_2}, \ldots, \boldsymbol{r}_{i_d}$ denote the corresponding set of rows of $G(D)$. Add

$$D^{\nu_{i_d}-\nu_{i_1}}\boldsymbol{r}_{i_1} + D^{\nu_{i_d}-\nu_{i_2}}\boldsymbol{r}_{i_2} + \cdots + D^{\nu_{i_d}-\nu_{i_{d-1}}}\boldsymbol{r}_{i_{d-1}}$$

to the i_dth row of $G(D)$.

Call the new matrix $G(D)$ and go to MB1.

By combining Theorem 2.16 and Corollary 2.23, we have the next corollary.

Corollary 2.24 Every rational generator matrix is equivalent to a minimal-basic encoding matrix.

Before we prove that the constraint lengths are invariants of equivalent minimal-basic encoding matrices, we need the following:

Lemma 2.25 Let V be a k-dimensional vector space over a field F, and let $\{\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2, \ldots, \boldsymbol{\alpha}_k\}$ be a basis of V. Let $\{\boldsymbol{\beta}_1, \boldsymbol{\beta}_2, \ldots, \boldsymbol{\beta}_\ell\}$ be a set of ℓ, $\ell < k$, linearly independent vectors of V. Then there exist $k - \ell$ vectors $\boldsymbol{\alpha}_{i_{\ell+1}}, \boldsymbol{\alpha}_{i_{\ell+2}}, \ldots, \boldsymbol{\alpha}_{i_k}$, $1 \leq i_{\ell+1} < i_{\ell+2} < \cdots < i_k \leq k$, such that $\{\boldsymbol{\beta}_1, \boldsymbol{\beta}_2, \ldots, \boldsymbol{\beta}_\ell, \boldsymbol{\alpha}_{i_{\ell+1}}, \boldsymbol{\alpha}_{i_{\ell+2}}, \ldots, \boldsymbol{\alpha}_{i_k}\}$ is also a basis of V.

Proof: Consider the vectors in the sequence $\boldsymbol{\beta}_1, \boldsymbol{\beta}_2, \ldots, \boldsymbol{\beta}_\ell, \boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2, \ldots, \boldsymbol{\alpha}_k$ one by one successively from left to right. If the vector under consideration is a linear combination of vectors to the left of it, then delete it; otherwise keep it. Finally, we obtain a basis $\boldsymbol{\beta}_1, \boldsymbol{\beta}_2, \ldots, \boldsymbol{\beta}_\ell, \boldsymbol{\alpha}_{i_{\ell+1}}, \boldsymbol{\alpha}_{i_{\ell+2}}, \ldots, \boldsymbol{\alpha}_{i_k}$, $1 \leq i_{\ell+1} < \cdots < i_k \leq k$, of V. ∎

Theorem 2.26 The constraint lengths of two equivalent minimal-basic encoding matrices are equal one by one up to a rearrangement.

Proof: Let $G(D)$ and $G'(D)$ be two equivalent minimal-basic encoding matrices with constraint lengths $\nu_1, \nu_2, \ldots, \nu_b$ and $\nu'_1, \nu'_2, \ldots, \nu'_b$, respectively. Without loss of generality, we assume that $\nu_1 \leq \nu_2 \leq \cdots \leq \nu_b$ and $\nu'_1 \leq \nu'_2 \leq \cdots \leq \nu'_b$.

Now suppose that ν_i and ν'_i are not equal for all i, $1 \leq i \leq b$. Let j be the smallest index such that $\nu_j \neq \nu'_j$. Then without loss of generality we assume that $\nu_j < \nu'_j$. From the sequence $\boldsymbol{g}_1(D), \boldsymbol{g}_2(D), \ldots, \boldsymbol{g}_j(D), \boldsymbol{g}'_1(D), \boldsymbol{g}'_2(D), \ldots, \boldsymbol{g}'_b(D)$ according to Lemma 2.25 we can obtain a basis $\boldsymbol{g}_1(D), \boldsymbol{g}_2(D), \ldots, \boldsymbol{g}_j(D), \boldsymbol{g}'_{i_{j+1}}(D)$,

$\boldsymbol{g}'_{i_{j+2}}(D), \ldots, \boldsymbol{g}'_{i_b}(D)$ of $\mathcal{C}$. These b row vectors form an encoding matrix $G''(D)$ which is equivalent to $G'(D)$. Let

$$\left\{\boldsymbol{g}'_1(D), \boldsymbol{g}'_2(D), \ldots, \boldsymbol{g}'_b(D)\right\} \setminus \left\{\boldsymbol{g}'_{i_{j+1}}(D), \boldsymbol{g}'_{i_{j+2}}(D), \ldots, \boldsymbol{g}'_{i_b}(D)\right\} = \left\{\boldsymbol{g}'_{i_1}(D), \boldsymbol{g}'_{i_2}(D), \ldots, \boldsymbol{g}'_{i_j}(D)\right\} \tag{2.147}$$

From our assumptions it follows that

$$\sum_{\ell=1}^{j} \nu_i < \sum_{\ell=1}^{j} \nu'_i \leq \sum_{\ell=1}^{j} \nu'_{i_\ell} \tag{2.148}$$

Then we have

$$\nu'' = \sum_{\ell=1}^{j} \nu_i + \sum_{\ell=j+1}^{b} \nu'_{i_\ell} < \sum_{\ell=1}^{j} \nu'_{i_\ell} + \sum_{\ell=j+1}^{b} \nu'_{i_\ell} = \nu' \tag{2.149}$$

where ν' and ν'' are the overall constraint lengths of the encoding matrices $G'(D)$ and $G''(D)$, respectively. From Theorem 2.14 it follows that there exists a $b \times b$ nonsingular matrix $T(D)$ over $\mathbb{F}_2(D)$ such that

$$G''(D) = T(D)G'(D) \tag{2.150}$$

Since $G'(D)$ is basic, it has a polynomial right inverse $G'^{-1}(D)$, and it follows that

$$T(D) = G''(D)G'^{-1}(D) \tag{2.151}$$

is polynomial. Denote by μ' and μ'' the highest degrees of the $b \times b$ minors of $G'(D)$ and $G''(D)$, respectively. It follows from (2.150) that

$$\mu'' = \deg \mid T(D) \mid + \mu' \tag{2.152}$$

Clearly, $\nu'' \geq \mu''$ and, since $G'(D)$ is minimal-basic, $\nu' = \mu'$ by Theorem 2.22. Thus,

$$\nu'' \geq \deg \mid T(D) \mid + \nu' \geq \nu' \tag{2.153}$$

which contradicts (2.149) and the proof is complete. ∎

Corollary 2.27 Two equivalent minimal-basic encoding matrices have the same memory.

Next, we will consider the *predictable degree property* for polynomial generator matrices, which is a useful analytic tool when we study the structural properties of convolutional generator matrices.

Let $G(D)$ be a rate $R = b/c$ binary polynomial generator matrix with ν_i as the constraint length of its ith row $\boldsymbol{g}_i(D)$. For any polynomial input $\boldsymbol{u}(D) =$

$(u_1(D)\ u_2(D) \ldots u_b(D))$, the output $\boldsymbol{v}(D) = \boldsymbol{u}(D)G(D)$ is also polynomial. We have

$$\begin{aligned} \deg \boldsymbol{v}(D) &= \deg \boldsymbol{u}(D)G(D) = \deg \sum_{i=1}^{b} u_i(D)\boldsymbol{g}_i(D) \\ &\leq \max_{1 \leq i \leq b} \{\deg u_i(D) + \nu_i\} \end{aligned} \tag{2.154}$$

where the degree of a polynomial vector is defined to be the maximum of the degrees of its components.

Definition A polynomial generator matrix $G(D)$ is said to have the *predictable degree property (PDP)* if for all polynomial inputs $\boldsymbol{u}(D)$ we have equality in (2.154).

The predictable degree property guarantees that short codewords will be associated with short information sequences. We have the following:

Theorem 2.28 Let $G(D)$ be a polynomial generator matrix. Then $G(D)$ has the predictable degree property if and only if $[G(D)]_h$ has full rank.

Proof: Without loss of generality, we assume that

$$\nu_1 \geq \nu_2 \geq \cdots \geq \nu_b \tag{2.155}$$

Let us write

$$G(D) = G_0(D) + G_1(D) \tag{2.156}$$

where

$$G_1(D) = \begin{pmatrix} D^{\nu_1} & & & \\ & D^{\nu_2} & & \\ & & \ddots & \\ & & & D^{\nu_b} \end{pmatrix} [G(D)]_h \tag{2.157}$$

Then, all entries in the ith row of $G_0(D)$ are of degree $< \nu_i$.

Assume that $[G(D)]_h$ has full rank. For any input polynomial vector $\boldsymbol{u}(D)$ we have

$$\begin{aligned} \boldsymbol{v}(D) &= \boldsymbol{u}(D)G(D) = \boldsymbol{u}(D)(G_0(D) + G_1(D)) \\ &= \sum_{i=1}^{b} u_i(D)(\boldsymbol{g}_{0i}(D) + D^{\nu_i}[G(D)]_{hi}) \end{aligned} \tag{2.158}$$

where $\boldsymbol{g}_{0i}(D)$ and $[G(D)]_{hi}$ are the ith rows of $G_0(D)$ and $[G(D)]_h$, respectively. Since $[G(D)]_h$ has full rank, we have

$$[G(D)]_{hi} \neq \mathbf{0}, \quad i = 1, 2, \ldots, b \tag{2.159}$$

Thus,

$$\begin{aligned} &\deg(u_i(D)\boldsymbol{g}_{0i}(D) + u_i(D)D^{\nu_i}[G(D)]_{hi}) \\ &\quad = \deg(u_i(D)D^{\nu_i}[G(D)]_{hi}) = \deg u_i(D) + \nu_i \end{aligned} \tag{2.160}$$

It follows from (2.158) and (2.160) that

$$\deg \boldsymbol{v}(D) = \max_{1 \le i \le b} \{\deg u_i(D) + \nu_i\} \tag{2.161}$$

Now assume that $[G(D)]_h$ does not have full rank. Then, there exists a nonzero constant binary vector $\boldsymbol{u}^{(0)} = (u_1^{(0)} u_2^{(0)} \ldots u_b^{(0)})$ such that

$$\boldsymbol{u}^{(0)}[G(D)]_h = \mathbf{0} \tag{2.162}$$

Let $\boldsymbol{u}^{(0)}(D) = (u_1^{(0)} \; u_2^{(0)} D^{\nu_1 - \nu_2} \ldots u_b^{(0)} D^{\nu_1 - \nu_b})$ be a polynomial input vector. Then,

$$\begin{aligned} \boldsymbol{v}^{(0)}(D) = \boldsymbol{u}^{(0)}(D)G(D) &= \boldsymbol{u}^{(0)}(D)(G_0(D) + G_1(D)) \\ &= \boldsymbol{u}^{(0)}(D)G_0(D) \\ &= \sum_{i=1}^{b} u_i^{(0)} D^{\nu_1 - \nu_i} \boldsymbol{g}_{0i}(D) \end{aligned} \tag{2.163}$$

Since $\deg \boldsymbol{g}_{0i}(D) < \nu_i$, it follows that

$$\deg u_i^{(0)} D^{\nu_1 - \nu_i} \boldsymbol{g}_{0i}(D) < \nu_1, \;\; i = 1, 2, \ldots, b \tag{2.164}$$

and, hence, that

$$\deg \boldsymbol{v}^{(0)}(D) < \nu_1 \tag{2.165}$$

But

$$\max_{1 \le i \le b} \{\deg u_i^{(0)} D^{\nu_1 - \nu_i} + \nu_i\} = \nu_1 \tag{2.166}$$

Therefore, the $G(D)$ does not have the predictable degree property. ■

Since a basic encoding matrix is minimal-basic if and only if $[G(D)]_h$ has full rank (Theorem 2.22), we immediately have the following theorem [For73]:

Theorem 2.29 Let $G(D)$ be a basic encoding matrix. Then $G(D)$ has the predictable degree property if and only if it is minimal-basic.

■ **EXAMPLE 2.19**

The catastrophic (and, hence, not basic) encoding matrix

$$G(D) = (1 + D^3 \quad 1 + D + D^2 + D^3) \tag{2.167}$$

has the predictable degree property since

$$[G(D)]_h = \begin{pmatrix} 1 & 1 \end{pmatrix} \tag{2.168}$$

has full rank.

■ **EXAMPLE 2.20**

The basic encoding matrix (cf. Example 2.18)

$$G(D) = \begin{pmatrix} 1+D & D & 1 \\ 1+D^2+D^3 & 1+D+D^2+D^3 & 0 \end{pmatrix} \tag{2.169}$$

has

$$[G(D)]_h = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \end{pmatrix} \tag{2.170}$$

of rank 1 and, hence, does not have the predictable degree property.

2.7 MINIMAL ENCODING MATRICES AND MINIMAL ENCODERS

We will now proceed to show that a minimal-basic encoding matrix is also minimal in a more general sense, but first we need the following definitions:

Definition The *encoder state* $\boldsymbol{\sigma}$ of a realization of a rational generator matrix $G(D)$ is the contents of its memory elements. The set of encoder states is called the *encoder state space*.

If $G(D)$ is polynomial, then the dimension of the encoder state space of its controller canonical form is equal to the overall constraint length ν.

Definition Let $G(D)$ be a rational generator matrix. The *abstract state* $\boldsymbol{s}(D)$ associated with an input sequence $\boldsymbol{u}(D)$ is the sequence of outputs at time 0 and later, which are due to that part of $\boldsymbol{u}(D)$ that occurs up to time -1 and to the allzero inputs thereafter. The set of abstract states is called the *abstract state space*.

The abstract state depends only on the generator matrix and not on its realization. Distinct abstract states must spring from distinct encoder states at time 0. The number of encoder states is greater than or equal to the number of abstract states.

Let P be the projection operator that truncates sequences to end at time -1, and let $Q = 1 - P$ be the projection operator that truncates sequences to start at time 0. That is, if

$$\boldsymbol{u}(D) = \boldsymbol{u}_d D^d + \boldsymbol{u}_{d+1} D^{d+1} + \cdots \tag{2.171}$$

then

$$\boldsymbol{u}(D)P = \begin{cases} \boldsymbol{u}_d D^d + \boldsymbol{u}_{d+1} D^{d+1} + \cdots + \boldsymbol{u}_{-1} D^{-1}, & d < 0 \\ \boldsymbol{0}, & d \geq 0 \end{cases} \tag{2.172}$$

and

$$\boldsymbol{u}(D)Q = \boldsymbol{u}_0 + \boldsymbol{u}_1 D + \boldsymbol{u}_2 D^2 + \cdots \tag{2.173}$$

Clearly,

$$P + Q = 1 \tag{2.174}$$

Thus, the abstract state $\boldsymbol{s}(D)$ associated with $\boldsymbol{u}(D)$ can be written concisely as

$$\boldsymbol{s}(D) = \boldsymbol{u}(D)PG(D)Q \tag{2.175}$$

The encoder in Fig. 2.14 has 16 encoder states and 8 abstract states. The correspondence between them is tabulated in Table 2.1.

Table 2.1 Correspondence between encoder and abstract states of encoder $G'(D)$ illustrated in Fig. 2.14.

Encoder states (at time 0)	Abstract state
$\begin{pmatrix} 0 \\ 000 \end{pmatrix}, \begin{pmatrix} 1 \\ 001 \end{pmatrix}$	$(0 \quad 0 \quad 0)$
$\begin{pmatrix} 0 \\ 001 \end{pmatrix}, \begin{pmatrix} 1 \\ 000 \end{pmatrix}$	$(1 \quad 1 \quad 0)$
$\begin{pmatrix} 0 \\ 010 \end{pmatrix}, \begin{pmatrix} 1 \\ 011 \end{pmatrix}$	$(1+D \quad 1+D \quad 0)$
$\begin{pmatrix} 0 \\ 011 \end{pmatrix}, \begin{pmatrix} 1 \\ 010 \end{pmatrix}$	$(D \quad D \quad 0)$
$\begin{pmatrix} 0 \\ 100 \end{pmatrix}, \begin{pmatrix} 1 \\ 101 \end{pmatrix}$	$(D+D^2 \quad 1+D+D^2 \quad 0)$
$\begin{pmatrix} 0 \\ 101 \end{pmatrix}, \begin{pmatrix} 1 \\ 100 \end{pmatrix}$	$(1+D+D^2 \quad D+D^2 \quad 0)$
$\begin{pmatrix} 0 \\ 110 \end{pmatrix}, \begin{pmatrix} 1 \\ 111 \end{pmatrix}$	$(1+D^2 \quad D^2 \quad 0)$
$\begin{pmatrix} 0 \\ 111 \end{pmatrix}, \begin{pmatrix} 1 \\ 110 \end{pmatrix}$	$(D^2 \quad 1+D^2 \quad 0)$

For example, the input sequence

$$\boldsymbol{u}(D) = \cdots + (0\ 1)D^{-3} + (0\ 1)D^{-2} + (0\ 0)D^{-1} + \cdots \tag{2.176}$$

will give the encoder state

$$\begin{pmatrix} 0 \\ 011 \end{pmatrix}$$

at time 0. The corresponding abstract state is

$$\begin{aligned}\boldsymbol{s}(D) &= \boldsymbol{u}(D)PG(D)Q \\ &= \Big((D^{-3}+D^{-2})(1+D^2+D^3) \\ &\qquad (D^{-3}+D^{-2})(1+D+D^2+D^3) \quad 0\Big)Q \\ &= (D \quad D \quad 0) \end{aligned} \tag{2.177}$$

In Fig. 2.15 we give the observer canonical form for the encoding matrix $G'(D)$ in Example 2.16. Note that in the observer canonical form of a polynomial generator matrix the abstract states are in one-to-one correspondence with the encoder states since the contents of the memory elements are simply shifted out in the absence of nonzero inputs.

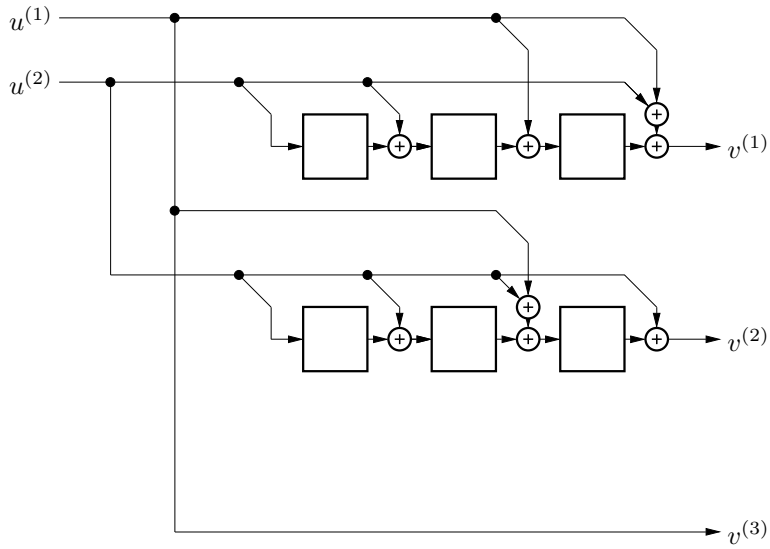


Figure 2.15 The observer canonical form of encoding matrix $G'(D)$ in Example 2.16.

Since the abstract state does not depend on the realization, we have the same abstract states in the observer canonical form as in the controller canonical form.

At a first glance it might be surprising that we have only 8 abstract states but as many as 64 encoder states in the observer canonical form in Fig. 2.15. We have some (in this case 56) encoder states that cannot be reached from the zero state—this realization is not controllable.

The encoder state space of the controller canonical form of a generator matrix of overall constraint length ν contains 2^ν states. This type of realization plays an important role in connection with minimal-basic encoding matrices, as we shall see in the sequel, but first we prove a technical lemma.

Lemma 2.30 Let $G(D)$ be a minimal-basic encoding matrix, and let

$$\boldsymbol{u}(D) = \sum_i (u_i^{(1)} u_i^{(2)} \dots u_i^{(b)}) D^i \tag{2.178}$$

be of finite weight. If $\boldsymbol{u}(D)G(D)Q = \mathbf{0}$, then $u_i^{(j)} = 0$ for each $j = 1, 2, \dots, b$ and each i such that $i \geq -\nu_j$.

Proof: If $\boldsymbol{u}(D) = \mathbf{0}$, then the claim is obvious. Let therefore $\boldsymbol{u}(D) \neq \mathbf{0}$. Denote

$$\boldsymbol{v}(D) = \boldsymbol{u}(D)G(D) \tag{2.179}$$

and let k denote the time in which the controller canonical form encoder of $G(D)$ will contain the last nonzero elements of $\boldsymbol{u}(D)$. Then

$$k = \max_j \left\{ \deg \boldsymbol{u}^{(j)}(D) + \nu_j \right\} \tag{2.180}$$

It follows from (2.180) that

$$\boldsymbol{\nu}_k = \sum_{j=1}^{b} u_{k-\nu_j}^{(j)} [\boldsymbol{r}_j] \tag{2.181}$$

where $[\boldsymbol{r}_j]$ is the jth row of $[G(D)]_h$.

Since $G(D)$ is minimal-basic, the rows $[\boldsymbol{r}_j]$ are linearly independent. Since $u_{k-\nu_j}^{(j)} \neq 0$ for all j for which the maximum in the definition of k is achieved, we obtain $\boldsymbol{\nu}_k \neq \mathbf{0}$. From $\boldsymbol{\nu}(D)Q = \mathbf{0}$, we have $k < 0$ and, hence, $\deg \boldsymbol{u}^{(j)} < -\nu_j$ for all j as claimed. ∎

Remark: The authors are grateful to Stepan Holub, who found an error in Lemma 2.30 in the first edition. This formulation of the lemma is due to Holub. The proof is a slightly shortened version of his proof.

The following theorem shows that the controller canonical form is a minimal realization (minimal number of memory elements) of a given minimal-basic encoding matrix.

Theorem 2.31 Let $G(D)$ be a minimal-basic encoding matrix whose overall constraint length is ν. Then[7]

$$\#\{\text{abstract states}\} = 2^{\nu} \tag{2.182}$$

Proof: Consider the controller canonical form of the minimal-basic encoding matrix $G(D)$. Input sequences of the form

$$\boldsymbol{u}(D) = \left(\sum_{i=1}^{\nu_1} u_{-i}^{(1)} D^{-i} \quad \sum_{i=1}^{\nu_2} u_{-i}^{(2)} D^{-i} \quad \dots \quad \sum_{i=1}^{\nu_b} u_{-i}^{(b)} D^{-i} \right) \tag{2.183}$$

[7] # { } denotes the cardinality of the set { }.

will carry us to all encoder states at time 0. Then we have the abstract states

$$\boldsymbol{s}(D) = \boldsymbol{u}(D)G(D)Q \tag{2.184}$$

where $\boldsymbol{u}(D)$ is of the form given in (2.183). Every abstract state can be obtained in this way, and we have

$$\#\{\text{abstract states}\} \leq 2^{\nu} \tag{2.185}$$

To prove that the equality sign holds in (2.185), it is enough to show that $\boldsymbol{u}(D) = \mathbf{0}$ is the only input that produces the abstract state $\boldsymbol{s}(D) = \mathbf{0}$. This follows from Lemma 2.30. ■

■ **EXAMPLE 2.21**

The encoder illustrated in Fig. 2.9 has the following eight abstract states:

$$(0 \quad 0 \quad 0), (1 \quad 0 \quad 1), (D \quad 0 \quad D), (1+D \quad 0 \quad D), (1 \quad 1 \quad 0),$$
$$(0 \quad 1 \quad 1), (1+D \quad 1 \quad 1+D), (D \quad 1 \quad D)$$

Definition A convolutional generator matrix is *minimal* if its number of abstract states is minimal over all equivalent generator matrices.

Before we can show that every minimal-basic encoding matrix is also a (basic) minimal encoding matrix, we have to prove the following lemmas:

Lemma 2.32 Only the zero abstract state of a minimal-basic encoding matrix $G(D)$ can be a codeword.

Proof: We can assume that the abstract state $\boldsymbol{s}(D)$ arises from an input $\boldsymbol{u}(D)$, which is polynomial in D^{-1} and of degree $\leq m$ and without a constant term, that is, $\boldsymbol{u}(0) = \mathbf{0}$. Thus,

$$\boldsymbol{s}(D) = \boldsymbol{u}(D)G(D)Q \tag{2.186}$$

Then it follows that

$$\boldsymbol{u}(D)G(D) = \boldsymbol{w}(D) + \boldsymbol{s}(D) \tag{2.187}$$

where $\boldsymbol{w}(D)$ is polynomial in D^{-1} without a constant term. Assume that $\boldsymbol{s}(D)$ is a codeword, that is, there is an input $\boldsymbol{u}'(D) \in \mathbb{F}_2((D))$ such that

$$\boldsymbol{s}(D) = \boldsymbol{u}'(D)G(D) \tag{2.188}$$

Since $\boldsymbol{s}(D)$ is polynomial and $G(D)$ has a polynomial inverse, it follows that $\boldsymbol{u}'(D) \in \mathbb{F}_2[D]$. Combining (2.187) and (2.188), we have

$$(\boldsymbol{u}(D) + \boldsymbol{u}'(D))G(D) = \boldsymbol{w}(D) \tag{2.189}$$

Consequently

$$(\boldsymbol{u}(D) + \boldsymbol{u}'(D))G(D)Q = \mathbf{0} \tag{2.190}$$

By Lemma 2.30,

$$\boldsymbol{u}(D) + \boldsymbol{u}'(D) = \mathbf{0} \tag{2.191}$$

and, since $\boldsymbol{u}(D)$ is polynomial in D^{-1} without a constant term and $\boldsymbol{u}'(D)$ is polynomial, we conclude that $\boldsymbol{u}'(D) = \mathbf{0}$. It follows from (2.188) that $\boldsymbol{s}(D) = \mathbf{0}$. ∎

Lemma 2.33 Let $G(D)$ and $G'(D)$ be equivalent generator matrices. Then, every abstract state of $G(D)$ can be expressed as a sum of an abstract state of $G'(D)$ and a codeword. Furthermore, if $G'(D)$ is minimal-basic, then the expression is unique.

Proof: Assume that $G(D) = T(D)G'(D)$, where $T(D)$ is a $b \times b$ nonsingular matrix over $\mathbb{F}_2(D)$. Any abstract state of $G(D)$, $\boldsymbol{s}_G(D)$, can be written in the form $\boldsymbol{u}(D)G(D)Q$, where $\boldsymbol{u}(D)$ is polynomial in D^{-1} without a constant term. Thus, we have

$$\begin{aligned}\boldsymbol{s}_G(D) &= \boldsymbol{u}(D)G(D)Q = \boldsymbol{u}(D)T(D)G'(D)Q \\ &= \boldsymbol{u}(D)T(D)(P+Q)G'(D)Q \\ &= \boldsymbol{u}(D)T(D)PG'(D)Q + \boldsymbol{u}(D)T(D)QG'(D)Q\end{aligned} \tag{2.192}$$

Since $\boldsymbol{u}(D)T(D)P$ is polynomial in D^{-1} without a constant term, it follows from (2.175) that

$$\boldsymbol{s}_{G'}(D) = \boldsymbol{u}(D)T(D)PG'(D)Q \tag{2.193}$$

is an abstract state of $G'(D)$. Furthermore, $\boldsymbol{u}(D)T(D)Q$ is a formal power series, and so is $\boldsymbol{u}(D)T(D)QG'(D)$. Hence,

$$\begin{aligned}\boldsymbol{v}(D) &\stackrel{\text{def}}{=} \boldsymbol{u}(D)T(D)QG'(D)Q \\ &= \boldsymbol{u}(D)T(D)QG'(D)\end{aligned} \tag{2.194}$$

is a codeword encoded by $G'(D)$. Combining (2.192), (2.193), and (2.194), we obtain

$$\boldsymbol{s}_G(D) = \boldsymbol{s}_{G'}(D) + \boldsymbol{v}(D) \tag{2.195}$$

and we have proved that every abstract state of $G(D)$ can be written as a sum of an abstract state of $G'(D)$ and a codeword.

Assume now that $G'(D) = G_{\text{mb}}(D)$ is minimal-basic. To prove uniqueness we assume that

$$\boldsymbol{s}_G(D) = \boldsymbol{s}_{\text{mb}}(D) + \boldsymbol{v}(D) = \boldsymbol{s}'_{\text{mb}}(D) + \boldsymbol{v}'(D) \tag{2.196}$$

where $\boldsymbol{s}_{\text{mb}}(D), \boldsymbol{s}'_{\text{mb}}(D)$ are abstract states of $G_{\text{mb}}(D)$ and $\boldsymbol{v}(D)$, $\boldsymbol{v}'(D)$ are codewords. Since the sum of two abstract states is an abstract state and the sum of two codewords is a codeword, it follows from (2.196) that

$$\boldsymbol{s}''_{\text{mb}}(D) = \boldsymbol{s}_{\text{mb}}(D) + \boldsymbol{s}'_{\text{mb}}(D) = \boldsymbol{v}(D) + \boldsymbol{v}'(D) = \boldsymbol{v}''(D) \tag{2.197}$$

is both an abstract state of $G_{\text{mb}}(D)$ and a codeword. From Lemma 2.32 we deduce that

$$\boldsymbol{s}''_{\text{mb}}(D) = \mathbf{0} \tag{2.198}$$

and, hence, that

$$\boldsymbol{s}_{\text{mb}}(D) = \boldsymbol{s}'_{\text{mb}}(D) \tag{2.199}$$

and

$$\boldsymbol{v}(D) = \boldsymbol{v}'(D) \tag{2.200}$$

which completes the proof. ∎

Theorem 2.34 Let $G(D)$ be any generator matrix equivalent to a minimal-basic encoding matrix $G_{\text{mb}}(D)$. Then

$$\#\left\{\text{abstract states of } G(D)\right\} \geq \#\left\{\text{abstract states of } G_{\text{mb}}(D)\right\} \tag{2.201}$$

Proof: Consider the following map:

$$\begin{gathered}\phi : \left\{\text{abstract states of } G(D)\right\} \rightarrow \left\{\text{abstract states of } G_{\text{mb}}(D)\right\} \\ \boldsymbol{s}_G(D) \mapsto \boldsymbol{s}_{\text{mb}}(D)\end{gathered}$$

where

$$\boldsymbol{s}_G(D) = \boldsymbol{s}_{\text{mb}}(D) + \boldsymbol{v}(D) \tag{2.202}$$

in which $\boldsymbol{v}(D)$ is a codeword. From Lemma 2.33 it follows that ϕ is well-defined.

By the first statement of Lemma 2.33 we can prove that every abstract state $\boldsymbol{s}_{\text{mb}}(D)$ can be written as a sum of an abstract state of $G(D)$ and a codeword. Hence, we conclude that ϕ is surjective, which completes the proof. ∎

Remark: The map ϕ in Theorem 2.34 is linear. Moreover, if $G(D)$ is a minimal encoding matrix, then ϕ is necessarily an isomorphism of the abstract state space of $G(D)$ and that of $G_{\text{mb}}(D)$.

From Theorem 2.34 the corollary below follows immediately.

Corollary 2.35 Every minimal-basic encoding matrix is a (basic) minimal encoding matrix.

Next we will prove the following little lemma:

Lemma 2.36 Let $G(D)$ be a $b \times c$ matrix of rank b whose entries are rational functions of D. Then a necessary and sufficient condition for $G(D)$ to have a polynomial inverse is: for each $\boldsymbol{u}(D) \in \mathbb{F}_2^b(D)$ satisfying $\boldsymbol{u}(D)G(D) \in \mathbb{F}_2^c[D]$ we must have $\boldsymbol{u}(D) \in \mathbb{F}_2^b[D]$.

Proof: Since the necessity of the condition is obvious, we shall prove only the sufficiency. Let us assume that $G(D)$ does not have a polynomial inverse. Then,

from Theorem 2.8 it follows that

$$G(D) = A(D) \begin{pmatrix} \frac{\alpha_1(D)}{\beta_1(D)} & & & & & \\ & \frac{\alpha_2(D)}{\beta_2(D)} & & & & \\ & & \ddots & & & \\ & & & \frac{\alpha_b(D)}{\beta_b(D)} & 0 & \dots & 0 \end{pmatrix} B(D) \tag{2.203}$$

where $\alpha_b(D) \neq 1$. Clearly,

$$\boldsymbol{u}(D) = \left(0 \dots 0 \, \frac{\beta_b(D)}{\alpha_b(D)}\right) A^{-1}(D) \notin \mathbb{F}_2^b[D] \tag{2.204}$$

but

$$\begin{aligned} \boldsymbol{u}(D)G(D) &= \left(0 \dots 0 \, \frac{\beta_b(D)}{\alpha_b(D)}\right) A^{-1}(D)G(D) \\ &= (0 \dots 0 \, 1 \, 0 \dots 0)B(D) \in \mathbb{F}_2^c[D] \end{aligned} \tag{2.205}$$

Hence, we have proved our lemma. ∎

Define the *span* of a Laurent series $f(D)$ in D as the interval from the index of the first nonzero component of $f(D)$ to the index of the last nonzero component, if there is one, or to infinity otherwise. In other words, if $f(D)$ is a Laurent series, then

$$\operatorname{span} f(D) = \begin{cases} [\operatorname{del} f(D), \deg f(D)], & \text{if } \deg f(D) < \infty \\ [\operatorname{del} f(D), \infty), & \text{otherwise} \end{cases} \tag{2.206}$$

We are now well prepared to prove the following beautiful theorem on minimal generator matrices:

Theorem 2.37 Let $G(D)$ be a generator matrix and $G_{\text{mb}}(D)$ be an equivalent minimal-basic encoding matrix. Then the following statements are equivalent:

(i) $G(D)$ is a minimal generator matrix.

(ii) # {abstract states of $G(D)$ } = # { abstract states of $G_{\text{mb}}(D)$ } .

(iii) Only the zero abstract state of $G(D)$ can be a codeword.

(iv) $G(D)$ has a polynomial right inverse in D and a polynomial right inverse in D^{-1}.

(v) $\operatorname{span} \boldsymbol{u}(D) \subseteq \operatorname{span} \boldsymbol{u}(D)G(D)$ for all rational input sequences $\boldsymbol{u}(D)$.

Proof: It follows immediately from Theorem 2.34 that (i) and (ii) are equivalent.

Next, we prove that (ii) and (iii) are equivalent. In the proof of Theorem 2.34 we have defined a surjective (linear) map

$$\begin{aligned}\phi : \{\text{abstract states of } G(D)\} &\to \{\text{abstract states of } G_{\text{mb}}(D)\}\\ \boldsymbol{s}_G(D) &\mapsto \boldsymbol{s}_{\text{mb}}(D)\end{aligned}$$

where

$$\boldsymbol{s}_G(D) = \boldsymbol{s}_{\text{mb}}(D) + \boldsymbol{v}(D) \tag{2.207}$$

in which $\boldsymbol{v}(D)$ is a codeword. Clearly, ϕ is injective if and only if (ii) holds and if and only if (iii) holds. Hence, (ii) and (iii) are equivalent.

Then we prove that (iii) and (iv) are equivalent.

(iii $\Rightarrow$ iv). Suppose that (iii) holds. First we shall prove that $G(D)$ has a polynomial right inverse in D^{-1}. Let $\boldsymbol{u}(D) \in \mathbb{F}_2^b(D)$ and assume that $\boldsymbol{v}(D) = \boldsymbol{u}(D)G(D)$ is polynomial in D^{-1}. Then $D^{-1}\boldsymbol{v}(D)$ is polynomial in D^{-1} without a constant term, that is,

$$D^{-1}\boldsymbol{v}(D)Q = \boldsymbol{0} \tag{2.208}$$

But

$$\begin{aligned}D^{-1}\boldsymbol{v}(D)Q &= D^{-1}\boldsymbol{u}(D)(P+Q)G(D)Q\\ &= D^{-1}\boldsymbol{u}(D)PG(D)Q + D^{-1}\boldsymbol{u}(D)QG(D)Q = \boldsymbol{0}\end{aligned} \tag{2.209}$$

where

$$D^{-1}\boldsymbol{u}(D)QG(D)Q = D^{-1}\boldsymbol{u}(D)QG(D) \tag{2.210}$$

is a codeword. Hence, from (2.209) and (2.210) it follows that the abstract state $D^{-1}\boldsymbol{u}(D)PG(D)Q$ is a codeword and, then, since (iii) holds, it is the zero codeword. Thus,

$$D^{-1}\boldsymbol{u}(D)QG(D) = \boldsymbol{0} \tag{2.211}$$

and, since $G(D)$ has full rank,

$$D^{-1}\boldsymbol{u}(D)Q = \boldsymbol{0} \tag{2.212}$$

or, in other words, $\boldsymbol{u}(D)$ is polynomial in D^{-1}. Since every rational function in D can be written as a rational function in D^{-1}, $G(D)$ can be written as a matrix whose entries are rational functions in D^{-1}, we can apply Lemma 2.36 and conclude that $G(D)$ has a polynomial right inverse in D^{-1}.

Now we prove that $G(D)$ has a polynomial right pseudoinverse in D. Let $G_{-1}^{-1}(D)$ be a polynomial right inverse in D^{-1} of $G(D)$. Then there exists an integer $s \geq 0$ such that $D^s G_{-1}^{-1}(D)$ is a polynomial matrix in D and

$$G(D)D^s G_{-1}^{-1}(D) = D^s I_b \tag{2.213}$$

That is, $D^s G_{-1}^{-1}(D)$ is a polynomial right pseudoinverse in D of $G(D)$.

Next, we prove that $G(D)$ also has a polynomial right inverse in D. Let $\boldsymbol{u}(D) \in \mathbb{F}_2^b(D)$ and assume that $\boldsymbol{v}(D) = \boldsymbol{u}(D)G(D)$ is polynomial in D. Then,

$$\boldsymbol{v}(D) = \boldsymbol{u}(D)PG(D) + \boldsymbol{u}(D)QG(D) \tag{2.214}$$

where $\boldsymbol{u}(D)Q$ is a formal power series. Thus, $\boldsymbol{u}(D)QG(D)$ is also a formal power series and, since $\boldsymbol{v}(D)$ is polynomial in D, it follows that $\boldsymbol{u}(D)PG(D)$ is a formal power series. Then

$$\boldsymbol{u}(D)PG(D) = \boldsymbol{u}(D)PG(D)Q \tag{2.215}$$

is an abstract state. From (2.214) it follows that it is also a codeword, and, since (iii) holds, we conclude that

$$\boldsymbol{u}(D)PG(D) = \mathbf{0} \tag{2.216}$$

Since $G(D)$ has full rank,

$$\boldsymbol{u}(D)P = \mathbf{0} \tag{2.217}$$

or, in other words, $\boldsymbol{u}(D)$ is a formal power series. Since $\boldsymbol{v}(D)$ is polynomial and $D^sG_{-1}^{-1}(D)$ is a polynomial matrix in D, it follows that

$$\boldsymbol{v}(D)D^sG_{-1}^{-1}(D) = \boldsymbol{u}(D)G(D)D^sG_{-1}^{-1}(D) = \boldsymbol{u}(D)D^s \tag{2.218}$$

is polynomial; that is, $\boldsymbol{u}(D)$ has finitely many terms. But $\boldsymbol{u}(D)$ is a formal power series. Hence, we conclude that it is polynomial in D. By Lemma 2.36, $G(D)$ has a polynomial right inverse in D.

(iv $\Rightarrow$ iii). Assume that the abstract state $\boldsymbol{s}_G(D)$ of $G(D)$ is a codeword; that is,

$$\begin{aligned}\boldsymbol{s}_G(D) &= \boldsymbol{u}(D)G(D)Q \\ &= \boldsymbol{u}'(D)G(D)\end{aligned} \tag{2.219}$$

where $\boldsymbol{u}(D)$ is polynomial in D^{-1} but without a constant term and $\boldsymbol{u}'(D) \in \mathbb{F}_2^b((D))$. Since $\boldsymbol{s}_G(D)$ is a formal power series and $G(D)$ has a polynomial right inverse, it follows that $\boldsymbol{u}'(D)$ is also a formal power series. Let us use the fact that

$$Q = 1 + P \tag{2.220}$$

and rewrite (2.219) as

$$\boldsymbol{s}_G(D) = \boldsymbol{u}(D)G(D) + \boldsymbol{u}(D)G(D)P = \boldsymbol{u}'(D)G(D) \tag{2.221}$$

Let $G_{-1}^{-1}(D)$ be a right inverse of $G(D)$ whose entries are polynomials in D^{-1}. Then

$$\boldsymbol{u}(D)G(D)G_{-1}^{-1}(D) + \boldsymbol{u}(D)G(D)PG_{-1}^{-1}(D) = \boldsymbol{u}'(D)G(D)G_{-1}^{-1}(D) \tag{2.222}$$

which can be simplified to

$$\boldsymbol{u}(D) + \boldsymbol{u}(D)G(D)PG_{-1}^{-1}(D) = \boldsymbol{u}'(D) \tag{2.223}$$

Since $\boldsymbol{u}(D)G(D)P$ is polynomial in D^{-1} without a constant term, it follows that $\boldsymbol{u}(D)G(D)PG_{-1}^{-1}(D)$ is polynomial in D^{-1} without a constant term. Furthermore, $\boldsymbol{u}(D)$ is polynomial in D^{-1} without a constant term and $\boldsymbol{u}'(D)$ is a formal power series. Thus, we conclude that $\boldsymbol{u}'(D) = \mathbf{0}$ and, hence, that $\boldsymbol{s}_G(D) = \mathbf{0}$.

It remains to prove that (iv) and (v) are equivalent.

(iv $\Rightarrow$ v). Assume that $G(D)$ has a polynomial right inverse in D, $G^{-1}(D)$, and let $\boldsymbol{v}(D) = \boldsymbol{u}(D)G(D)$ be any sequence in $\mathcal{C}$. Then $\boldsymbol{u}(D) = \boldsymbol{v}(D)G^{-1}(D)$, so

(i) $\deg \boldsymbol{u}(D) < \infty$ if $\deg \boldsymbol{v}(D) < \infty$

(ii) $\operatorname{del} \boldsymbol{u}(D) \geq \operatorname{del} \boldsymbol{v}(D)$

Similarly, if $G(D)$ has a polynomial right inverse in D^{-1}, then

(iii) $\deg \boldsymbol{u}(D) \leq \deg \boldsymbol{v}(D)$

Conditions (i)–(iii) imply that $\operatorname{span} \boldsymbol{u}(D) \subseteq \operatorname{span} \boldsymbol{v}(D)$.

(v $\Rightarrow$ iv). It follows from (v) that for each $\boldsymbol{u}(D) \in \mathbb{F}_2^b(D)$ satisfying $\boldsymbol{u}(D)G(D) \in \mathbb{F}_2^c[D]$ we must have $\boldsymbol{u}(D) \in \mathbb{F}_2^b[D]$. By Lemma 2.36, $G(D)$ has a polynomial right inverse in D. Similarly, (v) implies that for each $\boldsymbol{u}(D) \in \mathbb{F}_2^b(D)$ satisfying $\boldsymbol{u}(D)G(D) \in \mathbb{F}_2^c[D^{-1}]$ we must have $\boldsymbol{u}(D) \in \mathbb{F}_2^b[D^{-1}]$. Again, by Lemma 2.36, $G(D)$ has a polynomial right inverse in D^{-1}. ∎

Corollary 2.38 A minimal generator matrix is a minimal encoding matrix.

Proof: Follows immediately from Theorems 2.5 and 2.37 (iv). ∎

Corollary 2.39 A minimal encoding matrix is noncatastrophic.

Proof: Follows immediately from Corollary 2.13 and Theorem 2.37 (iv). ∎

Theorem 2.37 (v) is closely related to the following (non)minimality criteria [LFM94]:

Theorem 2.40 A generator matrix can be nonminimal in only three ways:

(i) If there is an infinite nontrivial sequence of (encoder) states (not the zero state sequence) that produces an allzero output sequence.

(ii) If there is a nontrivial transition from the zero state (not to the zero state) that produces a zero output.

(iii) If there is a nontrivial transition to the zero state (not from the zero state) that produces a zero output.

Proof: Condition (i) corresponds to a case in which there is an infinite input $\boldsymbol{u}(D)$ that produces a finite output $\boldsymbol{u}(D)G(D)$ (the "catastrophic" case); condition (ii) corresponds to a case in which there is an input $\boldsymbol{u}(D)$ that produces an output $\boldsymbol{u}(D)G(D)$ with $\operatorname{del} \boldsymbol{u}(D)G(D) > \operatorname{del} \boldsymbol{u}(D)$; and condition (iii) corresponds to a case in which there is a finite input $\boldsymbol{u}(D)$ that produces a finite output $\boldsymbol{u}(D)G(D)$ with $\deg \boldsymbol{u}(D)G(D) < \deg \boldsymbol{u}(D)$. It is easy to see that $\operatorname{span} \boldsymbol{u}(D)G(D)$ does not cover $\operatorname{span} \boldsymbol{u}(D)$ if and only if one of these three conditions is satisfied. ∎

The following simple example shows that not all basic encoding matrices are minimal.

■ **EXAMPLE 2.22**

Consider the basic encoding matrix

$$G(D) = \begin{pmatrix} 1+D & D \\ D & 1+D \end{pmatrix} \tag{2.224}$$

which has $\mu = 0$ but $\nu = 2$. Clearly, it is not minimal-basic.

The *equivalent* minimal-basic encoding matrix,

$$G_{\text{mb}}(D) = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix} \tag{2.225}$$

has only one abstract state, viz., $\boldsymbol{s}_{\text{mb}} = (0, 0)$, and can be realized without any memory element.

Since $G(D)$ has two abstract states, viz., $\boldsymbol{s}_0 = (0, 0)$ and $\boldsymbol{s}_1 = (1, 1)$, it is not minimal!

Before we state a theorem on when a basic encoding matrix is minimal, we shall prove two lemmas:

Lemma 2.41 Let $f_1(D), f_2(D), \ldots, f_\ell(D) \in \mathbb{F}_2[D]$ with

$$\gcd(f_1(D), f_2(D), \ldots, f_\ell(D)) = 1 \tag{2.226}$$

and let

$$n = \max\{\deg f_1(D), \deg f_2(D), \ldots, \deg f_\ell(D)\} \tag{2.227}$$

Then, for $m \geq n$, $D^{-m}f_1(D), D^{-m}f_2(D), \ldots, D^{-m}f_\ell(D) \in \mathbb{F}_2[D^{-1}]$ and

$$\gcd(D^{-m}f_1(D), D^{-m}f_2(D), \ldots, D^{-m}f_\ell(D)) = D^{-(m-n)} \tag{2.228}$$

Proof: Let

$$f_i(D) = D^{s_i}g_i(D), \quad i = 1, 2, \ldots, \ell \tag{2.229}$$

where s_i is the start of $f_i(D)$ and $g_i(D) \in \mathbb{F}_2[D]$ is delayfree. From (2.226) follows

$$\min\{s_1, s_2, \ldots, s_\ell\} = 0 \tag{2.230}$$

and

$$\gcd(g_1(D), g_2(D), \ldots, g_\ell(D)) = 1 \tag{2.231}$$

For $m \geq n$

$$\begin{aligned} D^{-m}f_i(D) &= D^{-m}D^{s_i}g_i(D) \\ &= D^{-(m-s_i-\deg g_i(D))}\left(D^{-\deg g_i(D)}g_i(D)\right) \\ &= D^{-(m-\deg f_i(D))}\left(D^{-\deg g_i(D)}g_i(D)\right), \ i = 1, 2, \ldots, \ell \end{aligned} \tag{2.232}$$

where the last equality follows from the fact that

$$\deg f_i(D) = s_i + \deg g_i(D), \quad i = 1, 2, \ldots, \ell \tag{2.233}$$

Since $D^{-\deg g_i(D)} g_i(D)$, $i = 1, 2, \ldots, \ell$, are delayfree polynomials in $\mathbb{F}_2[D^{-1}]$, it follows from (2.232) that

$$\gcd\left(D^{-m} f_1(D), D^{-m} f_2(D), \ldots, D^{-m} f_\ell(D)\right) = D^{-(m-n)} \times \gcd\left(D^{-\deg g_1(D)} g_1(D), D^{-\deg g_2(D)} g_2(D), \ldots, D^{-\deg g_\ell(D)} g_\ell(D)\right) \tag{2.234}$$

Clearly,

$$\gcd\left(D^{-\deg g_1(D)} g_1(D), D^{-\deg g_2(D)} g_2(D), \ldots, D^{-\deg g_\ell(D)} g_\ell(D)\right) = 1 \tag{2.235}$$

and the proof is complete. ∎

Lemma 2.42 Let $G(D)$ be a basic encoding matrix, and let r and s be the maximum degree of its $b \times b$ minors and $(b-1) \times (b-1)$ minors, respectively. Then the bth invariant factor of $G(D)$ regarded as a matrix over $\mathbb{F}_2(D^{-1})$ is $1/D^{-(r-s)}$.

Proof: Let $G(D) = (\boldsymbol{g}_{ij}(D))$, $1 \le i \le b$, $1 \le j \le c$, and let

$$n = \max_{i,j} \{\deg g_{ij}(D)\}$$

Write $G(D)$ as a matrix over $\mathbb{F}_2(D^{-1})$ as

$$G(D) = \left(\frac{D^{-n} g_{ij}(D)}{D^{-n}}\right)_{i,j} = \frac{1}{D^{-n}} G_{-1}(D) \tag{2.236}$$

where

$$G_{-1}(D) = \left(D^{-n} g_{ij}(D)\right)_{i,j} \tag{2.237}$$

is a matrix of polynomials in D^{-1}.

Since $G(D)$ is basic, it follows, by definition, that it has a polynomial right inverse. Hence, it follows from Theorem 2.8 and (2.64) that

$$\alpha_1(D) = \alpha_2(D) = \cdots = \alpha_b(D) = 1 \tag{2.238}$$

Let $\Delta_i(G(D))$ be the greatest common divisor of the $i \times i$ minors of $G(D)$. Since from (2.40)

$$\Delta_i(G(D)) = \alpha_1(D)\alpha_2(D) \ldots \alpha_i(D) \tag{2.239}$$

we have in particular

$$\Delta_b(G(D)) = \Delta_{b-1}(G(D)) = 1 \tag{2.240}$$

An $i \times i$ minor of $G_{-1}(D)$ is equal to the corresponding minor of $G(D)$ multiplied by D^{-ni}. Hence, by Lemma 2.41, we have

$$\Delta_b(G_{-1}(D)) = D^{-(nb-r)} \tag{2.241}$$

and

$$\Delta_{b-1}(G_{-1}(D)) = D^{-(n(b-1)-s)} \tag{2.242}$$

Thus, the bth invariant factor of $G_{-1}(D)$ is (2.41),

$$\gamma_b(G_{-1}(D)) = \frac{\Delta_b(G_{-1}(D))}{\Delta_{b-1}(G_{-1}(D))} = \frac{D^{-n}}{D^{-(r-s)}} \tag{2.243}$$

From (2.236) and (2.243) it follows that the bth invariant factor of $G(D)$, regarded as a matrix over $\mathbb{F}_2[D^{-1}]$, is

$$\frac{1}{D^{-n}} \cdot \frac{D^{-n}}{D^{-(r-s)}} = \frac{1}{D^{-(r-s)}} \tag{2.244}$$

■

We are now ready to prove the following theorem:

Theorem 2.43 A basic encoding matrix $G(D)$ is minimal if and only if the maximum degree of its $b \times b$ minors is not less than the maximum degree of its $(b-1) \times (b-1)$ minors.

Proof: From Theorem 2.37 it follows that a basic encoding matrix $G(D)$ is minimal if and only if it has a polynomial right inverse in D^{-1}. By the invariant factor decomposition, $G(D)$, regarded as a matrix over $\mathbb{F}_2[D^{-1}]$, has a polynomial right inverse in D^{-1} if and only if the inverse of its bth invariant factor is a polynomial in D^{-1}. By applying Lemma 2.42 the theorem follows. ■

We will now briefly describe a "greedy" construction of a minimal-basic encoding matrix for a convolutional code $\mathcal{C}$ [Roo79]. The construction goes as follows.

Choose the generator $\boldsymbol{g}_1(D)$ as a nonzero polynomial code sequence of least degree. Choose $\boldsymbol{g}_2(D)$ as a nonzero polynomial code sequence of least degree not in the rate $R = 1/c$ code $\mathcal{C}_1$ encoded by $\boldsymbol{g}_1(D)$; choose $\boldsymbol{g}_3(D)$ as a nonzero polynomial code sequence of least degree not in the rate $R = 2/c$ code $\mathcal{C}_2$ encoded by $\boldsymbol{g}_1(D)$ and $\boldsymbol{g}_2(D)$, and so forth, until a set $G(D) = \{\boldsymbol{g}_i(D), \quad 1 \leq i \leq b\}$ of b generators that encodes $\mathcal{C}$ has been chosen.

It is easy to see that the degrees $\deg \boldsymbol{g}_i(D)$ are uniquely defined by $\mathcal{C}$; in fact, they are the constraint lengths ν_i. The sum $\sum_i \nu_i$ is minimal over all equivalent polynomial generator matrices; it follows from Corollary 2.35 that it is minimal and, thus, from Theorem 2.37 (iv) that it is also basic. Hence, $G(D)$ is minimal-basic.

■ **EXAMPLE 2.23**

Let $\mathcal{C}$ be the rate $R = 2/3$ convolutional code encoded by the encoding matrix in Example 2.16,

$$\begin{aligned} G'(D) &= \begin{pmatrix} \boldsymbol{g}'_1(D) \\ \boldsymbol{g}'_2(D) \end{pmatrix} \\ &= \begin{pmatrix} 1+D & D & 1 \\ 1+D^2+D^3 & 1+D+D^2+D^3 & 0 \end{pmatrix} \end{aligned} \tag{2.245}$$

The shortest nonzero polynomial code sequence is $\boldsymbol{g}'_1(D)$, so $\nu_1 = 1$. The next shortest code sequence not dependent on $\boldsymbol{g}'_1(D)$ has degree 2; for example,

$$\begin{aligned}\boldsymbol{g}_2(D) &= D^2\boldsymbol{g}'_1(D) + \boldsymbol{g}'_2(D) \\ &= \begin{pmatrix} 1 & 1+D+D^2 & D^2 \end{pmatrix}\end{aligned} \tag{2.246}$$

so $\nu_2 = 2$. A minimal-basic encoding matrix for $\mathcal{C}$ is

$$G(D) = \begin{pmatrix} 1+D & D & 1 \\ 1 & 1+D+D^2 & D^2 \end{pmatrix} \tag{2.247}$$

We return to our favorite basic encoding matrix given in Example 2.16, viz.,

$$G'(D) = \begin{pmatrix} 1+D & D & 1 \\ 1+D^2+D^3 & 1+D+D^2+D^3 & 0 \end{pmatrix} \tag{2.248}$$

In Example 2.18 we showed that $G'(D)$ is not minimal-basic, that is, $\mu < \nu$. Its controller canonical form (Fig. 2.14) requires four memory elements but the controller canonical form of an equivalent encoding matrix (Fig. 2.9) requires only three memory elements. However, $G'(D)$ is a basic encoding matrix, and, hence, it has a polynomial right inverse. Furthermore, it has a polynomial right inverse in D^{-1}, viz.,

$$G_{-1}^{-1}(D) = \begin{pmatrix} 1+D^{-1}+D^{-2}+D^{-3} & D^{-1} \\ 1+D^{-1}+D^{-3} & D^{-1} \\ D^{-2}+D^{-3} & D^{-1} \end{pmatrix} \tag{2.249}$$

Thus, from Theorem 2.37 we conclude that $G'(D)$ is indeed a minimal encoding matrix. Its eight abstract states are given in Table 2.1.

We will conclude this section by considering realizations of minimal encoding matrices.

Definition A *minimal encoder* is a realization of a minimal encoding matrix $G(D)$ with a minimal number of memory elements over all realizations of $G(D)$.

Theorem 2.44 The controller canonical form of a minimal-basic encoding matrix is a minimal encoder.

Proof: Follows immediately from Corollaries 2.23 and 2.35. ∎

■ **EXAMPLE 2.24**

The realization shown in Fig. 2.16 of the minimal encoding matrix $G'(D)$ given in (2.248) is a minimal encoder. (The realization is obtained by minimizing $G'(D)$ using a standard sequential circuit minimization method, see, for example, [Lee78]. See also Appendix A.)

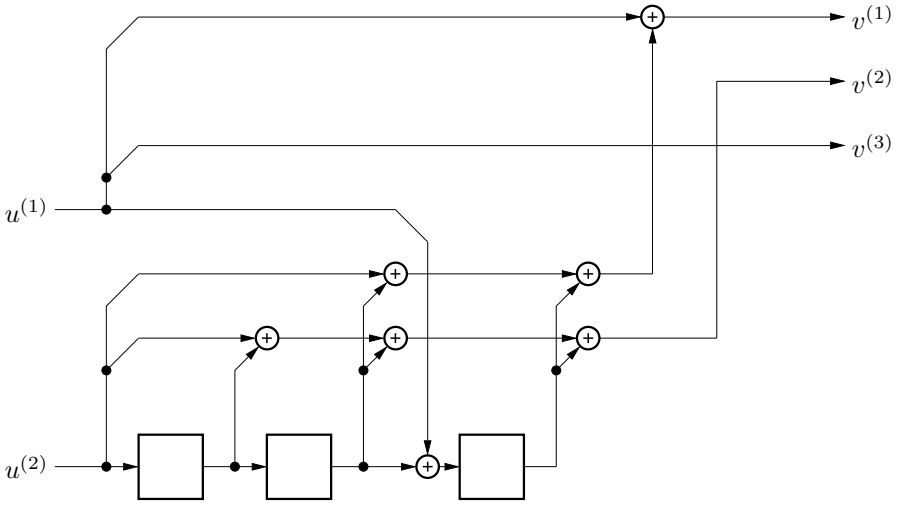


Figure 2.16 A minimal encoder for the basic encoding matrix $G'(D)$ given in (2.248).

Notice that the minimal realization shown in Fig. 2.16 is neither in controller canonical form nor in observer canonical form. This particular minimal encoding matrix does not have a *minimal* controller canonical form, but it has an *equivalent minimal-basic* encoding matrix whose controller canonical form (Fig. 2.9) is a minimal encoder for the same convolutional code.

Next we consider a rate $R = b/c$ convolutional code $\mathcal{C}$ with generator matrix $G(D)$. Let $G_k(D)$ be the rate $R = k/c$ generator matrix that consists of the k last rows of $G(D)$. The generator matrices $G_1(D), G_2(D), \ldots, G_b(D) = G(D)$ are called *nested* generator matrices and the corresponding convolutional codes $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_b = \mathcal{C}$ are called nested convolutional codes. The following theorem and its two corollaries deal with the minimality of nested convolutional generator matrices [JJB04].

Theorem 2.45 Given a rate $R = b/c$ minimal-basic encoding matrix $G_{\text{mb}}(D)$ and a $b \times b$ rational matrix $T(D)$ that has a polynomial right inverse $T^{-1}(D)$, then

$$G(D) = T(D)G_{\text{mb}}(D) \tag{2.250}$$

is minimal if and only if

$$\nu_i^{(T^{-1})} \leq \nu_i^{(G_{\text{mb}})}, \quad i = 1, 2, \ldots, b \tag{2.251}$$

where $\nu_i^{(T^{-1})}$ and $\nu_i^{(G_{\text{mb}})}$ denote the ith constraint lengths of $T^{-1}(D)$ and $G_{\text{mb}}(D)$, respectively.

Proof: Assume that $\nu_i^{(T^{-1})} \leq \nu_i^{(G_{mb})}$ is satisfied for all i. Since $G^{-1}(D) = G_{mb}^{-1}(D)T^{-1}(D)$ is polynomial, it follows that

$$\deg \boldsymbol{u}(D) = \deg \boldsymbol{v}(D)G^{-1}(D) < \infty \tag{2.252}$$

if $\deg \boldsymbol{v}(D) < \infty$. Hence, we need to consider only polynomial information sequences. Since $G_{\text{mb}}(D)$ is minimal, it follows from Theorem 2.37(v) that the span property,

$$\operatorname{span} \boldsymbol{u}(D) \subseteq \operatorname{span} \boldsymbol{u}(D)G_{\text{mb}}(D) \tag{2.253}$$

holds for all rational input sequences $\boldsymbol{u}(D)$. Hence, we conclude that

$$\operatorname{del} \boldsymbol{x}(D) = \operatorname{del} \boldsymbol{x}(D)G_{\text{mb}}(D) = \operatorname{del} \boldsymbol{u}(D)G_{\text{mb}}(D) \tag{2.254}$$

where $\boldsymbol{x}(D) = \boldsymbol{u}(D)T(D)$. Since $T^{-1}(0)$ has full rank, we have

$$\operatorname{del} \boldsymbol{x}(D) = \operatorname{del} \boldsymbol{x}(D)\, T^{-1}(D) = \operatorname{del} \boldsymbol{u}(D) \tag{2.255}$$

and, by combining (2.254) and (2.255), we conclude that

$$\operatorname{del} \boldsymbol{u}(D) = \operatorname{del} \boldsymbol{u}(D)G(D) \tag{2.256}$$

Since $G_{\text{mb}}(D)$ is minimal-basic, it follows that the predictable degree property holds, that is, for any polynomial input $\boldsymbol{x}(D)$ we have

$$\begin{aligned} \deg \boldsymbol{u}(D)G(D) &= \deg \boldsymbol{x}(D)G_{\text{mb}}(D) \\ &= \max_i \left\{ \deg x_i(D) + \nu_i^{(G_{\text{mb}})} \right\} \end{aligned} \tag{2.257}$$

Moreover, since $T^{-1}(D)$ is polynomial, we have

$$\begin{aligned} \deg \boldsymbol{u}(D) &= \deg \boldsymbol{x}(D)T^{-1}(D) \\ &\leq \max_i \left\{ \deg \boldsymbol{x}_i(D) + \nu_i^{(G_{\text{mb}})} \right\} \end{aligned} \tag{2.258}$$

and, by combining (2.257) and (2.258) with the assumption that $\nu_i^{(T^{-1})} \leq \nu_i^{(G_{\text{mb}})}$, it follows that

$$\deg \boldsymbol{u}(D) \leq \deg \boldsymbol{u}(D)G(D) \tag{2.259}$$

Combining (2.256) and (2.259) shows that, if the assumption $\nu_i^{(T^{-1})} \leq \nu_i^{(G_{\text{mb}})}$ holds, the span property also holds and, hence, from Theorem 2.37(i) it follows that $G(D)$ is minimal, which completes the first part of the proof.

Now we assume that $\nu_i^{(T^{-1})} > \nu_i^{(G_{\text{mb}})}$ for some i. Let $\boldsymbol{u'}(D) = \boldsymbol{x'}(D)T^{-1}(D)$, where $\boldsymbol{x'}(D) = (00\ldots010\ldots0)$; then

$$\deg \boldsymbol{u'}(D) = \nu_i^{(T^{-1})} \tag{2.260}$$

Since $\deg x_i(D) = 0$, we conclude from (2.258) that

$$\deg \boldsymbol{u'}(D)G(D) = \nu_i^{(G_{\text{mb}})} \tag{2.261}$$

Combining (2.260) and (2.261) with the assumption that $\nu_i^{(T^{-1})} > \nu_i^{(G_{\text{mb}})}$ for some i yields that the span property does not hold and, hence, by Theorem 2.37 we conclude that $G(D)$ is not minimal. This completes the proof. ∎

We show two important corollaries of this theorem.

Corollary 2.46 Given a rate $R = b/c$ convolutional code $\mathcal{C}$ with a minimal-basic encoding matrix $G_{\text{mb}}(D)$. Then any minimal (rational) encoding matrix $G_{\min}(D)$ of $\mathcal{C}$ can be written $G_{\min}(D) = T(D)G_{\text{mb}}(D)$, where $T(D) \in \mathcal{T}_{G_{\text{mb}}}$ and

$$\mathcal{T}_{G_{\text{mb}}} = \big\{T(D) | T^{-1}(D) \text{ is polynomial and } \nu_i^{(T^{-1})} \leq \nu_i^{(G_{\text{mb}})},\ i = 1, 2, \ldots, b\big\} \tag{2.262}$$

Moreover, there exists a one-to-one mapping between $\mathcal{T}_{G_{\text{mb}}}$ and the set of minimal encoding matrices in $\mathcal{C}$.

Proof: From Theorem 2.14 and Corollary 2.24 it follows that any minimal encoding matrix of $\mathcal{C}$ is equivalent to a minimal-basic encoding matrix $G_{\text{mb}}(D)$, that is, $G_{\min}(D) = T(D)G_{\text{mb}}(D)$, where $T(D)$ is nonsingular over $\mathbb{F}_2(D)$. By combining Theorems 2.14 and 2.45 it follows that $G_{\min}(D) = T(D)G_{\text{mb}}(D)$, if and only if $T(D)$ has a polynomial right inverse $T^{-1}(D)$ with constraint lengths $\nu_i^{(T^{-1})} \leq \nu_i^{(G_{\text{mb}})}$ for all i. Since $\mathcal{T}_{G_{\text{mb}}}$ is the set of all such matrices $T(D)$, we conclude that there exists a matrix $T(D) \in \mathcal{T}_{G_{\text{mb}}}$ such that $G_{\min}(D) = T(D)G_{\text{mb}}(D)$.

Given two matrices $T_1(D), T_2(D) \in \mathcal{T}_{G_{\text{mb}}}$, such that

$$G_{\min}(D) = T_1(D)G_{\text{mb}}(D) = T_2(D)G_{\text{mb}}(D) \tag{2.263}$$

it follows that $T_1(D) = T_2(D)$. Hence, the mapping between $\mathcal{T}_{G_{\text{mb}}}$ and the set of all minimal encoding matrices in $\mathcal{C}$ is one-to-one. ∎

Corollary 2.47 Given any minimal encoding matrix $G_{\min}(D)$ of a rate $R = b/c$ convolutional code $\mathcal{C}$. Let $G_{\text{mb}}(D)$ be an equivalent minimal-basic encoding matrix and let $T(D) \in \mathcal{T}_{G_{\text{mb}}}$ be given by $G_{\min}(D) = T(D)G_{\text{mb}}(D)$. Consider all nested generator matrices $G_k(D)$, $k = 1, 2, \ldots, b$, where

$$G_k(D) = T_k(D)G_{\text{mb}}(D),\ \ k = 1, 2, \ldots, b \tag{2.264}$$

and where $T_k(D)$ is given by the last k rows of the $b \times b$ matrix $T(D) = T_b(D)$. Then all the nested generator matrices $G_k(D)$, $k = 1, 2, \ldots, b$, are minimal.

Proof: Consider the nested generator matrices $G_k(D) = T_k(D)G_{\text{mb}}(D)$. The right inverse $T_k^{-1}(D)$ is given the last k columns of $T^{-1}(D) = T_b^{-1}(D)$. Hence,

$$\nu_i^{(T_k^{-1})} \leq \nu_i^{(T^{-1})},\ \ i = 1, 2, \ldots, b \tag{2.265}$$

for all $k = 1, 2, \ldots, b$. From Theorem 2.45 it follows that the generator matrix $G_k(D)$ is minimal if $\nu_i^{(T_k^{-1})} \leq \nu_i^{(G_{\text{mb}})}$ for all i. Hence, we conclude from (2.265) that all $G_k(D)$, $k = 1, 2, \ldots, b$, are minimal. ∎

■ **EXAMPLE 2.25**

Consider the rate $R = 2/3$ minimal-basic encoding matrix [Jor02]

$$G_{\text{mb}}(D) = \begin{pmatrix} 1+D+D^3 & 1+D+D^3 & D+D^2 \\ D^2+D^3 & D+D^2 & 1+D+D^3 \end{pmatrix} \tag{2.266}$$

with overall constraint length $\nu^{(G_{\text{mb}})} = \mu = 6$. Let $T(D)$ be given by its polynomial inverse

$$T^{-1}(D) = \begin{pmatrix} 1+D+D^3 & 1 \\ D+D^3 & 1 \end{pmatrix} \tag{2.267}$$

that is,

$$T(D) = \begin{pmatrix} 1 & 1 \\ D+D^3 & 1+D+D^3 \end{pmatrix} \tag{2.268}$$

The constraint lengths of $T^{-1}(D)$ and $G_{\text{mb}}(D)$ satisfy the inequalities (2.251) from Theorem 2.45. Hence, we conclude that

$$\begin{aligned} &G(D) = T(D)G_{\text{mb}}(D) \\ &= \begin{pmatrix} 1+D+D^2 & 1+D^2+D^3 & 1+D^2+D^3 \\ D+D^3+D^4+D^5 & D^2+D^4+D^5+D^6 & 1+D^3+D^4+D^5+D^6 \end{pmatrix} \end{aligned}$$

is minimal. It can be realized by $\nu = 9$ memory elements in controller cannonical form. It is basic, but since

$$[G(D)]_h = \begin{pmatrix} 0 & 1 & 1 \\ 0 & 1 & 1 \end{pmatrix} \tag{2.269}$$

does not have full rank, it is not minimal-basic. However, it can be realized by $\mu = 6$ memory elements but neither in controller canonical form nor in observer canonical form.

According to Corollary 2.47 all nested generator matrices are minimal and, hence, we have

$$\begin{aligned} &G_1(D) \\ &= (D+D^2+D^4+D^5 \quad D^2+D^4+D^5+D^6 \quad 1+D^3+D^4+D^5+D^6) \end{aligned}$$

Since $G_1(D)$ is minimal-basic, it can be realized in controller canonical form by $\nu_1 = \mu_1 = 6$ memory elements.

2.8 CANONICAL ENCODING MATRICES*

In this section we will revisit rational generator matrices and study them in depth.

For simplicity we consider first a rate $R = 1/c$ rational generator matrix $\boldsymbol{g}(D) = (g_1(D)\, g_2(D) \ldots g_c(D))$, where $g_1(D), g_2(D), \ldots, g_c(D)$ are rational realizable functions. We may write

$$g_i(D) = f_i(D)/q(D), \quad i = 1, 2, \ldots, c \tag{2.270}$$

where $f_1(D), f_2(D), \ldots, f_c(D), q(D) \in \mathbb{F}_2[D]$ and

$$\gcd(f_1(D), f_2(D), \ldots, f_c(D), q(D)) = 1 \tag{2.271}$$

We define the *constraint length* of the rational $1 \times c$ matrix

$$\boldsymbol{g}(D) = (g_1(D)\, g_2(D) \ldots g_c(D)) \tag{2.272}$$

as

$$\nu \stackrel{\text{def}}{=} \max\{\deg f_1(D), \deg f_2(D), \ldots, \deg f_c(D), \deg q(D)\} \tag{2.273}$$

Clearly, $\boldsymbol{g}(D)$ can be realized with ν delay elements in controller canonical form.

Next, we consider a rate $R = b/c$ rational generator matrix $G(D)$ and define its *ith constraint length* ν_i as the constraint length of the ith row of $G(D)$, its *memory* m as

$$m \stackrel{\text{def}}{=} \max_{1 \le i \le b} \{\nu_i\} \tag{2.274}$$

and its *overall constraint length* ν as

$$\nu \stackrel{\text{def}}{=} \sum_{i=1}^{b} \nu_i \tag{2.275}$$

For a polynomial generator matrix, these definitions coincide with original definitions of the ith constraint length, memory, and overall constraint length given by (2.133), (2.134), and (2.135), respectively.

A rational generator matrix with overall constraint length ν can be realized with ν memory elements in controller canonical form. This leads to the next definition.

Definition A *canonical* generator matrix is a rational generator matrix whose overall constraint length ν is minimal over all equivalent rational generator matrices.

We have immediately the following two theorems.

Theorem 2.48 A minimal-basic encoding matrix is canonical.

*Note: Sections marked with an asterisk (*) can be skipped at the first reading without loss of continuity.

Proof: Let $G_{\text{mb}}(D)$ be a minimal-basic encoding matrix with overall constraint length ν_{mb} and let $G_c(D)$ be an equivalent canonical generator matrix with overall constraint length ν_c. Then $\nu_c \leq \nu_{\text{mb}}$. Since $G_{\text{mb}}(D)$ is minimal, its number of abstract states, $2^{\nu_{\text{mb}}}$, is minimal over all equivalent encoding matrices. Thus, $2^{\nu_{\text{mb}}} \leq \#\{\text{abstract states of } G_c(D)\} \leq 2^{\nu_c}$. Therefore, $\nu_{\text{mb}} \leq \nu_c$. Hence, $\nu_{\text{mb}} = \nu_c$ and $G_{\text{mb}}(D)$ is canonical. ∎

Theorem 2.49 A canonical generator matrix is minimal.

Proof: From the proof of Theorem 2.48 it follows that the number of abstract states of $G_c(D)$, $2^{\nu_c} = 2^{\nu_{\text{mb}}}$, is minimal over all equivalent generator matrices. Hence, $G_c(D)$ is minimal. ∎

Since a canonical generator matrix $G(D)$ is minimal, the following theorem follows immediately from Corollary 2.38.

Theorem 2.50 A canonical generator matrix is a canonical encoding matrix.

In Theorem 2.26 we proved the invariance of the constraint lengths of minimal-basic encoding matrices. By a straightforward generalization of that proof we show the following:

Theorem 2.51 The constraint lengths of two equivalent canonical encoding matrices are equal one by one up to a rearrangement.

Proof: Let $\mathcal{C}$ be the code encoded by two equivalent canonical encoding matrices $G(D)$ and $G'(D)$ with constraint lengths $\nu_1, \nu_2, \ldots, \nu_b$ and $\nu'_1, \nu'_2, \ldots, \nu'_b$, respectively. Without loss of generality we assume that $\nu_1 \leq \nu_2 \leq \cdots \leq \nu_b$ and $\nu'_1 \leq \nu'_2 \leq \cdots \leq \nu'_b$.

Now suppose that ν_i and ν'_i are not equal for all i, $1 \leq i \leq b$. Let j be the smallest index such that $\nu_j \neq \nu'_j$. Then without loss of generality we can assume that $\nu_j < \nu'_j$. From the sequence $\boldsymbol{g}_1(D), \boldsymbol{g}_2(D), \ldots, \boldsymbol{g}_j(D), \boldsymbol{g}'_1(D), \boldsymbol{g}'_2(D) \ldots, \boldsymbol{g}'_b(D)$ we can obtain by Lemma 2.25 a basis $\boldsymbol{g}_1(D), \boldsymbol{g}_2(D), \ldots, \boldsymbol{g}_j(D), \boldsymbol{g}'_{i_{j+1}}(D), \boldsymbol{g}'_{i_{j+2}}(D) \ldots, \boldsymbol{g}'_{i_b}(D)$ of $\mathcal{C}$. These b row vectors form an encoding matrix $G''(D)$ which is equivalent to $G'(D)$. Let

$$\left\{\boldsymbol{g}'_1(D), \boldsymbol{g}'_2(D), \ldots, \boldsymbol{g}'_b(D)\right\} \setminus \left\{\boldsymbol{g}'_{i_{j+1}}(D), \boldsymbol{g}'_{i_{j+2}}(D), \ldots, \boldsymbol{g}'_{i_b}(D)\right\} = \left\{\boldsymbol{g}'_{i_1}(D), \boldsymbol{g}'_{i_2}(D), \ldots, \boldsymbol{g}'_{i_j}(D)\right\} \tag{2.276}$$

From our assumptions it follows that

$$\nu'' = \sum_{\ell=1}^{j} \nu_i + \sum_{\ell=j+1}^{b} \nu'_{i_\ell} < \sum_{\ell=1}^{j} \nu'_i + \sum_{\ell=j+1}^{b} \nu'_{i_\ell} \leq \sum_{\ell=1}^{j} \nu'_{i_\ell} + \sum_{\ell=j+1}^{b} \nu'_{i_\ell} = \nu' \tag{2.277}$$

where ν' and ν'' are the overall constraint lengths of the encoding matrices $G'(D)$ and $G''(D)$, respectively. The inequality (2.277) contradicts the assumption that $G'(D)$ is canonical. This completes the proof. ∎

By virtue of Theorem 2.51, we may define the constraint lengths of a convolutional code to be the constraint lengths of any canonical encoding matrix that encodes the code. By Theorem 2.28, a minimal-basic encoding matrix is canonical. Thus, we have the next theorem.

Theorem 2.52 The constraint lengths of a convolutional code are equal to the constraint lengths of any minimal-basic encoding matrix that encodes the code one by one up to a rearrangement.

The following lemma will be useful in the sequel when we prove our main results for canonical encoding matrices.

Lemma 2.53 Let $\boldsymbol{g}(D) = (g_1(D)\, g_2(D) \ldots g_c(D))$ be a $1 \times c$ rational generator matrix, where $g_1(D), g_2(D), \ldots, g_c(D) \in \mathbb{F}_2(D)$. Write

$$g_i(D) = f_i(D)/q(D), \quad i = 1, 2, \ldots, c \tag{2.278}$$

and assume that

$$\gcd(f_1(D), f_2(D), \ldots, f_c(D), q(D)) = 1 \tag{2.279}$$

Then $\boldsymbol{g}(D)$ is canonical if and only if both (i) and (ii) hold:

(i) $\deg q(D) \leq \max\{\deg f_1(D), \deg f_2(D), \ldots, \deg f_c(D)\}$.

(ii) $\gcd(f_1(D), f_2(D), \ldots, f_c(D)) = 1$.

Proof: Let $\boldsymbol{f}(D) = (f_1(D)\, f_2(D) \ldots f_c(D)) = \gcd(f_1(D), f_2(D), \ldots, f_c(D))\boldsymbol{\ell}(D)$. It is clear that $\boldsymbol{g}(D)$, $\boldsymbol{f}(D)$, and $\boldsymbol{\ell}(D)$ are equivalent generator matrices.

Suppose that $\boldsymbol{g}(D)$ is canonical. Then by the definitions of ν_g and ν_f

$$\begin{aligned} \nu_g &= \max\{\deg f_1(D), \deg f_2(D), \ldots, \deg f_c(D), \deg q(D)\} \\ &\leq \max\{\deg f_1(D), \deg f_2(D), \ldots, \deg f_c(D)\} = \nu_f \end{aligned} \tag{2.280}$$

from which (i) and $\nu_g = \nu_f$ follow.

Moreover,

$$\nu_g = \nu_f = \deg \gcd(f_1(D), f_2(D), \ldots, f_c(D)) + \nu_\ell \tag{2.281}$$

where ν_ℓ is the constraint length of $\boldsymbol{\ell}(D)$. From (2.281), the equivalence of the generator matrices $\boldsymbol{g}(D)$ and $\boldsymbol{\ell}(D)$, and the assumption that $\boldsymbol{g}(D)$ is canonical, it follows that

$$\deg \gcd(f_1(D), f_2(D), \ldots, f_c(D)) = 0 \tag{2.282}$$

which is equivalent to (ii).

Conversely, suppose that (i) does not hold; that is, that

$$\deg q(D) > \max\{\deg f_1(D), \deg f_2(D), \ldots, \deg f_c(D)\} \tag{2.283}$$

From (2.283) follows that

$$\nu_g = \deg q(D) > \nu_f \tag{2.284}$$

and, hence, since $\boldsymbol{g}(D)$ and $\boldsymbol{f}(D)$ are equivalent, that $\boldsymbol{g}(D)$ is not canonical.

Finally, suppose that (i) holds and that (ii) does not hold. Then from (2.281) follows that $\nu_g > \nu_\ell$ and, since $\boldsymbol{g}(D)$ and $\boldsymbol{\ell}(D)$ are equivalent, that $\boldsymbol{g}(D)$ is not canonical. ∎

Next we will introduce a few concepts from valuation theory [Jac89]:

Let $\mathcal{P}$ be the set of irreducible polynomials in $\mathbb{F}_2[D]$. For simplicity we write p for the irreducible polynomial $p(D) \in \mathcal{P}$. For any nonzero $g(D) \in \mathbb{F}_2(D)$ we can express $g(D)$ as

$$g(D) = p^{e_p(g(D))}\, h(D)/d(D) \tag{2.285}$$

where $e_p(g(D)) \in \mathbb{Z}$, $h(D)$ and $d(D) \in \mathbb{F}_2[D]$, $\gcd(h(D), d(D)) = 1$, and $p \nmid h(D)d(D)$. The exponents $e_p(g(D)), p \in \mathcal{P}$, that occur in this unique factorization are called the *valuations* of the rational function $g(D)$ at the primes p, or the *p-valuations* of $g(D)$. By convention we define $e_p(0) = \infty$. The map

$$\begin{aligned} e_p : \mathbb{F}_2(D) &\to \mathbb{Z} \cup \{\infty\} \\ g(D) &\mapsto e_p(g(D)) \end{aligned}$$

is called an *exponential valuation* of $\mathbb{F}_2(D)$. Moreover, for any nonzero $g(D) \in \mathbb{F}_2(D)$ we can express $g(D)$ as

$$g(D) = f(D)/q(D) \tag{2.286}$$

where $f(D), q(D) \in \mathbb{F}_2[D]$. We define

$$e_{D^{-1}}(g(D)) = \deg q(D) - \deg f(D) \tag{2.287}$$

and $e_{D^{-1}}(0) = \infty$. Then $e_{D^{-1}}$ is also called an exponential valuation of $\mathbb{F}_2(D)$.

For notational convenience we introduce

$$\mathcal{P}^* = \mathcal{P} \cup \left\{D^{-1}\right\} \tag{2.288}$$

It is easily verified that the p-valuation $e_p(g(D))$ for each $p \in \mathcal{P}^*$ satisfies the properties:

(i) (uniqueness of 0) $e_p(g(D)) = \infty$ if and only if $g(D) = 0$

(ii) (additivity) $e_p(g(D)h(D)) = e_p(g(D)) + e_p(h(D))$

(iii) (strong triangle inequality) $e_p(g(D) + h(D)) \geq \min\{e_p(g(D)), e_p(h(D))\}$

From (2.285) and (2.287) we have the important *product formula*, as it is called in valuation theory, written here in additive form since we are using exponential valuations:

$$\sum_{p \in \mathcal{P}^*} e_p(g(D)) \deg p = 0 \tag{2.289}$$

for all nonzero $g(D) \in \mathbb{F}_2(D)$, where the degree of D^{-1} is defined as 1.

■ **EXAMPLE 2.26**

Let $g(D) = (D^3 + D^5)/(1 + D + D^2)$. Then we have $e_{1+D+D^2}(g(D)) = -1$, $e_{1+D}(g(D)) = 2$, $e_D(g(D)) = 3$, $e_{D^{-1}}(g(D)) = -3$, and

$$e_p(g(D)) = 0 \quad \text{if } p \in \mathcal{P}^* \text{ and } p \neq 1 + D + D^2, 1 + D, D, D^{-1} \tag{2.290}$$

It is easy to verify that

$$\sum_{p \in \mathcal{P}^*} e_p(g(D)) \deg p = 0 \tag{2.291}$$

The *delay* of a rational function $g(D)$ can be expressed in terms of the exponential valuation e_D as

$$\operatorname{del} g(D) = e_D(g(D)) \tag{2.292}$$

Similarly, the *degree* of a rational function $g(D)$ may be expressed as

$$\deg g(D) = -e_{D^{-1}}(g(D)) \tag{2.293}$$

A rational function $g(D)$ is:

(i) *causal*, if $\operatorname{del} g(D) \geq 0$, i.e., if $e_D(g(D)) \geq 0$

(ii) *polynomial*, if $e_p(g(D)) \geq 0$ for all $p \in \mathcal{P}$

(iii) *finite*, if $e_p(g(D)) \geq 0$ for all $p \in \mathcal{P}$ except possibly D

Remark: Causal rational functions are also sometimes called proper (particularly when z-transforms are used rather than D-transforms).

We mentioned in Section 2.1 that a rational function $g(D)$ may be expanded by long division into a Laurent series in powers of D and thus identified with a semi-infinite sequence over $\mathbb{F}_2$ that begins with allzeros; for example,

$$1/(1 + D) = 1 + D + D^2 + \cdots \tag{2.294}$$

In this way, the set of rational functions $\mathbb{F}_2(D)$ may be identified with a subset of the set $\mathbb{F}_2((D))$ of Laurent series in D over $\mathbb{F}_2$, which we shall call the rational Laurent series. These are precisely the Laurent series that eventually become periodic. The first nonzero term of a Laurent series expansion of $g(D)$ in powers of D is the term involving $D^{e_D(g(D))} = D^{\operatorname{del} g(D)}$, that is, the Laurent series in D "starts" at a "time index" equal to the delay $e_D(g(D))$ of $g(D)$.

Remark: Alternatively, a rational function may be expanded similarly into a Laurent series in D^{-1}; for example,

$$1/(1 + D) = D^{-1} + D^{-2} + \cdots \tag{2.295}$$

In this way, $\mathbb{F}_2(D)$ may alternatively be identified with a subset of $\mathbb{F}_2((D^{-1}))$. If elements of $\mathbb{F}_2((D^{-1}))$ are identified with semi-infinite sequences over $\mathbb{F}_2$ that finish

with allzeros, then $g(D)$ "ends" at a time equal to the degree $-e_{D^{-1}}(g(D))$ of $g(D)$. This hints at why we use the notation $p = D^{-1}$ for this valuation.

We should emphasize that this second, alternative expansion is a purely mathematical construct. When we wish to identify a rational function in $g(D)$ with a physical sequence of elements of $\mathbb{F}_2$, we shall always use the first Laurent series expansion in powers of D.

A rational function $g(D)$ may be expanded as a Laurent series in powers of p with coefficients in the residue class field $\mathbb{F}_2[D]_p = \mathbb{F}_2[D]/p\mathbb{F}_2[D]$ for any p in $\mathcal{P}^*$, as follows. Let $g(D) = f(D)/q(D)$, where $f(D)$ and $q(D) \neq 0$ are polynomial. If $f(D) = 0$, then the Laurent series in powers of p of $g(D)$ is simply $f(D) = 0$. If $f(D) \neq 0$, then we may write

$$f(D) = [f(D)]_p p^{e_p(f(D))} + f^{(1)}(D) \tag{2.296}$$

where $[f(D)]_p$ is the residue of $f(D)p^{-e_p(f(D))}$ modulo p, which is an element in the residue class field $\mathbb{F}_2[D]_p = \mathbb{F}_2[D]/p\mathbb{F}_2[D]$ and $f^{(1)}(D)$ is a polynomial (possibly 0) whose p-valuation is greater than $e_p(f(D))$. Iterating this process, possibly indefinitely, we obtain a Laurent series in powers of p, which is an element in $\mathbb{F}_2[D]_p((p))$ and whose first nonzero term is $[f(D)]_p\, p^{e_p(f(D))}$. Similarly, we may expand the denominator $q(D)$ into a Laurent series in powers of p whose first nonzero term is $[q(D)]_p\, p^{e_p(q(D))}$. Then by long division we obtain a Laurent series expansion of $g(D)$ in powers of p whose first term is $[g(D)]_p p^{e_p(g(D))}$, where

$$e_p(g(D)) = e_p(f(D)) - e_p(q(D)) \tag{2.297}$$

and

$$[g(D)]_p = [f(D)]_p \,/\, [q(D)]_p \tag{2.298}$$

This division is well-defined because $[f(D)]_p$ and $[q(D)]_p$ are nonzero residues of polynomials in $\mathbb{F}_2[D]$ modulo p.

If $g(D) = 0$, then in addition to $e_p(0) = \infty$, we define $[0]_p = 0$ for all p in $\mathcal{P}^*$.

Note that this general expansion method works perfectly well for $p = D^{-1}$, if we take $[f(D)]_{D^{-1}}$ and $[q(D)]_{D^{-1}}$ to be the coefficients of the highest-order terms of $f(D)$ and $q(D)$, respectively, that is, the coefficients of $D^{\deg f(D)}$ and $D^{\deg q(D)}$, respectively. Again, this explains our use of the notation $p = D^{-1}$ for this valuation.

■ **EXAMPLE 2.27**

For $f(D) = D + D^2 + D^3$ (or indeed for any nonzero polynomial in D), the Laurent series in the polynomial D is simply $f(D)$. We have $e_D(f(D)) = 1$, $[f(D)]_D = 1$, and the first nonzero term of the series is $[f(D)]_D D^{e_D(f(D))} = D$. Similarly, for $p = D^{-1}$, we have $e_{D^{-1}}(f(D)) = -3$, $[f(D)]_{D^{-1}} = 1$, and the Laurent series in D^{-1} is

$$f(D) = (D^{-1})^{-3} + (D^{-1})^{-2} + (D^{-1})^{-1} \tag{2.299}$$

whose first nonzero term is $[f(D)]_{D^{-1}}(D^{-1})^{e_{D^{-1}}(f(D))} = (D^{-1})^{-3}$. For $p = 1 + D$, we have $e_{1+D}(f(D)) = 0$, $[f(D)]_{1+D} = 1$, and the Laurent series in

$1 + D$ is

$$f(D) = (1 + D)^0 + (1 + D)^3 \tag{2.300}$$

whose first nonzero term is $[f(D)]_{1+D}(1 + D)^{e_{1+D}(f(D))} = (1 + D)^0$. For $p = 1 + D + D^2$, we have $e_{1+D+D^2}(f(D)) = 1$, $[f(D)]_{1+D+D^2} = D$, and the Laurent series in $1 + D + D^2$ is simply

$$f(D) = D(1 + D + D^2)^1 \tag{2.301}$$

whose first and only nonzero term is $[f(D)]_{1+D+D^2}(1+D+D^2)^{e_{1+D+D^2}(f(D))} = D(1 + D + D^2)^1$.

■ **EXAMPLE 2.28**

Let $g(D) = (D^3 + D^5)/(1 + D + D^2)$. Then $e_D(g(D)) = 3 = \operatorname{del} g(D)$, $e_{D^{-1}}(g(D)) = -3 = -\deg g(D)$, $e_{1+D}(g(D)) = 2$, $e_{1+D+D^2}(g(D)) = -1$, and all other p-valuations are zero. It is easy to verify that the product formula holds. Also, $[g(D)]_D = [g(D)]_{D^{-1}} = [g(D)]_{1+D} = 1$ and $[g(D)]_{1+D+D^2} = D$.

Next we extend the valuations to vectors (or sets) of rational functions.

Let $\boldsymbol{g}(D) = (g_1(D)\, g_2(D)\, \ldots\, g_c(D))$, where $g_1(D), g_2(D), \ldots, g_c(D) \in \mathbb{F}_2(D)$. For any $p \in \mathcal{P}^*$ we define

$$e_p(\boldsymbol{g}(D)) = \min\{e_p(g_1(D)), e_p(g_2(D)), \ldots, e_p(g_c(D))\}. \tag{2.302}$$

Remark: Equality (2.302) generalizes the notion of the "greatest common divisor." Indeed, if $\boldsymbol{g}(D)$ is a set of polynomials, then the greatest common divisor of the set $\boldsymbol{g}(D)$ is

$$\gcd \boldsymbol{g}(D) = \prod_{p \in \mathcal{P}} p^{e_p(g(D))} \tag{2.303}$$

Now we can write Lemma 2.53 in a more symmetric form:

Lemma 2.54 Let $\boldsymbol{g}(D) = (g_1(D)\, g_2(D) \ldots g_c(D))$ be a $1 \times c$ generator matrix over $\mathbb{F}_2(D)$. Then $\boldsymbol{g}(D)$ is canonical if and only if

$$e_p(\boldsymbol{g}(D)) \le 0, \quad \text{all } p \in \mathcal{P}^* \tag{2.304}$$

Proof: We will prove that (2.304) is equivalent to (i) and (ii) of Lemma 2.53. From (2.278), (2.279), and (2.302) it follows that

$$e_{D^{-1}}(\boldsymbol{g}(D)) = \deg q(D) - \max\{\deg f_1(D), \deg f_2(D), \ldots, \deg f_c(D)\} \tag{2.305}$$

Hence,

$$\begin{aligned} & e_{D^{-1}}(\boldsymbol{g}(D)) \le 0 \\ \Leftrightarrow\ & \deg q(D) \le \max\{\deg f_1(D), \deg f_2(D), \ldots, \deg f_c(D)\} \end{aligned} \tag{2.306}$$

For the second half of the proof, let p be any irreducible polynomial of $\mathbb{F}_2[D]$. First, we assume that $p \mid q(D)$. Since (2.279) holds, $p \nmid f_i(D)$ for some i. Then we have both

$$\begin{aligned} e_p(\boldsymbol{g}(D)) &= \min\{e_p(f_1(D)/q(D)), e_p(f_2(D)/q(D)), \ldots, e_p(f_c(D)/q(D))\} \\ &= e_p(f_i(D)/q(D)) = -e_p(q(D)) < 0 \end{aligned} \tag{2.307}$$

and

$$p \nmid \gcd(f_1(D), f_2(D), \ldots, f_c(D))$$

Now we assume that $p \nmid q(D)$. Then

$$e_p(\boldsymbol{g}(D)) = \min\{e_p(f_1(D)), e_p(f_2(D)), \ldots, e_p(f_c(D))\} \geq 0 \tag{2.308}$$

Thus,

$$\begin{aligned} e_p(\boldsymbol{g}(D)) \leq 0 &\Leftrightarrow p \nmid f_i(D) \text{ for some } i \\ &\Leftrightarrow p \nmid \gcd(f_1(D), f_2(D), \ldots, f_c(D)) \end{aligned} \tag{2.309}$$

Therefore,

$$\begin{aligned} e_p(\boldsymbol{g}(D)) \leq 0 &\text{ for all irreducible polynomial } p \\ &\Leftrightarrow \gcd(f_1(D), f_2(D), \ldots, f_c(D)) = 1 \end{aligned} \tag{2.310}$$

which completes the proof. ∎

In the following we will give necessary and sufficient conditions for a $b \times c$ rational generator matrix to be canonical but first some prerequisites.

Properties (i)–(iii) given below (2.288), appropriately generalized, continue to hold:

(i) $e_p(\boldsymbol{g}(D)) = \infty$ if and only if $\boldsymbol{g}(D) = \mathbf{0}$

(ii) $e_p(k(D)\boldsymbol{g}(D)) = e_p(k(D)) + e_p(\boldsymbol{g}(D))$ for all $k(D) \in \mathbb{F}_2(D)$

(iii) $e_p(\boldsymbol{g}(D) + \boldsymbol{h}(D)) \geq \min\{e_p(\boldsymbol{g}(D)), e_p(\boldsymbol{h}(D))\}$

However, the product formula becomes an inequality, since for any i

$$\sum_{p \in \mathcal{P}^*} e_p(\boldsymbol{g}(D)) \deg p \leq \sum_{p \in \mathcal{P}^*} e_p(g_i(D)) \deg p = 0 \tag{2.311}$$

We therefore define the *defect* of a $1 \times c$ nonzero vector $\boldsymbol{g}(D) = (g_1(D)\, g_2(D) \ldots g_c(D))$ over $\mathbb{F}_2(D)$ to be [For75]

$$\operatorname{def} \boldsymbol{g}(D) \stackrel{\text{def}}{=} -\sum_{p \in \mathcal{P}^*} e_p(\boldsymbol{g}(D)) \deg p \tag{2.312}$$

We may generalize the definition of delay and degree to a vector $\boldsymbol{g}(D)$ as

$$\operatorname{del} \boldsymbol{g}(D) = e_D(\boldsymbol{g}(D)) = \min_i \{\operatorname{del} g_i(D)\} \tag{2.313}$$

$$\deg \boldsymbol{g}(D) = -e_{D^{-1}}(\boldsymbol{g}(D)) = \max_i \{\deg g_i(D)\} \tag{2.314}$$

Then (2.312) can also be written as

$$\begin{aligned} \operatorname{def} \boldsymbol{g}(D) &= \deg \boldsymbol{g}(D) - \sum_{p \in \mathcal{P}} e_p(\boldsymbol{g}(D)) \deg p \\ &= \deg \boldsymbol{g}(D) - \deg(\gcd \boldsymbol{g}(D)) \end{aligned} \tag{2.315}$$

In view of property (ii) and the product formula, we have for all nonzero $k(D) \in \mathbb{F}_2(D)$

$$\operatorname{def} k(D)\boldsymbol{g}(D) = \operatorname{def} \boldsymbol{g}(D) \tag{2.316}$$

Thus, every nonzero vector in a one-dimensional rational vector space has the same defect.

When $c = 1$, that is, when $\boldsymbol{g}(D)$ reduces to a nonzero $g(D) \in \mathbb{F}_2(D)$, we have

$$\operatorname{def} g(D) = -\sum_{p \in \mathcal{P}^*} e_p(g(D)) \deg p \tag{2.317}$$

From the product formula it follows that for any nonzero $g(D) \in \mathbb{F}_2(D)$, $\operatorname{def} g(D) = 0$.

The following lemma shows the significance of $\operatorname{def} \boldsymbol{g}(D)$.

Lemma 2.55 Let $\boldsymbol{g}(D) = (g_1(D)\, g_2(D) \ldots g_c(D))$ be a $1 \times c$ nonzero generator matrix over $\mathbb{F}_2(D)$. Write

$$g_i(D) = f_i(D)/q(D), \quad i = 1, 2, \ldots, c \tag{2.318}$$

where $f_i(D), q(D) \in \mathbb{F}_2[D]$, $i = 1, 2, \ldots, c$, and

$$\gcd\left(f_1(D), f_2(D), \ldots, f_c(D), q(D)\right) = 1 \tag{2.319}$$

and assume that $\boldsymbol{g}(D)$ is canonical. Then,

$$\operatorname{def} \boldsymbol{g}(D) = \max\{\deg f_1(D), \deg f_2(D), \ldots, \deg f_c(D)\} \tag{2.320}$$

and $\operatorname{def} \boldsymbol{g}(D)$ is the constraint length of $\boldsymbol{g}(D)$.

Proof: We have

$$\begin{aligned} \operatorname{def} \boldsymbol{g}(D) &= -\sum_{p \in \mathcal{P}^*} e_p(\boldsymbol{g}(D)) \deg p \\ &= -\Big(e_{D^{-1}}(\boldsymbol{g}(D)) + \sum_{p | q(D)} e_p(\boldsymbol{g}(D)) \deg p + \sum_{p \nmid q(D)} e_p(\boldsymbol{g}(D)) \deg p \Big) \\ &= -\Big(\big(\deg q(D) - \max\big\{ \deg f_1(D), \deg f_2(D), \ldots, \deg f_c(D) \big\} \big) \\ &\qquad - \sum_{p | q(D)} e_p(q(D)) \deg p + 0 \Big) \end{aligned} \tag{2.321}$$

where in the last equality the first term follows from (2.305), the second term from (2.307), and the last term from (2.308) and Lemma 2.54. The observation that

$$\deg q(D) = \sum_{p|q(D)} e_p(q(D)) \deg p \tag{2.322}$$

and application of Lemma 2.53 complete the proof. ∎

■ **EXAMPLE 2.29**

Let

$$\boldsymbol{g}(D) = \begin{pmatrix} \dfrac{D}{1+D} & \dfrac{1+D}{1+D+D^2} & D^2 \end{pmatrix} \tag{2.323}$$

By definition,

$$e_{1+D+D^2}(\boldsymbol{g}(D)) = \min\{0, -1, 0\} = -1 \tag{2.324}$$

Similarly, $e_{1+D}(\boldsymbol{g}(D)) = -1$, $e_D(\boldsymbol{g}(D)) = 0$, $e_{D^{-1}}(\boldsymbol{g}(D)) = -2$, and $e_p(\boldsymbol{g}(D)) = 0$ if $p \in \mathcal{P}^*$ and $p \neq 1 + D + D^2$, $1 + D$, D, D^{-1}. It follows from Lemma 2.54 that $\boldsymbol{g}(D)$ is canonical.

We can express $\boldsymbol{g}(D)$ as

$$\boldsymbol{g}(D) = \begin{pmatrix} \dfrac{D + D^2 + D^3}{1 + D^3} & \dfrac{1 + D^2}{1 + D^3} & \dfrac{D^2 + D^5}{1 + D^3} \end{pmatrix} \tag{2.325}$$

which can be implemented by five delay elements in controller canonical form.

Let $G(D) = \{\boldsymbol{g}_i(D), 1 \leq i \leq b\}$ be a set of vectors $\boldsymbol{g}_i(D) \in \mathbb{F}_2^c(D)$. In view of properties (ii) and (iii), for any vector $\boldsymbol{v}(D) = \sum_i u_i(D)\boldsymbol{g}_i(D)$ and any $p \in \mathcal{P}^*$, we have

$$\begin{aligned} e_p(\boldsymbol{v}(D)) &\geq \min_i \{e_p(u_i(D)\boldsymbol{g}_i(D))\} && \text{(iii)} \\ &= \min_i \{e_p(u_i(D)) + e_p(\boldsymbol{g}_i(D))\} && \text{(ii)} \end{aligned} \tag{2.326}$$

Monna [Mon70] defines the set $G(D)$ to be *p-orthogonal* if equality holds (2.326) for all rational b-tuples $\boldsymbol{u}(D)$; that is, if for all $\boldsymbol{u}(D)$ in $\mathbb{F}_2^b(D)$ we have

$$e_p(\boldsymbol{v}(D)) = \min_i \{e_p(u_i(D)) + e_p(\boldsymbol{g}_i(D))\} \tag{2.327}$$

If $G(D)$ is p-orthogonal for all p in $\mathcal{P}^*$, then the set $G(D)$ is called *globally orthogonal*.

A $b \times c$ polynomial matrix $G(D) = \{\boldsymbol{g}_i(D) \in \mathbb{F}_2^c[D], 1 \leq i \leq b\}$ was said in Section 2.5 to have the predictable degree property (PDP) if for all $\boldsymbol{v}(D) = \boldsymbol{u}(D)G(D)$, where $\boldsymbol{u}(D)$ and $\boldsymbol{v}(D)$ are polynomial vectors,

$$\deg \boldsymbol{v}(D) = \max_i \{\deg u_i(D) + \deg \boldsymbol{g}_i(D)\} \tag{2.328}$$

Equivalently, in the terminology we are using here, $G(D)$ has the PDP if for all $\boldsymbol{v}(D) = \boldsymbol{u}(D)G(D)$

$$e_{D^{-1}}(\boldsymbol{v}(D)) = \min_i \{e_{D^{-1}}(u_i(D)) + e_{D^{-1}}(\boldsymbol{g}_i(D))\} \tag{2.329}$$

that is, if $G(D)$ is D^{-1}-orthogonal. Hence, the PDP is naturally generalized as follows:

Definition A rational matrix $G(D)$ has the *predictable degree property* (PDP) if it is D^{-1}-orthogonal.

Definition For any $p \in \mathcal{P}^*$, a rational matrix $G(D)$ has the *predictable p-valuation property* (PVP$_p$) if it is p-orthogonal.

Definition A rational matrix $G(D)$ has the *global predictable valuation property* (GPVP) if it is globally orthogonal.

We will see below that the GPVP is an essential property of canonical encoding matrices of convolutional codes.

Let $\boldsymbol{g}(D) \in \mathbb{F}_2^c(D)$. We define the residue vector $[\boldsymbol{g}(D)]_p$ as the vector whose components are residues of the corresponding components of the vector $\boldsymbol{g}(D)p^{-e_p(\boldsymbol{g}(D))}$ modulo p in the ring of formal power series $\mathbb{F}_2[D]_p[[p]]$. Thus, if $e_p(g_i(D)) > e_p(\boldsymbol{g}(D))$, then $[g_i(D)]_p = 0$, even if $g_i(D) \neq 0$. If $\boldsymbol{g}(D)$ is expanded as a vector of Laurent series in powers of p with coefficients in $\mathbb{F}_2^c[D]_p$, then $[\boldsymbol{g}(D)]_p p^{e_p(\boldsymbol{g}(D))}$ is the first nonzero term in the expansion.

In Section 2.5 for a polynomial generator matrix $G(D)$ the matrix $[G(D)]_h$ was defined as consisting of the high-order coefficient vectors $[\boldsymbol{g}_i(D)]_h$ which we would call here the residue vectors $[\boldsymbol{g}_i(D)]_{D^{-1}}$. It was shown that for the PDP to hold for $G(D)$ it is necessary and sufficient that $[G(D)]_h$ have full rank. We have the following natural generalization [For75]:

Definition Given a rational matrix $G(D)$, its *p-residue matrix* $[G(D)]_p$ is the $\mathbb{F}_2[D]_p$-matrix whose ith row is the residue vector $[\boldsymbol{g}_i(D)]_p$, $1 \leq i \leq b$.

The following theorem then gives a basic test for p-orthogonality:

Theorem 2.56 For any $p \in \mathcal{P}^*$, a rational matrix $G(D)$ has the PVP$_p$ (is p-orthogonal) if and only if its p-residue matrix $[G(D)]_p$ has full rank over $\mathbb{F}_2[D]_p$.

Proof: In general, if $\boldsymbol{v}(D) = \boldsymbol{u}(D)G(D)$, where $\boldsymbol{u}(D) = (u_1(D)\, u_2(D) \ldots u_b(D))$ and $u_i(D) \in \mathbb{F}_2(D)$, $1 \leq i \leq b$, then

$$e_p(\boldsymbol{v}(D)) \geq d \tag{2.330}$$

where

$$d = \min_i \{e_p(u_i(D)) + e_p(\boldsymbol{g}_i(D))\} \tag{2.331}$$

Let $\mathcal{I}$ be the set of indices such that the minimum is achieved, that is,

$$\mathcal{I} = \{i \mid e_p(u_i(D)) + e_p(\boldsymbol{g}_i(D)) = d\} \tag{2.332}$$

Then, if $\boldsymbol{v}(D) \neq \mathbf{0}$, the Laurent series expansion of $\boldsymbol{v}(D)$ in powers of p with coefficients in $\mathbb{F}_2^c[D]_p$ may be written as

$$\boldsymbol{v}(D) = \boldsymbol{v}_d p^d + \boldsymbol{v}_{d+1} p^{d+1} + \cdots \tag{2.333}$$

$G(D)$ is p-orthogonal if and only if for all $\boldsymbol{u}(D) \neq \mathbf{0}$, $e_p(\boldsymbol{v}(D)) = d$, that is, $\boldsymbol{v}_d \neq \mathbf{0}$. We may write the Laurent series expansions of the nonzero $u_i(D)$ and of the $\boldsymbol{g}_i(D)$ as

$$u_i(D) = [u_i(D)]_p p^{e_p(u_i(D))} + u_i^{(1)}(D), \quad 1 \leq i \leq b \tag{2.334}$$

$$\boldsymbol{g}_i(D) = [\boldsymbol{g}_i(D)]_p p^{e_p(\boldsymbol{g}_i(D))} + \boldsymbol{g}_i^{(1)}(D), \quad 1 \leq i \leq b \tag{2.335}$$

where for all i $[u_i(D)]_p \neq 0$, $e_p(u_i^{(1)}(D)) > e_p(u_i(D))$, $[\boldsymbol{g}_i(D)]_p \neq \mathbf{0}$, and $e_p(\boldsymbol{g}_i^{(1)}(D)) > e_p(\boldsymbol{g}_i(D))$. Then the lowest-order coefficient of $\boldsymbol{v}(D)$ is given by

$$\boldsymbol{v}_d = \sum_{i \in \mathcal{I}} [u_i(D)]_p [\boldsymbol{g}_i(D)]_p \tag{2.336}$$

If $\boldsymbol{v}_d = \mathbf{0}$, then the p-residue vectors $[\boldsymbol{g}_i(D)]_p$ are linearly dependent over $\mathbb{F}_2[D]_p$ and $[G(D)]_p$ does not have full rank. Conversely, if $[G(D)]_p$ does not have full rank over $\mathbb{F}_2[D]_p$, then there exists some nontrivial linear combination of rows that equals zero:

$$\sum_i u_i(D)[\boldsymbol{g}_i(D)]_p = 0 \tag{2.337}$$

where $u_i(D) \in \mathbb{F}_2[D]_p$. Therefore with the input sequence

$$(u_1(D)p^{-e_p(\boldsymbol{g}_1(D))} \;\; u_2(D)p^{-e_p(\boldsymbol{g}_2(D))} \ldots u_b(D)p^{-e_p(\boldsymbol{g}_b(D))})$$

we have $d = 0$ and $\boldsymbol{v}_d = \mathbf{0}$, so

$$e_p(\boldsymbol{v}(D)) > 0 = \min_i \left\{ e_p(u_i(D)p^{e_p(\boldsymbol{g}_i(D))}) + e_p(\boldsymbol{g}_i(D)) \right\} \tag{2.338}$$

which implies that $G(D)$ is not p-orthogonal. This completes the proof. ∎

This test involves only the determination of the rank of a $b \times c$ matrix $[G(D)]_p$ over the field $\mathbb{F}_2[D]_p$ and is thus easy to carry out for any polynomial p of moderate degree.

From Theorem 2.56 the next corollary follows immediately

Corollary 2.57 A rational matrix $G(D)$ has the GPVP (is globally orthogonal) if and only if its p-residue matrix $[G(D)]_p$ has full rank over $\mathbb{F}_2[D]_p$ for all $p \in \mathcal{P}^*$.

■ **EXAMPLE 2.30**

Consider the rational encoding matrix

$$G(D) = \begin{pmatrix} \boldsymbol{g}_1(D) \\ \boldsymbol{g}_2(D) \end{pmatrix} = \begin{pmatrix} 1 & \frac{D}{1+D} & \frac{1}{1+D} \\ \frac{D^2}{1+D+D^2} & \frac{1}{1+D+D^2} & 1 \end{pmatrix} \tag{2.339}$$

Let $p = D$. Then we have $e_D(1) = e_D(\frac{1}{1+D}) = e_D(\frac{1}{1+D+D^2}) = 0$, $e_D(\frac{D}{1+D}) = 1$, and $e_D(\frac{D^2}{1+D+D^2}) = 2$. Hence, we have $e_D(\boldsymbol{g}_1(D)) = e_D(\boldsymbol{g}_2(D)) = 0$, and

$$\begin{aligned} [\boldsymbol{g}_1(D)]_D &= \begin{pmatrix} 1 & \frac{D}{1+D} & \frac{1}{1+D} \end{pmatrix} D^0 \\ &= \begin{pmatrix} 1 & 0 & 1 \end{pmatrix} \pmod{D} \end{aligned} \tag{2.340}$$

$$\begin{aligned} [\boldsymbol{g}_2(D)]_D &= \begin{pmatrix} \frac{D^2}{1+D+D^2} & \frac{1}{1+D+D^2} & 1 \end{pmatrix} D^0 \\ &= \begin{pmatrix} 0 & 1 & 1 \end{pmatrix} \pmod{D} \end{aligned} \tag{2.341}$$

Thus, we have

$$[G(D)]_D = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix} \tag{2.342}$$

which has full rank over $\mathbb{F}_2[D]_D = \mathbb{F}_2$.

For $p = 1 + D$ we have $e_{1+D}(1) = e_{1+D}(\frac{D^2}{1+D+D^2}) = e_{1+D}(\frac{1}{1+D+D^2}) = 0$ and $e_{1+D}(\frac{D}{1+D}) = e_{1+D}(\frac{1}{1+D}) = -1$.
Hence, $e_{1+D}(\boldsymbol{g}_1(D)) = -1$, $e_{1+D}(\boldsymbol{g}_2(D)) = 0$, and

$$\begin{aligned} [\boldsymbol{g}_1(D)]_{1+D} &= \begin{pmatrix} 1 & \frac{D}{1+D} & \frac{1}{1+D} \end{pmatrix} (1+D)^{-(-1)} \\ &= \begin{pmatrix} 0 & 1 & 1 \end{pmatrix} \pmod{(1+D)} \end{aligned} \tag{2.343}$$

$$\begin{aligned} [\boldsymbol{g}_2(D)]_{1+D} &= \begin{pmatrix} \frac{D^2}{1+D+D^2} & \frac{1}{1+D+D^2} & 1 \end{pmatrix} (1+D)^0 \\ &= \begin{pmatrix} 1 & 1 & 1 \end{pmatrix} \pmod{(1+D)} \end{aligned} \tag{2.344}$$

Thus, we have

$$[G(D)]_{1+D} = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix} \tag{2.345}$$

which has full rank over $\mathbb{F}_2[D]_{1+D}$.

Next, we let $p = 1 + D + D^2$ and obtain $e_{1+D+D^2}(1) = e_{1+D+D^2}(\frac{D}{1+D}) = e_{1+D+D^2}(\frac{1}{1+D}) = 0$ and $e_{1+D+D^2}(\frac{D^2}{1+D+D^2}) = e_{1+D+D^2}(\frac{1}{1+D+D^2}) = -1$.
Hence, $e_{1+D+D^2}(\boldsymbol{g}_1(D)) = 0$, $e_{1+D+D^2}(\boldsymbol{g}_2(D)) = -1$.

A simple way to calculate $D/(1+D) \pmod{(1+D+D^2)}$ is as follows:

First, we need the inverse of the denominator, that is, $(1+D)^{-1} \pmod{(1+D+D^2)}$. From Euclid's algorithm it follows that

$$1 = D(1+D) + (1+D+D^2) \tag{2.346}$$

and, hence,

$$(1+D)^{-1} = D \pmod{(1+D+D^2)} \tag{2.347}$$

Then we have

$$\frac{D}{1+D} = D(1+D)^{-1} = D^2 = 1+D \pmod{(1+D+D^2)} \tag{2.348}$$

Similarly, we have

$$\frac{1}{1+D} = D \pmod{(1+D+D^2)} \tag{2.349}$$

Thus,

$$\begin{aligned}[\boldsymbol{g}_1(D)]_{1+D+D^2} &= \begin{pmatrix} 1 & \frac{D}{1+D} & \frac{1}{1+D} \end{pmatrix} (1+D+D^2)^0 \\ &= \begin{pmatrix} 1 & 1+D & D \end{pmatrix} \pmod{(1+D+D^2)} \end{aligned} \tag{2.350}$$

$$\begin{aligned}[\boldsymbol{g}_2(D)]_{1+D+D^2} &= \begin{pmatrix} \frac{D^2}{1+D+D^2} & \frac{1}{1+D+D^2} & 1 \end{pmatrix} (1+D+D^2)^{-(-1)} \\ &= \begin{pmatrix} 1+D & 1 & 0 \end{pmatrix} \pmod{(1+D+D^2)} \end{aligned} \tag{2.351}$$

Thus,

$$[G(D)]_{1+D+D^2} = \begin{pmatrix} 1 & 1+D & D \\ 1+D & 1 & 0 \end{pmatrix} \tag{2.352}$$

which has full rank over $\mathbb{F}_2[D]_{1+D+D^2}$. Finally, let $p = D^{-1}$. Then we have $e_{D^{-1}}(1) = e_{D^{-1}}(\frac{D}{1+D}) = e_{D^{-1}}(\frac{D^2}{1+D+D^2}) = 0, e_{D^{-1}}(\frac{1}{1+D}) = 1$, and $e_{D^{-1}}(\frac{1}{1+D+D^2}) = 2$. Hence, $e_{D^{-1}}(\boldsymbol{g}_1(D)) =$
$e_{D^{-1}}(\boldsymbol{g}_2(D)) = 0$, and

$$\begin{aligned}[\boldsymbol{g}_1(D)]_{D^{-1}} &= \begin{pmatrix} 1 & \frac{D}{1+D} & \frac{1}{1+D} \end{pmatrix} (D^{-1})^0 \\ &= \begin{pmatrix} 1 & 1 & 0 \end{pmatrix} \pmod{D^{-1}} \end{aligned} \tag{2.353}$$

$$\begin{aligned}[\boldsymbol{g}_2(D)]_{D^{-1}} &= \begin{pmatrix} \frac{D^2}{1+D+D^2} & \frac{1}{1+D+D^2} & 1 \end{pmatrix} (D^{-1})^0 \\ &= \begin{pmatrix} 1 & 0 & 1 \end{pmatrix} \pmod{D^{-1}} \end{aligned} \tag{2.354}$$

Thus, we have

$$[G(D)]_{D^{-1}} = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix} \tag{2.355}$$

which has full rank over $\mathbb{F}_2[D]_{D^{-1}} = \mathbb{F}_2$.

For $p \neq D, 1+D, 1+D+D^2, D^{-1}$ we have $e_p(\boldsymbol{g}_i(D)) = 0$, $i = 1, 2$. Thus,

$$[G(D)]_p = \begin{pmatrix} 1 & \frac{D}{1+D} & \frac{1}{1+D} \\ \frac{D^2}{1+D+D^2} & \frac{1}{1+D+D^2} & 1 \end{pmatrix} \pmod{p} \quad (2.356)$$

which has full rank over $\mathbb{F}_2[D]_p$.

Since $[G(D)]_p$ has full rank over $\mathbb{F}_2[D]_p$ for all $p \in \mathcal{P}^*$, we conclude that $G(D)$ has the GPVP.

■ **EXAMPLE 2.31**

Consider the rational encoding matrix

$$G(D) = \begin{pmatrix} 1 & 0 & \frac{1+D^2}{1+D+D^2} & \frac{D^2}{1+D+D^2} \\ 0 & 1 & \frac{D^2}{1+D+D^2} & \frac{1}{1+D+D^2} \end{pmatrix}. \quad (2.357)$$

By repeating the steps in the previous example we can show that the p-residue matrix $[G(D)]_p$ has full rank over $\mathbb{F}_2[D]_p$ for $p = D$, $1+D$, and D^{-1}.

For $p = 1+D+D^2$ we have $e_{1+D+D^2}(1) = 0$, $e_{1+D+D^2}(0) = \infty$, and $e_{1+D+D^2}(\frac{1+D^2}{1+D+D^2}) = e_{1+D+D^2}(\frac{D^2}{1+D+D^2}) = e_{1+D+D^2}(\frac{1}{1+D+D^2}) = -1$. Hence, we have $e_{1+D+D^2}(\boldsymbol{g}_1(D)) = e_{1+D+D^2}(\boldsymbol{g}_2(D)) = -1$, and

$$\begin{aligned} [\boldsymbol{g}_1(D)]_{1+D+D^2} &= \begin{pmatrix} 1 & 0 & \frac{1+D^2}{1+D+D^2} & \frac{D^2}{1+D+D^2} \end{pmatrix} (1+D+D^2)^{-(-1)} \\ &= \begin{pmatrix} 0 & 0 & D & 1+D \end{pmatrix} \pmod{(1+D+D^2)} \quad (2.358) \\ [\boldsymbol{g}_2(D)]_{1+D+D^2} &= \begin{pmatrix} 0 & 1 & \frac{D^2}{1+D+D^2} & \frac{1}{1+D+D^2} \end{pmatrix} (1+D+D^2)^{-(-1)} \\ &= \begin{pmatrix} 0 & 0 & 1+D & 1 \end{pmatrix} \pmod{(1+D+D^2)} \quad (2.359) \end{aligned}$$

Thus, we have

$$[G(D)]_{1+D+D^2} = \begin{pmatrix} 0 & 0 & D & 1+D \\ 0 & 0 & 1+D & 1 \end{pmatrix} \quad (2.360)$$

whose rows are linearly dependent over $\mathbb{F}_2[D]_{1+D+D^2}$ and it follows that $G(D)$ does not have the GPVP.

Let $G(D)$ be a $b \times c$ generator matrix over $\mathbb{F}_2(D)$ and let $\mathcal{M}_b$ denote the set of all $b \times b$ submatrices of $G(D)$. For all $p \in \mathcal{P}^*$ we define [For75]

$$e_p(G(D)) \stackrel{\text{def}}{=} \min_{M_b(D) \in \mathcal{M}_b} \{e_p(|M_b(D)|)\} \quad (2.361)$$

and then correspondingly define the *internal defect* of $G(D)$ to be [For91]

$$\text{intdef}\, G(D) \stackrel{\text{def}}{=} -\sum_{p \in \mathcal{P}^*} e_p(G(D)) \deg p \tag{2.362}$$

Then we can prove the following important result:

Theorem 2.58 The internal defect $\text{intdef}\, G(D)$ is an invariant of the convolutional code $\mathcal{C}$ that is encoded by $G(D)$.

Proof: Let $T(D)$ be a $b \times b$ nonsingular matrix over $\mathbb{F}_2(D)$. Then

$$e_p(|T(D)M_b(D)|) = e_p(|T(D)|) + e_p(|M_b(D)|) \tag{2.363}$$

Hence,

$$e_p(T(D)G(D)) = e_p(|T(D)|) + e_p(G(D)) \tag{2.364}$$

It follows from (2.317), (2.362), and (2.364) that

$$\text{intdef}\, (T(D)G(D)) = \text{def}|T(D)| + \text{intdef}\, G(D) \tag{2.365}$$

But $|T(D)| \in \mathbb{F}_2(D)$ and $|T(D)| \neq 0$. By the product formula (2.289) and (2.317), $\text{def}|T(D)| = 0$. Hence,

$$\text{intdef}\, (T(D)G(D)) = \text{intdef}\, G(D) \tag{2.366}$$

■

Theorem 2.58 motivates us to introduce the *defect* of the *code* $\mathcal{C}$ encoded by the generator matrix $G(D)$ to be [For91]

$$\text{def}\, \mathcal{C} \stackrel{\text{def}}{=} \text{intdef}\, G(D) \tag{2.367}$$

We define the *external defect* of $G(D)$ as the sum of the generator defects:

$$\text{extdef}\, G(D) \stackrel{\text{def}}{=} \sum_{i=1}^{b} \text{def}\, \boldsymbol{g}_i(D) \tag{2.368}$$

Before we give five equivalent conditions for a generator matrix to be canonical we shall prove a lemma. Let

$$G(D) = (g_{ij}(D))_{1 \leq i \leq b,\ 1 \leq j \leq c} \tag{2.369}$$

be a generator matrix, where $g_{ij}(D) \in \mathbb{F}_2(D)$. Write

$$\boldsymbol{g}_i(D) = (g_{i1}(D)\, g_{i2}(D) \ldots g_{ic}(D)),\ \ i = 1, 2 \ldots, b \tag{2.370}$$

$$g_{ij}(D) = f_{ij}(D)/q_i(D),\ \ i = 1, 2, \ldots, b;\ \ j = 1, 2, \ldots, c \tag{2.371}$$

where $f_{ij}(D), q_i(D) \in \mathbb{F}_2[D]$, $i = 1, 2, \ldots, b$, $j = 1, 2, \ldots, c$, and assume that

$$\gcd(f_{i1}(D), f_{i2}(D), \ldots, f_{ic}(D), q_i(D)) = 1, \quad i = 1, 2, \ldots, b \tag{2.372}$$

Then define $G_1(D, p)$ by

$$G_1(D,p) = \begin{pmatrix} p^{e_p(\boldsymbol{g}_1(D))} & & & \\ & p^{e_p(\boldsymbol{g}_2(D))} & & \\ & & \ddots & \\ & & & p^{e_p(\boldsymbol{g}_b(D))} \end{pmatrix} [G(D)]_p \tag{2.373}$$

and $G_0(D, p)$ by

$$G(D) = G_0(D,p) + G_1(D,p) \tag{2.374}$$

From (2.374) and (2.373) we have the following:

Lemma 2.59 Let $G(D)$ be a $b \times c$ rational generator matrix and let $p \in \mathcal{P}^*$. Then

(i) $e_p([G(D)]_p) = 0$ if and only if $e_p(G(D)) = \sum_{i=1}^{b} e_p(\boldsymbol{g}_i(D))$

(ii) $e_p([G(D)]_p) \neq 0$ if and only if $e_p(G(D)) > \sum_{i=1}^{b} e_p(\boldsymbol{g}_i(D))$

We are now well prepared to prove the following:

Theorem 2.60 Let $G(D)$ be a $b \times c$ rational generator matrix with rows $\boldsymbol{g}_1(D), \boldsymbol{g}_2(D), \ldots, \boldsymbol{g}_b(D)$. Then the following statements are equivalent:

(i) $G(D)$ is a canonical encoding matrix.

(ii) For all $p \in \mathcal{P}^*$: $e_p(\boldsymbol{g}_i(D)) \leq 0$, $1 \leq i \leq b$, and $e_p([G(D)]_p) = 0$.

(iii) For all $p \in \mathcal{P}^*$: $e_p(\boldsymbol{g}_i(D)) \leq 0$, $1 \leq i \leq b$, and $e_p(G(D)) = \sum_{i=1}^{b} e_p(\boldsymbol{g}_i(D))$.

(iv) For all $p \in \mathcal{P}^*$: $e_p(\boldsymbol{g}_i(D)) \leq 0$, $1 \leq i \leq b$, and $\operatorname{intdef} G(D) = \operatorname{extdef} G(D)$.

(v) For all $p \in \mathcal{P}^*$: $e_p(\boldsymbol{g}_i(D)) \leq 0$, $1 \leq i \leq b$, and $G(D)$ has the GPVP.

Proof: (i $\Rightarrow$ ii). Assume that $G(D)$ is canonical. Suppose that $e_p(\boldsymbol{g}_i(D)) \leq 0$ does not hold for some $p \in \mathcal{P}^*$ and some i. Then, by Lemma 2.54, $\boldsymbol{g}_i(D)$ is not canonical, and, hence, $G(D)$ is not canonical.

Suppose that $e_p([G(D)]_p) = 0$ does not hold for some $p \in \mathcal{P}^*$. Then, by Lemma 2.59, for any $p \in \mathcal{P}^*$ such that $e_p([G(D)]_p) = 0$ does not hold, we have

$$e_p(G(D)) > \sum_{i=1}^{b} e_p(\boldsymbol{g}_i(D)) \tag{2.375}$$

and for any $p \in \mathcal{P}^*$ such that $e_p([G(D)]_p) = 0$ holds, we have

$$e_p(G(D)) = \sum_{i=1}^{b} e_p(\boldsymbol{g}_i(D)) \tag{2.376}$$

Thus, by combining (2.375) and (2.376) we obtain

$$\begin{aligned}\operatorname{intdef} G(D) &= -\sum_{p\in\mathcal{P}^*} e_p(G(D))\deg p < -\sum_{p\in\mathcal{P}^*}\sum_{i=1}^{b} e_p(\boldsymbol{g}_i(D))\deg p \\ &= \sum_{i=1}^{b}\left(-\sum_{p\in\mathcal{P}^*} e_p(\boldsymbol{g}_i(D))\deg p\right) \\ &= \sum_{i=1}^{b}\operatorname{def}\boldsymbol{g}_i(D) = \operatorname{extdef} G(D) \end{aligned} \tag{2.377}$$

Hence, $G(D)$ is not canonical.

(ii $\Rightarrow$ iii). Follows from Lemma 2.59.

(iii $\Rightarrow$ iv). Follows from (2.312) and (2.362).

(iv $\Rightarrow$ i). By Lemma 2.55, the hypothesis means that $\operatorname{intdef} G(D)$ is the overall constraint length of $G(D)$. Let $G_c(D)$ be a canonical encoding matrix equivalent to $G(D)$. Then, from Theorem 2.58 it follows that $\operatorname{intdef} G_c(D) = \operatorname{intdef} G(D)$. By (i $\Rightarrow$ iv), $\operatorname{intdef} G_c(D)$ is the overall constraint length of $G_c(D)$. Thus, $\operatorname{intdef} G_c(D)$ is minimum over all equivalent generator matrices, and so is $\operatorname{intdef} G(D)$. Hence, $G(D)$ is canonical.

(ii $\Leftrightarrow$ v). $e_p([G(D)]_p) = 0$ means that there exists at least one $b \times b$ minor of $[G(D)]_p$ not divisible by p, which together with Corollary 2.57 completes the proof. ■

■ EXAMPLE 2.32

In Example 2.30 we showed that the rational encoding matrix

$$G(D) = \begin{pmatrix} 1 & \dfrac{D}{1+D} & \dfrac{1}{1+D} \\[2ex] \dfrac{D^2}{1+D+D^2} & \dfrac{1}{1+D+D^2} & 1 \end{pmatrix} \tag{2.378}$$

has the GPVP. Clearly, for all $p \in \mathcal{P}^*, e_p(\boldsymbol{g}_i(D)) \leq 0$, $i = 1, 2$. Therefore condition (v) in Theorem 2.60 is satisfied, and we conclude that $G(D)$ is canonical and, hence, minimal.

■ EXAMPLE 2.33

The rational encoding matrix

$$G(D) = \begin{pmatrix} 1 & 0 & \dfrac{1+D^2}{1+D+D^2} & \dfrac{D^2}{1+D+D^2} \\[2ex] 0 & 1 & \dfrac{D^2}{1+D+D^2} & \dfrac{1}{1+D+D^2} \end{pmatrix} \tag{2.379}$$

has a (trivial) right inverse,

$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \end{pmatrix}$$

which is polynomial in both D and D^{-1}. Hence, from Theorem 2.37 (iv) it follows that $G(D)$ is minimal.

In Example 2.31 we showed that $G(D)$ does not have the GPVP. Hence, from Theorem 2.60 we conclude that $G(D)$ is *not* canonical although it is minimal.

Corollary 2.61 Let $\mathcal{C}$ be a convolutional code. Then any canonical encoding matrix of $\mathcal{C}$ has $\operatorname{def}\mathcal{C}$ as its overall constraint length. Moreover, the number of memory elements in any encoder of $\mathcal{C}$ is $\geq \operatorname{def}\mathcal{C}$.

Proof: Let $G(D)$ be a canonical encoding matrix of $\mathcal{C}$. By Theorem 2.60 (iv), $\operatorname{intdef} G(D) = \operatorname{extdef} G(D)$. By (2.367) and (2.368) $\operatorname{def}\mathcal{C} = \sum_{i=1}^{b} \operatorname{def} \boldsymbol{g}_i(D)$ and from Lemma 2.55 it follows that $\sum_{i=1}^{b} \operatorname{def} \boldsymbol{g}_i(D)$ is the overall constraint length of $G(D)$. ∎

Among the rational generator matrices that encode a convolutional code, we have singled out the class of canonical encoding matrices, which can be realized by the least number of delay elements in controller canonical form among all equivalent generator matrices. Thus the position of canonical encoding matrices within the class of rational generator matrices corresponds to that of minimal-basic encoding matrices within the class of polynomial generator matrices. The set of canonical encoding matrices is a proper subset of the set of minimal rational encoding matrices. This is a generalization of the previous result that the set of minimal-basic encoding matrices is a proper subset of the set of minimal polynomial encoding matrices.

2.9 MINIMALITY VIA THE INVARIANT-FACTOR THEOREM*

In this section we will use the invariant-factor theorem with respect to both $\mathbb{F}_2[D]$ and $\mathbb{F}_2[D^{-1}]$ to derive a result on minimality of generator matrices. First, we state the invariant-factor decomposition of a rational matrix introduced in Section 2.2 in the following form, where for simplicity we assume that the rational matrix has full rank.

Theorem 2.62 (Invariant-Factor Theorem) Let $G(D)$ be a full-rank $b \times c$ rational matrix, where $b \leq c$. Then $G(D)$ may be written as

$$G(D) = A(D)\Gamma(D)B(D) \tag{2.380}$$

where $A(D)$ and $B(D)$ are, respectively, $b \times b$ and $c \times c$ matrices with unit determinants and where $\Gamma(D)$ is a diagonal matrix with diagonal elements $\gamma_i(D)$, $1 \leq i \leq b$,

called the *invariant factors* of $G(D)$ relative to the ring $\mathbb{F}_2[D]$. The invariant factors are uniquely determined by $G(D)$ as

$$\gamma_i(D) = \Delta_i(D)/\Delta_{i-1}(D) \tag{2.381}$$

where $\Delta_0(D) = 1$ by convention and

$$\Delta_i(D) = \prod_{p\in\mathcal{P}} p^{\min\{e_p(\det M_i(D))|M_i(D)\in\mathcal{M}_i\}} \tag{2.382}$$

where $\mathcal{M}_i$ is the set of $i \times i$ submatrices of $G(D)$, $1 \le i \le b$. Consequently,

$$\prod_{i=1}^{b} \gamma_i(D) = \Delta_b(D) \tag{2.383}$$

$$\sum_{i=1}^{b} e_p(\gamma_i(D)) = e_p(\Delta_b(D)), \;\; p \in \mathcal{P} \tag{2.384}$$

For all p in $\mathcal{P}$, the invariant factors satisfy the divisibility property

$$e_p(\gamma_i(D)) \le e_p(\gamma_{i+1}(D)), \;\; 1 \le i < b \tag{2.385}$$

It is easy to show that if $G(D)$ is regarded as a matrix over $\mathbb{F}_2(D^{-1})$, then the invariant factors $\widetilde{\gamma}_i(D^{-1})$ of $G(D)$ with respect to $\mathbb{F}_2[D^{-1}]$ have the same p-valuations as the invariant factors $\gamma_i(D)$ for all p in $\mathcal{P}$ except for D. Therefore, it makes sense to define the p-valuations of the invariant factors of $G(D)$ for all p in $\mathcal{P}^*$ and all i by

$$\gamma_{D,i} = e_D(\gamma_i(D)) \qquad \text{if } p = D \tag{2.386}$$

$$\gamma_{D^{-1},i} = e_{D^{-1}}(\widetilde{\gamma}_i(D^{-1})) \qquad \text{if } p = D^{-1} \tag{2.387}$$

$$\gamma_{p,i} = e_p(\gamma_i(D)) = e_p(\widetilde{\gamma}_i(D^{-1})) \qquad \text{otherwise} \tag{2.388}$$

If we define $\widetilde{\Delta}_0(D^{-1}) = 1$ and

$$\widetilde{\Delta}_i(D^{-1}) = \prod_{p\in\mathcal{P}^*\setminus\{D\}} p^{\min\{e_p(\det M_i(D^{-1}))|M_i(D^{-1})\in\mathcal{M}_i\}} \tag{2.389}$$

then

$$\widetilde{\gamma}_i(D^{-1}) = \widetilde{\Delta}_i(D^{-1})/\widetilde{\Delta}_{i-1}(D^{-1}) \tag{2.390}$$

To simplify the computation of these p-valuations for small generator matrices we define $\delta_{p,0} = 0$ for all p in $\mathcal{P}^*$ and

$$\delta_{p,i} = e_p(\Delta_i(D)), \;\; p \in \mathcal{P} \tag{2.391}$$

$$\delta_{D^{-1},i} = e_{D^{-1}}(\widetilde{\Delta}_i(D^{-1})) \tag{2.392}$$

and then we have for all p in $\mathcal{P}^*$ and $1 \le i \le b$

$$\gamma_{p,i} = \delta_{p,i} - \delta_{p,i-1} \tag{2.393}$$

which implies

$$\delta_{p,b} = \sum_{i=1}^{b} \gamma_{p,i} \tag{2.394}$$

Remark: We can now recognize that for $p \in \mathcal{P}$

$$\begin{aligned} e_p(G(D)) &= \min \{e_p(\det M_b(D)) \mid M_b(D) \in \mathcal{M}_b\} \\ &= e_p(\Delta_b(D)) = \delta_{p,b} \end{aligned} \tag{2.395}$$

and

$$\begin{aligned} e_{D^{-1}}(G(D)) &= \min \left\{e_{D^{-1}}(\det M_b(D^{-1})) \mid M_b(D^{-1}) \in \mathcal{M}_b\right\} \\ &= e_{D^{-1}}(\widetilde{\Delta}_b(D^{-1})) = \delta_{D^{-1},b} \end{aligned} \tag{2.396}$$

Thus, the internal defect can be computed directly from the p-valuations of the invariant factors of $G(D)$ by

$$\text{intdef}\, G(D) = -\sum_{p \in \mathcal{P}^*} \delta_{p,b} \deg p = -\sum_{p \in \mathcal{P}^*} \left(\sum_{i=1}^{b} \gamma_{p,i} \right) \deg p \tag{2.397}$$

Now we have the following theorem:

Theorem 2.63 Let $G(D)$ be a $b \times c$ rational generator matrix. Then $G(D)$ is minimal if and only if $\gamma_{p,b} \leq 0$ for all p in $\mathcal{P}^*$.

Proof: If $G(D)$ is minimal, then Theorem 2.37 (v) implies that a polynomial output sequence $\boldsymbol{u}(D)G(D)$ must be generated by a polynomial input sequence $\boldsymbol{u}(D)$, and an output sequence $\boldsymbol{u}(D^{-1})G(D^{-1})$ that is polynomial in D^{-1} must be generated by an input sequence $\boldsymbol{u}(D^{-1})$ that is polynomial in D^{-1}. Let $G(D) = A(D)\Gamma(D)B(D)$ be an invariant-factor decomposition of $G(D)$; then $G^{-1}(D) = B^{-1}(D)\Gamma^{-1}(D)A^{-1}(D)$ is a right inverse of $G(D)$, where $A^{-1}(D)$ and $B^{-1}(D)$ are polynomial since $A(D)$ and $B(D)$ have unit determinants. Suppose that $\gamma_{p,b} > 0$ for some p in $\mathcal{P}$. Then $\boldsymbol{u}(D) = (0\,0 \ldots 1/p)A^{-1}(D)$ is nonpolynomial (since $\boldsymbol{u}(D)A(D)$ is nonpolynomial), but $\boldsymbol{u}(D)G(D)$ is polynomial and we have a contradiction. Using the invariant-factor theorem with respect to $\mathbb{F}_2[D^{-1}]$, we can show a similar contradiction if $\gamma_{D^{-1},b} > 0$.

Conversely, assume that $\gamma_{p,b} \leq 0$ for p in $\mathcal{P}$. Then $\gamma_{p,i} \leq 0$ since, by the invariant-factor theorem, $\gamma_{p,i} \leq \gamma_{p,b}$ for $i \leq b$. Hence, if $\gamma_{p,b} \leq 0$ for all p in $\mathcal{P}$, $\Gamma^{-1}(D)$ is polynomial, and then $G^{-1}(D) = B^{-1}(D)\Gamma^{-1}(D)A^{-1}(D)$ is the desired polynomial right inverse of $G(D)$. Similarly, if also $\gamma_{D^{-1},b} \leq 0$, then $\Gamma^{-1}(D^{-1})$ is polynomial in D^{-1}, and the invariant-factor theorem with respect to $\mathbb{F}_2[D^{-1}]$ yields an $\mathbb{F}_2[D^{-1}]$-inverse of $G(D)$. The minimality of $G(D)$ follows immediately from Theorem 2.37. ∎

Remark: The condition that $\gamma_{p,b} \leq 0$ for $p = D$ is equivalent to the condition that no nontrivial zero-output (encoder) state transition starting from the zero state occurs in the minimal state realization of $G(D)$ when it is used as a state realization of $\mathcal{C}$. Similarly, the condition $\gamma_{p,b} \leq 0$ for $p = D^{-1}$ is equivalent to the condition that there is no nontrivial zero-output state transition ending in the zero state. Finally, the condition $\gamma_{p,b} \leq 0$ for all other $p \in \mathcal{P}^*$ is equivalent to the condition that there is no nontrivial infinite zero-output state path (the "noncatastrophic" condition).

■ **EXAMPLE 2.34**

Let

$$G(D) = \begin{pmatrix} 1 & 0 \\ D & 1 \end{pmatrix} \tag{2.398}$$

be an encoding matrix over $\mathbb{F}_2$. Then the 1×1 minors of $G(D)$ are $\{1, 0, D, 1\}$, and the 2×2 minor is the determinant $\det G(D) = 1$. The greatest common polynomial divisor of the 1×1 minors is 1, so $\delta_{p,1} = 0$ for all $p \in \mathcal{P}$. However, the maximum degree of the 1×1 minors is 1, so it follows that $\delta_{D^{-1},1} = -1$. Since $\det G(D) = 1$, we have $\delta_{p,2} = 0$ for all $p \in \mathcal{P}^*$. Therefore,

$$\gamma_{p,2} = \delta_{p,2} - \delta_{p,1} = \begin{cases} 0, & p \in \mathcal{P} \\ 1, & p = D^{-1} \end{cases} \tag{2.399}$$

so $G(D)$ is not minimal.

Indeed, $G(D)$ does have an $\mathbb{F}_2[D]$-inverse, namely, its unique inverse

$$G^{-1}(D) = \begin{pmatrix} 1 & 0 \\ D & 1 \end{pmatrix} \tag{2.400}$$

but $G^{-1}(D)$ is not an $\mathbb{F}_2[D^{-1}]$-matrix, so $G(D)$ has no $\mathbb{F}_2[D^{-1}]$-inverse.

■ **EXAMPLE 2.35**

Let

$$G(D) = \begin{pmatrix} 1 & \dfrac{D}{1+D} & \dfrac{1}{1+D} \\ \dfrac{D^2}{1+D+D^2} & \dfrac{1}{1+D+D^2} & 1 \end{pmatrix} \tag{2.401}$$

be an encoding matrix over $\mathbb{F}_2$. The greatest common divisor of the 1×1 minors of $G(D)$ is $\Delta_1(D) = 1/(1+D^3)$ and that of the 2×2 minors is $\Delta_2(D) = 1/(1+D^3)$. Therefore, $\gamma_2(D) = \Delta_2(D)/\Delta_1(D) = 1$.

$G(D)$ can also be written as a rational matrix in D^{-1}, viz.,

$$G(D) = \begin{pmatrix} 1 & \dfrac{1}{1+D^{-1}} & \dfrac{D^{-1}}{1+D^{-1}} \\ \dfrac{1}{1+D^{-1}+D^{-2}} & \dfrac{D^{-2}}{1+D^{-1}+D^{-2}} & 1 \end{pmatrix} \tag{2.402}$$

We have, similarly, $\widetilde{\Delta}_1(D^{-1}) = 1/(1 + D^{-3})$ and $\widetilde{\Delta}_2(D^{-1}) = 1/(1 + D^{-3})$. Therefore, we also have $\gamma_2(D^{-1}) = 1$. Thus, $\gamma_{p,2} = 0$ for all $p \in \mathcal{P}^*$. By Theorem 2.63, $G(D)$ is minimal.

Corollary 2.64 Let $G(D)$ be a minimal $b \times c$ rational generator matrix with rows $\boldsymbol{g}_1(D), \boldsymbol{g}_2(D), \ldots, \boldsymbol{g}_b(D)$. Then

$$e_p(\boldsymbol{g}_i(D)) \leq 0, \quad 1 \leq i \leq b, \text{ all } p \in \mathcal{P}^* \tag{2.403}$$

Proof: Suppose that $e_p(\boldsymbol{g}_j(D)) > 0$ for some j with $1 \leq j \leq b$ and p in $\mathcal{P}$. Then $\boldsymbol{u}(D) = (0 \ldots 0\, 1/p\, 0 \ldots 0)$, where $1/p$ is in the jth position, is nonpolynomial, but $\boldsymbol{u}(D)G(D)$ is polynomial. Similarly, if $e_{D^{-1}}(\boldsymbol{g}_j(D)) > 0$, then $\boldsymbol{u}(D^{-1}) = (0 \ldots 0\, 1/D^{-1}\, 0 \ldots 0)$ is nonpolynomial in D^{-1}, but $\boldsymbol{u}(D^{-1})G(D^{-1})$ is polynomial in D^{-1}. Hence, Theorem 2.37 (v) implies that $G(D)$ is nonminimal, which is a contradiction. ∎

It is easily seen that the converse of Corollary 2.64 does not hold; for example, the basic encoding matrix (cf. Example 2.22)

$$G(D) = \begin{pmatrix} 1+D & D \\ D & 1+D \end{pmatrix} \tag{2.404}$$

is not minimal, although its rows satisfy (2.403).

By combining Theorems 2.48 and 2.60 (v) and Corollary 2.64, we have the following:

Corollary 2.65 Let $G(D)$ be a $b \times c$ rational generator matrix with rows $\boldsymbol{g}_1(D)$, $\boldsymbol{g}_2(D), \ldots, \boldsymbol{g}_b(D)$. Then $G(D)$ is canonical if and only if it is minimal and has the GPVP.

The encoding matrix given in Example 2.33 is minimal but does not have the GPVP. Hence, it is not canonical. But the encoding matrix given in Example 2.35 is both minimal and has the GPVP (cf. Example 2.30) and, hence, canonical.

We have shown that canonicality is the intersection of two independent properties: minimality and the global predictable valuation property. A minimal encoding matrix need not have the GPVP, and a matrix with the GPVP need not be minimal.

2.10 SYNDROME FORMERS AND DUAL ENCODERS

We will now use the invariant-factor decomposition of a rational convolutional generator matrix to construct generator matrices for a dual code.

Let $G(D) = A(D)\Gamma(D)B(D)$. In Section 2.4 we have shown that the first b rows of the $c \times c$ polynomial matrix $B(D)$ can be taken as a basic encoding matrix $G'(D)$ equivalent to $G(D)$. A polynomial right inverse $G'^{-1}(D)$ consists of the first b columns of $B^{-1}(D)$. Let $H^{\mathrm{T}}(D)$, where T denotes transpose, be the last $c - b$

columns of $B^{-1}(D)$. Then the last $c-b$ rows of $B(D)$, which is a $(c-b)\times c$ matrix, is a left inverse of $H^{\mathrm{T}}(D)$. Thus, the transpose of the matrix formed by the last $c-b$ rows of $B(D)$ is a right inverse of $H(D)$ and can be denoted $H^{-1}(D)$. Hence, the last $c-b$ rows of $B(D)$ can be denoted $(H^{-1}(D))^{\mathrm{T}}$. Summarizing, we have

$$B(D) = \begin{pmatrix} G'(D) \\ (H^{-1}(D))^{\mathrm{T}} \end{pmatrix} \tag{2.405}$$

and

$$B^{-1}(D) = (G'^{-1}(D)\, H^{\mathrm{T}}(D)) \tag{2.406}$$

The matrix $H^{\mathrm{T}}(D)$ has full rank and is both realizable and delayfree.

Let $\boldsymbol{g}'(D)$ be a row among the *first* b in $B(D)$. Since $H^{\mathrm{T}}(D)$ consists of the *last* $c-b$ columns in $B^{-1}(D)$, it follows that

$$\boldsymbol{g}'(D)H^{\mathrm{T}}(D) = \mathbf{0} \tag{2.407}$$

Then for each codeword $\boldsymbol{v}(D) = \boldsymbol{u}(D)G'(D)$ we have

$$\boldsymbol{v}(D)H^{\mathrm{T}}(D) = \boldsymbol{u}(D)G'(D)H^{\mathrm{T}}(D) = \mathbf{0} \tag{2.408}$$

Conversely, suppose that $\boldsymbol{v}(D)H^{\mathrm{T}}(D) = \mathbf{0}$. Since rank $H^{\mathrm{T}}(D) = c-b$, it follows from (2.407) that $\boldsymbol{v}(D)$ is a linear combination of the first b rows of $B(D)$ with Laurent series as coefficients, say $\boldsymbol{v}(D) = \boldsymbol{u}(D)G'(D)$. Thus, $\boldsymbol{v}(D)$ is a codeword. It follows that the output of the c-input, $(c-b)$-output linear sequential circuit, whose transfer function is the polynomial matrix $H^{\mathrm{T}}(D)$, is the allzero sequence if and only if the input sequence is a codeword of the code $\mathcal{C}$ encoded by $G(D)$. Thus, $\mathcal{C}$ could equally well be defined as the sequences $\boldsymbol{v}(D)$ such that

$$\boldsymbol{v}(D)H^{\mathrm{T}}(D) = \mathbf{0} \tag{2.409}$$

or the null space of $H^{\mathrm{T}}(D)$.

We call the matrix $H(D)$ the *parity-check matrix* and the matrix $H^{\mathrm{T}}(D)$ the *syndrome former* corresponding to $G(D) = A(D)\Gamma(D)B(D)$. In general, any $c\times(c-b)$ realizable, delayfree transfer function matrix $S^{\mathrm{T}}(D)$ of rank $c-b$ is called a syndrome former of $\mathcal{C}$ if

$$G(D)S^{\mathrm{T}}(D) = \mathbf{0} \tag{2.410}$$

The syndrome former $H^{\mathrm{T}}(D)$ can be expanded as

$$H^{\mathrm{T}}(D) = H_0^{\mathrm{T}} + H_1^{\mathrm{T}}D + \cdots + H_{m_{\mathrm{s}}}^{\mathrm{T}}D^{m_{\mathrm{s}}} \tag{2.411}$$

where H_i^{T}, $0 \le i \le m_{\mathrm{s}}$, is a $c\times(c-b)$ matrix and m_{s} is the memory of the syndrome former. In general, the memory of a syndrome former is not equal to the memory of the generator matrix $G(D)$.

Using (2.411), we can write equation (2.409) as

$$\boldsymbol{v}_t H_0^{\mathrm{T}} + \boldsymbol{v}_{t-1}H_1^{\mathrm{T}} + \cdots + \boldsymbol{v}_{t-m_{\mathrm{s}}}H_{m_{\mathrm{s}}}^{\mathrm{T}} = \mathbf{0}, \quad t \in \mathbb{Z} \tag{2.412}$$

For causal codewords $\boldsymbol{v} = \boldsymbol{v}_0\, \boldsymbol{v}_1\, \boldsymbol{v}_2\, \ldots$ we have equivalently

$$\boldsymbol{v}\boldsymbol{H}^{\mathrm{T}} = \boldsymbol{0} \tag{2.413}$$

where

$$\boldsymbol{H}^{\mathrm{T}} = \begin{pmatrix} H_0^{\mathrm{T}} & H_1^{\mathrm{T}} & \ldots & H_{m_{\mathrm{s}}}^{\mathrm{T}} & & \\ & H_0^{\mathrm{T}} & H_1^{\mathrm{T}} & \ldots & H_{m_{\mathrm{s}}}^{\mathrm{T}} & \\ & & \ddots & \ddots & & \ddots \end{pmatrix} \tag{2.414}$$

is a semi-infinite syndrome former matrix corresponding to the semi-infinite generator matrix $\boldsymbol{G}$ given in (1.91). Clearly, we have

$$\boldsymbol{G}\boldsymbol{H}^{\mathrm{T}} = \boldsymbol{0} \tag{2.415}$$

■ **EXAMPLE 2.36**

For the basic encoding matrix $G(D)$ whose encoder is illustrated in Fig. 2.9 we have (Example 2.4)

$$B^{-1}(D) = \begin{pmatrix} 0 & 1+D+D^2 & 1+D+D^2+D^3 \\ 0 & D+D^2 & 1+D^2+D^3 \\ 1 & 1+D^2 & 1+D+D^3 \end{pmatrix} \tag{2.416}$$

Hence, we have the syndrome former

$$H^{\mathrm{T}}(D) = \begin{pmatrix} 1+D+D^2+D^3 \\ 1+D^2+D^3 \\ 1+D+D^3 \end{pmatrix} \tag{2.417}$$

whose controller canonical form is illustrated in Fig. 2.17. Its observer canonical form is much simpler (Fig. 2.18).

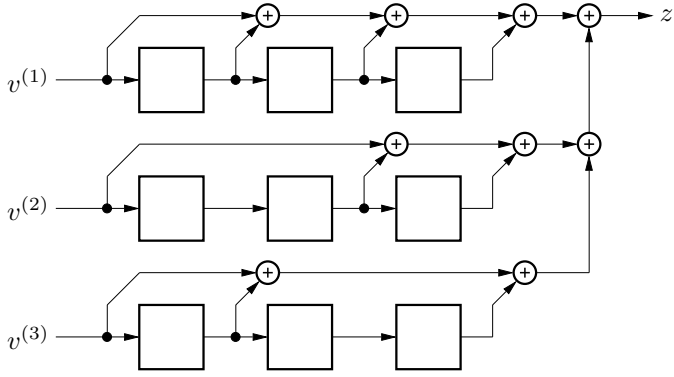


Figure 2.17 The controller canonical form of the syndrome former in Example 2.36.

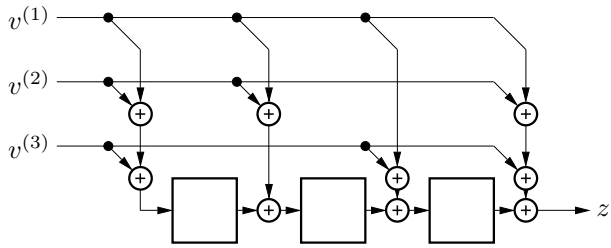


Figure 2.18 The observer canonical form of the syndrome former in Example 2.36.

The corresponding semi-infinite syndrome former matrix is

$$\boldsymbol{H}^{\mathrm{T}} = \begin{pmatrix} 1 & 1 & 1 & 1 & & \\ 1 & 0 & 1 & 1 & & \\ 1 & 1 & 0 & 1 & & \\ & & & & & \\ & 1 & 1 & 1 & 1 & \\ & 1 & 0 & 1 & 1 & \\ & 1 & 1 & 0 & 1 & \\ & & & & & \\ & & \ddots & \ddots & \ddots & \ddots \end{pmatrix} \tag{2.418}$$

We notice that in the previous example the observer canonical form of the syndrome former requires exactly the same number of memory elements as the controller canonical form of the corresponding minimal-basic encoding matrix $G(D)$. Before we prove that this holds in general, we shall consider $H(D)$ as a generator matrix.

Definition The *dual code* $\mathcal{C}^{\perp}$ to a convolutional code $\mathcal{C}$ is the set of all c-tuples of sequences $\boldsymbol{v}^{\perp}$ such that the inner product

$$(\boldsymbol{v}, \boldsymbol{v}^{\perp}) \stackrel{\text{def}}{=} \boldsymbol{v}(\boldsymbol{v}^{\perp})^{\mathrm{T}} \tag{2.419}$$

is zero, that is, $\boldsymbol{v}$ and $\boldsymbol{v}^{\perp}$ are *orthogonal*, for all finite $\boldsymbol{v}$ in $\mathcal{C}$.

The dual code $\mathcal{C}^{\perp}$ to a rate $R = b/c$ convolutional code is a rate $R = (c-b)/c$ convolutional code.

Theorem 2.66 Let the rate $R = b/c$ convolutional code $\mathcal{C}$ be generated by the semi-infinite generator matrix $\boldsymbol{G}$ and the rate $R = (c-b)/c$ dual code $\mathcal{C}^{\perp}$ be generated by the semi-infinite generator matrix $\boldsymbol{G}^{\perp}$, where $\boldsymbol{G}$ is given in (1.91) and

$$\boldsymbol{G}^{\perp} = \begin{pmatrix} G_0^{\perp} & G_1^{\perp} & \dots & G_{m^{\perp}}^{\perp} & & \\ & G_0^{\perp} & G_1^{\perp} & \dots & G_{m^{\perp}}^{\perp} & \\ & & \ddots & \ddots & & \ddots \end{pmatrix} \tag{2.420}$$

Then

$$\boldsymbol{G}(\boldsymbol{G}^{\perp})^{\mathrm{T}} = \boldsymbol{0} \tag{2.421}$$

Proof: Let $\boldsymbol{v} = \boldsymbol{u}\boldsymbol{G}$ and $\boldsymbol{v}^\perp = \boldsymbol{u}^\perp\boldsymbol{G}^\perp$, where $\boldsymbol{v}$ and $\boldsymbol{v}^\perp$ are orthogonal. Then we have

$$\boldsymbol{v}(\boldsymbol{v}^\perp)^{\mathrm{T}} = \boldsymbol{u}\boldsymbol{G}(\boldsymbol{u}^\perp\boldsymbol{G}^\perp)^{\mathrm{T}} = \boldsymbol{u}\boldsymbol{G}(\boldsymbol{G}^\perp)^{\mathrm{T}}(\boldsymbol{u}^\perp)^{\mathrm{T}} = \boldsymbol{0} \tag{2.422}$$

and (2.421) follows. ∎

The concept of a dual convolutional code is a straightforward generalization of the corresponding concept for block codes (see Section 1.2). For convolutional codes, we also have a closely related concept:

Definition The *convolutional dual code* $\mathcal{C}_\perp$ to a convolutional code $\mathcal{C}$, which is encoded by the rate $R = b/c$ generator matrix $G(D)$, is the set of all codewords encoded by any rate $R = (c-b)/c$ generator matrix $G_\perp(D)$ such that

$$G(D)G_\perp^{\mathrm{T}}(D) = \boldsymbol{0} \tag{2.423}$$

Consider a rate $R = b/c$ convolutional code $\mathcal{C}$ encoded by the polynomial generator matrix

$$G(D) = G_0 + G_1 D + \cdots + G_m D^m \tag{2.424}$$

Let $\widetilde{G}^\perp(D)$ denote the rate $R = (c-b)/c$ polynomial generator matrix

$$\widetilde{G}^\perp(D) = G^\perp_{m^\perp} + G^\perp_{m^\perp-1} D + \cdots + G_0 D^{m^\perp} \tag{2.425}$$

which is the reciprocal of the generator matrix

$$G^\perp(D) = G_0^\perp + G_1^\perp D + \cdots + G^\perp_{m^\perp} D^{m^\perp} \tag{2.426}$$

for the dual code $\mathcal{C}^\perp$. Then we have

$$\begin{aligned} G(D)(\widetilde{G}^\perp(D))^{\mathrm{T}} &= G_0(G^\perp_{m^\perp})^{\mathrm{T}} + (G_0(G^\perp_{m^\perp-1})^{\mathrm{T}} + G_1(G^\perp_{m^\perp})^{\mathrm{T}})D \\ &\quad + \cdots + G_m(G_0^\perp)^{\mathrm{T}} D^{m+m^\perp} \\ &= \left(\sum_{j=-m}^{m^\perp} \left(\sum_{i=0}^{m} G_i (G^\perp_{i+j})^{\mathrm{T}} \right) \right) D^{m+j} = \boldsymbol{0} \end{aligned} \tag{2.427}$$

where the last equality follows from (2.421).

Let $\widetilde{\mathcal{C}}^\perp$ be the *reversal* of the dual code $\mathcal{C}^\perp$, that is, the rate $R = (c-b)/c$ convolutional code encoded by $\widetilde{G}^\perp(D)$. Then, we have the following theorem:

Theorem 2.67 The convolutional dual to the code encoded by the generator matrix $G(D)$ is the reversal of the convolutional code dual to the code encoded by $G(D)$, that is, if $\mathcal{C}$ is encoded by $G(D)$, then

$$\mathcal{C}_\perp = \widetilde{\mathcal{C}}^\perp \tag{2.428}$$

Remark: Often the convolutional dual code $\mathcal{C}_\perp$ is simply called the dual of $\mathcal{C}$, which could cause confusion since it is in general not equal to $\mathcal{C}^\perp$.

It follows from (2.408) and (2.423) that the transpose of the syndrome former for the code $\mathcal{C}$ can be used as a generator matrix for the convolutional dual code $\mathcal{C}_\perp$; that is, we may take

$$G_\perp(D) = H(D) \tag{2.429}$$

and, equivalently, the transpose of the reciprocal of the syndrome former for the code $\mathcal{C}$ can be used as a generator matrix for the dual code $\mathcal{C}^\perp$; that is, we may take

$$G^\perp(D) = \widetilde{H}(D) \tag{2.430}$$

■ **EXAMPLE 2.37**

Let the rate $R = 2/3$ encoding matrix $G(D)$ in Example 2.4 encode the code $\mathcal{C}$. Two realizations of its syndrome former $H^{\mathrm{T}}(D)$ are shown in the previous example. The convolutional dual code is encoded by the rate $R = 1/3$ encoding matrix

$$H(D) = (1 + D + D^2 + D^3 \quad 1 + D^2 + D^3 \quad 1 + D + D^3) \tag{2.431}$$

whose controller canonical form is illustrated in Fig. 2.19.

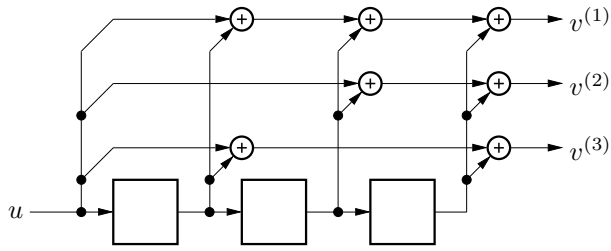


Figure 2.19 The controller canonical form of the encoding matrix for the convolutional dual code in Example 2.37.

The following lemma leads to an important theorem relating a code to its convolutional dual code.

Lemma 2.68 Let $i_1, i_2, \ldots, i_c$ be a permutation of $1, 2, \ldots, c$. Then the $b \times b$ subdeterminant of the basic encoding matrix $G'(D)$ formed by the $i_1, i_2, \ldots, i_b$ columns is equal to the $(c - b) \times (c - b)$ subdeterminant of the syndrome former $H^{\mathrm{T}}(D)$ formed by the $i_{b+1}, i_{b+2}, \ldots, i_c$ rows.

Proof: It is sufficient to consider the case that $i_1 = 1, i_2 = 2, \ldots, i_c = c$. Recall that $G'(D)$ is the first b rows of $B(D)$ and that $H^{\mathrm{T}}(D)$ is the last $(c - b)$ columns of $B^{-1}(D)$. Write

$$B(D) = \begin{pmatrix} B_{11}(D) & B_{12}(D) \\ B_{21}(D) & B_{22}(D) \end{pmatrix} \tag{2.432}$$

and

$$B^{-1}(D) = \begin{pmatrix} B'_{11}(D) & B'_{12}(D) \\ B'_{21}(D) & B'_{22}(D) \end{pmatrix} \tag{2.433}$$

where

$$G'(D) = (B_{11}(D)\ B_{12}(D)) \tag{2.434}$$

and

$$H^{\mathrm{T}}(D) = \begin{pmatrix} B'_{12}(D) \\ B'_{22}(D) \end{pmatrix} \tag{2.435}$$

where $B_{11}(D)$ is a $b \times b$ matrix and $B_{22}(D)$ is a $(c-b) \times (c-b)$ matrix. Consider the matrix product

$$\begin{pmatrix} B_{11}(D) & B_{12}(D) \\ \mathbf{0} & I_{c-b} \end{pmatrix} \begin{pmatrix} B'_{11}(D) & B'_{12}(D) \\ B'_{21}(D) & B'_{22}(D) \end{pmatrix} = \begin{pmatrix} I_b & \mathbf{0} \\ B'_{21}(D) & B'_{22}(D) \end{pmatrix} \tag{2.436}$$

Taking the determinants, we have

$$\det(B_{11}(D)) \det(B^{-1}(D)) = \det(B'_{22}(D)) \tag{2.437}$$

Since $\det(B^{-1}(D)) = 1$, we have now shown that the leftmost subdeterminant of $G'(D)$ is equal to the lower subdeterminant of $H^{\mathrm{T}}(D)$. ∎

Theorem 2.69 If $G(D)$ is a minimal-basic encoding matrix with maximum degree μ among its $b \times b$ subdeterminants, then the convolutional dual code has a minimal-basic encoding matrix $H(D)$ with overall constraint length μ.

Proof: Follows directly from Theorems 2.20 and 2.22, the Corollary 2.23, and Lemma 2.68. ∎

Since $\mu = \nu$ for a minimal-basic encoding matrix we have the next corollary.

Corollary 2.70 If $G(D)$ is a minimal-basic encoding matrix for a convolutional code $\mathcal{C}$ and $H(D)$ is a minimal-basic encoding matrix for the convolutional dual code $\mathcal{C}_\perp$, then

$$\#\,\{\text{abstract states of } G(D)\,\} = \#\,\{\text{abstract states of } H(D)\,\} \tag{2.438}$$

If we connect the encoder outputs directly to the syndrome former input, then the output of the syndrome former will be the allzero sequence. From this it follows that a close connection exists between the abstract states of a generator matrix and the abstract states of its syndrome former:

Theorem 2.71 Let the output of the encoder with generator matrix $G(D)$ drive its syndrome former $H^{\mathrm{T}}(D)$. Then, whenever the abstract state of $G(D)$ is $\boldsymbol{s}(D)$, the abstract state of $H^{\mathrm{T}}(D)$ will be $\boldsymbol{s}_{\mathrm{s}}(D) = \boldsymbol{s}(D)H^{\mathrm{T}}(D)$.

Proof: Let $\boldsymbol{u}(D)$ be the input of $G(D)$ associated with the abstract state $\boldsymbol{s}(D)$, that is, $\boldsymbol{s}(D) = \boldsymbol{u}(D)PG(D)Q$, and let $\boldsymbol{v}(D)$ be the output when we truncate the input at time zero, that is, $\boldsymbol{v}(D) = \boldsymbol{u}(D)PG(D)$. Then it follows that $\boldsymbol{s}(D) = \boldsymbol{v}(D)Q$. Since $\boldsymbol{v}(D)$ is a codeword, we have (2.409) $\boldsymbol{v}(D)H^{\mathrm{T}}(D) = \boldsymbol{0}$. Using $P + Q = 1$ we get

$$\begin{aligned}\boldsymbol{0} = \boldsymbol{v}(D)H^{\mathrm{T}}(D) &= \boldsymbol{v}(D)H^{\mathrm{T}}(D)Q = \boldsymbol{v}(D)(P+Q)H^{\mathrm{T}}(D)Q \\ &= \boldsymbol{v}(D)PH^{\mathrm{T}}(D)Q + \boldsymbol{v}(D)QH^{\mathrm{T}}(D)Q \\ &= \boldsymbol{v}(D)PH^{\mathrm{T}}(D)Q + \boldsymbol{s}(D)H^{\mathrm{T}}(D)Q \\ &= \boldsymbol{s}_{\mathrm{s}}(D) + \boldsymbol{s}(D)H^{\mathrm{T}}(D) \end{aligned} \tag{2.439}$$

where $\boldsymbol{s}_{\mathrm{s}}(D) = \boldsymbol{v}(D)PH^{\mathrm{T}}(D)Q$ is the abstract state of the syndrome former corresponding to the input $\boldsymbol{v}(D)$. ∎

Corollary 2.72 Assume that both the encoding matrix $G(D)$ of a convolutional code and the encoding matrix $H(D)$ of its convolutional dual code are minimal-basic. Then the abstract state spaces of the encoding matrix $G(D)$ and its syndrome former $H^{\mathrm{T}}(D)$ are isomorphic under the map

$$\boldsymbol{s}(D) \mapsto \boldsymbol{s}_{\mathrm{s}}(D) = \boldsymbol{s}(D)H^{\mathrm{T}}(D) \tag{2.440}$$

Proof: Following the notation of Theorem 2.71, it is clear that

$$\boldsymbol{s}(D) \mapsto \boldsymbol{s}_{\mathrm{s}}(D) = \boldsymbol{s}(D)H^{\mathrm{T}}(D) \tag{2.441}$$

is a well-defined linear map from the abstract state space of $G(D)$ to that of $H^{\mathrm{T}}(D)$. By Lemma 2.32, the map is injective. Furthermore, we have

$$\begin{aligned}&\#\,\{\text{abstract states of } G(D)\} = \#\,\{\text{abstract states of } H(D)\} \\ &\quad = \#\,\{\text{encoder states of the controller canonical form of } H(D)\} \\ &\quad = \#\,\{\text{states of the observer canonical form of } H^{\mathrm{T}}(D)\} \\ &\quad = \#\,\{\text{abstract states of } H^{\mathrm{T}}(D)\} \end{aligned} \tag{2.442}$$

Therefore, the map is also surjective. Hence, we have an isomorphism. ∎

Suppose that the codeword $\boldsymbol{v}(D)$, where $\boldsymbol{v}(D) = \boldsymbol{u}(D)G(D)$, is transmitted over a noisy additive, memoryless channel. Let $\boldsymbol{r}(D)$ be the received sequence. Then

$$\boldsymbol{r}(D) = \boldsymbol{v}(D) + \boldsymbol{e}(D) \tag{2.443}$$

where $\boldsymbol{e}(D)$ is the error sequence. If we pass the received sequence through the syndrome former $H^{\mathrm{T}}(D)$, we obtain the *syndrome*

$$\boldsymbol{z}(D) = \boldsymbol{r}(D)H^{\mathrm{T}}(D) = (\boldsymbol{v}(D) + \boldsymbol{e}(D))H^{\mathrm{T}}(D) = \boldsymbol{e}(D)H^{\mathrm{T}}(D) \tag{2.444}$$

We notice that the syndrome is $\boldsymbol{0}$ if and only if the error sequence is a codeword. Furthermore, the syndrome is independent of the transmitted codeword—it depends only on the error sequence. If we use a syndrome former, then the decoding rule can be simply a map from the syndromes to the apparent errors $\widehat{\boldsymbol{e}}(D)$.

2.11 SYSTEMATIC CONVOLUTIONAL ENCODERS

Systematic convolutional generator matrices are in general simpler to implement than general generator matrices. They have trivial right inverses, but unless we use rational generator matrices (i.e., allow feedback in the encoder), they are in general less powerful when used together with maximum-likelihood decoding.

Since a systematic generator matrix has a submatrix that is a $b \times b$ identity matrix, we immediately have the following:

Theorem 2.73 A systematic generator matrix is a systematic encoding matrix.

■ **EXAMPLE 2.38**

Consider the rate $R = 2/3$ systematic convolutional encoder with the basic encoding matrix

$$G(D) = \begin{pmatrix} 1 & 0 & 1+D^2 \\ 0 & 1 & 1+D+D^2 \end{pmatrix} \tag{2.445}$$

In Figs. 2.20 and 2.21, we show the controller canonical form and the observer canonical form, respectively.

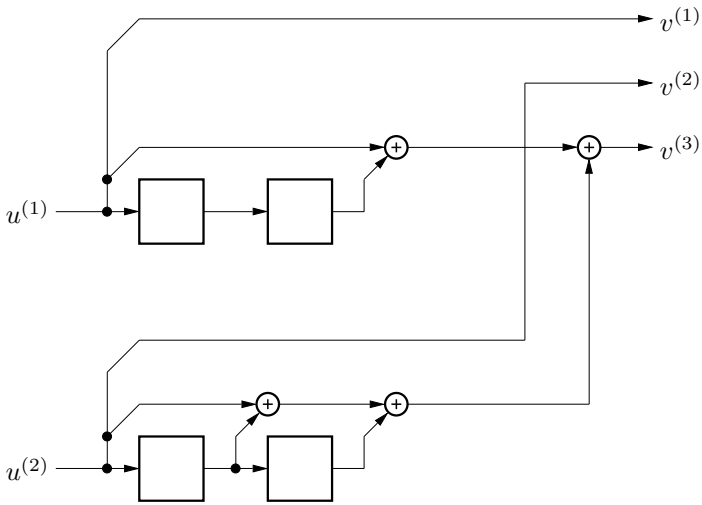


Figure 2.20 The controller canonical form of the systematic encoder in Example 2.38.

From Theorem 2.17 and equation (2.40) it follows that every basic encoding matrix has the greatest common divisor of all $b \times b$ minors equal to 1. Thus, it must have some $b \times b$ submatrix whose determinant is a delayfree polynomial, since otherwise all subdeterminants would be divisible by D. Premultiplication by the inverse of such a submatrix yields an equivalent systematic encoding matrix, possibly rational. Thus, we have the following:

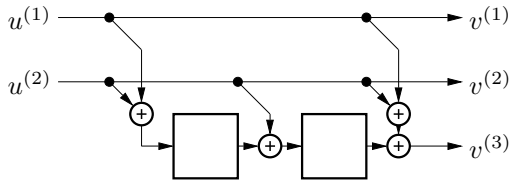


Figure 2.21 The observer canonical form of the systematic encoder in Example 2.38.

Theorem 2.74 Every convolutional generator matrix is equivalent to a systematic rational encoding matrix.

Remark: If the determinant of the *leftmost* $b \times b$ submatrix of $G(D)$ is *not* a delayfree polynomial, then we can always, by permuting the columns of $G(D)$, find a weakly equivalent generator matrix $G'(D)$ whose leftmost $b \times b$ submatrix has a determinant which is a delayfree polynomial, where $G'(D)$ encodes an equivalent code. Hence, without loss of generality we can write a systematic encoding matrix $G(D) = (I_b \, R(D))$.

■ **EXAMPLE 2.39**

Consider the rate $R = 2/3$ nonsystematic convolutional encoder illustrated in Fig. 2.9. It has the minimal-basic encoding matrix

$$G(D) = \begin{pmatrix} 1+D & D & 1 \\ D^2 & 1 & 1+D+D^2 \end{pmatrix} \tag{2.446}$$

with $\mu = \nu = 3$. Let $T(D)$ be the matrix consisting of the first two columns of $G(D)$:

$$T(D) = \begin{pmatrix} 1+D & D \\ D^2 & 1 \end{pmatrix} \tag{2.447}$$

We have $\det(T(D)) = 1 + D + D^3$, and

$$T^{-1}(D) = \frac{1}{1+D+D^3} \begin{pmatrix} 1 & D \\ D^2 & 1+D \end{pmatrix} \tag{2.448}$$

Multiplying $G(D)$ by $T^{-1}(D)$ yields a systematic encoding matrix $G_{\text{sys}}(D)$ equivalent to $G(D)$:

$$\begin{aligned} G_{\text{sys}}(D) &= T^{-1}(D)G(D) \\ &= \frac{1}{1+D+D^3} \begin{pmatrix} 1 & D \\ D^2 & 1+D \end{pmatrix} \begin{pmatrix} 1+D & D & 1 \\ D^2 & 1 & 1+D+D^2 \end{pmatrix} \\ &= \begin{pmatrix} 1 & 0 & \dfrac{1+D+D^2+D^3}{1+D+D^3} \\ 0 & 1 & \dfrac{1+D^2+D^3}{1+D+D^3} \end{pmatrix} \end{aligned} \tag{2.449}$$

Its realization requires a linear sequential circuit with feedback and $\mu = 3$ memory elements as shown in Fig. 2.22.

The systematic encoding matrix in the previous example was realized with the same number of memory elements as the equivalent minimal-basic encoding matrix (Example 2.17). Hence, it is a minimal encoding matrix.

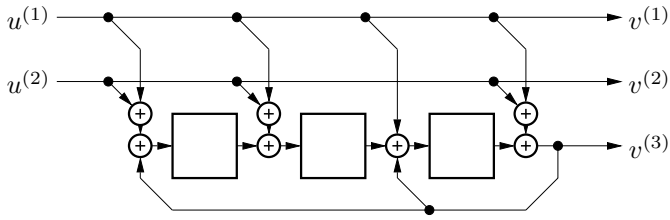


Figure 2.22 The observer canonical form of the systematic encoding matrix in Example 2.39.

Every systematic encoding matrix (2.29),

$$G(D) = (I_b \, R(D)) \tag{2.450}$$

where I_b is a $b \times b$ identity matrix and $R(D)$ is a $b \times (c - b)$ matrix whose entries are rational functions of D, has a trivial right inverse, viz., the $c \times b$ matrix

$$G^{-1}(D) = \begin{pmatrix} I_b \\ \mathbf{0} \end{pmatrix} \tag{2.451}$$

which is polynomial in both D and D^{-1}. Hence, it follows from Theorem 2.37 that this minimality holds in general:

Theorem 2.75 Every systematic encoding matrix is minimal.

■ **EXAMPLE 2.40**

Consider the rate $R = 2/4$ minimal-basic encoding matrix

$$G(D) = \begin{pmatrix} 1+D & D & 1 & D \\ D & 1 & D & 1+D \end{pmatrix} \tag{2.452}$$

with $\mu = \nu = 2$. Let

$$T(D) = \begin{pmatrix} 1+D & D \\ D & 1 \end{pmatrix} \tag{2.453}$$

Then, we have

$$T^{-1}(D) = \frac{1}{1+D+D^2} \begin{pmatrix} 1 & D \\ D & 1+D \end{pmatrix} \tag{2.454}$$

and

$$\begin{aligned} G_{\text{sys}}(D) &= T^{-1}(D)G(D) \\ &= \frac{1}{1+D+D^2}\begin{pmatrix} 1 & D \\ D & 1+D \end{pmatrix}\begin{pmatrix} 1+D & D & 1 & D \\ D & 1 & D & 1+D \end{pmatrix} \\ &= \begin{pmatrix} 1 & 0 & \dfrac{1+D^2}{1+D+D^2} & \dfrac{D^2}{1+D+D^2} \\ 0 & 1 & \dfrac{D^2}{1+D+D^2} & \dfrac{1}{1+D+D^2} \end{pmatrix} \end{aligned} \tag{2.455}$$

$G_{\text{sys}}(D)$ has neither a minimal controller canonical form nor a minimal observer canonical form, but by a standard minimization method for sequential circuits [Lee78] we obtain the *minimal* realization shown in Fig. 2.23. (This minimization is described in Appendix A.)

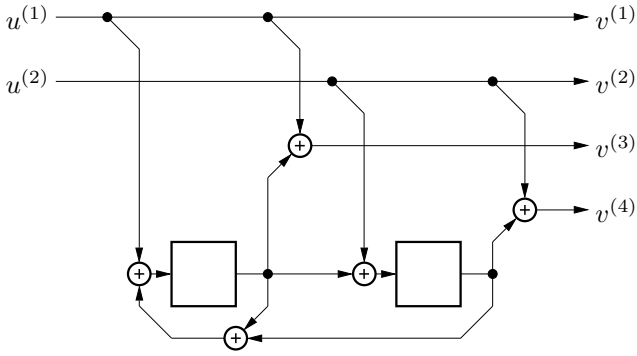


Figure 2.23 A minimal realization of the systematic encoding matrix in Example 2.40.

Consider the $c \times 1$ polynomial syndrome former

$$H^{\mathrm{T}}(D) = \begin{pmatrix} h_1^{\mathrm{T}}(D) \\ h_2^{\mathrm{T}}(D) \\ \vdots \\ h_c^{\mathrm{T}}(D) \end{pmatrix} \tag{2.456}$$

with

$$\gcd(h_1^{\mathrm{T}}(D), h_2^{\mathrm{T}}(D), \ldots, h_c^{\mathrm{T}}(D)) = 1 \tag{2.457}$$

for a rate $R = (c-1)/c$ convolutional code $\mathcal{C}$. From (2.457) it follows that at least one of the polynomials $h_i^{\mathrm{T}}(D)$ is delayfree. We can without loss of essential generality assume that $h_c^{\mathrm{T}}(D)$ is delayfree. Then we can rewrite (2.409) as

$$(v^{(1)}(D)\, v^{(2)}(D)\, \ldots\, v^{(c)}(D)) \begin{pmatrix} h_1^{\mathrm{T}}(D) \\ h_2^{\mathrm{T}}(D) \\ \vdots \\ h_c^{\mathrm{T}}(D) \end{pmatrix} = \mathbf{0} \tag{2.458}$$

which can be used to construct a systematic encoder as follows.

Assume that the first $c-1$ output sequences are identical to the $c-1$ input sequences; that is,

$$v^{(i)}(D) = u^{(i)}(D), \quad i = 1, 2, \ldots, c-1 \tag{2.459}$$

Inserting (2.459) in (2.458) gives the following expression for determining the last output sequence

$$v^{(c)}(D) = \left(h_c^{\mathrm{T}}(D)\right)^{-1}\left(u^{(1)}(D)h_1^{\mathrm{T}}(D) + \cdots + u^{(c-1)}(D)h_{c-1}^{\mathrm{T}}(D)\right) \tag{2.460}$$

Hence, we have

$$\boldsymbol{v}(D) = \boldsymbol{u}(D)G_{\mathrm{sys}}(D) \tag{2.461}$$

where

$$G_{\mathrm{sys}}(D) = \begin{pmatrix} 1 & & & h_1^{\mathrm{T}}(D)/h_c^{\mathrm{T}}(D) \\ & 1 & & h_2^{\mathrm{T}}(D)/h_c^{\mathrm{T}}(D) \\ & & \ddots & \vdots \\ & & 1 & h_{c-1}^{\mathrm{T}}(D)/h_c^{\mathrm{T}}(D) \end{pmatrix} \tag{2.462}$$

is a $(c-1) \times c$ systematic rational encoding matrix for the code $\mathcal{C}$ obtained directly from the syndrome former $H^{\mathrm{T}}(D)$. $G_{\mathrm{sys}}(D)$ is realizable since $h_c^{\mathrm{T}}(D)$ is assumed to be delayfree.

■ **EXAMPLE 2.36** ***(Cont'd)***

The syndrome former corresponding to the generator matrix $G(D)$ given in (2.446) was determined in Example 2.36:

$$H^{\mathrm{T}}(D) = \begin{pmatrix} h_1^{\mathrm{T}}(D) \\ h_2^{\mathrm{T}}(D) \\ \vdots \\ h_c^{\mathrm{T}}(D) \end{pmatrix} = \begin{pmatrix} 1 + D + D^2 + D^3 \\ 1 + D^2 + D^3 \\ 1 + D + D^3 \end{pmatrix} \tag{2.463}$$

By inserting (2.463) in (2.462) we again obtain the systematic encoding matrix given in (2.449).

Only a slight modification of the syndrome former in Fig. 2.18 is required to obtain the observer canonical form of the systematic encoding matrix in Fig. 2.22.

Next we consider the $c \times (c-b)$ polynomial syndrome former

$$H^{\mathrm{T}}(D) = \begin{pmatrix} h_{11}^{\mathrm{T}}(D) & h_{12}^{\mathrm{T}}(D) & \ldots & h_{1(c-b)}^{\mathrm{T}}(D) \\ h_{21}^{\mathrm{T}}(D) & h_{22}^{\mathrm{T}}(D) & \ldots & h_{2(c-b)}^{\mathrm{T}}(D) \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ h_{c1}^{\mathrm{T}}(D) & h_{c2}^{\mathrm{T}}(D) & \ldots & h_{c(c-b)}^{\mathrm{T}}(D) \end{pmatrix} \tag{2.464}$$

for a rate $R = b/c$ convolutional code $\mathcal{C}$. Assume that $H(D)$ is an encoding matrix for the convolutional dual code $\mathcal{C}_\perp$; then $H(0)$ has full rank and there exists a $(c-b) \times (c-b)$ submatrix of $H(0)$ whose determinant is 1. It follows that the determinant of the same submatrix of $H(D)$ is a delayfree polynomial and, thus, has a realizable inverse. Assume without loss of essential generality that this submatrix consists of the last $(c-b)$ rows in $H^{\mathrm{T}}(D)$.

We can now construct a systematic rational encoding matrix for $\mathcal{C}$ from (2.409) as follows.

Assume that the first b output sequences are identical to the b input sequences; that is,

$$v^{(i)}(D) = u^{(i)}(D), \quad i = 1, 2, \ldots, b \tag{2.465}$$

Inserting (2.465) into (2.409) where $H^{\mathrm{T}}(D)$ is given by (2.464) gives

$$\begin{aligned}(u^{(1)}(D)\, u^{(2)}(D) \ldots u^{(b)}(D)\, v^{(b+1)}(D)\, v^{(b+2)}(D) \ldots v^{(c)}(D)) \\ \times \begin{pmatrix} h^{\mathrm{T}}_{11}(D) & h^{\mathrm{T}}_{12}(D) & \ldots & h^{\mathrm{T}}_{1(c-b)}(D) \\ h^{\mathrm{T}}_{21}(D) & h^{\mathrm{T}}_{22}(D) & \ldots & h^{\mathrm{T}}_{2(c-b)}(D) \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ h^{\mathrm{T}}_{c1}(D) & h^{\mathrm{T}}_{c2}(D) & \ldots & h^{\mathrm{T}}_{c(c-b)}(D) \end{pmatrix} = \mathbf{0}\end{aligned} \tag{2.466}$$

or, equivalently,

$$\begin{aligned}&(v^{(b+1)}(D)\, v^{(b+2)}(D)\ \ldots\ v^{(c)}(D)) \\ &\quad \times \begin{pmatrix} h^{\mathrm{T}}_{(b+1)1}(D) & h^{\mathrm{T}}_{(b+1)2}(D) & \ldots & h^{\mathrm{T}}_{(b+1)(c-b)}(D) \\ h^{\mathrm{T}}_{(b+2)1}(D) & h^{\mathrm{T}}_{(b+2)2}(D) & \ldots & h^{\mathrm{T}}_{(b+2)(c-b)}(D) \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ h^{\mathrm{T}}_{c1}(D) & h^{\mathrm{T}}_{c2}(D) & \ldots & h^{\mathrm{T}}_{c(c-b)}(D) \end{pmatrix} \\ &= \Big(u^{(1)}(D)\, u^{(2)}(D) \ldots u^{(b)}(D)\Big) \\ &\quad \times \begin{pmatrix} h^{\mathrm{T}}_{11}(D) & h^{\mathrm{T}}_{12}(D) & \ldots & h^{\mathrm{T}}_{1(c-b)}(D) \\ h^{\mathrm{T}}_{21}(D) & h^{\mathrm{T}}_{22}(D) & \ldots & h^{\mathrm{T}}_{2(c-b)}(D) \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ h^{\mathrm{T}}_{b1}(D) & h^{\mathrm{T}}_{b2}(D) & \ldots & h^{\mathrm{T}}_{b(c-b)}(D) \end{pmatrix}\end{aligned} \tag{2.467}$$

Then we have

$$\begin{aligned}&\Big(v^{(b+1)}(D)\, v^{(b+2)}(D)\ \ldots\ v^{(c)}(D)\Big) \\ &\quad = \Big(u^{(1)}(D)\, u^{(2)}(D)\ \ldots\ u^{(b)}(D)\Big)\, H'(D)\end{aligned} \tag{2.468}$$

where

$$H'(D) = \begin{pmatrix} h_{11}^{(T)}(D) & h_{12}^{(T)}(D) & \dots & h_{1(c-b)}^{(T)}(D) \\ h_{21}^{(T)}(D) & h_{22}^{(T)}(D) & \dots & h_{2(c-b)}^{(T)}(D) \\ \dots & \dots & \dots & \dots \\ h_{b1}^{(T)}(D) & h_{b2}^{(T)}(D) & \dots & h_{b(c-b)}^{(T)}(D) \end{pmatrix} \times \begin{pmatrix} h_{(b+1)1}^{(T)}(D) & h_{(b+1)2}^{(T)}(D) & \dots & h_{(b+1)(c-b)}^{(T)}(D) \\ h_{(b+2)1}^{(T)}(D) & h_{(b+2)2}^{(T)}(D) & \dots & h_{(b+2)(c-b)}^{(T)}(D) \\ \dots & \dots & \dots & \dots \\ h_{c1}^{(T)}(D) & h_{c2}^{(T)}(D) & \dots & h_{c(c-b)}^{(T)}(D) \end{pmatrix}^{-1} \quad (2.469)$$

is a $b \times (c-b)$ realizable rational matrix. Thus, we have a rational systematic encoding matrix for the convolutional code $\mathcal{C}$,

$$G_{\text{sys}}(D) = (I_b \quad H'(D)) \quad (2.470)$$

where $H'(D)$ is given in equation (2.469).

■ **EXAMPLE 2.41**

The rate $R = 2/4$ minimal-basic encoding matrix (2.452) in Example 2.40 has the Smith form decomposition

$$\begin{aligned} G(D) &= A(D)\Gamma(D)B(D) \\ &= \begin{pmatrix} 1 & 0 \\ D & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \\ &\quad \times \begin{pmatrix} 1+D & D & 1 & D \\ D^2 & 1+D^2 & 0 & 1+D+D^2 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \end{aligned} \quad (2.471)$$

and, hence,

$$B^{-1}(D) = \begin{pmatrix} 0 & 1 & 1+D^2 & 1+D+D^2 \\ 0 & 1 & D^2 & 1+D+D^2 \\ 1 & 1 & 1+D+D^2 & 1+D^2 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (2.472)$$

Since the syndrome former is the last $c-b$ columns of $B^{-1}(D)$ (2.406), we have

$$H^{(T)}(D) = \begin{pmatrix} 1+D^2 & 1+D+D^2 \\ D^2 & 1+D+D^2 \\ 1+D+D^2 & 1+D^2 \\ 0 & 1 \end{pmatrix} \quad (2.473)$$

The encoding matrix for the convolutional dual code $\mathcal{C}_\perp$, viz.,

$$H(D) = \begin{pmatrix} 1+D^2 & D^2 & 1+D+D^2 & 0 \\ 1+D+D^2 & 1+D+D^2 & 1+D^2 & 1 \end{pmatrix} \quad (2.474)$$

is not minimal-basic; $[H(D)]_h$ does not have full rank. Since $H(D)$ is obtained from the unimodular matrix $B(D)$, it is basic and, hence, we can apply our minimization algorithm (Algorithm MB, given in Section 2.5) and obtain an equivalent minimal-basic encoding matrix for the convolutional dual code $\mathcal{C}_\perp$,

$$H_{\text{mb}}(D) = \begin{pmatrix} 1 & D & 1+D & D \\ D & 1+D & D & 1 \end{pmatrix} \tag{2.475}$$

The $(c-b) \times (c-b)$ matrix formed by last two rows of $H_{\text{mb}}^{\text{T}}(0)$ has full rank, which implies that the determinant of the same submatrix of $H_{\text{mb}}^{\text{T}}(D)$ is a delayfree polynomial and, thus, has a realizable inverse. Hence, we have from (2.468) that

$$\begin{aligned} &\left(v^{(3)}(D)\, v^{(4)}(D)\right) \\ &\quad = \left(u^{(1)}(D)\, u^{(2)}(D)\right) \begin{pmatrix} 1 & D \\ D & 1+D \end{pmatrix} \begin{pmatrix} 1+D & D \\ D & 1 \end{pmatrix}^{-1} \\ &\quad = \left(u^{(1)}(D)\, u^{(2)}(D)\right) \begin{pmatrix} \dfrac{1+D^2}{1+D+D^2} & \dfrac{D^2}{1+D+D^2} \\ \dfrac{D^2}{1+D+D^2} & \dfrac{1}{1+D+D^2} \end{pmatrix} \end{aligned} \tag{2.476}$$

Finally, we have the following systematic rational encoding matrix obtained via the syndrome former:

$$G_{\text{sys}}(D) = \begin{pmatrix} 1 & 0 & \dfrac{1+D^2}{1+D+D^2} & \dfrac{D^2}{1+D+D^2} \\ 0 & 1 & \dfrac{D^2}{1+D+D^2} & \dfrac{1}{1+D+D^2} \end{pmatrix} \tag{2.477}$$

which is identical to (2.455).

Next we consider the following construction of a *systematic* encoder for a convolutional code. Let the last $c-b$ rows of H_0^{T} in (2.412) form the $(c-b) \times (c-b)$ identity matrix. Then the code symbols $\boldsymbol{v} = v_t^{(1)} v_t^{(2)} \ldots v_t^{(c)}$ at time t determined by

$$\begin{aligned} v_t^{(j)} &= u_t^{(j)}, && j = 1, 2, \ldots, b \\ v_t^{(j)} &= \sum_{k=1}^{b} v_t^{(k)} h_0^{\text{T}(k,j-b)} \\ &\quad + \sum_{i=1}^{m_{\text{s}}} \sum_{k=1}^{c} v_{t-i}^{(k)} h_i^{\text{T}(k,j-b)}, && j = b+1, b+2, \ldots, c \end{aligned} \tag{2.478}$$

satisfy (2.412). The encoder for this convolutional code can be implemented by c shift registers of length m_{s} with tap weights corresponding to the entries $h_i^{\text{T}(k,j-b)}$ of the matrix H_i^{T}. It is a *syndrome (former)* realization of a convolutional encoder. The overall constraint length of the encoder, that is, the number of binary symbols which the encoder keeps in its memory, is at most cm_{s}.

■ **EXAMPLE 2.41** ***(Cont'd)***

The syndrome encoder based on the minimal-basic parity-check matrix $H_{\text{mb}}(D)$ given by (2.475) has memory $m_s = 1$ and can be derived from the syndrome former

$$H_{\text{mb}}^{\text{T}}(D) = \begin{pmatrix} 1 & D \\ D & 1+D \\ 1+D & D \\ D & 1 \end{pmatrix} = \underbrace{\begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 0 \\ 0 & 1 \end{pmatrix}}_{H_0^{\text{T}}} + \underbrace{\begin{pmatrix} 0 & 1 \\ 1 & 1 \\ 1 & 1 \\ 1 & 0 \end{pmatrix}}_{H_1^{\text{T}}} D \tag{2.479}$$

According to (2.478) its code tuples $\boldsymbol{v}_t = v_t^{(1)} v_t^{(2)} v_t^{(3)} v_t^{(4)}$ satisfy

$$\begin{aligned} v_t^{(1)} &= u_t^{(1)} \\ v_t^{(2)} &= u_t^{(2)} \\ v_t^{(3)} &= v_t^{(1)} h_0^{\text{T}(1,1)} + v_t^{(2)} h_0^{\text{T}(2,1)} \\ &\quad + v_{t-1}^{(1)} h_1^{\text{T}(1,1)} + v_{t-1}^{(2)} h_1^{\text{T}(2,1)} + v_{t-1}^{(3)} h_1^{\text{T}(3,1)} + v_{t-1}^{(4)} h_1^{\text{T}(4,1)} \\ v_t^{(4)} &= v_t^{(1)} h_0^{\text{T}(1,2)} + v_t^{(2)} h_0^{\text{T}(2,2)} \\ &\quad + v_{t-1}^{(1)} h_1^{\text{T}(1,2)} + v_{t-1}^{(2)} h_1^{\text{T}(2,2)} + v_{t-1}^{(3)} h_1^{\text{T}(3,2)} + v_{t-1}^{(4)} h_1^{\text{T}(4,2)} \end{aligned} \tag{2.480}$$

A shift register realization of the systematic rate $R = 2/4$ encoder for the convolutional code based on the syndrome former (2.479) with memory $m_s = 1$, is shown in Fig. 2.24.

We have described a particular syndrome realization of the encoder. Such a realization is far from being minimal. Another, less complex, realization of a convolutional encoder is the *partial syndrome* realization [PJS08].

A partial syndrome encoder keeps at time t in its memory the *partial syndrome*

$$\boldsymbol{p}_t = \left(\boldsymbol{p}_{t1}\, \boldsymbol{p}_{t2}\, \ldots\, \boldsymbol{p}_{tm_{\text{s}}}\right) \tag{2.481}$$

where $\boldsymbol{p}_{ti} = (p_{ti}^{(1)}\, p_{ti}^{(2)}\, \ldots\, p_{ti}^{(c-b)})$, $i = 1, 2, \ldots, m_{\text{s}}$; that is, only $(c-b)m_{\text{s}}$ bits instead of cm_{s} bits are stored. We consider $\boldsymbol{p}_t$ to be the *state* of the partial syndrome encoder at time t.

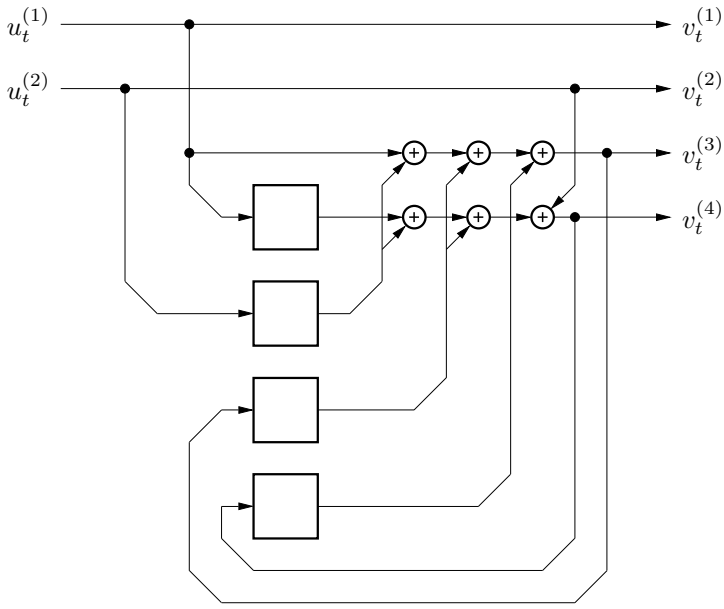


Figure 2.24 Realization of a systematic syndrome encoder for the convolutional code specified by (2.445).

To obtain a recursion for the partial syndrome we truncate the infinite syndrome former H^{T} (2.414) after the row that starts with H_0^{T} at time t, that is, we obtain

$$\boldsymbol{H}_{\mathrm{tr}}^{\mathrm{T}} = \begin{pmatrix} \ddots & H_{m_\mathrm{s}}^{\mathrm{T}} & & & & & \\ \ddots & \vdots & H_{m_\mathrm{s}}^{\mathrm{T}} & & & & \\ \ddots & \vdots & \dots & H_{m_\mathrm{s}}^{\mathrm{T}} & & & \\ \ddots & H_1^{\mathrm{T}} & H_2^{\mathrm{T}} & \dots & \ddots & & \\ \ddots & H_0^{\mathrm{T}} & H_1^{\mathrm{T}} & H_2^{\mathrm{T}} & \dots & H_{m_\mathrm{s}}^{\mathrm{T}} & \\ & & H_0^{\mathrm{T}} & H_1^{\mathrm{T}} & \dots & \dots & H_{m_\mathrm{s}}^{\mathrm{T}} \end{pmatrix} \tag{2.482}$$

time: $t-1$ t $t+1$ $\cdots$ $t+m_\mathrm{s}$

Consider the triangle in (2.482). At time t we define the partial syndrome $\boldsymbol{p}_t = (\boldsymbol{p}_{t1}\, \boldsymbol{p}_{t2}\, \cdots\, \boldsymbol{p}_{tm_\mathrm{s}})$ as

$$\begin{aligned} \boldsymbol{p}_{t1} &= \boldsymbol{v}_{t-1}H_1^{\mathrm{T}} + \boldsymbol{v}_{t-2}H_2^{\mathrm{T}} + \cdots + \boldsymbol{v}_{t-m_\mathrm{s}+1}H_{m_\mathrm{s}-1}^{\mathrm{T}} + \boldsymbol{v}_{t-m_\mathrm{s}}H_{m_\mathrm{s}}^{\mathrm{T}} \\ \boldsymbol{p}_{t2} &= \boldsymbol{v}_{t-1}H_2^{\mathrm{T}} + \boldsymbol{v}_{t-2}H_3^{\mathrm{T}} + \cdots + \boldsymbol{v}_{t-m_\mathrm{s}+1}H_{m_\mathrm{s}}^{\mathrm{T}} \\ &\vdots \\ \boldsymbol{p}_{tm_\mathrm{s}} &= \boldsymbol{v}_{t-1}H_{m_\mathrm{s}}^{\mathrm{T}} \end{aligned} \tag{2.483}$$

If we slide the triangle one step backwards along the $45°$ slope to time $t-1$, its bottom row will start with H_1^{T}. Then we obtain its second component of the partial syndrome $\boldsymbol{p}_{t-1}$ as

$$\boldsymbol{p}_{(t-1)2} = \boldsymbol{v}_{t-2}H_2^{\mathrm{T}} + \boldsymbol{v}_{t-3}H_3^{\mathrm{T}} + \cdots + \boldsymbol{v}_{t-m_{\mathrm{s}}}H_{m_{\mathrm{s}}}^{\mathrm{T}} \tag{2.484}$$

Comparing (2.483) and (2.484) yields

$$\boldsymbol{p}_{t1} = \boldsymbol{v}_{t-1}H_1^{\mathrm{T}} + \boldsymbol{p}_{(t-1)2} \tag{2.485}$$

We conclude that, in general, we have the recursion

$$\boldsymbol{p}_{ti} = \begin{cases} \boldsymbol{v}_{t-1}H_i^{\mathrm{T}} + \boldsymbol{p}_{(t-1)(i+1)}, & t \in \mathbb{Z}^+, 1 \le i < m_{\mathrm{s}} \\ \boldsymbol{v}_{t-1}H_{m_{\mathrm{s}}}^{\mathrm{T}}, & t \in \mathbb{Z}^+, i = m_{\mathrm{s}} \end{cases} \tag{2.486}$$

If we begin the encoding at time $t = 0$, say, then $\boldsymbol{p}_0$ is set equal to the allzero $(c-b)m_{\mathrm{s}}$-tuple.

Assume that a systematic realization of the partial syndrome encoder is in state $\boldsymbol{p}_t$ at time t. Then the first b symbols of the c-tuples $\boldsymbol{v}_t = v_t^{(1)}v_t^{(2)}\ldots v_t^{(b)}v_t^{(b+1)}v_t^{(b+2)}\ldots v_t^{(c)}$ are the information symbols and the last $c-b$ symbols are the parity-check symbols. It follows from (2.412) and (2.483) that we can obtain the $c-b$ parity-check symbols from the equation

$$\boldsymbol{v}_t H_0^{\mathrm{T}} = \boldsymbol{p}_{t1} \tag{2.487}$$

Since the last $c-b$ rows of H_0^{T} form the $(c-b)\times(c-b)$ identity matrix, the code symbols at time t can be determined as

$$\begin{aligned} v_t^{(j)} &= u_t^{(j)}, && j = 1, 2, \ldots, b \\ v_t^{(j)} &= \sum_{k=1}^{b} v_t^{(k)} h_0^{\mathrm{T}(k,j-b)} + p_{t1}^{(j-b)}, && j = b+1, b+2, \ldots, c \end{aligned} \tag{2.488}$$

A partial syndrome encoder can be implemented using $(c-b)m_s$ delay elements.

■ **EXAMPLE 2.41** ***(Cont'd)***

Since the memory of the syndrome former is $m_{\mathrm{s}} = 1$, we can construct a systematic partial syndrome encoder for the rate $R = 2/4$ convolutional code given in Example 2.40 with only $(c-b)m_{\mathrm{s}} = (4-2)1 = 2$ delay elements. From (2.488) we conclude that the code tuples $\boldsymbol{v}_t = v_t^{(1)}v_t^{(2)}v_t^{(3)}v_t^{(4)}$ must satisfy

$$\begin{aligned} v_t^{(1)} &= u_t^{(1)} \\ v_t^{(2)} &= u_t^{(2)} \\ v_t^{(3)} &= v_t^{(1)}h_0^{\mathrm{T}(1,1)} + v_t^{(2)}h_0^{\mathrm{T}(2,1)} + p_{t1}^{(1)} \\ v_t^{(4)} &= v_t^{(1)}h_0^{\mathrm{T}(1,2)} + v_t^{(2)}h_0^{\mathrm{T}(2,2)} + p_{t1}^{(2)} \end{aligned} \tag{2.489}$$

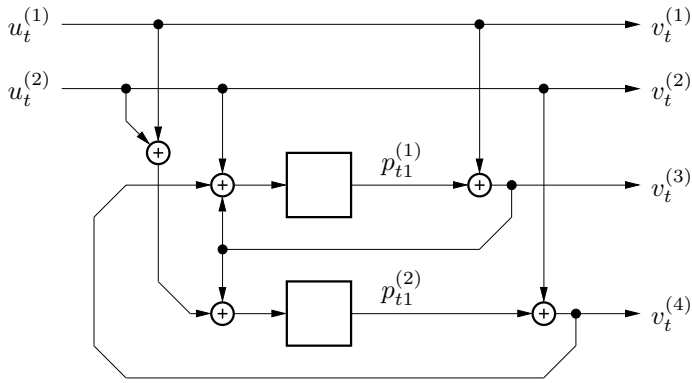


Figure 2.25 A realization of a systematic $R = 2/4$ partial syndrome encoder for the convolutional code specified in Example 2.40.

Since $m_s = 1$ we conclude from (2.486) that

$$\boldsymbol{p}_{t1} = \boldsymbol{v}_{t-1} H_1^{\mathrm{T}} \tag{2.490}$$

or, equivalently, that

$$\begin{aligned} \boldsymbol{p}_{t1}^{(1)} &= v_{t-1}^{(2)} + v_{t-1}^{(3)} + v_{t-1}^{(4)} \\ \boldsymbol{p}_{t1}^{(2)} &= v_{t-1}^{(1)} + v_{t-1}^{(2)} + v_{t-1}^{(3)} \end{aligned} \tag{2.491}$$

Hence, a realization of the systematic partial syndrome encoder based on (2.489) is shown in Fig. 2.25. This realization is a minimal encoder. An equivalent minimal encoder for the same convolutional code is shown in Fig. 2.23.

2.12 SOME PROPERTIES OF GENERATOR MATRICES—AN OVERVIEW

In Fig. 2.26 we show a Venn diagram that illustrates the relations between various properties of convolutional generator matrices.

2.13 COMMENTS

Massey made early contributions of the greatest importance to the structural theory of convolutional encoders. Together with Sain [MaS67], he defined two convolutional generator matrices to be equivalent if they encode the same code. They also proved that every convolutional code can be encoded by a polynomial generator matrix. Later, they studied conditions for a convolutional generator matrix to have a polynomial right inverse [MaS68, SaM69]. Massey's work in this area was continued

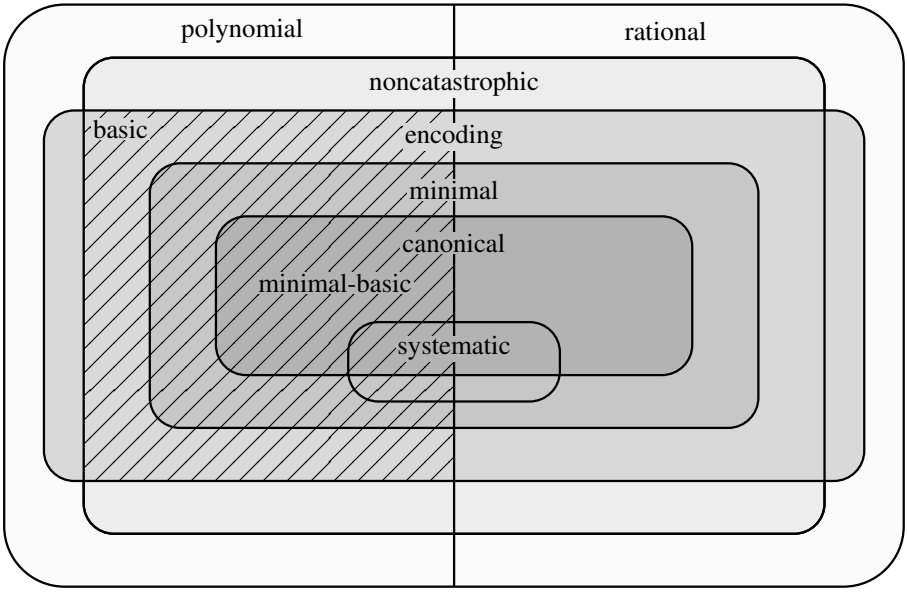


Figure 2.26 Relations between properties of generator matrices. (Courtesy of Maja Lončar.)

by his students Costello [Cos69] and Olson [Ols70]. Costello was apparently the first to notice that every convolutional generator matrix is equivalent to a systematic encoding matrix, in general nonpolynomial.

By exploiting the invariant-factor theorem and the realization theory of linear systems, Forney generalized, deepened, and extended these results in a series of landmark papers [For70, For73, For75].

Among the pre-Forney structural contributions we also have an early paper by Bussgang [Bus65].

In the late 1980s and early 1990s, there was a renewed interest in the structural analysis of convolutional codes. In 1988 Piret published his monograph [Pir88], which contains an algebraic approach to convolutional codes. In a semitutorial paper, Johannesson and Wan [JoW93] rederived many of Forney's important results and derived some new ones using only linear algebra—an approach that permeates this chapter. In [For91] Forney extended and deepened his results in [For75]. This inspired Johannesson and Wan to write [JoW94], which together with [FJW96] constitute the basis for Sections 2.7 and 2.8.

Other important developments are reported in [FoT93, LoM96].

Finally, we would like to mention Massey's classic introduction to convolutional codes [Mas75] and McEliece's 1998 "spin" on Forney's papers mentioned above [McE98].

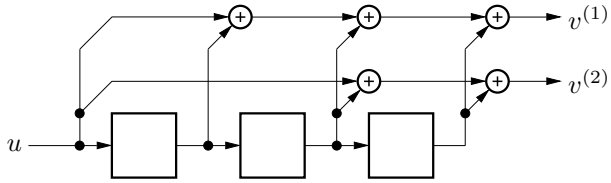


Figure 2.27 Encoder used in Problem 2.1.

PROBLEMS

2.1 Consider the rate $R = 1/2$ convolutional encoder illustrated in Fig. 2.27.

a) Find the generator matrix G(D).

b) Let $G(D) = A(D)\Gamma(D)B(D)$. Find $A(D), \Gamma(D)$, and $B(D)$.

c) Find $G'(D)$, where $G'(D)$ is the first b rows in $B(D)$.

d) Is $G'(D)$ minimal? If not, find a minimal encoding matrix $G_{\min}(D)$ equivalent to $G'(D)$.

e) Find $A^{-1}(D), B^{-1}(D)$, and $G^{-1}(D)$.

f) Find $H(D)$ and verify that $G(D)H^{\mathrm{T}}(D) = \mathbf{0}$.

2.2 Repeat Problem 2.1 for the encoding matrix

$$\begin{pmatrix} 11 & 10 & 01 & 11 & & & \\ & 11 & 10 & 01 & 11 & & \\ & & 11 & 10 & 01 & 11 & \\ & & & \ddots & \ddots & \ddots & \ddots \end{pmatrix}$$

2.3 Draw the encoder block diagram for the controller canonical form of the encoding matrix

$$G(D) = \begin{pmatrix} 1+D & D & 1+D \\ 1 & 1 & D \end{pmatrix}$$

and repeat Problem 2.1.

2.4 Consider the encoder shown in Fig. 2.28.

a) Find the generator matrix $G(D)$.

b) Find the Smith form.

c) Is the generator matrix catastrophic or noncatastrophic?

d) Find $G^{-1}(D)$.

2.5 Consider the rate $R = 1/2$ convolutional encoding matrix

$$G(D) = (1 + D^3 \quad 1 + D + D^2 + D^3)$$

a) Find the Smith form decomposition.

b) Find $G^{-1}(D)$.

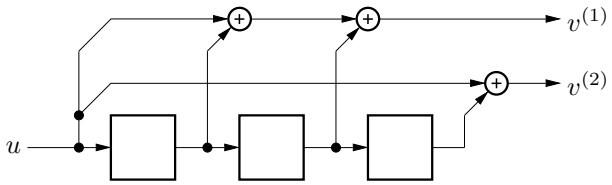


Figure 2.28 Encoder used in Problem 2.4.

2.6 Consider the rate $R = 2/3$ rational convolutional encoding matrix

$$G(D) = \begin{pmatrix} \dfrac{1}{1+D^2} & \dfrac{D}{1+D^2} & \dfrac{1}{1+D} \\ \dfrac{D}{1+D^2} & \dfrac{1}{1+D^2} & 1 \end{pmatrix}$$

a) Find the invariant-factor decomposition.

b) Find $G^{-1}(D)$.

2.7 The behavior of a linear sequential circuit can be described by the matrix equations

$$\begin{aligned} \boldsymbol{\sigma}_{t+1} &= A\boldsymbol{\sigma}_t + B\boldsymbol{u}_t \\ \boldsymbol{v}_t &= C\boldsymbol{\sigma}_t + H\boldsymbol{u}_t \end{aligned}$$

where $\boldsymbol{u}, \boldsymbol{v}$, and $\boldsymbol{\sigma}_t$ are the input, output, and encoder state, respectively, at time t.

a) Show that applying the D-transforms to each term in the matrices yields

$$\begin{aligned} D^{-1}(\boldsymbol{\sigma}(D) + \boldsymbol{\sigma}_0) &= \boldsymbol{\sigma}(D)A + \boldsymbol{u}(D)B \\ \boldsymbol{v}(D) &= \boldsymbol{\sigma}(D)C + \boldsymbol{u}(D)H \end{aligned}$$

where $\boldsymbol{\sigma}_0$ is the initial value of $\boldsymbol{\sigma}_t$.

b) Show that the transfer function matrix is

$$G(D) = H + B(I + AD)^{-1}CD$$

c) Show that the abstract state $\boldsymbol{s}(D)$ corresponding to the encoder state $\boldsymbol{\sigma}$ can be expressed as

$$\boldsymbol{s}(D) = \boldsymbol{\sigma}(I + AD)^{-1}C$$

2.8 Consider the encoding matrix in Problem 2.3. Find the correspondence between the encoder and abstract states.

2.9 Consider the two minimal-basic encoding matrices

$$G_1(D) = \begin{pmatrix} 1+D & D & 1 \\ D^2 & 1 & 1+D+D^2 \end{pmatrix}$$

and

$$G_2(D) = \begin{pmatrix} 1+D & D & 1 \\ 1 & 1+D+D^2 & D^2 \end{pmatrix}$$

a) Show that $G_1(D)$ and $G_2(D)$ are equivalent.

b) Find an isomorphism between the state spaces of $G_1(D)$ and $G_2(D)$. *Hint*: Study the proof of Lemma 2.33.

2.10 Consider the rate $R = 1/2$ convolutional encoding matrix

$$G(D) = (1+D+D^2 \quad 1+D^2)$$

a) Find the syndrome former $H^{\mathrm{T}}(D)$ and draw its controller and observer canonical forms.

b) Find a systematic encoding matrix $G_{\text{sys}}(D)$ equivalent to $G(D)$ and draw its controller and observer canonical forms.

c) Find an encoding matrix for the convolutional dual code and draw its observer canonical form.

2.11 Consider the rate $R = 2/3$ convolutional encoding matrix

$$G(D) = \begin{pmatrix} 1+D & D & 1+D \\ 1 & 1 & D \end{pmatrix}$$

a) Find the syndrome former $H^{\mathrm{T}}(D)$ and draw both its controller and observer canonical forms.

b) Find a systematic encoding matrix $G_{\text{sys}}(D)$ equivalent to $G(D)$ and draw its controller and observer canonical forms.

c) Find an encoding matrix for the convolutional dual code and draw its controller canonical form.

d) Find a right inverse to the encoding matrix in Problem 2.11(b) and draw its observer canonical form.

2.12 Consider the rate $R = 2/3$ convolutional encoding matrix

$$G(D) = \begin{pmatrix} 1 & D^2 & 1+D+D^2 \\ 1 & 1+D & D \end{pmatrix}$$

and repeat Problem 2.11.

2.13 Consider the rate $R = 1/2$ nonsystematic encoding matrix

$$G(D) = (1+D+D^2 \quad 1+D^2)$$

Find two systematic polynomial encoding matrices that are equivalent over a memory length.

2.14 Consider the rate $R = 2/3$ convolutional encoding matrix

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

a) Find without permuting the columns an equivalent systematic rational encoding matrix and draw both its controller and observer canonical forms.
b) Find a systematic polynomial encoding matrix that is equivalent to $G(D)$ over a memory length.
c) Compare the result with the systematic encoding matrix in Example 2.39.

2.15 Consider the rate $R = 2/3$ convolutional encoding matrix

$$G(D) = \begin{pmatrix} 1 & D^2 & 1+D+D^2 \\ 1 & 1+D & D \end{pmatrix}$$

a) Is $G(D)$ basic?
b) Is $G(D)$ minimal?
c) Let $G'(D)$ be the b first rows in the unimodular matrix B in the Smith form decomposition of $G(D)$. Is $G'(D)$ minimal?

2.16 Consider the rate $R = 2/3$ convolutional encoding matrix

$$G(D) = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 1+D & 1 \end{pmatrix}$$

a) Find $G^{-1}(D)$.
b) Is $G(D)$ basic?
c) Is $G(D)$ minimal? If not, find a minimal encoding matrix $G_{\min}(D)$ equivalent to $G(D)$.

2.17 Consider the rate $R = 2/3$ convolutional encoding matrix

$$G = \begin{pmatrix} \dfrac{1}{1+D^3} & \dfrac{1}{1+D+D^2} & \dfrac{D}{1+D^3} \\ \dfrac{D^2}{1+D^3} & \dfrac{D}{1+D^3} & \dfrac{1+D+D^2+D^3}{1+D^3} \end{pmatrix}$$

a) Find $G^{-1}(D)$.
b) Is $G^{-1}(D)$ realizable?
c) Is $G(D)$ catastrophic?

2.18 Consider the rate $R = 2/3$ convolutional encoding matrix

$$G(D) = \begin{pmatrix} 1 & 1 & D+D^3 & D^2+D^3 \\ D^2 & 1+D^3 & 1+D^2+D^3+D^4 & D+D^2+D^3+D^4 \end{pmatrix}$$

a) Is $G(D)$ basic?

b) Is $G(D)$ minimal?

2.19 Consider the rate $R = 4/5$ convolutional generator matrix

$$G(D) = \begin{pmatrix} 1+D+D^4 & D^4 & 1+D^2 & 1+D^4 & D^2+D^4 \\ 1+D & D+D^3 & 1+D^3 & D^2+D^3 & 1+D^2+D^3 \\ 1+D & 1 & D & 0 & 1 \\ 1+D^2 & D+D^2 & 1+D & D^2 & 1+D+D^2 \end{pmatrix}$$

a) Is $G(D)$ an encoding matrix?

b) Is $G(D)$ basic?

c) Is $G(D)$ minimal? If not, find a minimal encoding matrix $G_{\min}(D)$ equivalent to $G(D)$.

2.20 Consider the rate $R = 2/3$ convolutional encoder illustrated in Fig. 2.29.

a) Find the generator matrix $G(D)$.

b) Is $G(D)$ minimal-basic? If not, find a minimal-basic encoding matrix $G_{\text{mb}}(D)$ equivalent to $G(D)$.

c) Is $G(D)$ minimal?

d) Is the encoder in Fig. 2.29 minimal?

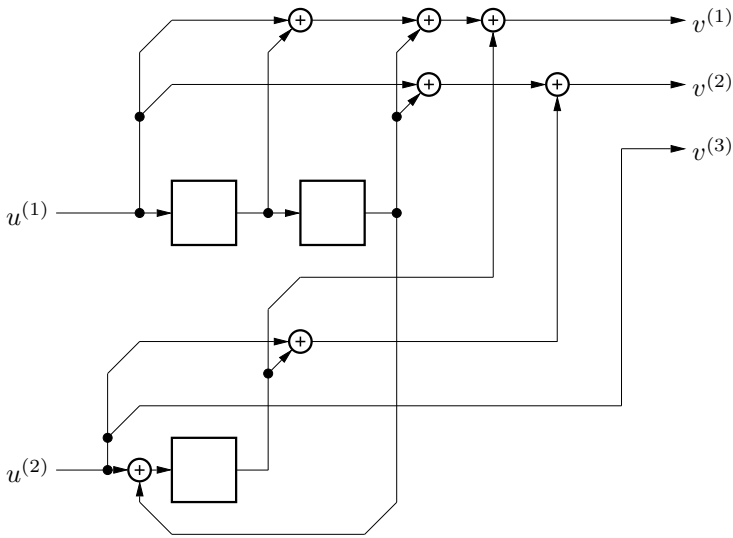


Figure 2.29 Encoder used in Problem 2.20.

2.21 Consider the rate $R = 2/3$ convolutional encoder illustrated in Fig. 2.30.

a) Find the generator matrix $G(D)$.

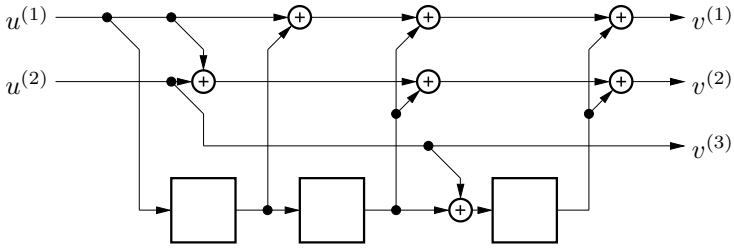


Figure 2.30 Encoder used in Problems 2.21 and 2.26.

b) Is $G(D)$ minimal-basic? If not, find a minimal-basic encoding matrix $G_{\text{mb}}(D)$ equivalent to $G(D)$.

2.22 Consider the rate $R = 2/3$ convolutional encoding matrix

$$G(D) = \begin{pmatrix} D & D & 1 \\ 1 & D^2 & 1 + D + D^2 \end{pmatrix}$$

a) Is $G(D)$ minimal?
b) Find a systematic encoding matrix $G_{\text{sys}}(D)$ equivalent to $G(D)$.

2.23 Consider the rate $R = 2/3$ convolutional encoding matrix

$$G(D) = \begin{pmatrix} \dfrac{1}{1 + D + D^2} & \dfrac{D}{1 + D^3} & \dfrac{1}{1 + D^3} \\ \dfrac{D^2}{1 + D^3} & \dfrac{1}{1 + D^3} & \dfrac{1}{1 + D} \end{pmatrix}$$

a) Is $G(D)$ minimal?
b) Find a systematic encoding matrix equivalent to $G(D)$.
c) Find $G^{-1}(D)$.

2.24 Consider the two rate $R = 2/3$ convolutional encoding matrices

$$G_1(D) = \begin{pmatrix} 1 & \dfrac{(1 + D)^3}{1 + D + D^3} & 0 \\ 0 & \dfrac{1 + D^2 + D^3}{1 + D + D^3} & 1 \end{pmatrix}$$

and

$$G_2(D) = \begin{pmatrix} 1 + D & 1 & D \\ D^2 & 1 + D + D^2 & 1 \end{pmatrix}$$

a) Are $G_1(D)$ and $G_2(D)$ equivalent?
b) Is $G_1(D)$ minimal?
c) Is $G_2(D)$ minimal?

2.25 Consider the rate $R = 1/3$ convolutional code with encoding matrix

$$G(D) = \left(\frac{1+D+D^2}{1+D^4} \quad \frac{1+D^2+D^4}{1+D^4} \quad \frac{1+D+D^2}{1+D^2} \right)$$

a) Is 111 010 011 000 000 000 . . . a codeword?

b) Find a minimal encoder whose encoding matrix is equivalent to $G(D)$.

2.26 Consider the rate $R = 2/3$ convolutional encoder illustrated in Fig. 2.30.

a) Is the encoder in Fig. 2.30 minimal?

b) Find a systematic encoder that is equivalent to the encoder in Fig. 2.30.

2.27 Consider the rate $R = 2/4$ convolutional encoder illustrated in Fig. 2.31. Is the encoder in Fig. 2.31 minimal?

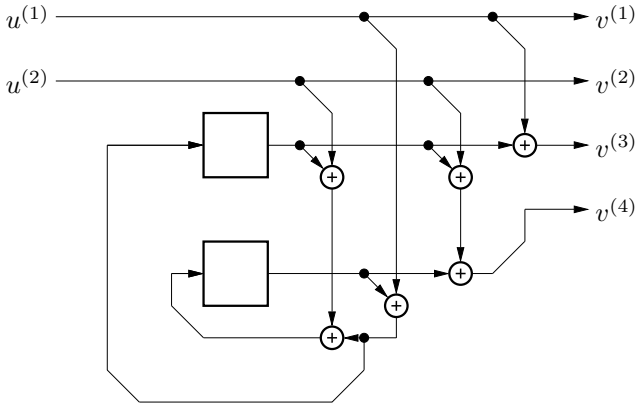


Figure 2.31 Encoder used in Problem 2.27.

2.28 Consider the cascade of a rate $R_o = b_o/c_o$ outer convolutional encoder with generator matrix $G^o(D)$ and a rate $R_i = b_i/c_i$ inner convolutional encoder with generator matrix $G^i(D)$, where $b_i = c_o$ (Fig. 2.32). Show that if $G^o(D)$ and $G^i(D)$ are both minimal, then the cascaded generator matrix defined by their product $G^c(D) = G^o(D)G^i(D)$ is also minimal [HJS98].

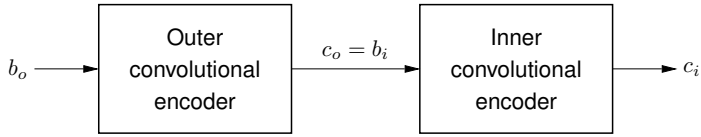


Figure 2.32 A cascade of two consecutive convolutional encoders used in Problem 2.28.

2.29 Let $G_{\text{mb}}(D)$ be a minimal-basic encoding matrix with memory m_{mb}. Show that m_{mb} is minimal over all equivalent polynomial generator matrices.
Hint: Use the predictable degree property.

2.30 Let $G_{\mathrm{c}}(D)$ be a canonical encoding matrix with memory m_{c}. Show that m_{c} is minimal over all equivalent rational generator matrices.
Hint: Use the GPVP.

CHAPTER 3

DISTANCE PROPERTIES OF CONVOLUTIONAL CODES

Several important distance measures for convolutional codes and encoders are defined. We also derive upper and lower bounds for most of these distances. Some of the bounds might be useful guidelines when we construct new encoders, others when we analyze and design coding systems. The Viterbi (distance) spectrum for a convolutional encoder is obtained via the path enumerators that are determined from the state-transition diagram for the encoder in controller canonical form.

3.1 DISTANCE MEASURES—A FIRST ENCOUNTER

In this section we discuss the most common distance measures for convolutional codes.

Consider a binary, rate $R = b/c$ convolutional code with a rational generator matrix $G(D)$ of memory m. The causal information sequence

$$\boldsymbol{u}(D) = \boldsymbol{u}_0 + \boldsymbol{u}_1 D + \boldsymbol{u}_2 D^2 + \cdots \tag{3.1}$$

is encoded as the causal codeword

$$\boldsymbol{v}(D) = \boldsymbol{v}_0 + \boldsymbol{v}_1 D + \boldsymbol{v}_2 D^2 + \cdots \tag{3.2}$$

Fundamentals of Convolutional Coding, 2. Edition. By Rolf Johannesson, Kamil Sh. Zigangirov

where

$$\boldsymbol{v}(D) = \boldsymbol{u}(D)G(D) \tag{3.3}$$

For simplicity we sometimes write $\boldsymbol{u} = \boldsymbol{u}_0\,\boldsymbol{u}_1\,\ldots$ and $\boldsymbol{v} = \boldsymbol{v}_0\,\boldsymbol{v}_1\,\ldots$ instead of $\boldsymbol{u}(D)$ and $\boldsymbol{v}(D)$, respectively.

First we consider the most fundamental distance measure, which is the *column distance* [Cos69].

Definition Let $\mathcal{C}$ be a convolutional code encoded by a rational generator matrix $G(D)$. The *jth order column distance* d_j^{c} of the generator matrix $G(D)$ is the minimum Hamming distance between two encoded sequences $\boldsymbol{v}_{[0,j]}$ resulting from causal information sequences $\boldsymbol{u}_{[0,j]}$ with differing $\boldsymbol{u}_0$.

From the linearity of the code it follows that d_j^{c} is also the minimum of the Hamming weights of the paths $\boldsymbol{v}_{[0,j]}$ resulting from causal information sequences with $\boldsymbol{u}_0 \neq \mathbf{0}$. Thus,

$$d_j^{\mathrm{c}} = \min_{\boldsymbol{u}_0 \neq \mathbf{0}} \left\{ w_{\mathrm{H}}(\boldsymbol{v}_{[0,j]}) \right\} \tag{3.4}$$

where $w_{\mathrm{H}}(\ )$ denotes the Hamming weight of a sequence.

Let

$$G(D) = G_0 + G_1 D + \cdots + G_m D^m \tag{3.5}$$

be a *polynomial* generator matrix of memory m and let the corresponding semi-infinite matrix $\boldsymbol{G}$ be (1.91)

$$\boldsymbol{G} = \begin{pmatrix} G_0 & G_1 & \ldots & G_m & & \\ & G_0 & G_1 & \ldots & G_m & \\ & & \ddots & \ddots & & \ddots \end{pmatrix} \tag{3.6}$$

where G_i, $0 \leq i \leq m$, are binary $b \times c$ matrices.

Denote by $\boldsymbol{G}_j^{\mathrm{c}}$ the truncation of $\boldsymbol{G}$ after $j+1$ columns, that is,

$$\boldsymbol{G}_j^{\mathrm{c}} = \begin{pmatrix} G_0 & G_1 & G_2 & \ldots & G_j \\ & G_0 & G_1 & & G_{j-1} \\ & & G_0 & & G_{j-2} \\ & & & \ddots & \vdots \\ & & & & G_0 \end{pmatrix} \tag{3.7}$$

where $G_i = \mathbf{0}$ when $i > m$.

Making use of (1.89), we can rewrite (3.4) as

$$d_j^{\mathrm{c}} = \min_{\boldsymbol{u}_0 \neq \mathbf{0}} \left\{ w_{\mathrm{H}} \left(\boldsymbol{u}_{[0,j]} \boldsymbol{G}_j^{\mathrm{c}} \right) \right\} \tag{3.8}$$

From (3.7) and (3.8) it follows that to obtain the jth order column distance d_j^{c} of the polynomial generator matrix (3.5), we truncate the matrix $\boldsymbol{G}$ after $j+1$ columns.

■ **EXAMPLE 3.1**

Consider the convolutional code $\mathcal{C}$ encoded by the encoding matrix

$$G(D) = \begin{pmatrix} 0 & 1 & 1 \\ 1 & D & 0 \end{pmatrix} \tag{3.9}$$

where

$$G(0) = G_0 = \begin{pmatrix} 011 \\ 100 \end{pmatrix} \tag{3.10}$$

has full rank. The encoding matrix $G(D)$ has the column distances

$$d_0^{\mathrm{c}} = \min_{\boldsymbol{u}_0 \neq \mathbf{0}} \{w_{\mathrm{H}}(\boldsymbol{u}_0 G_0)\} = w_{\mathrm{H}}\left((01)\begin{pmatrix} 011 \\ 100 \end{pmatrix}\right) = 1 \tag{3.11}$$

and

$$\begin{aligned} d_1^{\mathrm{c}} &= \min_{\boldsymbol{u}_0 \neq \mathbf{0}} \left\{ w_{\mathrm{H}}\left((\boldsymbol{u}_0 \boldsymbol{u}_1) \begin{pmatrix} G_0 & G_1 \\ & G_0 \end{pmatrix} \right) \right\} \\ &= w_{\mathrm{H}}\left((01\,00) \begin{pmatrix} 011 & 000 \\ 100 & 010 \\ & 011 \\ & 100 \end{pmatrix} \right) = 2 \end{aligned} \tag{3.12}$$

The equivalent generator matrix

$$\begin{aligned} G'(D) &= \begin{pmatrix} 1+D & 1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} 0 & 1 & 1 \\ 1 & D & 0 \end{pmatrix} \\ &= \begin{pmatrix} 1 & 1 & 1+D \\ 1 & 1+D & 1 \end{pmatrix} \end{aligned} \tag{3.13}$$

has

$$G_0' = \begin{pmatrix} 111 \\ 111 \end{pmatrix} \tag{3.14}$$

of rank $G_0' = 1$. By choosing $\boldsymbol{u}_0 = 11$ we obtain

$$d_0^{\mathrm{c}\prime} = w_{\mathrm{H}}(\boldsymbol{u}_0 G_0') = 0 \tag{3.15}$$

Hence, there is a nontrivial transition from the zero state (not *to* the zero state) that produces a zero output. The two equivalent generator matrices for the code $\mathcal{C}$ have different column distances.

From Example 3.1 it follows that the column distance is an encoder property, not a code property.

In Chapter 2 we defined an encoding matrix as a generator matrix with $G(0)$ of full rank. The main reason for this restriction on $G(0)$ is given in the following:

Theorem 3.1 The column distance is invariant over the class of equivalent encoding matrices.

Proof: Let $\mathcal{C}_{\text{cd}}$ be the set of causal and delayfree codewords,

$$\mathcal{C}_{\text{cd}} \stackrel{\text{def}}{=} \{\boldsymbol{v} \in \mathcal{C} \mid \boldsymbol{v}_i = \mathbf{0}, \;\; i < 0, \;\; \text{and} \;\; \boldsymbol{v}_0 \neq \mathbf{0}\} \tag{3.16}$$

The set $\mathcal{C}_{\text{cd}}$ is a subset of $\mathcal{C}, \mathcal{C}_{\text{cd}} \subset \mathcal{C}$, but it is not a subcode since it is not closed under addition. The set of causal and delayfree codewords $\mathcal{C}_{\text{cd}}$ depends only on $\mathcal{C}$ and is independent of the chosen generator matrix. Then the theorem follows from the observation that for encoding matrices the minimization over $\boldsymbol{u}_0 \neq \mathbf{0}$ in (3.4) is a minimization over $\{\boldsymbol{v}_{[0,j]} \mid \boldsymbol{v} \in \mathcal{C}_{\text{cd}}\}$. ∎

Theorem 3.1 leads to the following:

Definition Let $\mathcal{C}$ be a convolutional code. The jth order column distance of $\mathcal{C}$ is the jth order column distance of any encoding matrix of $\mathcal{C}$.

The mth order column distance d_m^{c} of a rational generator matrix of memory m, where the memory of a rational generator matrix is defined by (2.274), is sometimes called the *minimum distance* (of the generator matrix) and denoted $d_{\min}$. It determines the error-correcting capability of a decoder that estimates the information symbol $\boldsymbol{u}_0$ based on the received symbols over the *first memory length* only, that is, over the first $n_m = (m+1)c$ received symbols.

A good computational performance for sequential decoding (to be discussed in Chapter 7) requires a rapid initial growth of the column distances [MaC71]. This led to the introduction of the *distance profile* [Joh75]:

Definition Let $G(D)$ be a rational generator matrix of memory m. The $(m+1)$-tuple

$$\boldsymbol{d}^{\text{p}} = (d_0^{\text{c}}, d_1^{\text{c}}, \ldots, d_m^{\text{c}}) \tag{3.17}$$

where d_j^{c}, $0 \leq j \leq m$, is the jth order column distance, is called the *distance profile of the generator matrix* $G(D)$.

The distance profile of the generator matrix is an encoder property. However, since the jth order column distance is the same for equivalent encoding matrices and the memory is the same for all equivalent minimal-basic (canonical) encoding matrices, we can also define the distance profile of a code:

Definition Let $\mathcal{C}$ be a convolutional code encoded by a minimal-basic encoding matrix $G_{\text{mb}}(D)$ of memory m. The $(m+1)$-tuple

$$\boldsymbol{d}^{\text{p}} = (d_0^{\text{c}}, d_1^{\text{c}}, \ldots, d_m^{\text{c}}) \tag{3.18}$$

where d_j^{c}, $0 \leq j \leq m$, is the jth order column distance of $G_{\text{mb}}(D)$, is called the *distance profile of the code* $\mathcal{C}$.

A generator matrix of memory m is said to have a distance profile $\boldsymbol{d}^{\text{p}}$ *superior* to a distance profile $\boldsymbol{d}^{\text{p}'}$ of another generator matrix of the same rate R and memory m

if there is some ℓ such that

$$d_j^{\mathrm{c}} \begin{cases} = d_j^{\mathrm{c}\prime}, & j = 0, 1, \ldots, \ell - 1 \\ > d_j^{\mathrm{c}\prime}, & j = \ell \end{cases} \tag{3.19}$$

Moreover, a convolutional code $\mathcal{C}$ is said to have an *optimum distance profile* (is an ODP *code*) if there exists no generator matrix of the same rate and memory as $\mathcal{C}$ with a better distance profile.

A generator matrix $G(D)$ with optimum d_0^{c} must have $G(0)$ of full rank. Hence, a generator matrix of an ODP code is an ODP *encoding matrix*.

An ODP encoding matrix yields the fastest possible initial growth of the minimal separation between the encoded paths diverging at the root in a code tree.

Remark: We notice that only in the range $0 \leq j \leq m$ is each branch on a code sequence $\boldsymbol{v}_{[0,j]}$ affected by a new portion of the generator matrix as one penetrates into the trellis. The great dependence of the branches thereafter militates against the choice

$$\boldsymbol{d}_\infty^{\mathrm{p}} = (d_0^{\mathrm{c}}, d_1^{\mathrm{c}}, \ldots, d_\infty^{\mathrm{c}}) \tag{3.20}$$

as does the fact that d_∞^{c} is probably a description of the remainder of the column distances, which is quite adequate for all practical purposes.

Let $G'(D)$ be a rational encoding matrix of memory m'. We denote by $G'(D) \mid_m$, where $m \leq m'$, the truncation of all numerator and denominator polynomials in $G'(D)$ to degree m. Then it follows that the encoding matrix $G(D)$ of memory m and the encoding matrix $G''(D) = T(D)G(D) \mid_m$, where $T(D)$ is a $b \times b$ nonsingular rational matrix, are equivalent over the first memory length m. Hence, they have the same distance profile. Let, for example, $G(D)$ be a systematic encoding matrix. Then we can use this property to generate a set of nonsystematic encoding matrices with the same distance profile.

■ EXAMPLE 3.2

The systematic encoding matrix

$$G_{\mathrm{sys}}(D) = (1 \quad 1 + D + D^2) \tag{3.21}$$

has the optimum distance profile $\boldsymbol{d}^{\mathrm{p}} = (2, 3, 3)$.

The nonsystematic encoding matrix

$$\begin{aligned} G(D) &= (1 + D + D^2)(1 \quad 1 + D + D^2) \mid_2 \\ &= (1 + D + D^2 \quad 1 + D^2) \end{aligned} \tag{3.22}$$

is equivalent to $G_{\mathrm{sys}}(D)$ over the first memory length and, hence, has the same distance profile.

Theorem 3.2 The column distances of a generator matrix satisfy the following conditions:

(i)
$$d_j^c \leq d_{j+1}^c, \quad j = 0, 1, 2, \ldots \tag{3.23}$$

(ii) The sequence $d_0^c, d_1^c, d_2^c, \ldots$ is bounded from above.

(iii) d_j^c becomes stationary as j increases.

Proof: (i) Assume that
$$d_{j+1}^c = w_H(\boldsymbol{v}_{[0,j+1]}) \tag{3.24}$$
where $\boldsymbol{v}_0 \neq \boldsymbol{0}$. Then,
$$d_{j+1}^c \geq w_H(\boldsymbol{v}_{[0,j]}) \geq d_j^c \tag{3.25}$$

(ii) It follows from the controller canonical form of a generator matrix $G(D)$ that for any j there exists an input sequence $\boldsymbol{u}_{[0,j]}$ such that the Hamming weight of the output sequence $\boldsymbol{v}_{[0,j]}$ is less than or equal to the number of nonzero coefficients in the numerator polynomials of the equivalent generator matrix whose denominator polynomials are the lcm of the denominator polynomials of $G(D)$. This number is finite.

(iii) Follows immediately from (i) and (ii). ∎

Thus, the column distance d_j^c is a nondecreasing function of j. It is sometimes called the *column distance function* [ChC76]. Moreover, the limit
$$d_\infty^c = \lim_{j \to \infty} d_j^c \tag{3.26}$$
exists, and we have the relations
$$d_0^c \leq d_1^c \leq \cdots \leq d_\infty^c \tag{3.27}$$

Definition Let $\mathcal{C}$ be a convolutional code. The minimum Hamming distance between any two differing codewords,
$$d_{\text{free}} = \min_{\boldsymbol{v} \neq \boldsymbol{v}'} \{d_H(\boldsymbol{v}, \boldsymbol{v}')\} \tag{3.28}$$
is called the *free distance* of the code.

From the linearity of a convolutional code it follows immediately that d_{free} is also the minimum Hamming weight over the nonzero codewords.

The free distance is a *code* property!

In Fig. 3.1 we illustrate an example of two codewords, $\boldsymbol{v}$ and $\boldsymbol{v}'$, in a trellis. Assume that $\boldsymbol{v} = \boldsymbol{0}$. Then the free distance of a code $\mathcal{C}$ is the smallest Hamming weight of a codeword $\boldsymbol{v}'$ that makes a detour from the allzero codeword.

The free distance is the principal determiner for the error-correcting capability of a code when we are communicating over a channel with small error probability and use maximum-likelihood (or nearly so) decoding.

Let $\mathcal{E}_t$ be the set of all error patterns with t or fewer errors. As a counterpart to Theorem 1.1 for block codes we have:

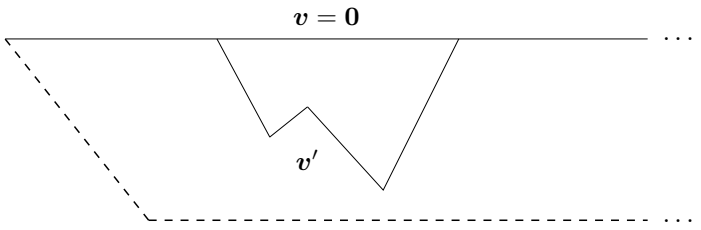


Figure 3.1 Two codewords in a trellis.

Theorem 3.3 A convolutional code $\mathcal{C}$ can correct all error patterns in $\mathcal{E}_t$ if and only if $d_{\text{free}} > 2t$.

Proof: See the proof of Theorem 1.1. ∎

Theorem 3.4 For every convolutional code $\mathcal{C}$,

$$d_{\text{free}} = d_{\infty}^{\text{c}} \tag{3.29}$$

Proof: From Theorem 3.2 (iii), it follows that there is an integer k such that

$$d_k^{\text{c}} = d_{k+1}^{\text{c}} = \cdots = d_{\infty}^{\text{c}} \tag{3.30}$$

Clearly, there exists a codeword of weight d_k^{c}. Let $G(D)$ be a minimal polynomial encoding matrix of $\mathcal{C}$. By definition of d_k^{c} an encoded sequence $\boldsymbol{v}_{[0,k]}$ results from a causal information sequence $\boldsymbol{u}_{[0,k]}$ with $\boldsymbol{u}_0 \neq \mathbf{0}$ such that $w_{\text{H}}(\boldsymbol{v}_{[0,k]}) = d_k^{\text{c}}$. Since $G(D)$ is minimal, it follows from Theorem 2.40 that (3.30) can only be satisfied if $\boldsymbol{v}_{[k+1,\infty)}$ follows the path of zero states. Then,

$$(\boldsymbol{u}_{[0,k]}\boldsymbol{u}_{k+1}\boldsymbol{u}_{k+2}\ldots)\boldsymbol{G} = (\boldsymbol{v}_{[0,k]}\mathbf{0}\,\mathbf{0}\ldots) \in \mathcal{C} \tag{3.31}$$

where $\boldsymbol{G}$ is the matrix (3.6), and we conclude that

$$w_{\text{H}}(\boldsymbol{v}_{[0,k]}\mathbf{0}\,\mathbf{0}\ldots) = w_{\text{H}}(\boldsymbol{v}_{[0,k]}) = d_k^{\text{c}} \tag{3.32}$$

Hence,

$$d_{\text{free}} \leq d_k^{\text{c}} = d_{\infty}^{\text{c}} \tag{3.33}$$

If $G(D)$ is nonminimal, (3.33) follows from the (non)minimality criteria given in Theorem 2.40. We can assume that $\mathcal{C}$ has a codeword $\boldsymbol{v} = \boldsymbol{v}_0\,\boldsymbol{v}_1\,\ldots$ of weight d_{free} with $\boldsymbol{v}_0 \neq \mathbf{0}$. For all j we have

$$d_j^{\text{c}} \leq w_{\text{H}}(\boldsymbol{v}_0\,\boldsymbol{v}_1\,\ldots\,\boldsymbol{v}_j) \leq d_{\text{free}} \tag{3.34}$$

Let $j \geq k$ and the proof is complete. ∎

Often we need more detailed knowledge of the distance structure of a convolutional code. Let $n_{d_{\text{free}}+i}$ denote the number of weight $d_{\text{free}} + i$ paths which depart from

the allzero path at the root in the code trellis and do not reach the zero state until their termini. We call $n_{d_{\text{free}}+i}$ the $(i+1)$th *Viterbi spectral component* [Vit71]. The sequence

$$n_{d_{\text{free}}+i}, \quad i = 0, 1, 2, \ldots \tag{3.35}$$

is called the *Viterbi weight spectrum* of the convolution encoder. The generating function for the weight spectrum,

$$T(W) = \sum_{i=0}^{\infty} n_{d_{\text{free}}+i} W^{d_{\text{free}}+i} \tag{3.36}$$

is called the *path weight enumerator* and will be studied in depth in Section 3.10.

The *zero state driving information sequence* for a rational generator matrix with a given encoder realized in controller canonical form is the sequence of information tuples that causes the memory elements to be successively filled with zeros. The length of this sequence is at most m, where m is the length of the longest shift register. Denote the zero state driving information sequence starting at time $t+1$ by $\boldsymbol{u}^{\text{zs}}_{(t,t+m]}$. In general, it depends on the encoder state at time t and it can happen that the encoder is driven to the zero state in fewer than m steps. To simplify notations we also use $\boldsymbol{u}^{\text{zs}}_{(t,t+m]}$ as the zero driving information sequence also in these cases. For a polynomial generator matrix of memory m we have $\boldsymbol{u}^{\text{zs}}_{(t,t+m]} = \mathbf{0}$.

As a counterpart to the column distance we have the *row distance* [Cos69].

Definition The jth order *row distance* d^{r}_j of a rational generator matrix of memory m realized in controller canonical form is the minimum of the Hamming weights of the paths $\boldsymbol{v}_{[0,j+m]}$ resulting from information sequences $\boldsymbol{u}_{[0,j+m]} = \boldsymbol{u}_{[0,j]}\boldsymbol{u}^{\text{zs}}_{(j,j+m]}$, where $\boldsymbol{u}_{[0,j]} \neq \mathbf{0}$, and remerging with the allzero path at depth at most $j+m+1$.

Let $G(D)$ be a polynomial generator matrix and let the corresponding semi-infinite matrix $\boldsymbol{G}$ be (3.6). Denote by $\boldsymbol{G}^{\text{r}}_j$ the matrix formed by truncating the semi-infinite matrix $\boldsymbol{G}$ after its first $j+1$ rows, that is,

$$\boldsymbol{G}^{\text{r}}_j = \begin{pmatrix} G_0 & G_1 & \ldots & G_m & & & & \\ & G_0 & G_1 & \ldots & G_m & & & \\ & & G_0 & G_1 & \ldots & G_m & & \\ & & & \ddots & & & \ddots & \\ & & & & G_0 & G_1 & \ldots & G_m \end{pmatrix} \tag{3.37}$$

Suppose that $G(D)$ is realized in controller canonical form. Then we have

$$d^{\text{r}}_j = \min_{\boldsymbol{u}_{[0,j]} \neq \mathbf{0}} \left\{ w_{\text{H}} \left(\boldsymbol{u}_{[0,j]} \boldsymbol{G}^{\text{r}}_j \right) \right\} \tag{3.38}$$

Theorem 3.5 The row distances of a rational generator matrix realized in controller canonical form satisfy the following conditions:

(i)

$$d^{\text{r}}_{j+1} \leq d^{\text{r}}_j, \quad j = 0, 1, 2, \ldots \tag{3.39}$$

(ii)

$$d_j^{\mathrm{r}} > 0, \quad j = 0, 1, 2, \ldots \tag{3.40}$$

(iii) d_j^{r} becomes stationary as j increases.

Proof: (i) Assume that

$$d_j^{\mathrm{r}} = w_{\mathrm{H}}(\boldsymbol{v}_{[0,j+m]}) \tag{3.41}$$

where $\boldsymbol{v}_{[0,j+m]}$ results from an information sequence $\boldsymbol{u}_{[0,j+m]} = \boldsymbol{u}_{[0,j]}\boldsymbol{u}^{\mathrm{zs}}_{(j,j+m]}$ with $\boldsymbol{u}_{[0,j]} \neq \mathbf{0}$ and such that the path remerges with the allzero path at depth at most $j + m + 1$. Then there exists an information tuple $\boldsymbol{u}'_{j+1}$ such that

$$\begin{aligned} d_j^{\mathrm{r}} &= w_{\mathrm{H}}(\boldsymbol{v}_{[0,j+m]}\mathbf{0}) \\ &\geq \min_{\boldsymbol{u}'_{j+1}} \left\{ w_{\mathrm{H}}(\boldsymbol{v}'_{[0,j+1+m]}) \right\} \geq d_{j+1}^{\mathrm{r}} \end{aligned} \tag{3.42}$$

where $\boldsymbol{v}'_{[0,j+1+m]}$ results from an information sequence $\boldsymbol{u}'_{[0,j+1+m]}$ with $\boldsymbol{u}'_{[0,j+1]} = \boldsymbol{u}_{[0,j]}\boldsymbol{u}'_{j+1}$ and such that the path remerges with the allzero path at depth at most $j + m + 2$.

(ii) Assume that $d_j^{\mathrm{r}} = 0$ for some j. Then there is an input sequence $\boldsymbol{u}_{[0,j+m]} = \boldsymbol{u}_{[0,j]}\boldsymbol{u}^{\mathrm{zs}}_{(j,j+m]}$ with $\boldsymbol{u}_{[0,j]} \neq \mathbf{0}$ which produces the output sequence $\boldsymbol{v}_{[0,j+m]} = \mathbf{0}_{[0,j+m]}$ and the zero state. This contradicts $G(D)$ having full rank.

(iii) Follows immediately from (i) and (ii). ∎

We define

$$d_\infty^{\mathrm{r}} \stackrel{\mathrm{def}}{=} \lim_{j\to\infty} d_j^{\mathrm{r}} \tag{3.43}$$

and have the relations

$$0 < d_\infty^{\mathrm{r}} \leq \cdots \leq d_2^{\mathrm{r}} \leq d_1^{\mathrm{r}} \leq d_0^{\mathrm{r}} \tag{3.44}$$

If we think of the row distances in terms of a state-transition diagram for the encoder, it follows that d_0^{r} is the minimum weight of the paths resulting from only one freely chosen nonzero information tuple followed by k zero state driving information tuples, where $\nu_{\min} \leq k \leq m$ and $\nu_{\min} = \min_i\{\nu_i\}$. These paths diverge from the zero state at some time instant and return to the zero state at $k + 1$ time instants later. Higher order row distances are obtained by allowing successively more freedom in finding the minimum-weight path diverging from and returning to the zero state. The jth order row distance d_j^{r} is the minimum weight of paths of length at most $j + m + 1$ branches diverging from and returning to the zero state. Eventually, d_∞^{r} is the minimum weight of paths diverging from and returning to the zero state.

Since the column distance d_i^{c} is the minimum weight of a path of length $i + 1$ branches and with a first branch diverging from the zero state, it is obvious that

$$d_i^{\mathrm{c}} \leq d_j^{\mathrm{r}}, \quad \text{all } i \text{ and } j \tag{3.45}$$

and, thus, that

$$d_0^{\mathrm{c}} \leq d_1^{\mathrm{c}} \leq \cdots \leq d_i^{\mathrm{c}} \leq \cdots \leq d_\infty^{\mathrm{c}} \leq d_\infty^{\mathrm{r}} \leq \cdots \leq d_j^{\mathrm{r}} \leq \cdots \leq d_1^{\mathrm{r}} \leq d_0^{\mathrm{r}} \tag{3.46}$$

Furthermore, if there are no closed circuits of zero weight in the state diagram except the trivial zero-weight self-loop at the zero state, it follows that

$$d_\infty^{\mathrm{c}} = d_\infty^{\mathrm{r}} \tag{3.47}$$

We will give a formal proof of the important equality (3.47) below.

First, we remark that the existence of a zero-weight nontrivial circuit in the state-transition diagram determined by a nonzero subsequence of information digits is equivalent to the existence of an information sequence with infinite weight that is encoded as a finite-weight sequence—a catastrophic encoder.

Theorem 3.6 Let $G(D)$ be a noncatastrophic generator matrix. Then,

$$d_\infty^{\mathrm{c}} = d_\infty^{\mathrm{r}} \tag{3.48}$$

Proof: If $G(D)$ is noncatastrophic, then there exists a codeword $\boldsymbol{v}$ resulting from an information sequence of the form $\boldsymbol{u} = \boldsymbol{u}_{[0,i]}\boldsymbol{u}^{\mathrm{zs}}_{(i,i+m]}\mathbf{0}$, where $\boldsymbol{u}_{[0,i]} \neq \mathbf{0}$, such that

$$\begin{aligned} d_\infty^{\mathrm{c}} &= w_{\mathrm{H}}(\boldsymbol{v}) \\ &= w_{\mathrm{H}}(\boldsymbol{v}_{[0,i+m]}) \geq d_i^{\mathrm{r}} \end{aligned} \tag{3.49}$$

Let k be the least positive integer such that

$$d_k^{\mathrm{r}} = d_{k+1}^{\mathrm{r}} = \cdots = d_\infty^{\mathrm{r}} \tag{3.50}$$

Then $d_i^{\mathrm{r}} > d_k^{\mathrm{r}}$ for $i < k$. By combining (3.46), (3.49), and (3.50) we obtain

$$d_i^{\mathrm{r}} \leq d_\infty^{\mathrm{c}} \leq d_k^{\mathrm{r}} \tag{3.51}$$

It follows that $i \geq k$ and $d_i^{\mathrm{r}} = d_k^{\mathrm{r}}$. Hence,

$$d_\infty^{\mathrm{c}} = d_k^{\mathrm{r}} = d_\infty^{\mathrm{r}} \tag{3.52}$$

■

The row distance could probably blame equality (3.48) for not getting much attention in the literature. However, its significance should not be underestimated. It is easy to calculate and serves as an excellent rejection rule when generator matrices are tested in search for convolutional codes with large free distance.

■ **EXAMPLE 3.3**

The state-transition diagram for the rate $R = 1/2$, binary, convolutional encoder of memory $m = 3$ and encoding matrix $G(D) = (1+D+D^2+D^3 \quad 1+D^2+D^3)$ shown in Fig. 3.2 is given in Fig. 3.3.

The row distance d_0^{r} is obtained from the circuit $000 \to 100 \to 010 \to 001 \to 000$, and $d_1^{\mathrm{r}}, d_2^{\mathrm{r}}, \ldots, d_\infty^{\mathrm{r}}$ are obtained from the circuit $000 \to 100 \to 110 \to$

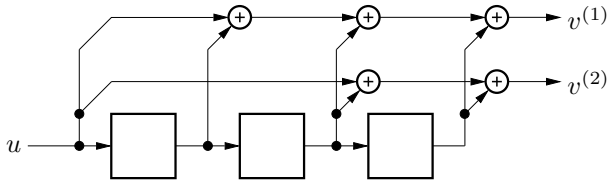
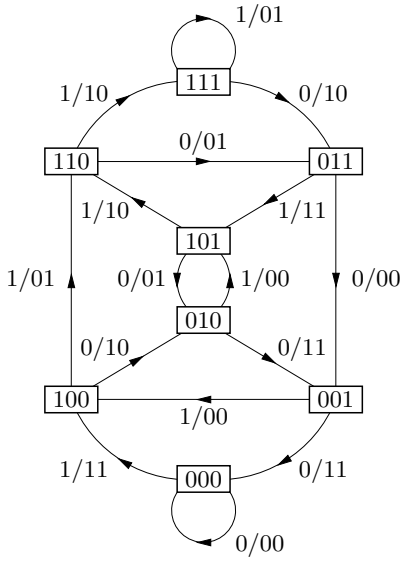


Figure 3.2 A rate $R = 1/2$ ODP convolutional encoder.



$$d_0^{\mathrm{r}} = 7$$
$$d_1^{\mathrm{r}} = d_2^{\mathrm{r}} = \cdots = d_\infty^{\mathrm{r}} = 6$$
$$d_0^{\mathrm{c}} = 2$$
$$d_1^{\mathrm{c}} = d_2^{\mathrm{c}} = 3$$
$$d_3^{\mathrm{c}} = d_4^{\mathrm{c}} = 4$$
$$d_5^{\mathrm{c}} = d_6^{\mathrm{c}} = 5$$
$$d_7^{\mathrm{c}} = d_8^{\mathrm{c}} = \cdots = d_\infty^{\mathrm{c}} = 6$$

Figure 3.3 The state-transition diagram for the encoder in Fig. 3.2.

$011 \to 001 \to 000$. The column distance d_0^{c} is obtained from $000 \to 100$, d_1^{c} from, for example, $000 \to 100 \to 110$, d_2^{c} from $000 \to 100 \to 010 \to 101$, and so on. The row and column distances are shown in Fig. 3.4. We have the free distance $d_{\mathrm{free}} = d_\infty^{\mathrm{c}} = d_\infty^{\mathrm{r}} = 6$ and the distance profile $\boldsymbol{d}^{\mathrm{p}} = (2, 3, 3, 4)$. This ODP code has an *optimum free distance* (OFD), which will be shown in Section 3.5.

3.2 ACTIVE DISTANCES

The column distance introduced in the previous section will not increase any more when the lowest weight path has merged with the allzero path. In this section we shall introduce a family of distances that stay "active" in the sense that we consider only those codewords that do not pass two consecutive zero states [HJZ99]. As we

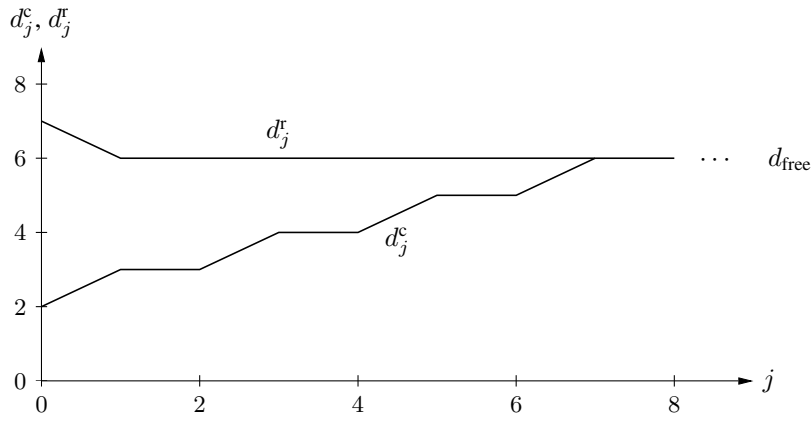


Figure 3.4 Row and column distances for the encoder in Fig. 3.2.

shall see in Section 3.9 these distances are of particular importance when we consider concatenated encoders.

Let the binary m-dimensional vector of b-tuples $\boldsymbol{\sigma}_t$ be the encoder state at depth t of a realization in controller canonical form of the generator matrix and let $\boldsymbol{\sigma}_t^{(i)}$ be the b-tuple representing the contents of position i of the shift registers (counted from the input connections). (When the jth constraint length $\nu_j < m$ for some j, then we set the jth component of $\boldsymbol{\sigma}_t^{(i)}$ to be 0.) Then we have $\boldsymbol{\sigma}_t = \boldsymbol{\sigma}_t^{(1)}\boldsymbol{\sigma}_t^{(2)}\ldots\boldsymbol{\sigma}_t^{(m)}$. To the information sequence $\boldsymbol{u} = \boldsymbol{u}_0\boldsymbol{u}_1\ldots$ corresponds the state sequence $\boldsymbol{\sigma} = \boldsymbol{\sigma}_0\boldsymbol{\sigma}_1\ldots$.

Let $\mathcal{S}_{[t_1,t_2]}^{\boldsymbol{\sigma}_1,\boldsymbol{\sigma}_2}$ denote the set of state sequences $\boldsymbol{\sigma}_{[t_1,t_2]}$ that start in state $\boldsymbol{\sigma}_1$ and terminate in state $\boldsymbol{\sigma}_2$ and do not have two consecutive zero states with zero input in between, i.e.,

$$\mathcal{S}_{[t_1,t_2]}^{\boldsymbol{\sigma}_1,\boldsymbol{\sigma}_2} \stackrel{\text{def}}{=} \{\boldsymbol{\sigma}_{[t_1,t_2]} \mid \boldsymbol{\sigma}_{t_1} = \boldsymbol{\sigma}_1, \boldsymbol{\sigma}_{t_2} = \boldsymbol{\sigma}_2 \text{ and } \boldsymbol{\sigma}_i, \boldsymbol{\sigma}_{i+1} \text{ not both } = \mathbf{0} \text{ for } \boldsymbol{u}_i = \mathbf{0},\ t_1 \le i < t_2\} \tag{3.53}$$

The most important distance parameter for a convolutional generator matrix is the active row distance; it determines the error-correcting capability of the code.

Definition Let $\mathcal{C}$ be a convolutional code encoded by a rational generator matrix $G(D)$ of memory m which is realized in controller canonical form. The *jth order active row distance* is

$$a_j^{\mathrm{r}} \stackrel{\text{def}}{=} \min_{\mathcal{S}_{[0,j+1]}^{\mathbf{0},\boldsymbol{\sigma}}, \boldsymbol{\sigma}_{j+1+i}^{(1,i)}=\mathbf{0},\ 1\le i\le m} \left\{w_{\mathrm{H}}(\boldsymbol{v}_{[0,j+m]})\right\} \tag{3.54}$$

where $\boldsymbol{\sigma}$ denotes any value of the state $\boldsymbol{\sigma}_{j+1}$ such that $\boldsymbol{\sigma}_{j+1}^{(1)} \neq \mathbf{0}$, and $\boldsymbol{\sigma}_{j+1+i}^{(1,i)}$ denotes the i first positions of the shift registers (counted from the input connections), i.e., $\boldsymbol{\sigma}_{j+1+i}^{(1,i)} = \boldsymbol{\sigma}_{j+1+i}^{(1)}\boldsymbol{\sigma}_{j+1+i}^{(2)}\cdots\boldsymbol{\sigma}_{j+1+i}^{(i)}$.

Let $\nu_{\min}$ be the minimum of the constraint lengths ν_i, $i = 1, 2, \ldots, b$, of the generator matrix $G(D)$ of memory m, i.e., $\nu_{\min} = \min_i\{\nu_i\}$ and $m = \max_i\{\nu_i\}$. Then the active row distance of order j is the minimum weight of paths that diverge from the zero state at depth 0 (or stay at the zero state with a nonzero input), for zero inputs possibly "touches" the allzero path only in nonconsecutive zero states at depth k, where $1 + \nu_{\min} \leq k \leq j$, and, finally, remerges with the allzero path at depth ℓ, where $j + 1 + \nu_{\min} \leq \ell \leq j + 1 + m$.

For a polynomial generator matrix realized in controller canonical form we have the following equivalent formulation:

$$a_j^{\mathrm{r}} = \min_{\boldsymbol{u}_j \neq 0,\ \mathcal{S}_{[0,j+1]}^{\mathbf{0},\boldsymbol{\sigma}}} \left\{ w_{\mathrm{H}} \left(\boldsymbol{u}_{[0,j]} \boldsymbol{G}_j^{\mathrm{r}} \right) \right\} \tag{3.55}$$

where $\boldsymbol{\sigma}$ denotes any value of the state $\boldsymbol{\sigma}_{j+1}$ with $\boldsymbol{\sigma}_{j+1}^{(1)} = \boldsymbol{u}_j$ and

$$\boldsymbol{G}_j^{\mathrm{r}} = \begin{pmatrix} G_0 & G_1 & \ldots & G_m & & & \\ & G_0 & G_1 & \ldots & G_m & & \\ & & \ddots & \ddots & & \ddots & \\ & & & G_0 & G_1 & \ldots & G_m \end{pmatrix} \tag{3.56}$$

is a $(j + 1) \times (j + 1 + m)$ truncated version of the semi-infinite matrix $\boldsymbol{G}$ given in (3.6).

Notice that the active row distance sometimes can decrease. As we shall show in Section 3.8, however, in the ensemble of convolutional codes encoded by periodically time-varying generator matrices there exists a convolutional code encoded by a generator matrix such that its active row distance can be lower-bounded by a linearly increasing function.

From the definition the next theorem follows immediately.

Theorem 3.7 (Triangle inequality) Let $G(D)$ be a rational generator matrix with $\nu_{\min} = m$. Then its active row distance satisfies the triangle inequality

$$a_j^{\mathrm{r}} \leq a_i^{\mathrm{r}} + a_{j-i-1-m}^{\mathrm{r}} \tag{3.57}$$

where $j > i + m$ and the sum of the lengths of the paths to the right of the inequality is

$$i + m + 1 + (j - i - m - 1) + m + 1 = j + m + 1 \tag{3.58}$$

that is, equal to the length of the path to the left of the inequality.

Furthermore, we have immediately the following important theorem:

Theorem 3.8 Let $\mathcal{C}$ be a convolutional code encoded by a minimal encoder. Then

$$\min_j\{a_j^{\mathrm{r}}\} = d_{\mathrm{free}} \tag{3.59}$$

The following simple example shows that the triangle inequality in Theorem 3.7 will not hold if we do not include state sequences that contain isolated inner zero states in the definition of $\mathcal{S}_{[t_1,t_2]}^{\boldsymbol{\sigma}_1,\boldsymbol{\sigma}_2}$.

■ **EXAMPLE 3.4**

Consider the memory $m = 1$ encoding matrix

$$G(D) = (1 \quad D) \tag{3.60}$$

The code sequences corresponding to the state sequences $(0, 1, 0, 1, 0)$ and $(0, 1, 1, 1, 0)$ are $(10, 01, 10, 01)$ and $(10, 11, 11, 01)$, respectively. It is easily verified that $a_0^{\mathrm{r}} = 2$, $a_1^{\mathrm{r}} = 4$, and $a_2^{\mathrm{r}} = 4$, which satisfy the triangle inequality

$$a_2^{\mathrm{r}} \leq a_0^{\mathrm{r}} + a_0^{\mathrm{r}} \tag{3.61}$$

If we consider only state sequences without isolated inner zero states, the lowest weight sequence of length 4 would pick up distance 6 and exceed the sum of the weight for two length 2 sequences, which would still be four, violating the triangle inequality.

Remark: If we consider the ensemble of periodically time-varying generator matrices $\boldsymbol{G}$ (or $G(D)$) to be introduced in Section 3.6 and require that the corresponding code sequences consist of only randomly chosen code symbols (i.e., we do not allow transitions from the zero state to itself), then for a given length the set of state sequences defined by $\mathcal{S}_{[t_1,t_2]}^{\boldsymbol{\sigma}_1,\boldsymbol{\sigma}_2}$ is as large as possible.

Next, we will consider an "active" counterpart to the column distance:

Definition Let $\mathcal{C}$ be a convolutional code encoded by a rational generator matrix $G(D)$ of memory m realized in controller canonical form. The *jth order active column distance* is

$$a_j^{\mathrm{c}} \stackrel{\mathrm{def}}{=} \min_{\mathcal{S}_{[0,j+1]}^{\mathbf{0},\boldsymbol{\sigma}}} \left\{ w_{\mathrm{H}}(\boldsymbol{v}_{[0,j]}) \right\} \tag{3.62}$$

where $\boldsymbol{\sigma}$ denotes any encoder state.

For a polynomial generator matrix we have the following equivalent formulation:

$$a_j^{\mathrm{c}} = \min_{\mathcal{S}_{[0,j+1]}^{\mathbf{0},\boldsymbol{\sigma}}} \left\{ w_{\mathrm{H}} \left(\boldsymbol{u}_{[0,j]} \boldsymbol{G}_j^{\mathrm{c}} \right) \right\} \tag{3.63}$$

where $\boldsymbol{\sigma}$ denotes any encoder state and

$$\boldsymbol{G}_j^{\mathrm{c}} = \begin{pmatrix} G_0 & G_1 & \dots & G_m & & & \\ & G_0 & G_1 & \dots & G_m & & \\ & & \ddots & \ddots & & \ddots & \\ & & & G_0 & G_1 & \dots & G_m \\ & & & & G_0 & & G_{m-1} \\ & & & & & \ddots & \vdots \\ & & & & & & G_0 \end{pmatrix} \tag{3.64}$$

is a $(j + 1) \times (j + 1)$ truncated version of the semi-infinite matrix $\boldsymbol{G}$ given in (3.6).

It follows from the definitions that

$$a_j^{\mathrm{c}} \le a_{j-k}^{\mathrm{r}} \tag{3.65}$$

where $k \le \min\{j, \nu_{\min}\}$ and, in particular, if $\nu_{\min} = m \le j$, then we have

$$a_j^{\mathrm{c}} \le a_{j-m}^{\mathrm{r}} \tag{3.66}$$

From (3.65) it follows that when $j \ge \nu_{\min}$ the active column distance of order j is upper-bounded by the active row distance of order $j - \nu_{\min}$, that is, by the minimum weight of paths of length $j+1$ starting at a zero state and terminating at a zero state without passing consecutive zero states in between.

The active column distance a_j^{c} is a nondecreasing function of j. As we will show in Section 3.8, however, in the ensemble of convolutional codes encoded by periodically time-varying generator matrices there exists a convolutional code encoded by a generator matrix such that its active column distance can be lower-bounded by a linearly increasing function.

Definition Let $\mathcal{C}$ be a convolutional code encoded by a rational generator matrix $G(D)$ of memory m. The *jth order active reverse column distance* is

$$a_j^{\mathrm{rc}} \stackrel{\mathrm{def}}{=} \min_{\mathcal{S}_{[m,m+j+1]}^{\boldsymbol{\sigma},\mathbf{0}}} \left\{ w_{\mathrm{H}}(\boldsymbol{v}_{[m,j+m]}) \right\} \tag{3.67}$$

where $\boldsymbol{\sigma}$ denotes any encoder state.

For a polynomial generator matrix we have the following equivalent formulation to (3.67):

$$a_j^{\mathrm{rc}} = \min_{\mathcal{S}_{[m,m+j+1]}^{\boldsymbol{\sigma},\mathbf{0}}} \left\{ w_{\mathrm{H}} \left(\boldsymbol{u}_{[0,j+m]} \boldsymbol{G}_j^{\mathrm{rc}} \right) \right\} \tag{3.68}$$

where $\boldsymbol{\sigma}$ denotes any encoder state and

$$\boldsymbol{G}_j^{\mathrm{rc}} = \begin{pmatrix} G_m & & & \\ G_{m-1} & G_m & & \\ \vdots & G_{m-1} & \ddots & \\ G_0 & \vdots & & G_m \\ & G_0 & & G_{m-1} \\ & & \ddots & \vdots \\ & & & G_0 \end{pmatrix} \tag{3.69}$$

is a $(j+m+1) \times (j+1)$ truncated version of the semi-infinite matrix $\boldsymbol{G}$ given in (3.6).

The active reverse column distance a_j^{rc} is a nondecreasing function of j. As we will show in Section 3.8, however, in the ensemble of convolutional codes encoded by periodically time-varying generator matrices there exists a convolutional code

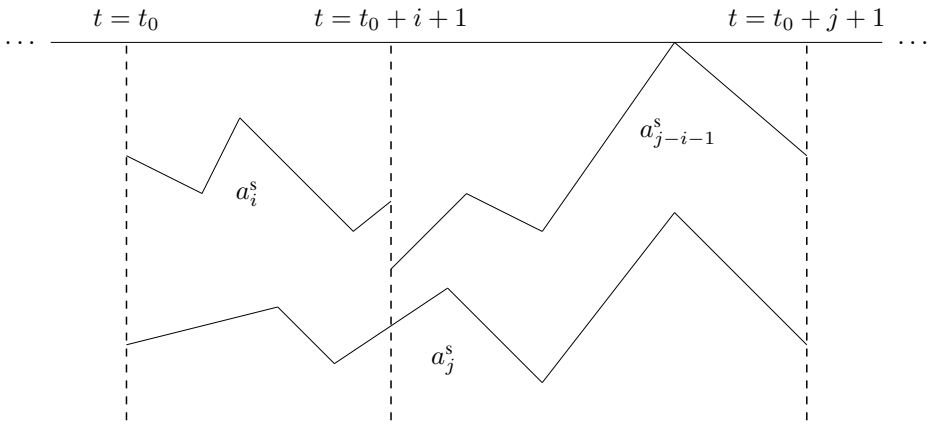


Figure 3.5 An illustration for Theorem 3.9.

encoded by a generator matrix such that its active reverse column distance can be lower-bounded by a linearly increasing function.

Furthermore, the active reverse column distance of a polynomial generator matrix $G(D)$ is equal to the active column distance of the *reciprocal generator matrix*

$$\mathrm{diag}(D^{\nu_1} D^{\nu_2} \dots D^{\nu_b}) G(D^{-1})$$

Definition Let $\mathcal{C}$ be a convolutional code encoded by a rational generator matrix $G(D)$ of memory m. The *jth order active segment distance* is

$$a^{\mathrm{s}}_j \stackrel{\mathrm{def}}{=} \min_{\mathcal{S}^{\boldsymbol{\sigma}_1,\boldsymbol{\sigma}_2}_{[m,m+j+1]}} \left\{ w_{\mathrm{H}}(\boldsymbol{v}_{[m,j+m]}) \right\} \tag{3.70}$$

where $\boldsymbol{\sigma}_1$ and $\boldsymbol{\sigma}_2$ denote any encoder states.

For a polynomial generator matrix we have the following equivalent formulation:

$$a^{\mathrm{s}}_j = \min_{\mathcal{S}^{\boldsymbol{\sigma}_1,\boldsymbol{\sigma}_2}_{[m,m+j+1]}} \left\{ w_{\mathrm{H}} \left(\boldsymbol{u}_{[0,j+m]} \boldsymbol{G}^{\mathrm{s}}_j \right) \right\} \tag{3.71}$$

where $\boldsymbol{\sigma}_1$ and $\boldsymbol{\sigma}_2$ denote any encoder states and $\boldsymbol{G}^{\mathrm{s}}_j = \boldsymbol{G}^{\mathrm{rc}}_j$.

If we consider the segment distances for two sets of consecutive paths of lengths $i + 1$ and $(j - i - 1) + 1$, respectively, then the terminating state of the first path is not necessarily identical to the starting state of the second path (see Fig. 3.5). Hence, the active segment distance for the set of paths of the total length $j + 1$ does not necessarily satisfy the triangle inequality. However, we have immediately the following:

Theorem 3.9 Let $G(D)$ be a generator matrix of memory m. Then its active segment distance satisfies the inequality

$$a_j^{\mathrm{s}} \geq a_i^{\mathrm{s}} + a_{j-i-1}^{\mathrm{s}} \tag{3.72}$$

where $j > i$ and the sum of the lengths of the paths to the right of the inequality is

$$i + 1 + j - i - 1 + 1 = j + 1 \tag{3.73}$$

that is, equal to the length of the path to the left of the inequality.

The active segment distance a_j^{s} is a nondecreasing function of j. As we will show in Section 3.8, however, in the ensemble of convolutional codes encoded by periodically time-varying generator matrices there exists a convolutional code encoded by a generator matrix such that its active segment distance can be lower-bounded by a linearly increasing function.

The *start* of the active segment distance is the largest j for which $a_j^{\mathrm{s}} = 0$ and is denoted j_{s}.

The jth order active row distance is characterized by a fixed number of almost freely chosen information tuples, $j + 1$, followed by a varying number, between $\nu_{\min}$ and m, of zero state driving information tuples ("almost" since we have to avoid consecutive zero states $\boldsymbol{\sigma}_i\boldsymbol{\sigma}_{i+1}$ for $0 \leq i < j + 1$ and ensure that $\boldsymbol{\sigma}_{j+1}^{(1)} \neq \mathbf{0}$). Sometimes we find it useful to consider a corresponding distance between two paths of fixed total length, $j + 1$, but with a varying number of almost freely chosen information tuples. Hence, we introduce the following (final) active distance:

Definition Let $\mathcal{C}$ be a convolutional code encoded by a rational generator matrix $G(D)$ of memory m. The *jth order active burst distance* is

$$a_j^{\mathrm{b}} \stackrel{\mathrm{def}}{=} \min_{\mathcal{S}_{[0,j+1]}^{\mathbf{0},\mathbf{0}}} \left\{ w_{\mathrm{H}}(\boldsymbol{v}_{[0,j]}) \right\} \tag{3.74}$$

where $j \geq j_{\mathrm{b}}$ and $j_{\mathrm{b}+1}$ is the length of the shortest detour from the allzero sequence.

For a polynomial generator matrix we have the following equivalent formulation:

$$a_j^{\mathrm{b}} \stackrel{\mathrm{def}}{=} \min_{\mathcal{S}_{[0,j+1]}^{\mathbf{0},\mathbf{0}}} \left\{ w_{\mathrm{H}}(\boldsymbol{u}_{[0,j]}\boldsymbol{G}_j^{\mathrm{c}}) \right\} \tag{3.75}$$

where $\boldsymbol{G}_j^{\mathrm{c}}$ is given in (3.64).

The active row and burst distances are related via the following inequalities:

$$\begin{aligned} a_j^{\mathrm{b}} &\geq \min_i \{a_{j-\nu_i}^{\mathrm{r}}\} \\ a_j^{\mathrm{r}} &\geq \min_i \{a_{j+\nu_i}^{\mathrm{b}}\} \end{aligned} \tag{3.76}$$

When $\nu_{\min} = m$, we have

$$a_j^{\mathrm{b}} = \begin{cases} \text{undefined}, & 0 \leq j < m \\ a_{j-m}^{\mathrm{r}}, & j \geq m \end{cases} \tag{3.77}$$

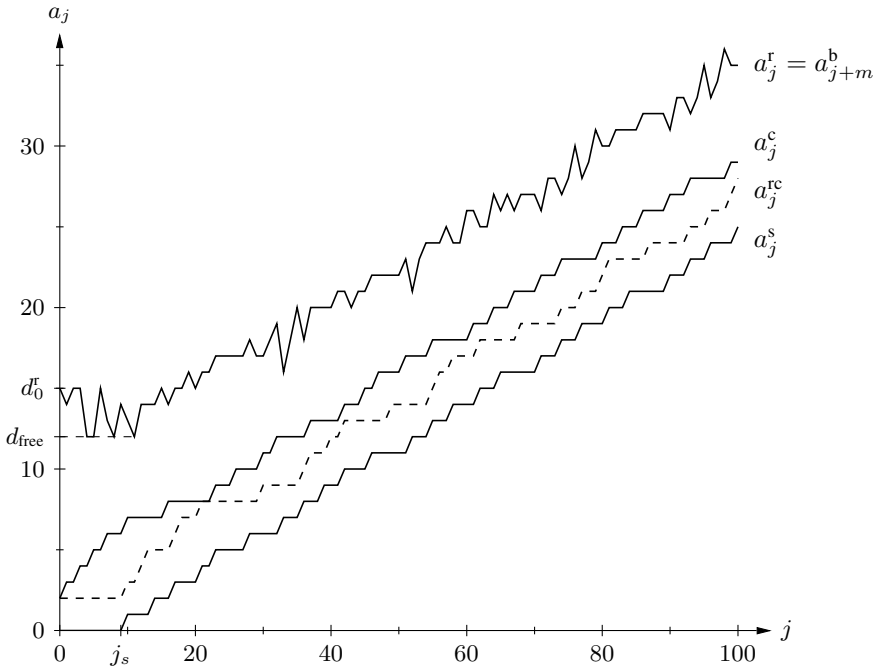


Figure 3.6 The active row distance for the encoding matrix in Example 3.5.

For a convolutional code encoded by a minimal encoder

$$\min_j \left\{ a_j^{\mathrm{b}} \right\} = d_{\mathrm{free}} \tag{3.78}$$

From the definition it follows that the active burst distance satisfies the triangle inequality.

■ **EXAMPLE 3.5**

In Fig. 3.6 we show the active distances for the encoding matrix $G(D) = (1 + D + D^2 + D^3 + D^7 + D^8 + D^9 + D^{11} \quad 1 + D^2 + D^3 + D^7 + D^8 + D^9 + D^{11})$. Notice that the active row distance of the 0th order, a_0^{r}, is identical to the row distance of the 0th order, $d_0^{\mathrm{r}} = 15$, which upper-bounds $d_{\mathrm{free}} = 12$, and the start $j_{\mathrm{s}} = 9$.

From the definitions it follows that the active distances are encoder properties, not code properties. However, it also follows that the active distances are invariants over the set of minimal-basic (or canonical if rational) encoding matrices for a code $\mathcal{C}$. Hence, in the sequel when we consider active distances for convolutional codes, it is understood that these distances are evaluated for the corresponding minimal-basic (canonical) encoding matrices.

3.3 PROPERTIES OF CONVOLUTIONAL CODES VIA THE ACTIVE DISTANCES

We define the *correct path* through a trellis to be the path determined by the encoded information sequence and we call the (encoder) states along the correct path *correct states*. Then we define an *incorrect segment* to be a segment starting in a correct state $\boldsymbol{\sigma}_{t_1}$ and terminating in a correct state $\boldsymbol{\sigma}_{t_2}$, $t_1 < t_2$, such that it differs from the correct path at some but not necessarily all states within this interval. Let $e_{[k,\ell)}$ denote the number of errors in the error pattern $\boldsymbol{e}_{[k,\ell)}$, where $\boldsymbol{e}_{[k,\ell)} = \boldsymbol{e}_k \boldsymbol{e}_{k+1} \ldots \boldsymbol{e}_{\ell-1}$.

For a convolutional code $\mathcal{C}$ with a generator matrix of memory m, consider any incorrect segment between two arbitrary correct states, $\boldsymbol{\sigma}_{t_1}$ and $\boldsymbol{\sigma}_{t_2}$. A minimum distance (MD) decoder can output an incorrect segment between $\boldsymbol{\sigma}_{t_1}$ and $\boldsymbol{\sigma}_{t_2}$ only if there exists a segment of length $j+1$ c-tuples, $\nu_{\min} \leq j < t_2 - t_1$, between these two states such that the number of channel errors $e_{[t_1,t_2)}$ within this interval is at least $a_j^{\mathrm{b}}/2$. Thus, we have the following:

Theorem 3.10 A convolutional code $\mathcal{C}$ encoded by a rational generator matrix of memory m can correct all error patterns $\boldsymbol{e}_{[t_1,t_2)}$ that correspond to incorrect segments between any two correct states, $\boldsymbol{\sigma}_{t_1}$ and $\boldsymbol{\sigma}_{t_2}$, and satisfy

$$e_{[t_1+k,t_1+1+i)} < a_{i-k}^{\mathrm{b}}/2 \tag{3.79}$$

for $0 \leq k \leq t_2 - t_1 - \nu_{\min} - 1$, $k + \nu_{\min} \leq i \leq t_2 - t_1 - 1$.

We have immediately the following:

Corollary 3.11 A convolutional code $\mathcal{C}$ encoded by a rational generator matrix of memory m and smallest constraint length $\nu_{\min} = m$ can correct all error patterns $\boldsymbol{e}_{[t_1,t_2)}$ that correspond to incorrect segments between any two correct states, $\boldsymbol{\sigma}_{t_1}$ and $\boldsymbol{\sigma}_{t_2}$, and satisfy

$$e_{[t_1+k,t_1+1+i)} < a_{i-k-m}^{\mathrm{r}}/2 \tag{3.80}$$

for $0 \leq k \leq t_2 - t_1 - m - 1$, $k + m \leq i \leq t_2 - t_1 - 1$.

Both the active column distance and the active reverse column distance are important parameters when we study the error-correcting capability of a convolutional code. As a counterpart to Theorem 3.10 we have the following:

Theorem 3.12 Let $\mathcal{C}$ be a convolutional code encoded by a rational generator matrix of memory m and let $\boldsymbol{e}_{[t_1,t_2)}$ be an error sequence between the two correct states $\boldsymbol{\sigma}_{t_1}$ and $\boldsymbol{\sigma}_{t_2}$. A minimum distance decoder will output a correct state $\boldsymbol{\sigma}_t$ at depth t, $t_1 < t < t_2$, if

$$\begin{aligned} e_{[i,t)} &< a_{t-i-1}^{\mathrm{c}}/2, \ t_1 \leq i < t \\ e_{[t,j)} &< a_{j-t-1}^{\mathrm{rc}}/2, \ t < j \leq t_2 \end{aligned} \tag{3.81}$$

Proof: Assume without loss of generality that the correct path is the allzero path. The weight of any path of length $t - i$ diverging from the correct path at depth i,

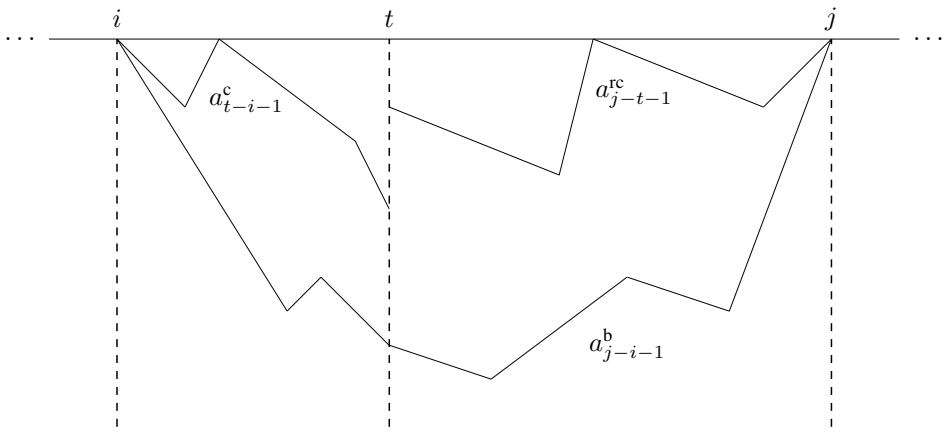


Figure 3.7 An illustration used in the proof of Theorem 3.12.

$i < t$, and not having two consecutive zero states is lower-bounded by a^{c}_{t-i-1} (see Fig. 3.7). Similarly, the weight of any path of length $j-t$, $j > t$, remerging with the correct path at depth j and not having two consecutive zero states is lower-bounded by a^{rc}_{j-t-1}. Hence, if $e_{[i,t)} < a^{\mathrm{c}}_{t-i-1}/2$ and $e_{[t,j)} < a^{\mathrm{rc}}_{j-t-1}/2$, then $\boldsymbol{\sigma}_t$ must be correct. ■

We also have the following inequality:

$$a^{\mathrm{c}}_{t-i-1} + a^{\mathrm{rc}}_{j-t-1} \leq a^{\mathrm{b}}_{j-i-1} \tag{3.82}$$

(see Fig. 3.7).

■ **EXAMPLE 3.6**

Assume that the binary, rate $R = 1/2$, memory $m = 2$ convolutional encoding matrix $G(D) = (1 + D + D^2 \quad 1 + D^2)$ is used to communicate over a binary symmetric channel (BSC) and that we have the error pattern

$$\boldsymbol{e}_{[0,20)} = 10000100000000010000000010000000000100001 \tag{3.83}$$

or, equivalently,

$$\boldsymbol{e}_{[0,20)}(D) = (10) + (01)D^2 + (01)D^7 + (10)D^{12} + (10)D^{17} + (01)D^{19} \tag{3.84}$$

The active distances for the encoding matrix are given in Fig. 3.8. From Theorem 3.10 it is easily seen that if we assume that $\boldsymbol{\sigma}_0$ is a correct state and that there exists a $t' \geq 20$ such that $\boldsymbol{\sigma}_{t'}$ is a correct state, then, despite the fact that the number of channel errors $e_{[0,20)} = 6 > d_{\text{free}} = 5$, the error pattern (3.83) is corrected by a minimum distance decoder.

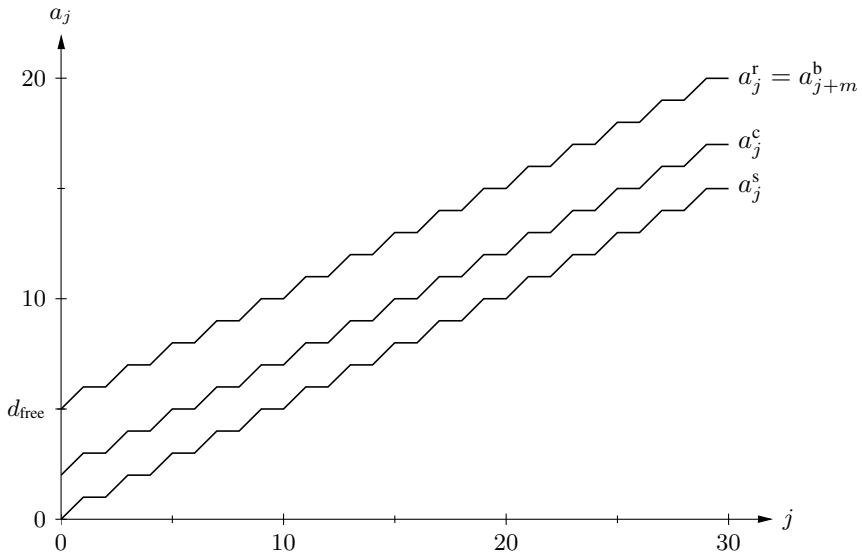


Figure 3.8 The active distances for the encoding matrix in Example 3.6.

The error pattern

$$\boldsymbol{e}'_{[0,20)} = 10100100000000000000000000000000101001 \tag{3.85}$$

or, equivalently,

$$\boldsymbol{e}'_{[0,20)}(D) = (10) + (10)D + (01)D^2 + (10)D^{17} + (10)D^{18} + (01)D^{19} \tag{3.86}$$

also contains six channel errors but with a different distribution; we have three channel errors in both the prefix and suffix 101001. Since $\nu_{\min} = m = 2$ and the active row distance $a_0^{\text{r}} = 5$, the active burst distance $a_2^{\text{b}} = 5$. Hence, Theorem 3.10 does not imply that the error pattern (3.85) is corrected by a minimum distance decoder; the states $\boldsymbol{\sigma}_1, \boldsymbol{\sigma}_2, \boldsymbol{\sigma}_{18}$, and $\boldsymbol{\sigma}_{19}$ will in fact be erroneous states. However, from Theorem 3.12 it follows that if $\boldsymbol{\sigma}_0$ is a correct state and if there exists a $t' \geq 20$ such that $\boldsymbol{\sigma}_{t'}$ is a correct state, then at least $\boldsymbol{\sigma}_{10}$ is also a correct state.

3.4 LOWER BOUND ON THE DISTANCE PROFILE

In this section we shall derive a lower bound on the distance profile for the *ensemble* of convolutional codes. First we give the following:

Definition The *ensemble* $\mathcal{E}(b, c, m)$ of binary, rate $R = b/c$ convolutional codes of memory m is the set of convolutional codes encoded by generator matrices $\boldsymbol{G}$ in

which each digit in each of the matrices G_i, $0 \leq i \leq m$, is chosen independently with probability $1/2$ and where $\boldsymbol{G}$ is given in (3.6).

Lemma 3.13 Let $\mathcal{E}(b, c, m)$ be the ensemble of binary, rate $R = b/c$ convolutional codes with memory m. The first $(m+1)c$ code symbols on a path diverging from the allzero path are independent and equally likely to be 0 and 1.

Proof: Assume without loss of generality that a path diverges from the allzero path at the root, that is, $\boldsymbol{u}_0 \neq \mathbf{0}$. Then we have

$$\boldsymbol{v}_j = \boldsymbol{u}_j G_0 + \boldsymbol{u}_{j-1} G_1 + \cdots + \boldsymbol{u}_0 G_j \tag{3.87}$$

where $0 \leq j \leq m$. Since $\boldsymbol{v}_j$ is a vector determined by $G_0, G_1, \ldots, G_{j-1}$ plus $\boldsymbol{u}_0 G_j$, where $\boldsymbol{u}_0 \neq \mathbf{0}$ and G_j is equally likely to be any $b \times c$ binary matrix, $\boldsymbol{v}_j$ assumes each of its 2^c possible values with the same probability. Furthermore, it is independent of the previous code symbols $\boldsymbol{v}_0, \boldsymbol{v}_1, \ldots, \boldsymbol{v}_{j-1}$ for $1 \leq j \leq m$ and the proof is complete. ∎

We will now prove the following:

Theorem 3.14 There exists a binary, rate $R = b/c$ convolutional code with a generator matrix of memory m whose column distances satisfy

$$d_j^{\mathrm{c}} > \rho c(j+1) \tag{3.88}$$

for $0 \leq j \leq m$ and where ρ, $0 \leq \rho < 1/2$, is the Gilbert-Varshamov parameter, that is, the solution of

$$h(\rho) = 1 - R \tag{3.89}$$

where $h(\rho)$ is the binary entropy function (1.22) (cf. p. 8).

Before proving this theorem we show the optimum distance profile together with the lower bound (3.88) for a rate $R = 1/2$ ($\rho = 0.11$), binary convolutional code in Fig. 3.9.

Proof: Let $d_{0,\ell}$, $0 < \ell \leq 2^b$, denote the weights of the branches with $\boldsymbol{u}_0 \neq \mathbf{0}$ stemming from the root. From Lemma 3.13 it follows that for the ensemble $\mathcal{E}(b, c, m)$ we have

$$P(d_{0,\ell} = k) = \binom{c}{k} \left(\frac{1}{2}\right)^c \tag{3.90}$$

for $0 \leq k \leq c$ and $0 < \ell \leq 2^b$. Consider all paths stemming from the ℓth initial branch with $\boldsymbol{u}_0 \neq \mathbf{0}$ for $0 < \ell \leq 2^b$. That is, these paths begin at depth 1 and not at the root!

Now let us introduce the random walk $S_0 = 0, S_1, S_2, \ldots$, where

$$S_j = \sum_{i=1}^{j} Z_i, \quad j = 1, 2, \ldots \tag{3.91}$$

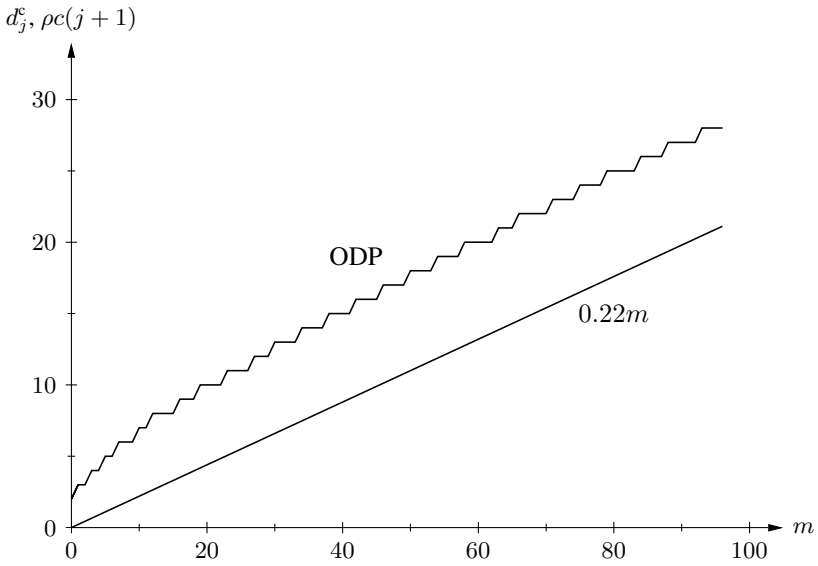


Figure 3.9 ODP and its lower bound for rate $R = 1/2$ and $0 \leq m \leq 96$.

where

$$Z_i = \sum_{\ell=1}^{c} Y_{i\ell} \tag{3.92}$$

with $Y_{i\ell} = \alpha$ if the ℓth symbol on the $(i+1)$th branch is 1 and $Y_{i\ell} = \beta$ if it is 0 and an absorbing barrier which will be specified later. According to Lemma 3.13,

$$P(Y_{il} = \alpha) = P(Y_{il} = \beta) = 1/2, \;\; 0 \leq i \leq m, \;\; 1 \leq l \leq c \tag{3.93}$$

(see Example B.2). The random walk begins at depth 1 with $S_0 = 0$. Let w_j be the weight of the corresponding path of length $j+1$ branches starting at the root. Then equation (3.91) can be rewritten as

$$S_j = (w_j - k)\alpha + (jc - w_j + k)\beta, \;\; j = 0, 1, \ldots \tag{3.94}$$

Furthermore, we notice that if we choose $\alpha = 1$ and $\beta = 0$, then $k + S_j$ should stay above the straight line in Fig. 3.9, that is,

$$k + S_j = w_j > \rho c(j+1) \tag{3.95}$$

for $0 \leq j \leq m$ for all paths in order to guarantee the existence of a code with a generator matrix of memory m satisfying our bound. Since it is more convenient to analyze a situation with an absorbing barrier parallel to the *j*-axis, we choose

$$\begin{aligned} \alpha &= 1 - \rho \\ \beta &= -\rho \end{aligned} \tag{3.96}$$

By inserting (3.96) into (3.94), we obtain

$$\begin{aligned} k + S_j &= w_j(1-\rho) - (jc - w_j)\rho \\ &= w_j - \rho cj,\ j = 0, 1, \ldots \end{aligned} \tag{3.97}$$

Thus, from (3.97) it follows that the random walk S_j stays above an absorbing barrier at $c\rho - k$, that is,

$$S_j > c\rho - k \tag{3.98}$$

or, equivalently,

$$k + S_j > c\rho \tag{3.99}$$

if and only if

$$w_j > \rho c(j+1) \tag{3.100}$$

for all paths with $\boldsymbol{u}_0 \neq \mathbf{0}$.

In the sequel we consider only $k > k_0 = \lfloor c\rho \rfloor$. If $k \leq k_0$, then we say that the random walk was absorbed already at the beginning. The probability of this event, P_0, is upper-bounded by

$$P_0 \leq (2^b - 1) \sum_{k=0}^{k_0} \binom{c}{k} \left(\frac{1}{2}\right)^c \tag{3.101}$$

To estimate the probability that the random walk is absorbed, we introduce a random variable (indicator function) ξ_{ji} such that $\xi_{ji} = 1$ if the random walk for a path leading to the ith node at depth $j+1$ will cross the barrier for the first time at depth $j+1$ and $\xi_{ji} = 0$ otherwise.

The average of the random variable ξ_{ji} is equal to the probability that the random walk S_j hits or drops below the barrier $c\rho - k$ for the first time at depth $j+1$, that is,

$$E[\xi_{ji}] = P(S_n > c\rho - k,\ 0 \leq n < j,\ \text{and } S_j = v, v \leq c\rho - k) \tag{3.102}$$

Let $P(k)$ denote the probability that the random walk is absorbed by the barrier at $c\rho - k$. Then, summing (3.102) over $1 \leq j \leq m$ and $1 \leq i \leq 2^{bj}$, we get

$$P(k) = \sum_{j=1}^{m} \sum_{i=1}^{2^{bj}} E[\xi_{ji}] < \sum_{j=1}^{\infty} E[\xi_{ji}] 2^{bj} \tag{3.103}$$

Using the notation from Appendix B (B.55) we have

$$E[\xi_{ji}] = \sum_{v \leq c\rho - k} f_{0,j}(c\rho - k, v) \tag{3.104}$$

where $f_{0,j}(c\rho - k, v)$ denotes the probability that the random walk is not absorbed at depth $i \leq j$ by the barrier at $(c\rho - k)$ and $S_j = v$. Hence, the right side of inequality (3.103) can be rewritten as

$$\sum_{j=1}^{\infty} \sum_{v \leq c\rho - k} f_{0,j}(c\rho - k, v) 2^{bj}$$

In Appendix B we prove (cf. Corollary B.7) that for any $\lambda_0 < 0$ such that the moment-generating function of the random variable Z_i given by (3.92) equals

$$g(\lambda_0) = E[2^{\lambda_0 Z_i}] = 2^{-b} \tag{3.105}$$

and such that $g'(\lambda_0) \leq 0$ we have

$$\sum_{j=1}^{\infty} \sum_{v \leq c\rho - k} f_{0,j}(c\rho - k, v) 2^{bj} \leq 2^{-\lambda_0(c\rho - k)} \tag{3.106}$$

Choose

$$\lambda_0 = \log \frac{\rho}{1-\rho} \tag{3.107}$$

Then we have (see also Example B.2)

$$\begin{aligned} g(\lambda_0) &= \left(\frac{1}{2} 2^{\lambda_0 \alpha} + \frac{1}{2} 2^{\lambda_0 \beta} \right)^c \\ &= \left(\frac{1}{2} 2^{(1-\rho) \log \frac{\rho}{1-\rho}} + \frac{1}{2} 2^{-\rho \log \frac{\rho}{1-\rho}} \right)^c \\ &= \left(\frac{1}{2} \left(\left(\frac{\rho}{1-\rho} \right)^{1-\rho} + \left(\frac{\rho}{1-\rho} \right)^{-\rho} \right) \right)^c \\ &= \left(\frac{1}{2} \rho^{-\rho} (1-\rho)^{1-\rho} \right)^c \\ &= 2^{(-1+h(\rho))c} = 2^{-Rc} = 2^{-b} \end{aligned} \tag{3.108}$$

and

$$\begin{aligned} g'(\lambda) \mid_{\lambda=\lambda_0} &= c \left(\frac{1}{2} 2^{\lambda_0 \alpha} + \frac{1}{2} 2^{\lambda_0 \beta} \right)^{c-1} \left(\frac{\alpha}{2} 2^{\lambda_0 \alpha} + \frac{\beta}{2} 2^{\lambda_0 \beta} \right) \ln 2 \\ &= c \left(\frac{1}{2} 2^{\lambda_0 \alpha} + \frac{1}{2} 2^{\lambda_0 \beta} \right)^{c-1} \left(\frac{1-\rho}{2} 2^{(1-\rho) \log \frac{\rho}{1-\rho}} - \frac{\rho}{2} 2^{-\rho \log \frac{\rho}{1-\rho}} \right) \ln 2 \\ &= c \left(\frac{1}{2} 2^{\lambda_0 \alpha} + \frac{1}{2} 2^{\lambda_0 \beta} \right)^{c-1} \left(\frac{\rho}{1-\rho} \right)^{-\rho} \left(\frac{\rho}{2} - \frac{\rho}{2} \right) \ln 2 = 0 \end{aligned} \tag{3.109}$$

Combining (3.106) and (3.108) we get (cf. (B.81))

$$P(k) \leq 2^{-\lambda_0(c\rho - k)}, \; k > k_0 \tag{3.110}$$

where we have used the facts that $\lambda_0 < 0$ and $g'(\lambda_0) = 0$. Since

$$2^{-\lambda_0(c\rho - k)} \geq 1 \tag{3.111}$$

for $k \leq k_0 = \lfloor c\rho \rfloor$, we can further upper-bound P_0 by

$$P_0 \leq (2^b - 1) \sum_{k=0}^{k_0} \binom{c}{k} \left(\frac{1}{2} \right)^c 2^{-\lambda_0(c\rho - k)} \tag{3.112}$$

Finally, using (3.112) and summing over all nodes at depth 1 with $\boldsymbol{u}_0 \neq \mathbf{0}$, we upper-bound the probability of absorption by

$$\begin{aligned} P_0 + (2^b - 1) \sum_{k=k_0+1}^{c} \binom{c}{k} \left(\frac{1}{2}\right)^c P(k) &\leq (2^b - 1) \sum_{k=0}^{c} \binom{c}{k} \left(\frac{1}{2}\right)^c 2^{-\lambda_0(c\rho - k)} \\ &= (2^{b-c} - 2^{-c}) 2^{c\rho \log \frac{1-\rho}{\rho}} \sum_{k=0}^{c} \binom{c}{k} 2^{-k \log \frac{1-\rho}{\rho}} \\ &= (2^{(R-1)c} - 2^{-c}) 2^{c\rho \log \frac{1-\rho}{\rho}} \left(1 + 2^{-\log \frac{1-\rho}{\rho}}\right)^c \\ &= (2^{-h(\rho)c} - 2^{-c}) \left((1-\rho)^{-(1-\rho)} \rho^{-\rho}\right)^c = (2^{-h(\rho)c} - 2^{-c}) 2^{ch(\rho)} \\ &= 1 - 2^{-c(1-h(\rho))} = 1 - 2^{-b} < 1 \end{aligned} \tag{3.113}$$

for $0 < \rho < 1/2$.

Since the probability of absorption is strictly less than 1, there exists a convolutional code with a generator matrix of memory m whose distance profile satisfies the bound and, hence, the proof is complete. ∎

We have immediately the following:

Corollary 3.15 There exists a binary, rate $R = b/c$ convolutional code with a generator matrix of memory m whose minimum distance satisfies

$$d_{\min} > \rho c(m+1) \tag{3.114}$$

where ρ is the Gilbert-Varshamov parameter.

3.5 UPPER BOUNDS ON THE FREE DISTANCE

We will now prove an upper bound on the free distance based on Plotkin's bound for block codes.

For the sake of completeness we start by proving Plotkin's upper bound on the minimum distance for block codes.

Lemma 3.16 (Plotkin) The minimum distance for any binary block code of M codewords and block length N satisfies

$$d_{\min} \leq \left\lfloor \frac{NM}{2(M-1)} \right\rfloor \tag{3.115}$$

Proof: Consider an arbitrary column in the list of M codewords. Suppose that the symbol 0 occurs n_0 times in this column. The contribution of this column to the sum of the distances between all ordered pairs of codewords is $2n_0(M - n_0)$, whose maximum value $M^2/2$ is achieved if and only if $n_0 = M/2$. Summing the distances

over all N columns, we have at most $NM^2/2$. Since $d_{\min}$ is the minimum distance between a pair of codewords and since there are $M(M-1)$ ordered pairs, we have

$$M(M-1)d_{\min} \leq \frac{NM^2}{2} \tag{3.116}$$

and the proof is complete. ∎

Heller [Hel68, LaM70] used Plotkin's bound for block codes to obtain a surprisingly tight bound on the free distance for convolutional codes. We regard Heller's bound as an immediate consequence of the following:

Theorem 3.17 The free distance for any binary, rate $R = b/c$ convolutional code encoded by a minimal-basic encoding matrix of memory m and overall constraint length ν satisfies

$$d_{\text{free}} \leq \min_{i \geq 1} \left\{ \left\lfloor \frac{(m+i)c}{2(1-2^{\nu-b(m+i)})} \right\rfloor \right\} \tag{3.117}$$

Proof: Any rate $R = b/c$ convolutional code can be encoded by a minimal-basic encoding matrix whose realization in controller canonical form has 2^ν encoder states. Consider $2^{b(m+i)}$, $i = 1, 2, \ldots$, information sequences. There exist $2^{b(m+i)}/2^\nu$ information sequences starting in the zero state leading to the zero state. The corresponding code sequences constitute a block code with $M = 2^{b(m+i)-\nu}$ codewords and block length $N = (m+i)c$ for $i = 1, 2, \ldots$. Apply Lemma 3.16 for $i = 1, 2, \ldots$ and the proof is complete. ∎

Since $\nu \leq bm$ we have the next corollary.

Corollary 3.18 (Heller) The free distance for any binary, rate $R = b/c$ convolutional code encoded by a minimal-basic encoding matrix of memory m satisfies

$$d_{\text{free}} \leq \min_{i \geq 1} \left\{ \left\lfloor \frac{(m+i)c}{2(1-2^{-bi})} \right\rfloor \right\} \tag{3.118}$$

From Heller's bound the next corollary follows immediately.

Corollary 3.19 (Heller asymptotic bound) The free distance for any binary, rate $R = b/c$ convolutional code encoded by a minimal generator matrix of memory m satisfies

$$\lim_{m \to \infty} \frac{d_{\text{free}}}{mc} \leq \frac{1}{2} \tag{3.119}$$

In fact, Heller's bound is valid not only for convolutional codes but also for a larger class of codes, viz., the so-called class of nonlinear, time-varying trellis codes. For convolutional codes (i.e, linear, time-constant trellis codes), we can use Griesmer's bound for block codes to obtain slight improvements for some memories [Gri60].

Lemma 3.20 (Griesmer bound for linear block codes) For a binary, linear, rate $R = K/N$ block code with minimum distance $d_{\min}$ we have

$$\sum_{i=0}^{K-1} \left\lceil \frac{d_{\min}}{2^i} \right\rceil \leq N \tag{3.120}$$

Proof: Without loss of generality, we assume that the first row of the generator matrix G is $(111\ldots10\ldots0)$ with $d_{\min}$ ones. Every other row has at least $\lceil d_{\min}/2 \rceil$ ones or $\lceil d_{\min}/2 \rceil$ zeros in the first $d_{\min}$ positions. Hence, in either case they have at least $\lceil d_{\min}/2 \rceil$ ones in the remaining $N - d_{\min}$ positions. Therefore, the residual code with respect to the first row is a rate $R = (K-1)/(N - d_{\min})$ code with minimum distance $\geq \lceil d_{\min}/2 \rceil$. Using induction on K completes the proof. ∎

Consider the binary, linear block code in the proof of Theorem 3.17 with $M = 2^{b(m+i)-\nu}$ codewords and block length $N = (m+i)c$, $i = 1, 2, \ldots$. The number of information symbols

$$K = \log M \geq b(m+i) - \nu \geq bi, \quad i = 1, 2, \ldots \tag{3.121}$$

The minimum distances for these block codes, $i = 1, 2, \ldots$, must satisfy the Griesmer bound. Hence, we have the following:

Theorem 3.21 (Griesmer bound for convolutional codes) The free distance for any binary, rate $R = b/c$ convolutional code encoded by a minimal-basic encoding matrix of memory m satisfies

$$\sum_{j=0}^{bi-1} \left\lceil \frac{d_{\text{free}}}{2^j} \right\rceil \leq (m+i)c \tag{3.122}$$

for $i = 1, 2, \ldots$.

■ **EXAMPLE 3.7**

(i) Let $R = 1/2$ and $m = 16$. Since $\min_{i \geq 1}\{\lfloor (16+i)/(1-2^{-i}) \rfloor\} = 21$, it follows from the Heller bound that any binary, rate $R = 1/2$ convolutional code with a generator matrix with memory $m = 16$ must have $d_{\text{free}} \leq 21$. Since $\sum_{j=0}^{3} \lceil 21/2^j \rceil = 41 \not\leq (16+4)2 = 40$, any binary, rate $R = 1/2$ convolutional code with a generator matrix with memory $m = 16$ must have $d_{\text{free}} < 21$. The Griesmer bound gives an improvement by one. Such a code exists with $d_{\text{free}} = 20$.

(ii) Let $R = 1/2$ and $m = 18$. Since $\min_{i \geq 1}\{\lfloor (18+i)/(1-2^{-i}) \rfloor\} = 23$, it follows from the Heller bound that any binary, rate $R = 1/2$ convolutional code with a generator matrix with memory $m = 18$ must have $d_{\text{free}} \leq 23$. Since $\sum_{j=0}^{i-1} \lceil 23/2^j \rceil \leq (18+i)2$ for all $i \geq 1$, the Griesmer bound for convolutional codes does not give any improvement over the Heller bound for convolutional codes in this case. The largest free distance for any binary,

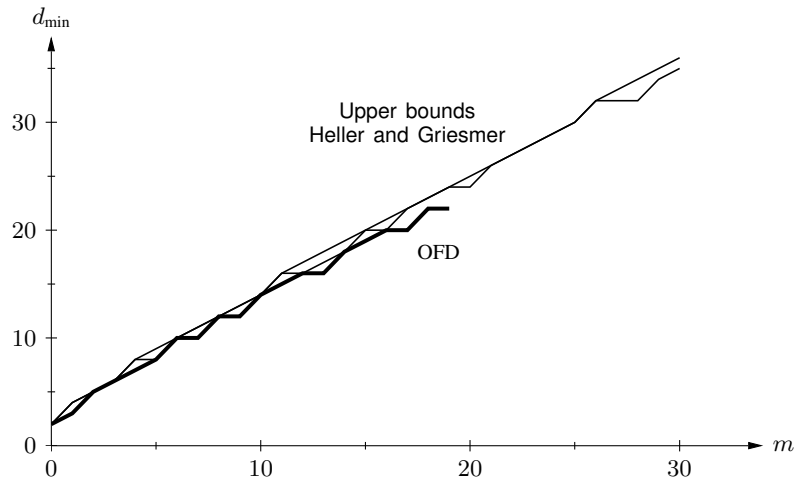


Figure 3.10 Heller and Griesmer upper bounds on the free distance and the free distance for rate $R = 1/2$ OFD codes.

rate $R = 1/2$ convolutional code with a generator matrix with memory $m = 18$ has been determined by exhaustive search to be $d_{\text{free}} = 22$. Thus, the Griesmer bound is not tight.

For rate $R = 1/2$, binary convolutional codes we have calculated Heller's upper bound for memories $1 \leq m \leq 39$. Using Griesmer's bound, we obtained an improved bound for some values of m by one or two (the bold values):

m	0	1	2	3	4	**5**	6	7	8	9	10	11	**12**	...
Heller	2	4	5	6	8	**9**	10	11	12	13	14	16	**17**	...
Griesmer	2	4	5	6	8	**8**	10	11	12	13	14	16	**16**	...

...	**13**	**14**	15	**16**	17	18	19	**20**	21	22	23	24	25	...
...	**18**	**19**	20	**21**	22	23	24	**25**	26	27	28	29	30	...
...	**17**	**18**	20	**20**	22	23	24	**24**	26	27	28	29	30	...

...	26	**27**	**28**	**29**	**30**	**31**	**32**	**33**	34	**35**	**36**	**37**	38	**39**
...	32	**33**	**34**	**35**	**36**	**37**	**38**	**39**	40	**41**	**42**	**43**	44	**45**
...	32	**32**	**32**	**34**	**35**	**36**	**37**	**38**	40	**40**	**41**	**42**	44	**44**

These results are shown in Fig. 3.10 and compared with the free distance for optimum free distance (OFD) fixed convolutional codes. The upper bound is surprisingly tight.

We notice, for example, that for memory $m = 4$ there is a gap. The Griesmer bound implies that $d_{\text{free}} \leq 8$, but the best rate $R = 1/2$ convolutional code of memory $m = 4$ has $d_{\text{free}} = 7$. However, there exists a rate $R = 2/4$ convolutional code of

the same complexity, that is, memory $m = 2$, with $d_{\text{free}} = 8$. Its encoding matrix is [JoW98]

$$G(D) = \begin{pmatrix} D + D^2 & 1 + D & D^2 & 1 + D + D^2 \\ 1 & D + D^2 & 1 + D & 1 + D + D^2 \end{pmatrix} \tag{3.123}$$

and the first few Viterbi spectral components are $12, 0, 52, 0, 260$. The optimum free distance rate $R = 1/2$ convolutional code has encoding matrix

$$G(D) = (1 + D + D^4 \quad 1 + D^2 + D^3 + D^4) \tag{3.124}$$

and the first few Viterbi spectral components $2, 3, 4, 16, 37$.

Next we will consider convolutional codes encoded by polynomial, systematic encoding matrices and derive counterparts to Heller's and Griesmer's bounds for convolutional codes encoded by general generator matrices.

Theorem 3.22 (Heller) The free distance for any binary, rate $R = b/c$ convolutional code encoded by a polynomial, systematic encoding matrix of memory m satisfies

$$d_{\text{free}} \leq \min_{i \geq 1} \left\{ \left\lfloor \frac{(m(1-R) + i)c}{2(1 - 2^{-bi})} \right\rfloor \right\} \tag{3.125}$$

Proof: Consider a convolutional code encoded by a polynomial(!), systematic encoding matrix realized in controller canonical form. The code sequences of length $(m + i)c$ starting at the zero state and remerging at the zero state constitute a block code of $M = 2^{b(m+i)-\nu}$ codewords, where ν is the overall constraint length. Append to the shift registers $m - \nu_i, \quad 1 \leq i \leq b$, memory elements *without* connecting them to the output. The corresponding block code encoded by this new encoder is an expurgated block code whose minimum distance is at least as large as the minimum distance of the original block code. In order to obtain the merger at the zero, we now have to feed the encoder with m allzero b-tuples. Hence, the expurgated block code has only $M_{\text{exp}} = 2^{bi}$ codewords, and each of them has b zeros as the first code symbols on each of the m last branches before merging with the allzero state. The "effective" block length is reduced to

$$N = (m + i)c - mb = (m(1 - R) + i)c \tag{3.126}$$

Applying Lemma 3.16 completes the proof. ∎

From Theorem 3.22 follows immediately the systematic counterpart to Corollary 3.19:

Corollary 3.23 (Heller asymptotic bound) The free distance for any binary, rate $R = b/c$ convolutional code encoded by a polynomial, systematic encoding matrix of memory m satisfies

$$\lim_{m \to \infty} \frac{d_{\text{free}}}{mc} \leq \frac{1 - R}{2} \tag{3.127}$$

Finally, by using (3.126) in the proof of Theorem 3.21, we obtain the following:

Theorem 3.24 (Griesmer bound for convolutional codes) The free distance for any rate $R = b/c$ convolutional code encoded by a polynomial, systematic encoding matrix of memory m satisfies

$$\sum_{j=0}^{bi-1} \left\lceil \frac{d_{\text{free}}}{2^j} \right\rceil \leq (m(1-R)+i)c \tag{3.128}$$

for $i = 1, 2, \ldots$.

3.6 TIME-VARYING CONVOLUTIONAL CODES

So far we have considered only *time-invariant* or *fixed* convolutional codes, that is, convolutional codes encoded by time-invariant generator matrices. When it is too difficult to analyze the performance of a communication system using time-invariant convolutional codes, we can often obtain powerful results if we study time-varying convolutional codes instead.

Assuming polynomial generator matrices, we have

$$\boldsymbol{v}_t = \boldsymbol{u}_t G_0 + \boldsymbol{u}_{t-1} G_1 + \cdots + \boldsymbol{u}_{t-m} G_m \tag{3.129}$$

where G_i, $0 \leq i \leq m$, is a binary $b \times c$ time-invariant matrix.

In general, a rate $R = b/c$, binary convolutional code can be *time-varying*. Then (3.129) becomes

$$\boldsymbol{v}_t = \boldsymbol{u}_t G_0(t) + \boldsymbol{u}_{t-1} G_1(t) + \cdots + \boldsymbol{u}_{t-m} G_m(t) \tag{3.130}$$

where $G_i(t)$, $i = 0, 1, \ldots, m$, is a binary $b \times c$ time-varying matrix. In Fig. 3.11 we illustrate a general time-varying polynomial convolutional encoder. As a counterpart to the semi-infinite matrix $\boldsymbol{G}$ given in (3.6), we have

$$\boldsymbol{G}_t = \begin{pmatrix} G_0(t) & G_1(t+1) & \cdots & G_m(t+m) & & \\ & G_0(t+1) & G_1(t+2) & \cdots & G_m(t+1+m) & \\ & & \ddots & \ddots & & \ddots \end{pmatrix} \tag{3.131}$$

Remark: With a slight abuse of terminology we call for simplicity a time-varying polynomial transfer function matrix a *generator matrix*, although it might not have full rank.

We have the general *ensemble of binary, rate $R = b/c$, time-varying convolutional codes* with generator matrices of memory m in which each digit in each of the matrices $G_i(t)$ for $0 \leq i \leq m$ and $t = 0, 1, 2, \ldots$ is chosen independently and is equally likely to be 0 and 1.

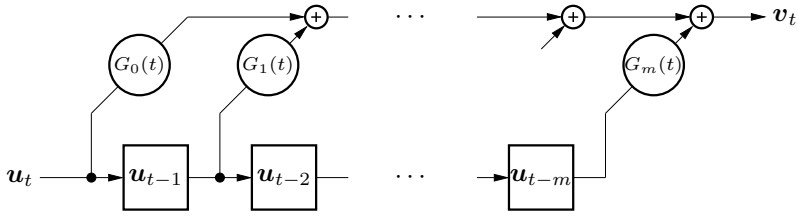


Figure 3.11 A general time-varying polynomial convolutional encoder.

As a special case of the ensemble of time-varying convolutional codes, we have the ensemble of binary, rate $R = b/c$, *periodically* time-varying convolutional codes encoded by a polynomial generator matrix $\boldsymbol{G}_t$ (3.131) of memory m and *period* T in which each digit in each of the matrices $G_i(t) = G_i(t+T)$ for $0 \le i \le m$ and $t = 0, 1, \ldots, T-1$ is chosen independently and is equally likely to be 0 and 1. We denote this ensemble $\mathcal{E}(b, c, m, T)$.

The ensemble $\mathcal{E}(b, c, m)$ of (time-invariant) convolutional codes that we encountered in Section 3.4 can be considered as the special case $\mathcal{E}(b, c, m, 1)$, and the ensemble of general time-varying convolutional codes defined above will be denoted $\mathcal{E}(b, c, m, \infty)$.

Before we define the active distances for periodically time-varying convolutional codes encoded by time-varying polynomial generator matrices, we introduce the following sets of information sequences, where we always assume that $t_1 \le t_2$.

Let $\mathcal{U}^{\mathrm{r}}_{[t_1-m,t_2+m]}$ denote the set of information sequences $\boldsymbol{u}_{t_1-m}\boldsymbol{u}_{t_1-m+1}\cdots\boldsymbol{u}_{t_2+m}$ such that the first m and the last m subblocks are zero and such that they do not contain $m+1$ consecutive zero subblocks, that is,

$$\begin{aligned}\mathcal{U}^{\mathrm{r}}_{[t_1-m,t_2+m]} \stackrel{\mathrm{def}}{=} \{&\boldsymbol{u}_{[t_1-m,t_2+m]} \mid \boldsymbol{u}_{[t_1-m,t_1-1]} = \boldsymbol{0},\\ &\boldsymbol{u}_{[t_2+1,t_2+m]} = \boldsymbol{0},\ \text{and}\ \boldsymbol{u}_{[i,i+m]} \neq \boldsymbol{0},\ t_1 - m \le i \le t_2\}\end{aligned} \tag{3.132}$$

Let $\mathcal{U}^{\mathrm{c}}_{[t_1-m,t_2]}$ denote the set of information sequences $\boldsymbol{u}_{t_1-m}\boldsymbol{u}_{t_1-m+1}\ldots\boldsymbol{u}_{t_2}$ such that the first m subblocks are zero and such that they do not contain $m+1$ consecutive zero subblocks, that is,

$$\begin{aligned}\mathcal{U}^{\mathrm{c}}_{[t_1-m,t_2]} \stackrel{\mathrm{def}}{=} \{&\boldsymbol{u}_{[t_1-m,t_2]} \mid \boldsymbol{u}_{[t_1-m,t_1-1]} = \boldsymbol{0},\\ &\text{and}\ \boldsymbol{u}_{[i,i+m]} \neq \boldsymbol{0},\ t_1 - m \le i \le t_2 - m\}\end{aligned} \tag{3.133}$$

Let $\mathcal{U}^{\mathrm{rc}}_{[t_1-m,t_2+m]}$ denote the set of information sequences $\boldsymbol{u}_{t_1-m}\boldsymbol{u}_{t_1-m+1}\ldots\boldsymbol{u}_{t_2+m}$ such that the last m subblocks are zero and such that they do not contain $m+1$ consecutive zero subblocks, that is,

$$\begin{aligned}\mathcal{U}^{\mathrm{rc}}_{[t_1-m,t_2+m]} \stackrel{\mathrm{def}}{=} \{&\boldsymbol{u}_{[t_1-m,t_2+m]} \mid \boldsymbol{u}_{[t_2+1,t_2+m]} = \boldsymbol{0},\\ &\text{and}\ \boldsymbol{u}_{[i,i+m]} \neq \boldsymbol{0},\ t_1 - m < i \le t_2\}\end{aligned} \tag{3.134}$$

Let $\mathcal{U}^{\mathrm{s}}_{[t_1-m,t_2]}$ denote the set of information sequences $\boldsymbol{u}_{t_1-m}\boldsymbol{u}_{t_1-m+1}\ldots\boldsymbol{u}_{t_2}$ such that they do not contain $m+1$ consecutive zero subblocks, that is,

$$\mathcal{U}^{\mathrm{s}}_{[t_1-m,t_2]} \stackrel{\mathrm{def}}{=} \{\boldsymbol{u}_{[t_1-m,t_2]} \mid \boldsymbol{u}_{[i,i+m]} \neq \boldsymbol{0}, t_1-m<i<t_2-m\} \tag{3.135}$$

Next, we introduce the $(j+m+1)\times(j+1)$ truncated, periodically time-varying generator matrix of memory m and period T:

$$\boldsymbol{G}_{[t,t+j]} = \begin{pmatrix} G_m(t) & & & \\ G_{m-1}(t) & G_m(t+1) & & \\ \vdots & G_{m-1}(t+1) & \ddots & \\ G_0(t) & \vdots & \ddots & G_m(t+j) \\ & G_0(t+1) & & G_{m-1}(t+j) \\ & & \ddots & \vdots \\ & & & G_0(t+j) \end{pmatrix} \tag{3.136}$$

where $G_i(t) = G_i(t+T)$ for $0 \le i \le m$.

We are now well-prepared to generalize the definitions of the active distances for convolutional codes encoded by polynomial generator matrices to time-varying convolutional codes encoded by polynomial time-varying generator matrices:

Definition Let $\mathcal{C}$ be a periodically time-varying convolutional code encoded by a periodically time-varying polynomial generator matrix of memory m and period T.

The *jth order active row distance* is

$$a^{\mathrm{r}}_j \stackrel{\mathrm{def}}{=} \min_t \min_{\mathcal{U}^{\mathrm{r}}_{[t-m,t+j+m]}} \{w_{\mathrm{H}}(\boldsymbol{u}_{[t-m,t+j+m]}\boldsymbol{G}_{[t,t+j+m]})\} \tag{3.137}$$

The *jth order active column distance* is

$$a^{\mathrm{c}}_j \stackrel{\mathrm{def}}{=} \min_t \min_{\mathcal{U}^{\mathrm{c}}_{[t-m,t+j]}} \{w_{\mathrm{H}}(\boldsymbol{u}_{[t-m,t+j]}\boldsymbol{G}_{[t,t+j]})\} \tag{3.138}$$

The *jth order active reverse column distance* is

$$a^{\mathrm{rc}}_j \stackrel{\mathrm{def}}{=} \min_t \min_{\mathcal{U}^{\mathrm{rc}}_{[t-m,t+j]}} \{w_{\mathrm{H}}(\boldsymbol{u}_{[t-m,t+j]}\boldsymbol{G}_{[t,t+j]})\} \tag{3.139}$$

The *jth order active segment distance* is

$$a^{\mathrm{s}}_j \stackrel{\mathrm{def}}{=} \min_t \min_{\mathcal{U}^{\mathrm{s}}_{[t-m,t+j]}} \{w_{\mathrm{H}}(\boldsymbol{u}_{[t-m,t+j]}\boldsymbol{G}_{[t,t+j]})\} \tag{3.140}$$

For a periodically time-varying convolutional code encoded by a periodically time-varying, noncatastrophic, polynomial generator matrix with active row distance a^{r}_j we define its free distance by a generalization of (3.59)

$$d_{\mathrm{free}} \stackrel{\mathrm{def}}{=} \min_j\{a^{\mathrm{r}}_j\} \tag{3.141}$$

In the following two sections we will derive lower bounds on the free distance and on the active distances. There we need the following:

Theorem 3.25 Consider a periodically time-varying, rate $R = b/c$, polynomial generator matrix of memory m and period T represented by $\boldsymbol{G}_t$, where $\boldsymbol{G}_t$ is given in (3.131).

(i) Let the information sequences be restricted to the set $\mathcal{U}^{\mathrm{r}}_{[t-m,t+j+m]}$. Then the code symbols in the segment $\boldsymbol{v}_{[t,t+j+m]}$ are mutually independent and equiprobable over the ensemble $\mathcal{E}(b,c,m,T)$ for all j, $0 \le j < T$.

(ii) Let the information sequences be restricted to the set $\mathcal{U}^{\mathrm{c}}_{[t-m,t+j]}$. Then the code symbols in the segment $\boldsymbol{v}_{[t,t+j]}$ are mutually independent and equiprobable over the ensemble $\mathcal{E}(b,c,m,T)$ for all j, $0 \le j < \max\{m+1, T\}$.

(iii) Let the information sequences be restricted to the set $\mathcal{U}^{\mathrm{rc}}_{[t-m,t+j]}$. Then the code symbols in the segment $\boldsymbol{v}_{[t,t+j]}$ are mutually independent and equiprobable over the ensemble $\mathcal{E}(b,c,m,T)$ for all j, $0 \le j < \max\{m+1, T\}$.

(iv) Let the information sequences be restricted to the set $\mathcal{U}^{\mathrm{s}}_{[t-m,t+j]}$. Then the code symbols in the segment $\boldsymbol{v}_{[t,t+j]}$ are mutually independent and equiprobable over the ensemble $\mathcal{E}(b,c,m,T)$ for all j, $0 \le j < T$.

Proof: It follows immediately that for $0 \le j < T$ the code tuples $\boldsymbol{v}_i$, $i = t, t+1, \ldots, t+j$, are mutually independent and equiprobable in all four cases. Hence, the proof of (iv) is complete. In cases (ii) and (iii) it remains to show that the statements also hold for $T \le j \le m$ when $m \ge T$.

(ii) Consider the information sequences in the set $\mathcal{U}^{\mathrm{c}}_{[t-m,t+j]}$, where $0 \le j \le m$. Let $t \le i \le t+j$. Then, in the expression

$$\boldsymbol{v}_i = \boldsymbol{u}_i G_0(i) + \boldsymbol{u}_{i-1} G_1(i) + \cdots + \boldsymbol{u}_{i-m} G_m(i) \tag{3.142}$$

there exists a k, $0 \le k \le m$, such that at least one of the b-tuples $\boldsymbol{u}_{i-k}$ is nonzero and all the previous b-tuples $\boldsymbol{u}_{i-k'}$, $k < k' \le m$, are zero. Hence, $\boldsymbol{v}_i$ and $\boldsymbol{v}_{i'}$, $t \le i < i' \le t+j$, are mutually independent and equiprobable. This completes the proof of (ii).

(iii) Consider the information sequences in the set $\mathcal{U}^{\mathrm{rc}}_{[t-m,t+j]}$, where $0 \le j \le m$. Let $t \le i \le t+j$; then, in (3.142) at least one of the b-tuples $\boldsymbol{u}_{i-k}$, $0 \le k \le m$, is nonzero, and all the following b-tuples $\boldsymbol{u}_{i-k'}$, $0 \le k' < k$, are zero. Hence, $\boldsymbol{v}_i$ and $\boldsymbol{v}_{i'}$, $t \le i < i' \le t+j$, are mutually independent and equiprobable.

(i) For the information sequences in $\mathcal{U}^{\mathrm{r}}_{[t-m,t+j+m]}$ it remains to show that $\boldsymbol{v}_i$ and $\boldsymbol{v}_{i'}$ are mutually independent and equiprobable also for $T \le i' - i < T + m$. From the definition of $\mathcal{U}^{\mathrm{r}}_{[t-m,t+j+m]}$ it follows that $\boldsymbol{u}_{[t-m,t-1]} = \boldsymbol{0}$, $\boldsymbol{u}_t \ne \boldsymbol{0}$, $\boldsymbol{u}_{t+j} \ne \boldsymbol{0}$, and $\boldsymbol{u}_{[t+j+1,t+j+m]} = \boldsymbol{0}$. For $j = T$, we can choose, for example, $\boldsymbol{u}_{[t-m,t+m]} = \boldsymbol{u}_{[t+T-m,t+T+m]} \in \mathcal{U}^{\mathrm{r}}_{[t-m,t+T+m]}$ which implies that $\boldsymbol{v}_{[t,t+m]} = \boldsymbol{v}_{[t+T,t+T+m]}$. However, for $T - m \le j < T$, $\boldsymbol{v}_i$, $t \le i < t+m$, and $\boldsymbol{v}_{i'}$, $t+j < i' \le t+j+m$, are mutually independent and equiprobable. ∎

From Theorem 3.25 the next corollary follows immediately.

Corollary 3.26 Consider a rate $R = b/c$ polynomial generator matrix of memory m represented by $\boldsymbol{G}$, where $\boldsymbol{G}$ is given in (3.6).

(i) Let the information sequences be restricted to the set $\mathcal{U}^{\text{c}}_{[t-m,t+j]}$. Then the code symbols in the segment $\boldsymbol{v}_{[t,t+j]}$ are mutually independent and equiprobable over the ensemble $\mathcal{E}(b, c, m, 1)$ for all j, $0 \leq j \leq m$.

(ii) Let the information sequences be restricted to the set $\mathcal{U}^{\text{rc}}_{[t-m,t+j+m]}$. Then the code symbols in the segment $\boldsymbol{v}_{[t,t+j]}$ are mutually independent and equiprobable over the ensemble $\mathcal{E}(b, c, m, 1)$ for all j, $0 \leq j \leq m$.

3.7 LOWER BOUND ON THE FREE DISTANCE

In this section we will derive a lower bound on the free distance for the ensemble of periodically time-varying convolutional codes. This bound is due to Costello [Cos74], but our proof is slightly different. Our goal is to find a nontrivial upper bound on the probability that $d_{\text{free}} < d$, that is, to prove that for the ensemble of periodically time-varying convolutional codes $P(d_{\text{free}} < d) < 1$, since then we know that at least one code exists within our ensemble that has $d_{\text{free}} \geq d$.

First, we combine the definition of the free distance (3.141) with the definition of the jth order active row distance (3.137) and obtain

$$d_{\text{free}} = \min_{t} \min_{j} \min_{\mathcal{U}^{\text{r}}_{[t-m,t+j+m]}} \{w_{\text{H}}(\boldsymbol{u}_{[t-m,t+j+m]}\boldsymbol{G}_{[t,t+j+m]})\} \tag{3.143}$$

Then we consider the ensemble of periodically time-varying convolutional codes encoded by polynomial generator matrices with the information sequences restricted to $\mathcal{U}^{\text{r}}_{[t_1-m,t_2+m]}$. The probability that the free distance for a randomly chosen code in this ensemble is less than d is equal to the probability that for at least one $\boldsymbol{u}_{[t-m,t+j+m]} \in \mathcal{U}^{\text{r}}_{[t-m,t+j+m]}$, $t = 0, 1, \ldots, T-1$ and $j = 0, 1, 2, \ldots$, we have

$$w_{\text{H}}(\boldsymbol{u}_{[t-m,t+j+m]}\boldsymbol{G}_{[t,t+j+m]}) < d \tag{3.144}$$

Thus, we have

$$\begin{aligned}
&P(d_{\text{free}} < d) \\
&\quad < P\left(\min_{0\le t<T}\min_{0\le j<\infty}\min_{\mathcal{U}^{\text{r}}_{[t-m,t+j+m]}}\{w_{\text{H}}(\boldsymbol{u}_{[t-m,t+j+m]}\boldsymbol{G}_{[t,t+j+m]})\} < d\right) \\
&\quad < P\left(\min_{0\le t<T}\min_{0\le j<T}\min_{\mathcal{U}^{\text{r}}_{[t-m,t+j+m]}}\{w_{\text{H}}(\boldsymbol{u}_{[t-m,t+j+m]}\boldsymbol{G}_{[t,t+j+m]})\} < d\right) \\
&\quad +P\left(\min_{0\le t<T}\min_{j\ge T}\min_{\mathcal{U}^{\text{r}}_{[t-m,t+j+m]}}\{w_{\text{H}}(\boldsymbol{u}_{[t-m,t+j+m]}\boldsymbol{G}_{[t,t+j+m]})\} < d\right) \\
&\quad < TP\left(\min_{0\le j<T}\min_{\mathcal{U}^{\text{r}}_{[t-m,t+j+m]}}\{w_{\text{H}}(\boldsymbol{u}_{[t-m,t+j+m]}\boldsymbol{G}_{[t,t+j+m]})\} < d\right) \\
&\quad +TP\left(\min_{j\ge T}\min_{\mathcal{U}^{\text{r}}_{[t-m,t+j+m]}}\{w_{\text{H}}(\boldsymbol{u}_{[t-m,t+j+m]}\boldsymbol{G}_{[t,t+j+m]})\} < d\right) \qquad (3.145)
\end{aligned}$$

where the second and third inequalities follow from the union bound.

In the sequel we will call code sequences

$$\boldsymbol{v}_{[t,t+j+m]} = \boldsymbol{u}_{[t-m,t+j+m]}\boldsymbol{G}_{[t,t+j+m]} \qquad (3.146)$$

where $\boldsymbol{u}_{[t-m,t+j+m]} \in \mathcal{U}^{\text{r}}_{[t-m,t+j+m]}$ for *incorrect sequences*. Then it follows from Theorem 3.25 that for $0 \le j < T$ the incorrect sequences are sequences of mutually independent, equiprobable, binary symbols.

For any fixed t, the set $\mathcal{U}^{\text{r}}_{[t-m,t+j+m]}$ contains $2^b - 1$ sequences for $j = 0$ and at most $2^{(j-1)b}(2^b - 1)^2$ sequences for $j \ge 1$. We use $2^{(j+1)b}$ as an upper bound on the cardinality of $\mathcal{U}^{\text{r}}_{[t-m,t+j+m]}$.

First, we consider only incorrect sequences $\boldsymbol{v}_{[t,t+j+m]}$ whose lengths are at most $T + m$. The probability for each of these sequences is $2^{-(j+m+1)c}$ and there are $\binom{(j+m+1)c}{i}$ ways of choosing exactly i ones among the $(j + m + 1)c$ code symbols. Hence, we have

$$\begin{aligned}
&P\left(\min_{\mathcal{U}^{\text{r}}_{[t-m,t+j+m]}}\{w_{\text{H}}(\boldsymbol{u}_{[t-m,t+j+m]}\boldsymbol{G}_{[t,t+j+m]})\} < d\right) \\
&\quad < 2^{(j+1)b}\sum_{i=0}^{d-1}\binom{(j+m+1)c}{i}2^{-(j+m+1)c} \qquad (3.147)
\end{aligned}$$

for $0 \le j < T$ and $t = 0, 1, 2, \ldots$.

Using the union bound we can obtain an upper bound on

$$P\left(\min_{j<T}\min_{\mathcal{U}^{\text{r}}_{[t-m,t+j+m]}}\{w_{\text{H}}(\boldsymbol{u}_{[t-m,t+j+m]}\boldsymbol{G}_{[t,t+j]})\} < d\right)$$

by summing the probabilities for all incorrect sequences of lengths $m+j+1 \le T+m$ with weights less than d. Thus, we have

$$\begin{aligned} & P\left(\min_{0\le j<T} \min_{\mathcal{U}^{\mathrm{r}}_{[t-m,t+j+m]}} \{w_{\mathrm{H}}(\boldsymbol{u}_{[t-m,t+j+m]}\boldsymbol{G}_{[t,t+j]})\} < d\right) \\ & < \sum_{j=0}^{T-1}\sum_{i=0}^{d-1} 2^{(j+1)b}\binom{(j+m+1)c}{i}2^{-(j+m+1)c} \\ & < \sum_{j=0}^{\infty}\sum_{i=0}^{d-1} 2^{(j+1)b}\binom{(j+m+1)c}{i}2^{-(j+m+1)c} \end{aligned} \tag{3.148}$$

We use the substitution

$$k = (j+m+1)c \tag{3.149}$$

and upper-bound (3.148) by summing over $k = 0, 1, 2, \ldots$,

$$\begin{aligned} & P\left(\min_{0\le j<T} \min_{\mathcal{U}^{\mathrm{r}}_{[t-m,t+j+m]}} \{w_{\mathrm{H}}(\boldsymbol{u}_{[t-m,t+j+m]}\boldsymbol{G}_{[t,t+j]})\} < d\right) \\ & < 2^{-mb}\sum_{i=0}^{d-1}\sum_{k=0}^{\infty}\binom{k}{i}2^{k(R-1)} \end{aligned} \tag{3.150}$$

Let

$$x = 2^{R-1}, \frac{1}{2} < x < 1 \tag{3.151}$$

and rearrange (3.150) as

$$\begin{aligned} & P\left(\min_{0\le j<T} \min_{\mathcal{U}^{\mathrm{r}}_{[t-m,t+j+m]}} \{w_{\mathrm{H}}(\boldsymbol{u}_{[t-m,t+j+m]}\boldsymbol{G}_{[t,t+j]})\} < d\right) \\ & < 2^{-mb}\sum_{i=0}^{d-1}\frac{x^i}{i!}\sum_{k=0}^{\infty}k(k-1)\ldots(k-i+1)x^{k-i} \\ & = 2^{-mb}\sum_{i=0}^{d-1}\frac{x^i}{i!}\left(\sum_{k=0}^{\infty}x^k\right)^{(i)} = 2^{-mb}\sum_{i=0}^{d-1}\frac{x^i}{i!}\left(\frac{1}{1-x}\right)^{(i)} \\ & = 2^{-mb}\frac{1}{1-x}\sum_{i=0}^{d-1}\left(\frac{x}{1-x}\right)^i = 2^{-mb}\frac{1}{1-x}\frac{(\frac{x}{1-x})^d-1}{\frac{x}{1-x}-1} \\ & < \frac{2^{-mb}}{(2x-1)(x^{-1}-1)^d} \end{aligned} \tag{3.152}$$

Using (3.151) we obtain

$$\begin{aligned} & P\left(\min_{0\le j<T} \min_{\mathcal{U}^{\mathrm{r}}_{[t-m,t+j+m]}} \{w_{\mathrm{H}}(\boldsymbol{u}_{[t-m,t+j+m]}\boldsymbol{G}_{[t,t+j]})\} < d\right) \\ & < \frac{2^{-mb}}{(2^R-1)(2^{1-R}-1)^d} \end{aligned} \tag{3.153}$$

Next, we will consider sequences of lengths greater than $T + m$, that is, sequences for which $j \geq T$. Then we have

$$\begin{aligned} P\left(\min_{j\geq T} \min_{\mathcal{U}^{\mathrm{r}}_{[t-m,t+j+m]}} \{w_{\mathrm{H}}(\boldsymbol{u}_{[t-m,t+j+m]}\boldsymbol{G}_{[t,t+j+m]})\} < d\right) \\ < P\left(\min_{\mathcal{U}^{\mathrm{c}}_{[t-m,t+T-1]}} \{w_{\mathrm{H}}(\boldsymbol{u}_{[t-m,t+T-1]}\boldsymbol{G}_{[t,t+T-1]})\} < d\right) \\ < 2^{Tb}\sum_{i=0}^{d-1}\binom{Tc}{i}2^{-Tc} \end{aligned} \tag{3.154}$$

where the first inequality follows from the fact that the weight of a sequence is a nondecreasing function of the length of the sequence. To obtain the second inequality, we use 2^{Tb} as an upper bound on the cardinality of $\mathcal{U}^{\mathrm{c}}_{[t-m,t+T-1]}$.

We will now interrupt our derivation and prove the following useful lemma:

Lemma 3.27 For $0 < \gamma < 1/2$,

$$\sum_{i=0}^{\lfloor \gamma n \rfloor}\binom{n}{i} < 2^{h(\gamma)n} \tag{3.155}$$

where $h(\gamma)$ is the binary entropy function (1.22).

Proof: Since $0 < \gamma < 1/2$, we have

$$\begin{aligned} \sum_{i=0}^{\lfloor \gamma n \rfloor}\binom{n}{i} &< \sum_{i=0}^{n}\binom{n}{i}\left(\frac{1-\gamma}{\gamma}\right)^{\gamma n - i} \\ &= \left(\frac{1-\gamma}{\gamma}\right)^{\gamma n}\sum_{i=0}^{n}\binom{n}{i}\left(\frac{\gamma}{1-\gamma}\right)^{i} = (1-\gamma)^{\gamma n}\gamma^{-\gamma n}\left(1+\frac{\gamma}{1-\gamma}\right)^{n} \\ &= (1-\gamma)^{\gamma n - n}\gamma^{-\gamma n} = \left(\gamma^{-\gamma}(1-\gamma)^{-(1-\gamma)}\right)^{n} \\ &= 2^{h(\gamma)n} \end{aligned} \tag{3.156}$$

■

From Lemma 3.27 it follows that

$$\sum_{i=0}^{d-1}\binom{Tc}{i} < 2^{h(\frac{d-1}{Tc})Tc} < 2^{h(\frac{d}{Tc})Tc} \tag{3.157}$$

Hence, we obtain

$$\begin{aligned} P\left(\min_{j\geq T} \min_{\mathcal{U}^{\mathrm{r}}_{[t-m,t+j+m]}} \{w_{\mathrm{H}}(\boldsymbol{u}_{[t-m,t+j+m]}\boldsymbol{G}_{[t,t+j+m]})\} < d\right) \\ < 2^{(h(\frac{d}{Tc})+R-1)Tc} \end{aligned} \tag{3.158}$$

By combining (3.145), (3.153), and (3.158) we obtain

$$P(d_{\text{free}} < d) < T\left(\frac{2^{-mb}}{(2^R-1)(2^{1-R}-1)^d} + 2^{(h(\frac{d}{Tc})+R-1)Tc}\right) \tag{3.159}$$

Let us now choose the period $T \gg m$, $T = m^2$, say, and choose $d = \widehat{d} < cm$, where $\widehat{d}$ satisfies

$$\begin{aligned} &m^2\left(\frac{2^{-mb}}{(2^R-1)(2^{1-R}-1)^{\widehat{d}}} + 2^{(h(\frac{\widehat{d}}{m^2c})+R-1)m^2c}\right) \\ &\quad < m^2\left(\frac{2^{-mb}}{(2^R-1)(2^{1-R}-1)^{\widehat{d}}} + 2^{(h(\frac{1}{m})+R-1)m^2c}\right) < 1 \end{aligned} \tag{3.160}$$

For large memories m such a $\widehat{d}$ always exists. From (3.160) it follows that

$$\begin{aligned} \widehat{d} &> \frac{-mb}{\log(2^{1-R}-1)} - \frac{\log(2^R-1) + \log(m^{-2} - 2^{(h(\frac{1}{m})+R-1)m^2c})}{\log(2^{1-R}-1)} \\ &= \frac{-mb}{\log(2^{1-R}-1)} - O(\log m) \end{aligned} \tag{3.161}$$

and, finally, we have proved the following:

Theorem 3.28 (Costello bound) There exists a binary, periodically time-varying, rate $R = b/c$ convolutional code with a polynomial generator matrix of memory m that has a free distance satisfying the inequality

$$\frac{d_{\text{free}}}{mc} \geq \frac{R}{-\log(2^{1-R}-1)} + O\left(\frac{\log m}{m}\right) \tag{3.162}$$

Since the overall constraint length

$$\nu \leq mb \tag{3.163}$$

we have the next corollary.

Corollary 3.29 There exists a binary, periodically time-varying, rate $R = b/c$ convolutional code with a polynomial generator matrix of overall constraint length ν that has a free distance satisfying the inequality

$$\frac{d_{\text{free}}}{\nu R^{-1}} \geq \frac{R}{-\log(2^{1-R}-1)} + O\left(\frac{\log \nu}{\nu}\right) \tag{3.164}$$

As a counterpart to Theorem 3.28 we can prove (see Problem 3.17) the following:

Theorem 3.30 There exists a binary, periodically time-varying, rate $R = b/c$ convolutional code with a polynomial, systematic generator matrix of memory m that has a free distance satisfying the inequality

$$\frac{d_{\text{free}}}{mc} \geq \frac{R(1-R)}{-\log(2^{1-R}-1)} + O\left(\frac{\log m}{m}\right) \tag{3.165}$$

Remark: The Costello bound is also valid for *time-invariant* convolutional codes with "sufficiently long" branches, that is, for large b [Zig86]. In [ZiC91, CSZ92] it was shown that the free distance for rate $R = 2/c$, $c \geq 4$, *time-invariant* convolutional codes asymptotically meets the Costello bound.

3.8 LOWER BOUNDS ON THE ACTIVE DISTANCES*

In this section we will derive lower bounds on the active distances for the ensemble of periodically time-varying convolutional codes. First, we consider the active row distance and begin by proving the following:

Lemma 3.31 Consider the ensemble $\mathcal{E}(b, c, m, T)$ of binary, rate $R = b/c$, periodically time-varying convolutional codes encoded by polynomial generator matrices of memory m. The fraction of convolutional codes in this ensemble whose jth order active row distance a_j^{r}, $0 \leq j < T$, satisfies

$$a_j^{\text{r}} \leq \widehat{a}_j^{\text{r}} < (j+m+1)c/2 \tag{3.166}$$

does not exceed

$$T 2^{\left(\frac{j+1}{j+m+1}R + h\left(\frac{\widehat{a}_j^{\text{r}}}{(j+m+1)c}\right) - 1\right)(j+m+1)c}$$

where $h(\ )$ is the binary entropy function (1.22).

Proof: Let

$$\boldsymbol{v}_{[t,t+j+m]} = \boldsymbol{u}_{[t-m,t+j+m]}\boldsymbol{G}_{[t,t+j+m]} \tag{3.167}$$

where $\boldsymbol{u}_{[t-m,t+j+m]} \in \mathcal{U}^{\text{r}}_{[t-m,t+j+m]}$ and assume that

$$\widehat{a}_j^{\text{r}} < (j+m+1)c/2 \tag{3.168}$$

Then, it follows from Theorem 3.25 that

$$P(w_{\text{H}}(\boldsymbol{v}_{[t,t+j+m]}) \leq \widehat{a}_j^{\text{r}}) = \sum_{i=0}^{\widehat{a}_j^{\text{r}}} \binom{(j+m+1)c}{i} 2^{-(j+m+1)c} \times 2^{\left(h\left(\frac{\widehat{a}_j^{\text{r}}}{(j+m+1)c}\right)-1\right)(j+m+1)c}, \quad 0 \leq j < T-m \tag{3.169}$$

where the last inequality follows from Lemma 3.27.

(Notice that we need the denominator "2" in the right inequality in (3.166) in order to be able to apply inequality (3.155).) Using

$$2^{(j+1)b} = 2^{(j+1)Rc} \tag{3.170}$$

as an upper bound on the cardinality of $\mathcal{U}^{\mathrm{r}}_{[t-m,t+j+m]}$, we have

$$\begin{aligned} P\left(\min_{\mathcal{U}^{\mathrm{r}}_{[t-m,t+j+m]}} \{w_{\mathrm{H}}(\boldsymbol{v}_{[t,t+j+m]})\} \le \widehat{a}^{\mathrm{r}}_j\right) &< 2^{(j+1)Rc} 2^{\left(h\left(\frac{\widehat{a}^{\mathrm{r}}_j}{(j+m+1)c}\right)-1\right)(j+m+1)c} \\ &= 2^{\left(\frac{j+1}{j+m+1}R + h\left(\frac{\widehat{a}^{\mathrm{r}}_j}{(j+m+1)c}\right)-1\right)(j+m+1)c} \end{aligned} \tag{3.171}$$

for each t, $0 \le t < T$. Using the union bound completes the proof. ∎

For a given f, $0 \le f < 1$, let j_0 be the smallest integer j satisfying the inequality

$$\left(1 - \frac{j+1}{j+m+1}R\right)(j+m+1)c \ge \log \frac{T^2}{1-f} \tag{3.172}$$

For large memories m such a value always exists. Let $\widehat{a}^{\mathrm{r}}_j$,

$$0 < \widehat{a}^{\mathrm{r}}_j < (j+m+1)c/2 \tag{3.173}$$

denote the largest integer that for given f, $0 \le f < 1$, and j, $j \ge j_0$, satisfies the inequality

$$\begin{aligned} &\left(\frac{j+1}{j+m+1}R + h\left(\frac{\widehat{a}^{\mathrm{r}}_j}{(j+m+1)c}\right) - 1\right)(j+m+1)c \\ &\quad \le -\log \frac{T^2}{1-f} \end{aligned} \tag{3.174}$$

Then, from Lemma 3.31 it follows that for each j, $j_0 \le j < T$, the fraction of convolutional codes with jth order active row distance satisfying (3.166) is upper-bounded by

$$T 2^{-\log \frac{T^2}{1-f}} = \frac{1-f}{T} \tag{3.175}$$

Hence, we use the union bound and conclude that the fraction of convolutional codes with active row distance $a^{\mathrm{r}}_j \le \widehat{a}^{\mathrm{r}}_j$ for at least one j, $j_0 \le j < T$, is upper-bounded by

$$\sum_{j=j_0}^{T-m-1} \frac{1-f}{T} < 1 - f \tag{3.176}$$

Thus, we have proved the following:

Lemma 3.32 In the ensemble $\mathcal{E}(b, c, m, T)$ of periodically time-varying convolutional codes, the fraction of codes with active row distance

$$a^{\mathrm{r}}_j > \widehat{a}^{\mathrm{r}}_j, \ j_0 \le j < T \tag{3.177}$$

is larger than f, where for a given f, $0 \leq f < 1$, j_0 is the smallest integer satisfying (3.172) and $\widehat{a}_j^{\mathrm{r}}$ the largest integer satisfying (3.174).

By taking $f = 0$, we immediately have the next corollary.

Corollary 3.33 There exists a binary, periodically time-varying, rate $R = b/c$, convolutional code encoded by a polynomial generator matrix of period T and memory m such that its jth order active row distance for $j_0 \leq j < T$ is lower-bounded by $\widehat{a}_j^{\mathrm{r}}$, where $\widehat{a}_j^{\mathrm{r}}$ is the largest integer satisfying

$$\left(\frac{j+1}{j+m+1}R + h\left(\frac{\widehat{a}_j^{\mathrm{r}}}{(j+m+1)c}\right) - 1\right)(j+m+1)c \leq -2\log T \tag{3.178}$$

and j_0 is the smallest integer satisfying

$$\left(1 - \frac{j+1}{j+m+1}R\right)(j+m+1)c \geq 2\log T \tag{3.179}$$

In order to get a better understanding for the significance of the previous lemma, we shall study the asymptotical behavior of the parameters j_0 and $\widehat{a}_j^{\mathrm{r}}$ for large memories.

Let the period T grow as a power of m greater than 1; choose $T = m^2$, say. Then, since j_0 is an integer, for large values of m we have $j_0 = 0$. Furthermore, the inequality (3.178) can be rewritten as

$$h\left(\frac{\widehat{a}_j^{\mathrm{r}}}{(j+m+1)c}\right) \leq 1 - \frac{j+1}{j+m+1}R + O\left(\frac{\log m}{m}\right) \tag{3.180}$$

or, equivalently, as[8]

$$\widehat{a}_j^{\mathrm{r}} \leq h^{-1}\left(1 - \frac{j+1}{j+m+1}R\right)(j+m+1)c + O(\log m) \tag{3.181}$$

Finally, we have proved the following:

Theorem 3.34 There exists a binary, periodically time-varying, rate $R = b/c$, convolutional code encoded by a polynomial generator matrix of memory m that has a jth order active row distance satisfying the inequality

$$a_j^{\mathrm{r}} > h^{-1}\left(1 - \frac{j+1}{j+m+1}R\right)(j+m+1)c + O(\log m) \tag{3.182}$$

for $j \geq 0$.

The main term in (3.182) can also be obtained from the Gilbert-Varshamov bound for block codes using a geometrical construction that is similar to Forney's inverse concatenated construction [For74].

[8]Here and hereafter we write $h^{-1}(y)$ for the *smallest* x such that $y = h(x)$.

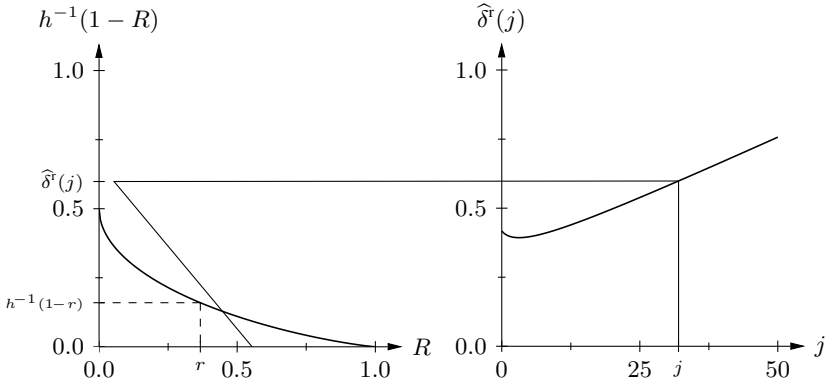


Figure 3.12 Geometrical construction of the relationship between the lower bound on the active row distance for convolutional codes and the Gilbert-Varshamov lower bound on the minimum distance for block codes.

Consider Gilbert-Varshamov's lower bound on the (normalized) minimum distance for block codes [MaS77], viz.,

$$\frac{d_{\min}}{N} \geq h^{-1}(1-R) \tag{3.183}$$

where N denotes the block length. Let

$$\widehat{\delta}^{\mathrm{r}}(j) = \frac{h^{-1}\left(1 - \frac{j+1}{j+1+m}R\right)(j+1+m)c}{mc} \tag{3.184}$$

denote the main term of the right-hand side of (3.182) normalized by mc.

The construction is illustrated in Fig. 3.12 for $R = 1/2$. The straight line between the points $(0, \widehat{\delta}^{\mathrm{r}}(j))$ and $(R, 0)$ intersects $h^{-1}(1-R)$ in the point $(r, h^{-1}(1-r))$. The rate r is chosen to be

$$r = \frac{j+1}{j+1+m}R \tag{3.185}$$

that is, it divides the line between $(0, 0)$ and $(R, 0)$ in the proportion $(j+1) : m$. Then we have

$$\frac{\widehat{\delta}^{\mathrm{r}}(j)}{h^{-1}(1-r)} = \frac{j+1+m}{m} \tag{3.186}$$

which is equivalent to (3.184). The relationship between r and j in Fig. 3.12 is given by (3.185).

We will now derive a corresponding lower bound on the active column distance. Let

$$\boldsymbol{v}_{[t,t+j]} = \boldsymbol{u}_{[t-m,t+j]}\boldsymbol{G}_{[t,t+j]} \tag{3.187}$$

where $\boldsymbol{u}_{[t-m,t+j]} \in \mathcal{U}^{\mathrm{c}}_{[t-m,t+j]}$ and let $\widehat{a}^{\mathrm{c}}_j$ be an integer satisfying the inequality

$$\widehat{a}^{\mathrm{c}}_j < (j+1)c/2 \tag{3.188}$$

Then, as a counterpart to (3.169) we have

$$P(w_{\mathrm{H}}(\boldsymbol{v}_{[t,t+j]}) \leq \widehat{a}_j^{\mathrm{c}}) = \sum_{i=0}^{\widehat{a}_j^{\mathrm{c}}} \binom{(j+1)c}{i} 2^{-(j+1)c}$$
$$< 2^{\left(h\left(\frac{\widehat{a}_j^{\mathrm{c}}}{(j+1)c}\right)-1\right)(j+1)c}, \quad 0 \leq j < T \tag{3.189}$$

We use (3.170) as an upper bound on the cardinality of $\mathcal{U}_{[t-m,t+j]}^{\mathrm{c}}$ and obtain

$$P\left(\min_{\mathcal{U}_{[t-m,t+j]}^{\mathrm{c}}} \{w_{\mathrm{H}}(\boldsymbol{v}_{[t,t+j]})\} \leq \widehat{a}_j^{\mathrm{c}}\right) < 2^{(j+1)Rc} 2^{\left(h\left(\frac{\widehat{a}_j^{\mathrm{c}}}{(j+1)c}\right)-1\right)(j+1)c}$$
$$= 2^{\left(R+h\left(\frac{\widehat{a}_j^{\mathrm{c}}}{(j+1)c}\right)-1\right)(j+1)c} \tag{3.190}$$

for each t, $0 \leq t < T$. Using the union bound completes the proof of the following:

Lemma 3.35 Consider the ensemble $\mathcal{E}(b, c, m, T)$ of binary, rate $R = b/c$, periodically time-varying convolutional codes encoded by polynomial generator matrices of memory m. The fraction of convolutional codes in this ensemble whose jth order active column distance a_j^{c}, $0 \leq j < T$, satisfies

$$a_j^{\mathrm{c}} \leq \widehat{a}_j^{\mathrm{c}} < (j+1)c/2 \tag{3.191}$$

does not exceed

$$T 2^{\left(R+h\left(\frac{\widehat{a}_j^{\mathrm{c}}}{(j+1)c}\right)-1\right)(j+1)c}$$

Next, we choose j_0 to be the smallest integer j satisfying the inequality

$$(1-R)(j+1)c \geq \log T^2 \tag{3.192}$$

Let $\widehat{a}_j^{\mathrm{c}}$,

$$0 < \widehat{a}_j^{\mathrm{c}} < (j+1)c/2 \tag{3.193}$$

denote the largest integer that for given j, $j \geq j_0$, satisfies the inequality

$$\left(R + h\left(\frac{\widehat{a}_j^{\mathrm{c}}}{(j+1)c}\right) - 1\right)(j+1)c \leq -\log T^2 \tag{3.194}$$

Then, from Lemma 3.35 it follows that for each j, $j_0 \leq j < T$, the fraction of convolutional codes with a jth order active column distance satisfying (3.191) is upper-bounded by

$$T 2^{-\log T^2} = \frac{1}{T} \tag{3.195}$$

Hence, we use the union bound and conclude that the fraction of convolutional codes with active column distance $a_j^{\mathrm{c}} \leq \widehat{a}_j^{\mathrm{c}}$ for at least one j, $j_0 \leq j < T$, is upper-bounded by

$$\sum_{j=j_0}^{T-1} \frac{1}{T} < 1 \tag{3.196}$$

Thus, we have proved the following:

Lemma 3.36 There exists a periodically time-varying, rate $R = b/c$, convolutional code encoded by a polynomial generator matrix of period T and memory m such that its jth order active column distance for $j_0 \leq j < T$ is lower-bounded by $\widehat{a}_j^{\mathrm{c}}$, where $\widehat{a}_j^{\mathrm{c}}$ is the largest integer satisfying

$$\left(R + h\left(\frac{\widehat{a}_j^{\mathrm{c}}}{(j+1)c}\right) - 1\right)(j+1)c \leq -2\log T \tag{3.197}$$

and j_0 is the smallest integer satisfying

$$(1-R)(j+1)c \geq 2\log T \tag{3.198}$$

If we as before choose $T = m^2$, then $j_0 = O(\log m)$, and the inequality (3.197) can be rewritten as

$$h\left(\frac{\widehat{a}_j^{\mathrm{c}}}{(j+1)c}\right) \leq 1 - R - \frac{4\log m}{(j+1)c} \tag{3.199}$$

for $j = O(m)$ or, equivalently, as

$$\widehat{a}_j^{\mathrm{c}} \leq h^{-1}(1-R)(j+1)c + O(\log m) \tag{3.200}$$

Thus, we have proved the following:

Theorem 3.37 There exists a binary, periodically time-varying, rate $R = b/c$, convolutional code encoded by a polynomial generator matrix of memory m that has a jth order active column distance satisfying the inequality

$$a_j^{\mathrm{c}} > \rho(j+1)c + O(\log m) \tag{3.201}$$

for $j = O(m) > j_0 = O(\log m)$ and ρ is the Gilbert-Varshamov parameter (3.89).

Analogously we can prove the next theorem.

Theorem 3.38 There exists a binary, periodically time-varying, rate $R = b/c$, convolutional code encoded by a polynomial generator matrix of memory m that has a jth order active reverse column distance a_j^{rc} which is lower-bounded by the right-hand side of the inequality (3.201) for all $j = O(m) > j_0 = O(\log m)$.

For the active segment distance we have the following:

Theorem 3.39 There exists a binary, periodically time-varying, rate $R = b/c$, convolutional code encoded by a polynomial generator matrix of memory m that has a jth order active segment distance satisfying the inequality

$$a_j^{\mathrm{s}} > h^{-1}\left(1 - \frac{j+m+1}{j+1}R\right)(j+1)c + O(\log m) \tag{3.202}$$

for $j = O(m) > j_s$, where

$$j_s < \frac{R}{1-R}m + O(\log m) \tag{3.203}$$

Proof: Consider the ensemble $\mathcal{E}(b, c, m, T)$. First, we notice that the cardinality of $\mathcal{U}^s_{[t,t+j]}$ is upper-bounded by

$$2^{mb}2^{(j+1)b} = 2^{(j+m+1)Rc} \tag{3.204}$$

Using (3.204) instead of (3.170) and repeating the steps in the derivation of the lower bound on the active column distance will give

$$h\left(\frac{\widehat{a}^s_j}{(j+1)c}\right) \leq 1 - \frac{j+m+1}{j+1}R - \frac{4\log m}{(j+1)c} \tag{3.205}$$

for all $j = O(m) > j_s$, or, equivalently,

$$\widehat{a}^s_j \leq h^{-1}\left(1 - \frac{j+m+1}{j+1}R\right)(j+1)c + O(\log m) \tag{3.206}$$

where

$$0 < \widehat{a}^s_j < (j+1)c/2 \tag{3.207}$$

instead of (3.199), (3.200), and (3.193), respectively, and the proof is complete. ∎

The parameter j_s is the start of the active segment distance which was introduced in Section 3.2 (cf. Fig. 3.6).

For $R \leq 1/2$ there exist binary, *time-invariant* convolutional codes with (see Problem 3.18)

$$j_s < \frac{R}{1-R}(m+1) \tag{3.208}$$

Next we consider our lower bounds on the active distances, viz., (3.182), (3.201), and (3.202), and introduce the substitution

$$\ell = (j+1)/m \tag{3.209}$$

Then we obtain asymptotically—for large memories m—the following lower bounds on the *normalized active distances*:

Theorem 3.40 (i) There exists a binary, periodically time-varying, rate $R = b/c$, convolutional code encoded by a polynomial generator matrix of memory m whose normalized active row distance asymptotically satisfies

$$\delta^r_\ell \stackrel{\text{def}}{=} \frac{a^r_j}{mc} \geq h^{-1}\left(1 - \frac{\ell}{\ell+1}R\right)(\ell+1) + O\left(\frac{\log m}{m}\right) \tag{3.210}$$

for $\ell \geq 0$.

(ii) There exists a binary, periodically time-varying, rate $R = b/c$, convolutional code encoded by a polynomial generator matrix of memory m whose normalized active column distance (active reverse column distance) asymptotically satisfies

$$\left.\begin{array}{lll} \delta_\ell^{\mathrm{c}} & \stackrel{\mathrm{def}}{=} & \frac{a_j^{\mathrm{c}}}{mc} \\ \delta_\ell^{\mathrm{rc}} & \stackrel{\mathrm{def}}{=} & \frac{a_j^{\mathrm{rc}}}{mc} \end{array}\right\} \geq h^{-1}(1-R)\ell + O\left(\frac{\log m}{m}\right) \tag{3.211}$$

for $\ell \geq \ell_0 = O\left(\frac{\log m}{m}\right)$.

(iii) There exists a binary, periodically time-varying, rate $R = b/c$, convolutional code encoded by a polynomial generator matrix of memory m whose normalized active segment distance asymptotically satisfies

$$\delta_\ell^{\mathrm{s}} \stackrel{\mathrm{def}}{=} \frac{a_j^{\mathrm{s}}}{mc} \geq h^{-1}\left(1 - \frac{\ell+1}{\ell}R\right)\ell + O\left(\frac{\log m}{m}\right) \tag{3.212}$$

for $\ell \geq \ell_{\mathrm{s}} = \frac{R}{1-R} + O\left(\frac{\log m}{m}\right)$.

The typical behavior of the bounds in Theorem 3.40 is shown in Fig. 3.13. Notice that by minimizing the lower bound on the normalized active row distance (3.210), we obtain nothing but the main term in Costello's lower bound on the free distance (3.162), viz.,

$$\frac{R}{-\log(2^{1-R}-1)}$$

3.9 DISTANCES OF CASCADED CONCATENATED CODES*

Consider the simplest concatenated scheme with two convolutional encoders, viz., a cascade of a rate $R_{\mathrm{o}} = b_{\mathrm{o}}/c_{\mathrm{o}}$ outer encoder of memory m_{o} and a rate $R_{\mathrm{i}} = b_{\mathrm{i}}/c_{\mathrm{i}}$ inner encoder of memory m_{i}, where $b_{\mathrm{i}} = c_{\mathrm{o}}$. The *cascaded concatenated code* $\mathcal{C}_c$ is encoded by the rate $R_{\mathrm{c}} = R_{\mathrm{o}}R_{\mathrm{i}} = b_{\mathrm{o}}/c_{\mathrm{i}} \stackrel{\mathrm{def}}{=} b/c$ convolutional encoder whose memory (2.134) in general could be less than the sum of the memories of the constituent encoders, that is, $m_{\mathrm{c}} \leq m_{\mathrm{o}} + m_{\mathrm{i}}$ (Fig. 3.14).

Consider the ensemble $\mathcal{EC}(b, c, m_{\mathrm{c}}, T)$ of periodically time-varying, cascaded convolutional codes constructed in the following way:

Choose as outer convolutional code a binary, periodically time-varying with period T, rate $R_{\mathrm{o}} = b_{\mathrm{o}}/c_{\mathrm{o}}$ convolutional code encoded by a minimal-basic encoding matrix of memory $m_{\mathrm{o}} = \min_k\{\nu_{\mathrm{o},k}\}$, where $\nu_{\mathrm{o},k}$ is the constraint length of the kth input and whose active segment distance has the start

$$j_{\mathrm{s}}^{\mathrm{o}} = \frac{R_{\mathrm{o}}}{1-R_{\mathrm{o}}}m_{\mathrm{o}} + O(\log m_{\mathrm{o}}) \tag{3.213}$$

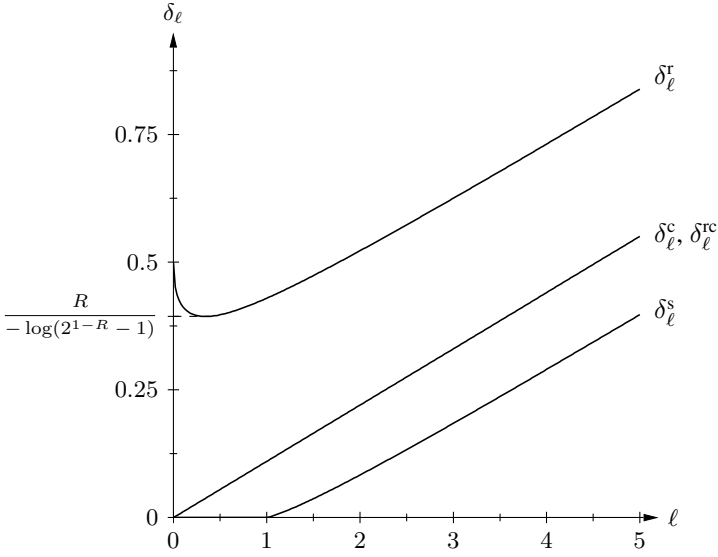


Figure 3.13 Typical behavior of the lower bounds on the normalized active distances of Theorem 3.40.

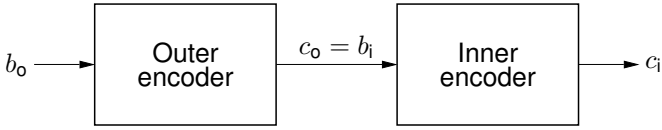


Figure 3.14 A cascade of two consecutive encoders.

The existence of such a convolutional code follows from Theorem 3.39. For $R_{\mathrm{o}} \leq 1/2$, the outer convolutional code can be chosen to be time invariant (cf. (3.208)).

The ensemble of inner convolutional codes is the ensemble of binary, periodically time-varying with period T, rate $R_{\mathrm{i}} = b_{\mathrm{i}}/c_{\mathrm{i}}$ convolutional codes encoded by time-varying polynomial generator matrices of memory $m_i \geq j_{\mathrm{s}}^{\mathrm{o}}$. Then we have the ensemble $\mathcal{EC}(b, c, m_{\mathrm{c}}, T)$ of periodically time-varying with period T, rate $R_{\mathrm{c}} = b/c$, cascaded convolutional codes encoded by convolutional encoders of memory m_{c}.

As a counterpart to Theorem 3.25 we have the following:

Theorem 3.41 Consider a periodically time-varying, rate $R_{\mathrm{c}} = b/c$, cascaded convolutional code encoded by a convolutional encoder of memory m_{c}.

(i) Let the information sequences be restricted to the set $\mathcal{U}^{\mathrm{r}}_{[t-m_{\mathrm{c}},t+j+m_{\mathrm{c}}]}$. Then the code symbols in the segment $\boldsymbol{v}_{[t,t+j+m_{\mathrm{c}}]}$ are mutually independent and equiprobable over the ensemble $\mathcal{EC}(b, c, m_{\mathrm{c}}, T)$ for all j, $0 \leq j < T$.

(ii) Let the information sequences be restricted to the set $\mathcal{U}^{\mathrm{c}}_{[t-m_{\mathrm{c}},t+j]}$. Then the code symbols in the segment $\boldsymbol{v}_{[t,t+j]}$ are mutually independent and equiprobable over the ensemble $\mathcal{EC}(b,c,m_{\mathrm{c}},T)$ for all j, $0 \le j < T$.

(iii) Let the information sequences be restricted to the set $\mathcal{U}^{\mathrm{rc}}_{[t-m_{\mathrm{c}},t+j]}$. Then the code symbols in the segment $\boldsymbol{v}_{[t,t+j]}$ are mutually independent and equiprobable over the ensemble $\mathcal{EC}(b,c,m_{\mathrm{c}},T)$ for all j, $0 \le j < T$.

(iv) Let the information sequences be restricted to the set $\mathcal{U}^{\mathrm{s}}_{[t-m_{\mathrm{c}},t+j]}$. Then the code symbols in the segment $\boldsymbol{v}_{[t,t+j]}$ are mutually independent and equiprobable over the ensemble $\mathcal{EC}(b,c,m_{\mathrm{c}},T)$ for all j, $0 \le j < T$.

Proof: Analogously to the proof of Theorem 3.25. ∎

If we let

$$\ell = (j+1)/m_{\mathrm{c}} \tag{3.214}$$

then for cascaded convolutional codes we have the following counterpart to Theorem 3.40:

Theorem 3.42 (i) There exists a cascaded convolutional code in the ensemble $\mathcal{EC}(b,c,m_{\mathrm{c}},T)$ whose normalized active row distance asymptotically satisfies

$$\delta^{\mathrm{r}}_{\ell} \stackrel{\mathrm{def}}{=} \frac{a^{\mathrm{r}}_j}{m_{\mathrm{c}}c} \ge h^{-1}\left(1 - \frac{\ell}{\ell+1}R_{\mathrm{c}}\right)(\ell+1) + O\left(\frac{\log m_{\mathrm{c}}}{m_{\mathrm{c}}}\right) \tag{3.215}$$

for $\ell \ge 0$.

(ii) There exists a cascaded convolutional code in the ensemble $\mathcal{EC}(b,c,\,m_{\mathrm{c}},T)$ whose normalized active column distance asymptotically satisfies

$$\delta^{\mathrm{c}}_{\ell} \stackrel{\mathrm{def}}{=} \frac{a^{\mathrm{c}}_j}{m_{\mathrm{c}}c} \ge h^{-1}(1-R_{\mathrm{c}})\ell + O\left(\frac{\log m_{\mathrm{c}}}{m_{\mathrm{c}}}\right) \tag{3.216}$$

for $\ell \ge \ell_0 = O\left(\frac{\log m_{\mathrm{c}}}{m_{\mathrm{c}}}\right)$.

(iii) There exists a cascaded convolutional code in the ensemble $\mathcal{EC}(b,c,\,m_{\mathrm{c}},T)$ whose normalized active segment distance asymptotically satisfies

$$\delta^{\mathrm{s}}_{\ell} \stackrel{\mathrm{def}}{=} \frac{a^{\mathrm{s}}_j}{m_{\mathrm{c}}c} \ge h^{-1}\left(1 - \frac{\ell+1}{\ell}R_{\mathrm{c}}\right)\ell + O\left(\frac{\log m_{\mathrm{c}}}{m_{\mathrm{c}}}\right) \tag{3.217}$$

for $\ell \ge \ell^{\mathrm{o}}_{\mathrm{s}} = \frac{R_{\mathrm{o}}}{1-R_{\mathrm{o}}} + O\left(\frac{\log m_{\mathrm{o}}}{m_{\mathrm{o}}}\right)$.

Proof: Analogously to the proof of Theorem 3.40. ∎

The behavior of the bounds in Theorem 3.42 is the same as that for the bounds in Theorem 3.40 (see Fig. 3.13).

The free distance for a convolutional code is obtained as the minimum weight of the nonzero codewords. The only restriction on the input sequence is that it should be nonzero. Now, let us consider the cascade in Fig. 3.14. The free distance for the cascaded convolutional code, $d_{\text{free}}^{\text{c}}$, is obtained as the minimum weight of the nonzero codewords; again the minimum is evaluated over all nonzero inputs. Since the inputs to the inner encoder are restricted to be codewords of the outer encoder, we will not obtain a useful estimate of $d_{\text{free}}^{\text{c}}$ from the free distance of the inner code, $d_{\text{free}}^{\text{i}}$.

It is somewhat surprising that, given only a restriction on the memory of the inner code, there exists a convolutional code obtained as a simple cascade with a free distance satisfying the Costello bound:

Theorem 3.43 (Costello bound) Consider a cascade $\mathcal{C}_{\text{c}}$ of an outer binary, periodically time-varying convolutional code $\mathcal{C}_{\text{o}}$ of rate $R_{\text{o}} = b_{\text{o}}/c_{\text{o}}$ and encoder memory m_{o} and an inner binary, time-varying convolutional code $\mathcal{C}_{\text{i}}$ of rate $R_{\text{i}} = b_{\text{i}}/c_{\text{i}}$ and encoder memory m_{i}, where $b_{\text{i}} = c_{\text{o}}$. If

$$m_i \geq \frac{R_{\text{o}}}{1 - R_{\text{o}}} m_{\text{o}} + O(\log m_{\text{o}}) \tag{3.218}$$

then there exists a pair of codes, $\mathcal{C}_{\text{o}}$ and $\mathcal{C}_{\text{i}}$, such that the code $\mathcal{C}_{\text{c}}$ of rate $R_{\text{c}} = R_{\text{o}} R_{\text{i}}$ and encoder memory $m_{\text{c}} = m_{\text{o}} + m_{\text{i}}$ has a free distance satisfying the inequalities

$$\begin{aligned} d_{\text{free}}^{\text{c}} &\geq \frac{m_{\text{c}} c_{\text{i}} R_{\text{c}}}{-\log(2^{1-R} - 1)} + O\left(\frac{\log m_{\text{c}}}{m_{\text{c}}}\right) \\ &\geq \frac{m_{\text{o}} c_{\text{o}} R_{\text{o}}}{-\log(2^{1-R_{\text{o}}} - 1)} + \frac{m_{\text{i}} c_{\text{i}} R_{\text{i}}}{-\log(2^{1-R_{\text{i}}} - 1)} \\ &\quad + O\left(\frac{\log m_{\text{o}}}{m_{\text{o}}}\right) + O\left(\frac{\log m_{\text{i}}}{m_{\text{i}}}\right) \end{aligned} \tag{3.219}$$

Proof: Choose an inner code $\mathcal{C}_i$ such that

$$m_{\text{i}} = \left\lceil \frac{R_{\text{o}}}{1 - R_{\text{o}}} m_{\text{o}} \right\rceil \tag{3.220}$$

Then, by minimizing (3.215) over ℓ, we obtain

$$\begin{aligned} d_{\text{free}}^{\text{c}} &\geq \frac{m_{\text{c}} b_{\text{o}}}{-\log(2^{1-R_{\text{c}}} - 1)} + O\left(\frac{\log m_{\text{c}}}{m_{\text{c}}}\right) \\ &= \frac{m_{\text{c}} c_{\text{i}} R_{\text{c}}}{-\log(2^{1-R_{\text{c}}} - 1)} + O\left(\frac{\log m_{\text{c}}}{m_{\text{c}}}\right) \\ &= \frac{m_{\text{o}} b_{\text{o}}}{-\log(2^{1-R_{\text{c}}} - 1)} + \frac{m_{\text{i}} c_{\text{i}} R_{\text{c}}}{-\log(2^{1-R_{\text{c}}} - 1)} + O\left(\frac{\log m_{\text{c}}}{m_{\text{c}}}\right) \\ &\geq \frac{m_{\text{o}} c_{\text{o}} R_{\text{o}}}{-\log(2^{1-R_{\text{o}}} - 1)} + \frac{m_{\text{i}} c_{\text{i}} R_{\text{i}}}{-\log(2^{1-R_{\text{i}}} - 1)} + O\left(\frac{\log m_{\text{c}}}{m_{\text{c}}}\right) \end{aligned} \tag{3.221}$$

where we in the last inequality have used the fact that

$$-\log(2^{1-R} - 1)$$

is an increasing and

$$\frac{R}{-\log(2^{1-R}-1)}$$

is a decreasing functions of R. ∎

Theorem 3.43 shows that, given the restriction (3.220) on the ratio $m_{\mathrm{i}}/m_{\mathrm{o}}$, from the Costello lower bound point of view, we lose nothing in free distance by splitting a given amount of convolutional encoder memory into two cascaded convolutional encoders.

Remark: Assume that for the cascaded encoder in Theorem 3.43 we have $\nu_{\min,\mathrm{o}} = m_{\mathrm{o}}$ and $\nu_{\min,\mathrm{i}} = m_{\mathrm{i}}$ and that $m_{\mathrm{i}} = \frac{R_{\mathrm{o}}}{1-R_{\mathrm{o}}} m_{\mathrm{o}}$ holds. Then the total number of states in the outer and inner encoders are

$$\begin{aligned} 2^{m_{\mathrm{i}}b_{\mathrm{i}}} + 2^{m_{\mathrm{o}}b_{\mathrm{o}}} &= 2^{m_{\mathrm{o}}b_{\mathrm{o}}}(2^{m_{\mathrm{i}}b_{\mathrm{i}}-m_{\mathrm{o}}b_{\mathrm{o}}}+1) \\ &= 2^{(m_{\mathrm{o}}+m_{\mathrm{i}})b_{\mathrm{o}}}(1+2^{-m_{\mathrm{i}}b_{\mathrm{o}}}) \end{aligned} \tag{3.222}$$

which is essentially equal to the total number of states of a generator matrix $G_{\mathrm{c}}(D)$ with $\nu_{\min,\mathrm{c}} = m_{\mathrm{c}}$. The second equality follows from the equality $m_{\mathrm{i}} = \frac{R_{\mathrm{o}}}{1-R_{\mathrm{o}}} m_{\mathrm{o}}$.

We have shown that the performances for all active distances for the ensemble of time-varying cascaded convolutional codes are the same as for the ensemble of time-varying convolutional codes, although the former ensemble is essentially smaller.

Consider again the concatenated scheme given in Fig. 3.14. Since the inputs to the inner encoder are restricted to the codewords of the outer encoder, we will not obtain a useful estimate of $d^{\mathrm{c}}_{\mathrm{free}}$ from the free distance of the inner code, $d^{\mathrm{i}}_{\mathrm{free}}$. Consider the situation when the shortest constraint length for the inner encoder exceeds the length of the allzero sequence considered when we determine the active segment distance for the outer encoder, that is, when $\min_k\{\nu_{\mathrm{i},k}\} \geq j^{\mathrm{o}}_{\mathrm{s}}$, where $\nu_{\mathrm{i},k}$ is the constraint length for the kth input of the inner encoder. Then a nontrivial lower bound on the free distance for the binary concatenated code, $d^{\mathrm{c}}_{\mathrm{free}}$,can be obtained as follows:

Assume a nonzero input sequence to the outer encoder. The length of its output sequence is at least $\min_k\{\nu_{\mathrm{o},k}\}+1$, where $\nu_{\mathrm{o},k}$ is the constraint length for the kth input of the outer encoder. Since this output sequence serves as input to the inner encoder and since the weight of the output sequence of the inner encoder is lower-bounded by the active row distance of the inner encoder, $a^{\mathrm{r,i}}_j$, at the length of its input sequence, it follows that

$$d^{\mathrm{c}}_{\mathrm{free}} \geq \min_{j \geq \left\lceil \frac{\ell^{\mathrm{o}}_{\min}}{b_{\mathrm{i}}} \right\rceil - 1} \{a^{\mathrm{r,i}}_j\} \tag{3.223}$$

where $\ell^{\mathrm{o}}_{\min}$ is the length of the shortest burst of code symbols, which is lower-bounded by

$$\ell^{\mathrm{o}}_{\min} \geq \max\{(\min_k\{\nu_{\mathrm{o},k}\}-1)c_{\mathrm{o}}+2, d^{\mathrm{o}}_{\mathrm{free}}\} \tag{3.224}$$

From (3.223) we conclude that, in order to obtain a large free distance for this cascaded concatenated code, the inner encoder should have a rapidly increasing active row distance and the outer encoder should have long nonzero sequences as outputs.

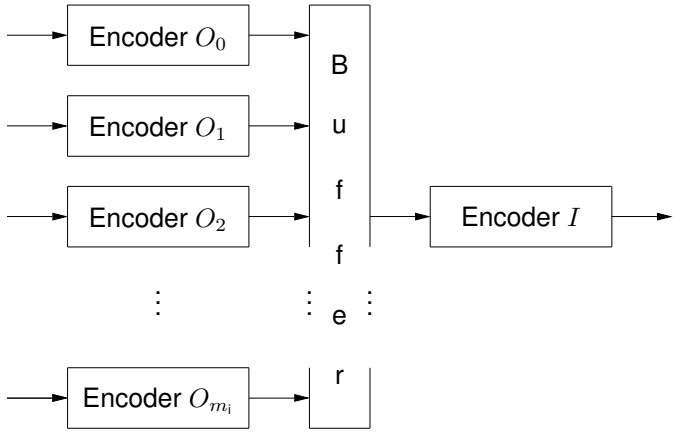


Figure 3.15 A cascade with a set of m_i parallel outer encoders, each of memory m_o and rate $R_o = b_o/c_o$, followed by a buffer of size $(m_i + 1) \times c_o$, and an inner encoder of memory m_i and rate $R_i = 1/c_i$.

Finally, we consider a cascade of a binary rate $R_i = 1/c_i$ convolutional inner encoder of memory m_i and a set of $m_i + 1$ parallel, binary, rate $R_o = b_o/c_o$ convolutional outer encoders each of memory m_o. The $m_i + 1$ outputs of the outer encoders are via a buffer of size $(m_i + 1) \times c_o$ connected to the single input of the inner encoder (Fig. 3.15). The overall concatenated code $\mathcal{C}_{pc}$ is a rate $R_{pc} = R_o R_i = b_o/c_i$ convolutional code with an encoder consisting of $m_{pc} = (m_o + c_o)(m_i + 1) + m_i$ memory elements.

Theorem 3.44 There exists a binary, periodically time-varying, rate $R_{pc} = R_o R_i$ cascaded concatenated convolutional code $\mathcal{C}_{pc}$ encoded by the scheme described above that has a free distance satisfying the inequality[9]

$$d_{\text{free}}^{pc} \geq \rho_i c_i (m_i + 1) \frac{m_o c_o R_o}{-\log(2^{1-R_o} - 1)} + o(m_o) \tag{3.225}$$

where ρ_i is the Gilbert-Varshamov parameter, that is, the solution of (3.89) for the inner convolutional code.

Proof: Choose outer convolutional codes satisfying the Costello bound and an inner convolutional code satisfying the active column distance bound. We have at least d_{free}^{o} ones in the output from any outer encoder. They will be at least $m_i + 1$ positions apart at the input of the inner encoder. Since each one at the input of the inner encoder will contribute at least $\rho_i c_i (m_i + 1)$ to the weight of the output sequence, it follows

[9]Here and hereafter we write $f(x) = o(g(x))$ if $\lim_{x \to a} \frac{f(x)}{g(x)} = 0$, where a is any real number or ∞. We have, e.g., $f(x) = x^2$ can be regarded as $f(x) = o(x)$ when $x \to 0$ and $f(x) = \sqrt{x}$ can be regarded as $f(x) = o(x)$ when $x \to \infty$. We also have $f(x) = x$ can be regarded as $f(x) = o(1)$ when $x \to 0$.

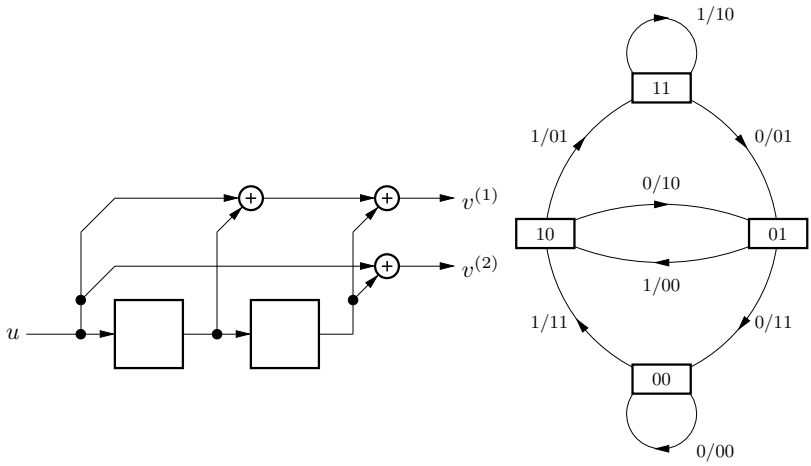


Figure 3.16 A rate $R = 1/2$ convolutional encoder and its state-transition diagram.

that its weight will be at least the product of the two lower bounds, and the proof is complete. ■

The lower bound in Theorem 3.44 can be interpreted as the product of a Gilbert-Varshamov-type lower bound on the minimum distance (Corollary 3.15) for the inner convolutional code and the Costello bound on the free distance (Theorem 3.28) for the outer convolutional code.

3.10 PATH ENUMERATORS

For a convolutional encoder the paths through the state-transition diagram beginning and ending in the (encoder) zero state when the self-loop at this state is removed determines the Viterbi distance spectrum. We shall now obtain a closed-form expression whose expansion yields the enumeration of all such paths. The method, which is due to Viterbi [Vit71], is best explained by an example.

Consider the rate $R = 1/2$ convolutional encoder and its state-transition diagram given in Fig. 1.16. For reference the figure is repeated as Fig. 3.16. The self-loop at the zero state in the state-transition diagram is removed, and the zero state is split in two—a source state and a sink state. Then the branches are labeled $W^0 = 1$, W, W^2, where the exponent corresponds to the weight of the particular branch. The result is the so-called *signal flowchart* shown in Fig. 3.17.

Let the input to the source (left zero state) be 1, and let $T(W)$ denote the generating function for the path weight W. We call $T(W)$ the *path weight enumerator*. In connection with signal flowcharts, it is often called *transmission gain* and can be found by the standard signal flowchart technique [MaZ60]. For some applications, this method is an efficient way of formulating and solving a system of linear equations. Here we prefer the straightforward method used by Viterbi.

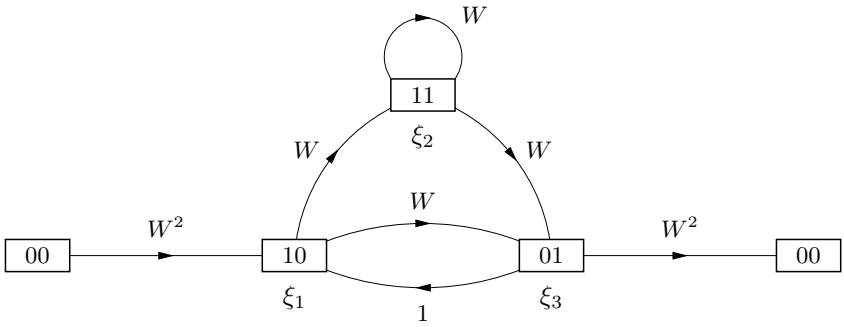


Figure 3.17 Signal flowchart for the encoder illustrated in Fig. 3.16.

Let ξ_1, ξ_2, and ξ_3 be dummy variables representing the weights of all paths from the left zero state to the intermediate states. Then from Fig. 3.17 we obtain the system of linear equations

$$\begin{aligned} \xi_1 &= \xi_3 + W^2 \\ \xi_2 &= W\xi_1 + W\xi_2 \\ \xi_3 &= W\xi_1 + W\xi_2 \end{aligned} \tag{3.226}$$

and

$$T(W) = W^2\xi_3 \tag{3.227}$$

Equation (3.226) can be rewritten as

$$\begin{pmatrix} 1 & 0 & -1 \\ -W & 1-W & 0 \\ -W & -W & 1 \end{pmatrix} \begin{pmatrix} \xi_1 \\ \xi_2 \\ \xi_3 \end{pmatrix} = \begin{pmatrix} W^2 \\ 0 \\ 0 \end{pmatrix} \tag{3.228}$$

Using Cramer's rule, we obtain

$$\begin{aligned} \xi_3 &= \left(\det \begin{pmatrix} 1 & 0 & -1 \\ -W & 1-W & 0 \\ -W & -W & 1 \end{pmatrix} \right)^{-1} \det \begin{pmatrix} 1 & 0 & W^2 \\ -W & 1-W & 0 \\ -W & -W & 0 \end{pmatrix} \\ &= W^3/(1-2W) \end{aligned} \tag{3.229}$$

Combining (3.227) and (3.229), we find

$$\begin{aligned} T(W) &= W^5/(1-2W) \\ &= W^5 + 2W^6 + 4W^7 + \cdots + 2^k W^{k+5} + \cdots \end{aligned} \tag{3.230}$$

Hence, we have $d_{\text{free}} = 5$, and the Viterbi spectral components $n_{d_{\text{free}}+i}$, $i = 0, 1, 2, \ldots$, are $1, 2, 4, \ldots$.

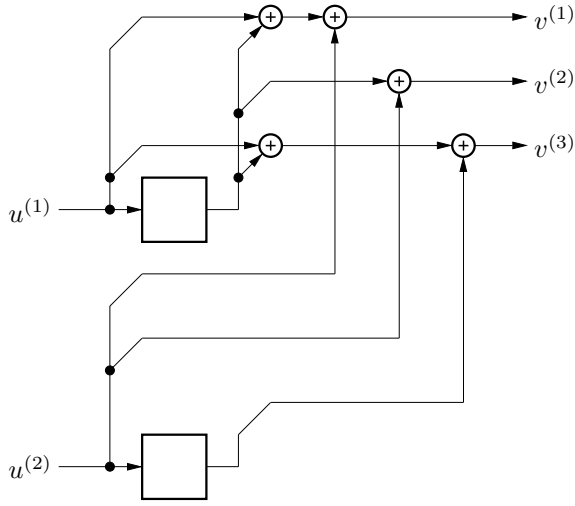


Figure 3.18 Controller canonical form of the encoding matrix in Example 3.8.

■ **EXAMPLE 3.8**

Consider the rate $R = 2/3$, memory $m = 1$, overall constraint length $\nu = 2$, convolutional encoding matrix

$$G(D) = \begin{pmatrix} 1+D & D & 1+D \\ 1 & 1 & D \end{pmatrix} \tag{3.231}$$

Its controller canonical form is shown in Fig. 3.18.

From $G(D)$ or from the encoder block diagram (Fig. 3.18) we can easily obtain the state-transition diagram which is shown in Fig. 3.19.

A modification of the state-transition diagram yields the signal flowchart given in Fig. 3.20. From the signal flowchart we obtain the system of linear equations

$$\begin{pmatrix} 1-W & -W^2 & -W \\ -W & 1-W^2 & -W \\ -W & -1 & 1-W^3 \end{pmatrix} \begin{pmatrix} \xi_1 \\ \xi_2 \\ \xi_3 \end{pmatrix} = \begin{pmatrix} W^2 \\ W^2 \\ W^2 \end{pmatrix} \tag{3.232}$$

and

$$T(W) = W^3\xi_1 + W^2\xi_2 + W\xi_3 \tag{3.233}$$

Solving these equations yields

$$T(W) = 2W^3 + 5W^4 + 15W^5 + 43W^6 + 118W^7 + 329W^8 + \cdots \tag{3.234}$$

Thus, the Viterbi spectrum has two codewords of weight $d_{\text{free}} = 3$, which can easily be verified by tracing paths in the signal flowchart or state-transition diagram.

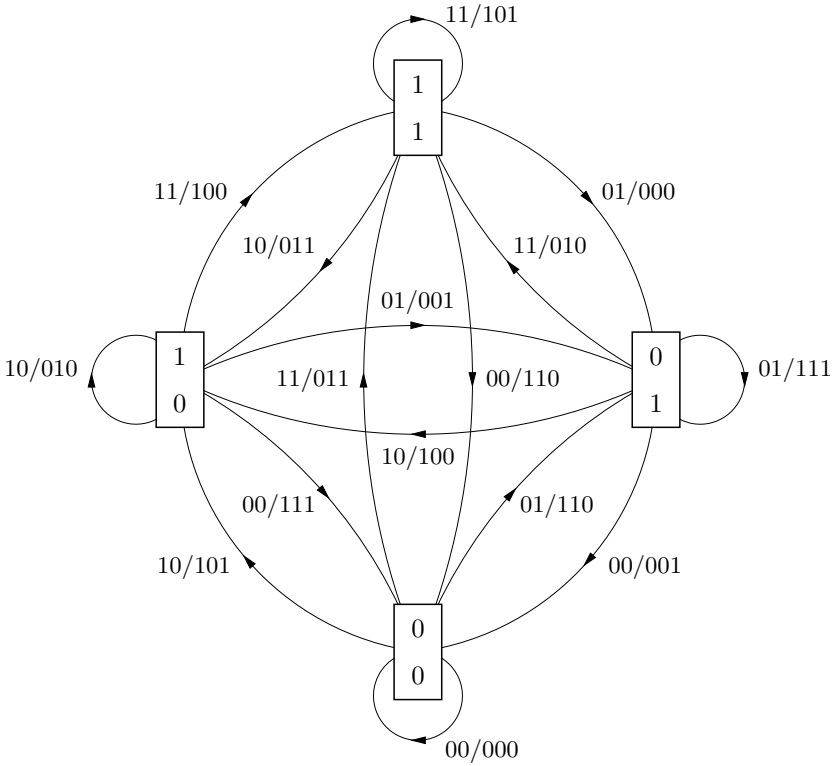


Figure 3.19 State-transition diagram for the encoder in Fig. 3.18.

Viterbi also used the signal flowchart to obtain an *extended path enumerator*, which counts the paths not only according to their weights but also according to their lengths L and to the number of 1's I in the corresponding information sequence.

We return to our encoder in Fig. 3.16 and label the branches not only by W^w, where w is the branch weight, but also by LI^i, where i is the number of 1's among the information symbols corresponding to the particular branch. Thus, we have the *extended signal flowchart* shown in Fig. 3.21.

From this extended signal flowchart we obtain the linear equations

$$\begin{pmatrix} 1 & 0 & -LI \\ -WLI & 1-WLI & 0 \\ -WL & -WL & 1 \end{pmatrix} \begin{pmatrix} \xi_1 \\ \xi_2 \\ \xi_3 \end{pmatrix} = \begin{pmatrix} W^2LI \\ 0 \\ 0 \end{pmatrix} \tag{3.235}$$

and

$$T(W, L, I) = W^2 L \xi_3 \tag{3.236}$$

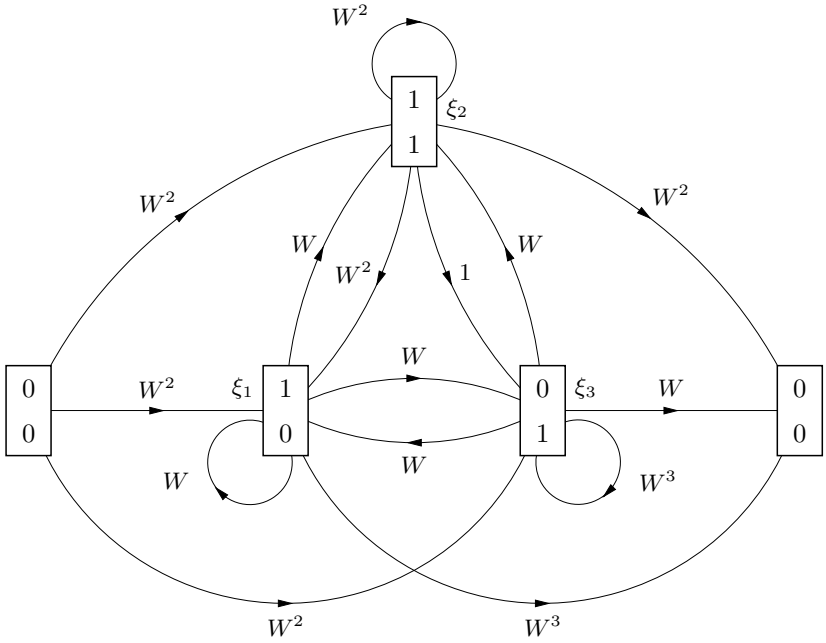


Figure 3.20 Signal flowchart for the encoder in Fig. 3.18.

Solving (3.235) and inserting the solution into (3.236) yield

$$\begin{aligned} T(W,L,I) &= \frac{W^5L^3I}{1-WL(1+L)I} \\ &= W^5L^3I + W^6L^4(1+L)I^2 + W^7L^5(1+L)^2I^3 \\ &\quad + \cdots + W^{5+k}L^{3+k}(1+L)^kI^{1+k} + \cdots \end{aligned} \tag{3.237}$$

Both the path weight enumerator and the extended path enumerator are encoder properties [BHJ10]. For example, consider the systematic encoding matrix

$$G(D) = \begin{pmatrix} 1 & 0 & 1+D \\ 0 & 1 & D \end{pmatrix} \tag{3.238}$$

which is minimal (we showed in Chapter 2 that all systematic encoding matrices are minimal) but not minimal-basic. So it cannot be realized by a minimal encoder in controller canonical form (ccf). However, it can be realized in observer canonical form (ocf) with only one memory element. Its path weight enumerator is

$$\begin{aligned} T_{\text{ocf}}(W) &= \frac{W^2+3W^3-W^5}{1-W-W^2} \\ &= W^2 + 4W^3 + 5W^4 + 8W^5 + 13W^6 + \cdots \end{aligned} \tag{3.239}$$

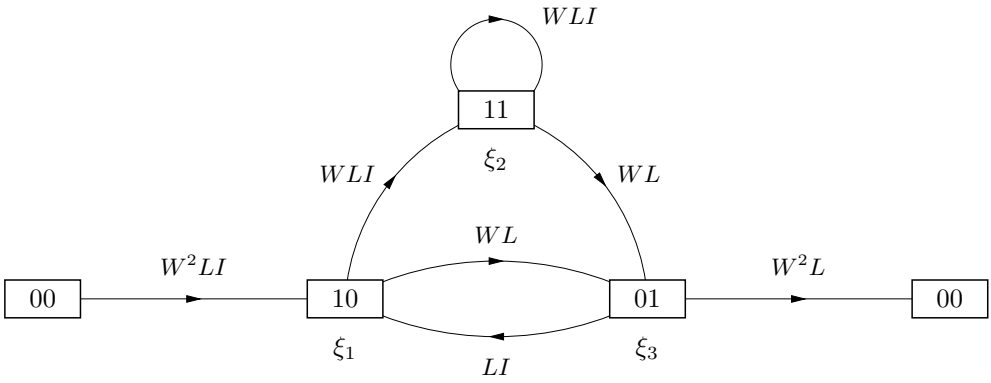


Figure 3.21 Extended signal flowchart for the encoder illustrated in Fig. 3.16.

The realization in controller canonical form requires two memory elements and has the path weight enumerator

$$\begin{aligned} T_{\text{ccf}}(W) &= \frac{W^2 + 3W^3 - W^5}{1 - W - W^2 - 2W^3 + W^5} \\ &= W^2 + 4W^3 + 5W^4 + 10W^5 + 23W^6 + \cdots \end{aligned} \tag{3.240}$$

In the state-transition diagram for a nonminimal encoder we might have an allzero branch between a nonzero state and the allzero state. Thus, a path passing such a nonzero state can reach its termini (the allzero state) either via the allzero branch or by passing other states. In the latter case it will pick up an additional weight of at least d_{free}. We conclude that equivalent generator matrices have the same path weight enumerators up to path weight $2d_{\text{free}} - 1$, but for higher path weights nonminimal realizations might have more paths than minimal ones.

In the next chapter we use these path enumerators to obtain error bounds for maximum-likelihood decoding.

Finally, using an example we will show how to determine the active burst distance by using a simple modification of the signal flowchart.

Consider the rate $R = 1/2$ convolutional encoder in Fig. 3.16 and its signal flowchart in Fig. 3.17. In order to determine the active burst distance, we add another zero state such that when we reach this zero state we will leave it in the next step corresponding to the "bounces" in the zero state. We also label all branches except the first one with J in order to count the order of the active burst distance. Hence, we obtain a modified signal flowchart as illustrated in Fig. 3.22.

As before, we use the dummy variables ξ_1, ξ_2, ξ_3, and ξ_4 to represent the weights and depths from the left zero state to the intermediate states. Thus, we obtain the set

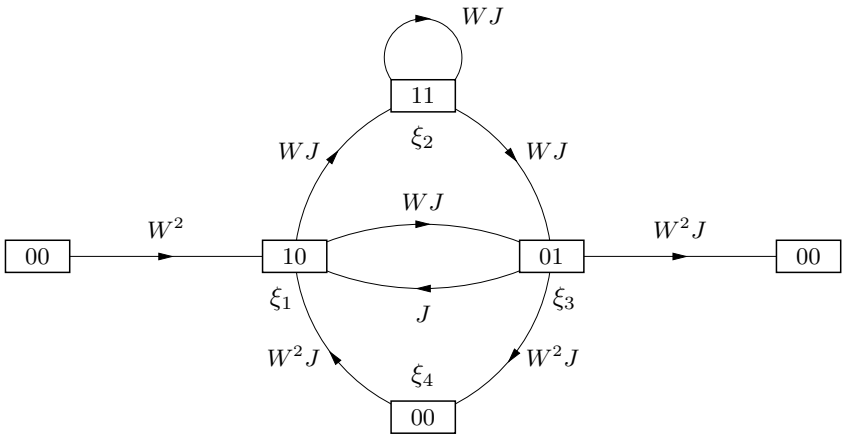


Figure 3.22 Modified signal flowchart for the encoder illustrated in Fig. 3.16.

of linear equations

$$\begin{array}{rcl} \xi_1 & = & J\xi_3 + W^2J\xi_4 + W^2 \\ \xi_2 & = & WJ\xi_1 + WJ\xi_2 \\ \xi_3 & = & WJ\xi_1 + WJ\xi_2 \\ \xi_4 & = & W^2J\xi_3 \end{array} \tag{3.241}$$

and

$$T(W, J) = W^2J\xi_3 \tag{3.242}$$

Equation (3.241) can be rewritten as

$$\begin{pmatrix} 1 & 0 & -J & -W^2J \\ -WJ & 1-WJ & 0 & 0 \\ -WJ & -WJ & 1 & 0 \\ 0 & 0 & -W^2J & 1 \end{pmatrix} \begin{pmatrix} \xi_1 \\ \xi_2 \\ \xi_3 \\ \xi_4 \end{pmatrix} = \begin{pmatrix} W^2 \\ 0 \\ 0 \\ 0 \end{pmatrix} \tag{3.243}$$

Using Cramer's rule we obtain

$$\begin{aligned} \xi_3 = & \left(\det \begin{pmatrix} 1 & 0 & -J & -W^2J \\ -WJ & 1-WJ & 0 & 0 \\ -WJ & -WJ & 1 & 0 \\ 0 & 0 & -W^2J & 1 \end{pmatrix} \right)^{-1} \\ & \times \det \begin{pmatrix} 1 & 0 & W^2 & -W^2J \\ -WJ & 1-WJ & 0 & 0 \\ -WJ & -WJ & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \\ = & \frac{W^3J}{1 - WJ - WJ^2 - W^5J^3} \end{aligned} \tag{3.244}$$

and, hence,

$$\begin{aligned} T(W,J) = W^2 J\xi^3 &= \frac{W^5J^2}{1-WJ-WJ^2-W^5J^3} \\ &= W^5J^2(1+WJ+(W+W^2)J^2+(2W^2+W^3+W^5)J^3 \\ &\quad +(W^2+3W^3+W^4+2W^6)J^4+\cdots) \\ &= W^5J^2+W^6J^3+(W^6+W^7)J^4+(2W^7+W^8+W^{10})J^5 \\ &\quad +(W^7+3W^8+W^9+2W^{11})J^6+\ldots \end{aligned} \tag{3.245}$$

From the expansion of $T(W,J)$ given in (3.245) we have the active burst distances $a_2^{\mathrm{b}}=5$, $a_3^{\mathrm{b}}=6$, $a_4^{\mathrm{b}}=6$, $a_5^{\mathrm{b}}=7$, $a_6^{\mathrm{b}}=7$, ... or since $\nu_{\min}=m=2$, equivalently, the active row distances $a_0^{\mathrm{r}}=5$, $a_1^{\mathrm{r}}=6$, $a_2^{\mathrm{r}}=6$, $a_3^{\mathrm{r}}=7$, $a_4^{\mathrm{r}}=7$, ..., in agreement with the initial part of the curve in Fig. 3.8.

3.11 COMMENTS

Most distance measures for convolutional codes were born at the University of Notre Dame: column distance, row distance, free distance [Cos69], and distance profile [Joh75].

In 1974 Costello published important bounds on the free distance [Cos74]. The lower bound on the distance profile was obtained by Johannesson and Zigangirov [JoZ89].

A family of extended distances was introduced for the class of unit memory (UM), that is, $m=1$, convolutional codes by Thommesen and Justesen [ThJ83]; see also [JTZ88]. They were generalized to $m>1$ convolutional codes by Höst, Johannesson, Zigangirov, and Zyablov and presented together with the corresponding bounds in 1995 [HJZ95, JZZ95]; they are closely related to the active distances [HJZ99].

PROBLEMS

3.1 Consider the convolutional encoding matrix (cf. Problem 1.30)

$$G = \begin{pmatrix} 11 & 10 & 01 & 11 & & \\ & 11 & 10 & 01 & 11 & \\ & & \ddots & \ddots & \ddots & \ddots \end{pmatrix}$$

a) Draw the state-transition diagram.

b) Find the path $\boldsymbol{u} = 11001$ in the state-transition diagram.

c) Find the lowest weight path that leaves the zero state and returns to the zero state.

3.2 Consider the rate $R = 2/3$, memory $m = 2$, overall constraint length $\nu = 3$, convolutional encoder illustrated in Fig. 1.14 (cf. Problem 1.32).

a) Draw the state-transition diagram.

b) Find the path $\boldsymbol{u} = 10\,11\,01\,10$ in the state-transition diagram.

3.3 Consider the rate $R = 1/2$ convolutional code with encoding matrix $G(D) = (1 + D + D^2 \quad 1 + D^2)$.

a) Find the column distances $d_0^{\mathrm{c}}, d_1^{\mathrm{c}}, \ldots, d_\infty^{\mathrm{c}}$.

b) Find the distance profile $\boldsymbol{d}^{\mathrm{p}}$.

c) Find the row distances $d_0^{\mathrm{r}}, d_1^{\mathrm{r}}, \ldots, d_\infty^{\mathrm{r}}$.

3.4 Consider the rate $R = 1/2$ convolutional code with encoding matrix $G(D) = (1 + D + D^2 + D^3 \quad 1 + D^2 + D^3)$ and repeat Problem 3.3.

3.5 Consider the rate $R = 1/3$ convolutional code with encoding matrix $G(D) = (1 + D + D^2 \quad 1 + D + D^2 \quad 1 + D^2)$ and repeat Problem 3.3.

3.6 Consider the rate $R = 2/3$ convolutional encoding matrix

$$G(D) = \begin{pmatrix} 1 + D & D & 1 + D \\ 1 & 1 & D \end{pmatrix}$$

and repeat Problem 3.3.

3.7 Consider the rate $R = 1/2$ convolutional code with encoding matrix $G(D) = (1 + D + D^2 \quad 1 + D^3)$ (cf. Problem 2.4).

a) Draw the state-transition diagram.

b) Find an infinite-weight information sequence that generates a codeword of finite weight.

c) Find d_∞^{c} and d_∞^{r}.

3.8 Find the distance profile and the free distance for the rate $R = 2/3$ convolutional code with encoding matrix

a)

$$G_1(D) = \begin{pmatrix} 1 + D & D & 1 \\ D^2 & 1 & 1 + D + D^2 \end{pmatrix}$$

b)

$$G_2(D) = \begin{pmatrix} 1+D & D & 1 \\ 1+D^2+D^3 & 1+D+D^2+D^3 & 0 \end{pmatrix}$$

c) Show that $G_1(D)$ and $G_2(D)$ encode the same code.

3.9 Consider the rate $R = 1/2$ systematic convolutional encoding matrix $G(D) = (1 \quad 1+D+D^2)$ realized in controller canonical form.

a) Draw its controller canonical form.

b) Draw the state-transition diagram.

c) Draw the signal flowchart.

d) Find the extended path enumerator $T(W, L, I)$.

3.10 Consider the rate $R = 1/2$ convolutional encoding matrix $G(D) = (1+D+D^2+D^3 \quad 1+D^2+D^3)$ realized in controller canonical form.

a) Draw the state-transition diagram.

b) Draw the signal flowchart.

c) Find the path weight enumerator $T(W)$.

3.11 Consider the rate $R = 2/3$ convolutional encoding matrix

$$G(D) = \begin{pmatrix} 1+D & D & 1+D \\ 1 & 1 & D \end{pmatrix}$$

realized in controller canonical form. Find the extended path enumerator $T(W, L, I)$.

3.12 Consider a rate $R = 1/2$ convolutional code with a memory $m = 4$ encoding matrix.

a) Calculate the Heller bound.

b) Calculate the Griesmer bound.

Remark: The optimum time-invariant code with an encoding matrix of memory $m = 4$ has $d^{\mathrm{c}}_{\infty} = d_{\mathrm{free}} = 7$, but there exists a catastrophic time-invariant encoding matrix with $d^{\mathrm{r}}_{\infty} = 8$.

3.13 Repeat Problem 3.12 for rate $R = 1/2$ and memory $m = 5$.

3.14 Repeat Problem 3.12 for rate $R = 1/2$ and memory $m = 28$.

3.15 Consider the rate $R = 2/3$ convolutional encoding matrix

$$G(D) = \begin{pmatrix} 1 & 1 & 0 \\ D & 1+D & 1 \end{pmatrix}$$

realized in controller canonical form.

a) Find the column distances $d^{\mathrm{c}}_0, d^{\mathrm{c}}_1, \ldots, d^{\mathrm{c}}_{\infty}$.

b) Find the extended path enumerator $T(W, L, I)$.

3.16 Consider the rate $R = 1/3$ convolutional code with encoding matrix $G(D) = (1 \quad 1+D+D^2 \quad 1+D^2)$ realized in controller canonical form.

Find the spectral component n_{16}.

3.17 Prove the Costello bound for periodically time-varying, rate $R = b/c$ convolutional codes with polynomial, systematic generator matrices (Theorem 3.30).
Hint: Modify the proof of Theorem 3.28 by using the idea from the proof of Theorem 3.22.

3.18 Prove that for $R \leq 1/2$ there exist binary, time-invariant convolutional codes whose start of the active segment distance is

$$j_{\mathrm{s}} < \frac{R}{1-R}(m+1)$$

3.19 Prove the Costello bound for periodically time-varying, rate $R = b/c$ convolutional codes with period

$$T = \left\lceil \frac{Rm/\rho}{-\log(2^{1-R}-1)} \right\rceil$$

where ρ is the Gilbert-Varshamov parameter.
Hint: Use inequalities (3.145) and (3.154).

CHAPTER 4

DECODING OF CONVOLUTIONAL CODES

In this chapter, we give general descriptions and analyses of important decoding algorithms for convolutional codes. First we study the *Viterbi algorithm*, which we encountered in Chapter 1 [Vit67]. It outputs the codeword that maximizes the probability of the received sequence conditioned on the information sequence, that is, it outputs the maximum-likelihood decision for the codeword. It is a maximum-likelihood (ML) sequence decision algorithm. If the information sequences are equally likely, then the Viterbi algorithm outputs the most probable codeword, that is, it minimizes the codeword error probability.

In the beginning of this chapter, we show that the Viterbi algorithm is an efficient decoding method, particularly when the advantage of soft decisions is fully exploited. The path weight enumerators and the extended path enumerators obtained from the state-transition diagram of the convolutional encoder are used to derive tight upper bounds on the decoding error probabilities for both hard and soft decisions. From these bounds we can estimate the coding gain without the need for experiments or simulations. Then we describe a Markovian technique for an exact calculation of the bit error probability for the Viterbi algorithm.

We consider also an *a posteriori* probability (APP) decoding algorithm that is often called the BCJR algorithm after its inventors Bahl, Cocke, Jelinek, and Raviv [BCJ74]. It is a symbol decoding algorithm and outputs the *a posteriori* probability

Fundamentals of Convolutional Coding, 2. Edition. By Rolf Johannesson, Kamil Sh. Zigangirov

for each of the transmitted information symbols. In other words, for each information symbol it calculates the probability conditioned on the entire received sequence. If we combine the APP decoder with a decision rule which decides in favor of the most likely information symbols given the received sequence, we have a decoder whose outputs are the sequence of the most probable information symbols. In general, the corresponding sequence of code symbols is not the same as the most probable codeword.

After having described APP decoding of convolutional codes, we turn to an efficient and interesting way to terminate convolutional codes into block codes, namely *tailbiting*. These so-called tailbiting (block) codes are discussed in depth. For decoding of tailbiting codes we describe both a maximum-likelihood codeword decoding algorithm called BEAST (Bidirectional Efficient Algorithm for Searching Trees) [BHJ04][Lon07] and an *a posteriori* probability algorithm for information symbol decoding.

4.1 THE VITERBI ALGORITHM REVISITED

First, we return to the Viterbi algorithm, which we introduced in Chapter 1. Suppose that the controller canonical form of the rate $R = 1/2$, memory $m = 2$ convolutional encoding matrix $G(D) = (1 + D + D^2\ 1 + D^2)$ is used to communicate over the BSC with crossover probability ϵ, $0 < \epsilon < 1/2$; that is, the decoder operates on hard decisions. For simplicity we encode only four information symbols followed by $m = 2$ dummy zeros in order to terminate the convolutional code into a block code. This kind of termination is called the zero-tail (ZT) method. Let $\boldsymbol{r} = 10\,01\,10\,01\,01\,00$ be the received sequence.

We recall that when comparing the subpaths leading to each state, the Viterbi algorithm discards all subpaths except the one closest (in Hamming distance) to the received sequence, since those discarded subpaths cannot possibly be the initial part of the path $\widehat{\boldsymbol{v}}$ that minimizes $d_{\mathrm{H}}(\boldsymbol{r}, \boldsymbol{v})$, that is,

$$\widehat{\boldsymbol{v}} = \arg\min_{\boldsymbol{v}}\{d_{\mathrm{H}}(\boldsymbol{r}, \boldsymbol{v})\} \tag{4.1}$$

This is the *principle of nonoptimality*. If we are true to the principle of nonoptimality when we discard subpaths, the path remaining at the end must be the optimal one. In case of a tie, we will assume that a *coin-flip tie-breaking rule* is used. It means that, if several subpaths leading to a trellis state have the same metric, one of them is randomly selected.

The Hamming distances and discarded subpaths at each state determined by the Viterbi algorithm are shown in Fig. 4.1. The decision for the information sequence is $\widehat{\boldsymbol{u}} = 1110$. The successive development of the surviving subpaths through the trellis is illustrated in Fig. 4.2. The decoding delay is as long as the codeword since an optimum decision cannot be made until the surviving paths to all states at a certain depth share a common initial subpath. In principle, this may not happen before the decision is made at the final node and only one path through the trellis remains. However, in practice, for rate $R = 1/2$ a fixed decoding delay of four to five times

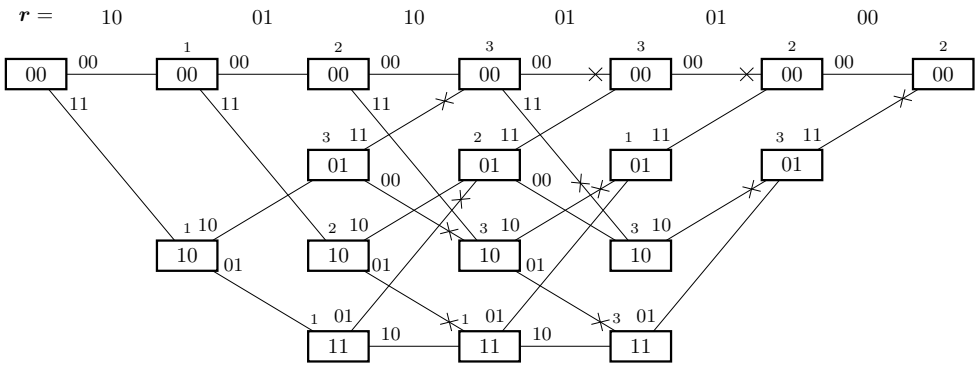


Figure 4.1 An example of Viterbi decoding—hard decisions.

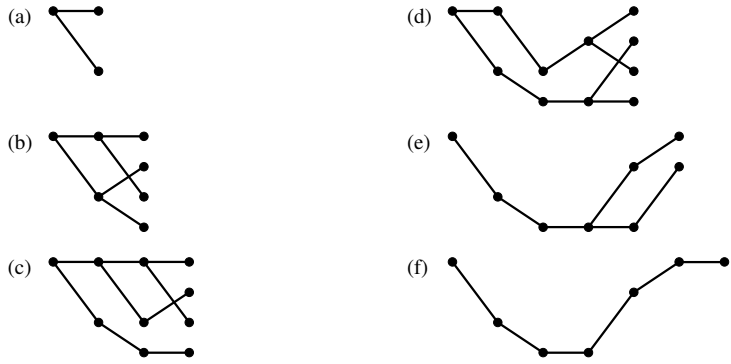


Figure 4.2 Development of subpaths through the trellis.

the encoder memory m causes negligible degradation. In our simple example the maximum decoding delay is four (see Fig. 4.2).

From this example it is clear that the Viterbi algorithm is a general method to do minimum distance decoding efficiently when the overall constraint length is small enough, $\nu < 10$ say.

The ML decoder selects as its decision $\widehat{\boldsymbol{v}}$ that encoded sequence $\boldsymbol{v}$ which maximizes $P(\boldsymbol{r} \mid \boldsymbol{v})$, or, equivalently, since we consider only memoryless channels (DMC),

$$\Pi_k P(\boldsymbol{r}_k \mid \boldsymbol{v}_k) = \Pi_k \Pi_\ell P\left(r_k^{(\ell)} \;\middle|\; v_k^{(\ell)}\right)$$

Taking the logarithm, subtracting a sum that depends only on the received sequence, and multiplying by an arbitrary positive constant A, we find that the maximum-likelihood rule reduces to choosing that encoded sequence $\widehat{\boldsymbol{v}}$ which maximizes the

Viterbi metric

$$\mu_{\mathrm{V}}(\boldsymbol{r}, \boldsymbol{v}) = \sum_k \mu_{\mathrm{V}}(\boldsymbol{r}_k, \boldsymbol{v}_k) = \sum_k \sum_\ell \mu_{\mathrm{V}}\left(r_k^{(\ell)}, v_k^{(\ell)}\right) \tag{4.2}$$

where [Mas84]

$$\mu_{\mathrm{V}}\left(r_k^{(\ell)}, v_k^{(\ell)}\right) = A\left(\log P\left(r_k^{(\ell)} \,\middle|\, v_k^{(\ell)}\right) - f_k^{(\ell)}\left(r_k^{(\ell)}\right)\right) \tag{4.3}$$

We call the quantity $\mu_{\mathrm{V}}(\boldsymbol{r}_k, \boldsymbol{v}_k)$ the *Viterbi branch metric*, and $\mu_{\mathrm{V}}\left(r_k^{(\ell)}, v_k^{(\ell)}\right)$ is the *Viterbi symbol metric*. It is convenient to choose the function $f_k^{(\ell)}\left(r_k^{(\ell)}\right)$ as

$$f_k^{(\ell)}\left(r_k^{(\ell)}\right) = \min_{v_k^{(\ell)}} \left\{\log P\left(r_k^{(\ell)} \,\middle|\, v_k^{(\ell)}\right)\right\} \tag{4.4}$$

if the minimum exists, since then the minimum value of the symbol metric for each received digit $r_k^{(\ell)}$ is zero. Furthermore, we choose the constant A so that the Viterbi symbol metric $\mu_{\mathrm{V}}\left(r_k^{(\ell)}, v_k^{(\ell)}\right)$ can be well approximated by integers.

Suppose that the information sequence $\boldsymbol{u}_{[0,n)}$ followed by mb dummy zeros are encoded by a convolutional encoder of memory m and that the corresponding codeword is transmitted over the BSC with crossover probability ϵ. Let $\boldsymbol{r}_{[0,n+m)}$ denote the received sequence. The ML decoder chooses as its decision $\widehat{\boldsymbol{v}}_{[0,n+m)}$ for the transmitted codeword that codeword $\boldsymbol{v}_{[0,n+m)}$ which maximizes the probability

$$\begin{aligned} P\left(\boldsymbol{r}_{[0,n+m)} \,\middle|\, \boldsymbol{v}_{[0,n+m)}\right) &= (1-\epsilon)^{(n+m)c - d_{\mathrm{H}}(\boldsymbol{r}_{[0,n+m)}, \boldsymbol{v}_{[0,n+m)})} \\ &\quad \times \epsilon^{d_{\mathrm{H}}(\boldsymbol{r}_{[0,n+m)}, \boldsymbol{v}_{[0,n+m)})} \\ &= (1-\epsilon)^{(n+m)c} \left(\frac{\epsilon}{1-\epsilon}\right)^{d_{\mathrm{H}}(\boldsymbol{r}_{[0,n+m)}, \boldsymbol{v}_{[0,n+m)})} \\ &= \prod_{t=0}^{(n+m)c-1} (1-\epsilon)\left(\frac{\epsilon}{1-\epsilon}\right)^{d_{\mathrm{H}}(r_t, v_t)} \end{aligned} \tag{4.5}$$

where the maximization is done over all codewords $\boldsymbol{v}_{[0,n+m)}$.

Hence, the symbol metric

$$\begin{aligned} \mu_{\mathrm{V}}(r_t, v_t) &= A\left(\log\left((1-\epsilon)\left(\frac{\epsilon}{1-\epsilon}\right)^{d_{\mathrm{H}}(r_t, v_t)}\right) - f_t(r_t)\right) \\ &= \left(A \log \frac{\epsilon}{1-\epsilon}\right) d_{\mathrm{H}}(r_t, v_t) + A\log(1-\epsilon) - A f_t(r_t) \end{aligned} \tag{4.6}$$

By choosing

$$A = -\left(\log \frac{\epsilon}{1-\epsilon}\right)^{-1} \tag{4.7}$$

and

$$f_t(r_t) = \log \epsilon \tag{4.8}$$

we get

$$\mu_{\mathrm{V}}(r_t, v_t) = 1 - d_{\mathrm{H}}(r_t, v_t) \tag{4.9}$$

or

$$\mu_{\mathrm{V}}(r_t, v_t) = \begin{cases} 1 & \text{if } r_t = v_t \\ 0 & \text{otherwise} \end{cases} \tag{4.10}$$

From (4.10) it is readily seen that, when communicating over the BSC, maximizing the Viterbi metric is equivalent to minimum (Hamming) distance (MD) decoding.

Before stating the Viterbi algorithm, we remark that finding the shortest route through a directed graph is an old problem in operations research. The following algorithm, first discovered by Viterbi in this context [Vit67], appears in many variants in the shortest-route literature and as dynamic programming in the control literature:

Algorithm V (Viterbi)

V1. Assuming that the convolutional encoder is at the zero state initially, assign the Viterbi metric zero to the initial node; set $t = 0$.

V2. For each node at depth $t + 1$, find for each of the predecessors at depth t the sum of the Viterbi metric of the predecessor and the branch metric of the connecting branch (ADD). Determine the maximum of these sums (COMPARE) and assign it to this node; label the node with the shortest path to it (SELECT).

V3. If we have reached the end of the trellis, then stop and choose as the decoded codeword a path to the terminating node with largest Viterbi metric; otherwise increment t by 1 and go to **V2**.

■ **EXAMPLE 4.1**

Consider the rate $R = 2/3$, memory $m = 1$, overall constraint length $\nu = 2$ encoding matrix

$$G(D) = \begin{pmatrix} 1+D & 1 & 0 \\ D & 1+D & 1+D \end{pmatrix} \tag{4.11}$$

The free distance of the convolutional code is $d_{\text{free}} = 3$. Its convolutional encoder is shown in Fig. 4.3. Suppose that two information two-bit symbols followed by a two-bit dummy **0** are encoded and that $\boldsymbol{r} = 010\,111\,000$ is received over a BSC with $0 < \epsilon < 1/2$.

The Hamming distances and discarded subpaths at each state determined by the Viterbi algorithm are shown in Fig. 4.4. The decided information sequence is $\widehat{\boldsymbol{u}} = 01\,00$.

Puncturing a given convolutional code is a method of constructing new convolutional codes with rates that are higher than the rate of the original code. The

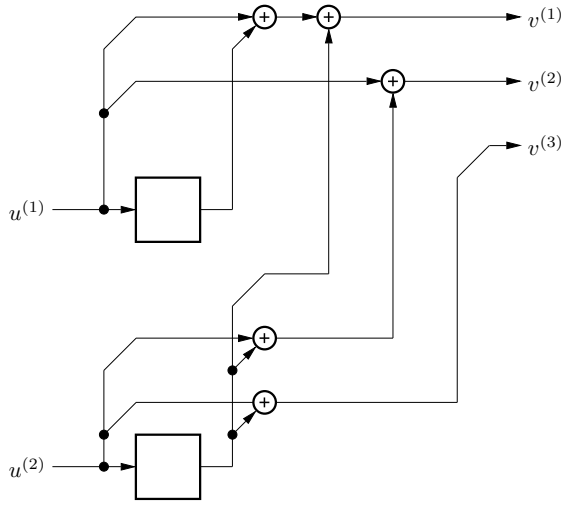


Figure 4.3 The rate $R = 2/3$ encoder used in Example 4.1.

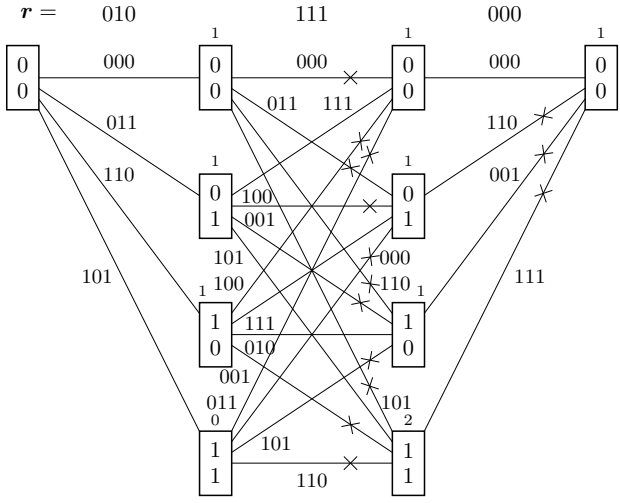


Figure 4.4 An example of Viterbi decoding—rate $R = 2/3$.

punctured codes are in general less powerful than nonpunctured codes of the same rate and memory, but they have two advantages:

From a given original low-rate convolutional code we can obtain a series of convolutional codes with successively higher rates. They can be decoded by the Viterbi algorithm with essentially the same structure as that for the original code [CCG79].

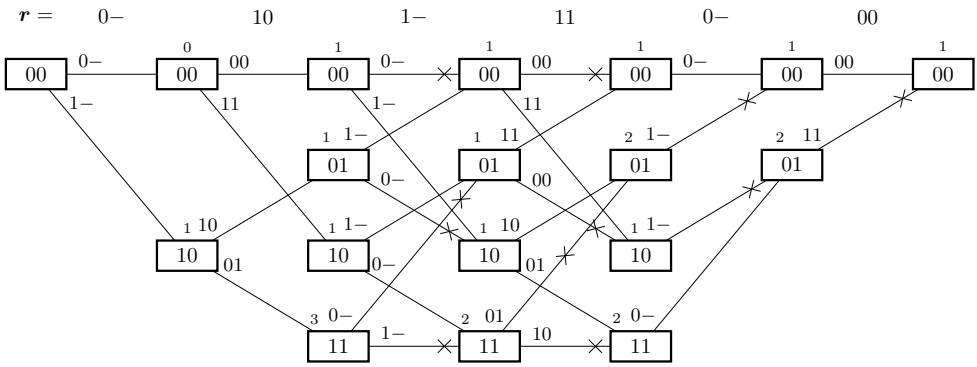


Figure 4.5 Viterbi decoding of a punctured convolutional code.

A *puncturing sequence* is a binary sequence; 0 and 1 means that the corresponding code symbol is not transmitted and transmitted, respectively. This is illustrated in the following little example:

11	10	00	10	original code symbols
10	11	10	11	puncturing sequence
1	10	0	10	punctured code symbols

We have used the periodic sequence $[1011]^\infty$ with period $T = 4$ as puncturing sequence, where $[\quad\quad]^\infty$ denotes a semi-infinite sequence that starts at time 0 and that consists of an infinite repetition of the subsequence between the square brackets.

Suppose that this puncturing sequence is used together with the encoding matrix $G(D) = (1 + D + D^2 \quad 1 + D^2)$ to communicate over a BSC with crossover probability ϵ.

Since the original convolutional code has rate $R = 1/2$, the puncturing sequence has period $T = 4$, and we puncture only one code symbol within the period, the punctured code has three code symbols per two information symbols; that is, the rate of the punctured convolutional code is in our example $R = 2/3$.

Next we use the punctured convolutional code to communicate over a BSC and assume that we receive the same 9 bits, as in Example 4.1; that is, $\boldsymbol{r} = 010111000$. In Fig. 4.5 we show the puctured trellis. The decided information sequence is $\widehat{\boldsymbol{u}} = 0100$, that is, the same as in Example 4.1. Is this a coincidence?

In order to answer that question we consider the semi-infinite generator matrix for the original convolutional code (see (1.95)),

$$\boldsymbol{G} = \begin{pmatrix} 11 & 10 & 11 & & & & \\ & 11 & 10 & 11 & & & \\ & & 11 & 10 & 11 & & \\ & & & 11 & 10 & 11 & \\ & & & & \ddots & \ddots & \ddots \end{pmatrix} \tag{4.12}$$

Then we puncture this generator matrix according to our puncturing sequence, that is,

$$\boldsymbol{G} = \left(\begin{array}{cc|cc|ccc} 11 & 10 & 11 & & & & \\ & 11 & 10 & 11 & & & \\ & & 11 & 10 & 11 & & \\ & & & 11 & 10 & 11 & \\ & & & & \ddots & \ddots & \ddots \end{array}\right) \tag{4.13}$$

Gathering the columns of (4.13) by three and three yields

$$\boldsymbol{G}_{\text{punct}} = \begin{pmatrix} 110 & 100 & & \\ 011 & 111 & & \\ & 110 & 100 & \\ & 011 & 111 & \\ & & \ddots & \ddots \end{pmatrix} \tag{4.14}$$

or, equivalently,

$$G_{\text{punct}}(D) = \begin{pmatrix} 1+D & 1 & 0 \\ D & 1+D & 1+D \end{pmatrix} \tag{4.15}$$

that is, via puncturing we have obtained the rate $R = 2/3$ convolutional code used in Example 4.1.

We will now consider how to do Viterbi decoding in a soft decisions situation. This gives us an "extra" 2–3 dB coding gain for "free" (cf. p. 250).

■ **EXAMPLE 4.2**

Consider the binary-input, 8-ary output DMC shown in Fig. 4.6 with transition probabilities $P(r \mid v)$ given by the following table:

		r							
		0_4	0_3	0_2	0_1	1_1	1_2	1_3	1_4
v	0	0.434	0.197	0.167	0.111	0.058	0.023	0.008	0.002
	1	0.002	0.008	0.023	0.058	0.111	0.167	0.197	0.434

Taking the logarithms, we have

		r							
		0_4	0_3	0_2	0_1	1_1	1_2	1_3	1_4
v	0	−0.83	−1.62	−1.79	−2.20	−2.85	−3.77	−4.83	−6.21
	1	−6.21	−4.83	−3.77	−2.85	−2.20	−1.79	−1.62	−0.83

For each r we subtract $\min_v \log P(r \mid v)$ and obtain

		r							
		0_4	0_3	0_2	0_1	1_1	1_2	1_3	1_4
v	0	5.38	3.21	1.98	0.65	0.00	0.00	0.00	0.00
	1	0.00	0.00	0.00	0.00	0.65	1.98	3.21	5.38

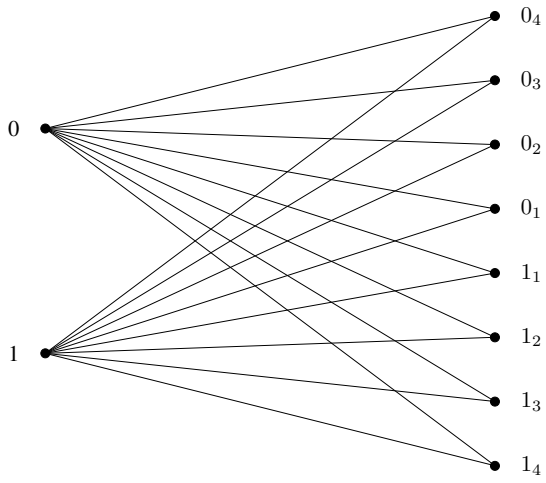


Figure 4.6 Binary-input, 8-ary output, DMC.

Finally, after scaling ($A = 1.5$) and rounding we have the following table:

		r							
		0_4	0_3	0_2	0_1	1_1	1_2	1_3	1_4
v	0	8	5	3	1	0	0	0	0
	1	0	0	0	0	1	3	5	8

These Viterbi symbol metrics will be used by the Viterbi algorithm when decoding a sequence received over the channel shown in Fig. 4.6.

Suppose that the same encoder as in the hard decisions example, viz., the controller canonical form of $G(D) = (1 + D + D^2 \quad 1 + D^2)$, is used and that again four information symbols followed by two dummy zeros are encoded. Let $\boldsymbol{r} = 1_1 0_4 \; 0_1 1_2 \; 1_1 0_1 \; 0_1 1_1 \; 0_1 1_3 \; 0_4 0_3$ be the received sequence. The Viterbi metrics L_{V} are calculated for all subpaths leading to each state. Only the subpath with the largest metric is kept—all other subpaths are discarded. The trellis with Viterbi metrics and discarded subpaths is shown in Fig. 4.7. The decision for the information sequence is $\widehat{\boldsymbol{u}} = 0110$. The successive development of the surviving subpaths through the trellis is shown in Fig. 4.8.

Notice that the received sequence in our hard decision example is exactly the received sequence, which we obtain if we merge the soft decision outputs $0_1, 0_2, 0_3, 0_4$ and $1_1, 1_2, 1_3, 1_4$ in Example 4.2 into the hard decision outputs 0 and 1, respectively. The decisions for the information sequences based on hard and soft decisions, respectively, differ in the first digit. Thus, here we have a specific example showing the importance of exploiting the full information provided by the soft output demodulator.

From this example it is clear that the Viterbi algorithm is an efficient maximum-likelihood decoding procedure that can easily exploit soft decisions.

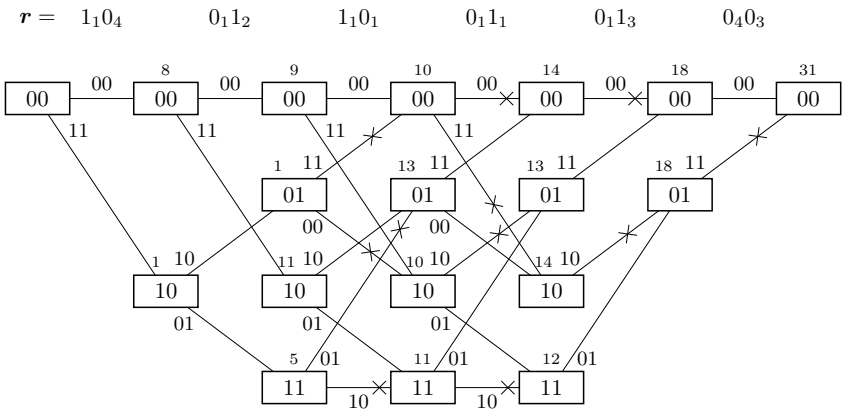


Figure 4.7 An example of Viterbi decoding—soft decisions.

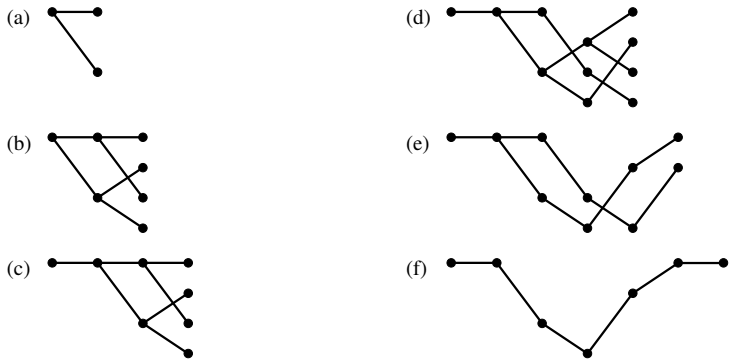


Figure 4.8 Development of subpaths through the trellis.

We conclude this section by observing that when comparing the subpaths leading to each state, we discard all subpaths except the one with the largest Viterbi metric, since those discarded subpaths cannot possibly be the initial part of the path $\widehat{\boldsymbol{v}}$ that maximizes the conditional probability of the received sequence $\boldsymbol{r}$ given the transmitted sequence $\boldsymbol{v}$, that is, $P(\boldsymbol{r} \mid \boldsymbol{v})$. In case of a tie, we choose one of the maximizing paths as the survivor by using the *coin-flip tie-breaking rule*, that is, in each trellis state where the path splits into two possible directions, one of them is selected at random. The path remaining at the end must be a maximum-likelihood decision. We summarize the discussion in this section as the following important theorem [For67, For94].

Theorem 4.1 The Viterbi algorithm is a maximum-likelihood decoding algorithm.

4.2 ERROR BOUNDS FOR TIME-INVARIANT CONVOLUTIONAL CODES

In practice, the Viterbi algorithm is often used for a rate $R = b/c$ convolutional code to decode long sequences of received symbols, typically a few thousand bits, before the decoder is forced back to the zero state by a tail of mb dummy zeros that are fed into the encoder in order to terminate the frame (the ZT method). An erroneously decoded path will in general remerge with the correct path long before it reaches the end of the trellis. Thus, a typical error event consists of a burst of erroneously decoded information digits. Such a burst always starts and ends with an error. Furthermore, if we have mb or more consecutively correct information digits among the erroneously decoded information digits, then the erroneous path has remerged with the correct path before it diverges again—a *multiple-error event*, which consists of separate error bursts.

The block error probability is not the appropriate quality measure for Viterbi decoding. If we use very long frames the block error probability will be close to 1 even if the system provides adequate protection of the information digits. In this case we shall use the *bit error probability*, P_{b}, considered in the previous section as our quality measure. Again we would like to stress that the bit error probability is not only a code property but also an encoding matrix property. It depends on the map between the information sequences and the codewords. We shall also introduce the *burst error probability*, P_{B}, which is the probability that an error burst starts at a given node. The burst error probability is a code property and is sometimes called *first-event error probability* or *node error probability*. Since it is easier to obtain good bounds on the burst error probability than on the bit error probability, we shall study the burst error probability first.

The burst error probability for a convolutional code consisting of finite-length codewords is always upper-bounded by the burst error probability for infinite-length codewords if they are encoded by the same encoder. This is readily seen from the fact that we cannot do better by adding more adversary paths.

The burst error probability is *not* the same for all nodes along the correct path. This is readily seen from the following argument. Suppose that the first burst starts at depth i, $i > 0$. Typically, this burst has not been caused by an error event containing many channel errors in the beginning of it since such an error event would have caused a burst starting at an earlier depth. Hence, the burst error probability at depth i, $i > 0$, is not greater than that at the root. However, our upper-bounding of the burst error probability is valid for any node.

Suppose that the convolutional code whose trellis diagram is shown in Fig. 4.9 is used for communication over a BSC with crossover probability ϵ, $0 < \epsilon < 1/2$.

What is the probability that the Viterbi decoder selects the codeword $11\,10\,11\,00\,00\ldots$ in favor of the transmitted allzero codeword? This particular decoding error will occur when there are three or more errors among the five digits in which the codewords differ, that is, in positions 1, 2, 3, 5, and 6. The probability for this event

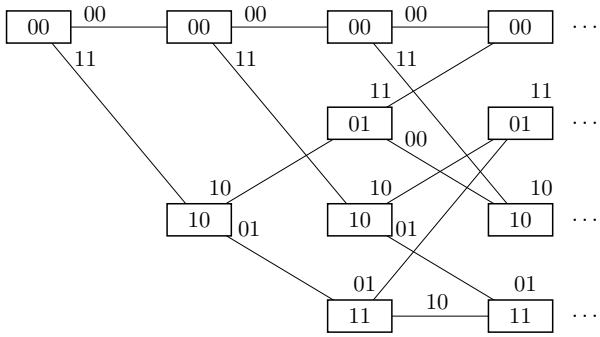


Figure 4.9 Trellis diagram for a convolutional encoder.

is

$$\begin{aligned} p_5 &= P(\text{3 or more 1's in 5 positions}) \\ &= \sum_{e=3}^{5} \binom{5}{e} \epsilon^e (1-\epsilon)^{5-e} \end{aligned} \tag{4.16}$$

If the distance d between the correct path and its adversary is *even*, which, for example, is the case if we consider the codeword $11\,01\,01\,11\,00\ldots$ instead, we do not necessarily make a decoding error when there are $d/2$ errors among the d digits in the positions in which the codewords differ. In this case there is a tie between the two paths, and we will discard the correct one only with probability $1/2$. For $d = 6$ we have

$$p_6 = \frac{1}{2}\binom{6}{3}\epsilon^3(1-\epsilon)^3 + \sum_{e=4}^{6}\binom{6}{e}\epsilon^e(1-\epsilon)^{6-e} \tag{4.17}$$

In general, we have

$$p_d = \begin{cases} \displaystyle\sum_{e=(d+1)/2}^{d} \binom{d}{e}\epsilon^e(1-\epsilon)^{d-e}, & d \text{ odd} \\ \displaystyle\frac{1}{2}\binom{d}{d/2}\epsilon^{d/2}(1-\epsilon)^{d/2} + \sum_{e=d/2+1}^{d}\binom{d}{e}\epsilon^e(1-\epsilon)^{d-e}, & d \text{ even} \end{cases} \tag{4.18}$$

Since $\epsilon^e(1-\epsilon)^{d-e}$ is increasing with decreasing e, we notice that for d odd

$$\begin{aligned} p_d &= \sum_{e=(d+1)/2}^{d}\binom{d}{e}\epsilon^e(1-\epsilon)^{d-e} < \sum_{e=(d+1)/2}^{d}\binom{d}{e}\epsilon^{d/2}(1-\epsilon)^{d/2} \\ &= \epsilon^{d/2}(1-\epsilon)^{d/2}\sum_{e=(d+1)/2}^{d}\binom{d}{e} < \epsilon^{d/2}(1-\epsilon)^{d/2}\sum_{e=0}^{d}\binom{d}{e} \\ &= \left(2\sqrt{\epsilon(1-\epsilon)}\right)^d \end{aligned} \tag{4.19}$$

It can be shown (Problem 4.17) that $\left(2\sqrt{\epsilon(1-\epsilon)}\right)^d$ is an upper bound on p_d also when d is even. Hence, we have the *Bhattacharyya bound* [Bha43]

$$p_d < \left(2\sqrt{\epsilon(1-\epsilon)}\right)^d \stackrel{\text{def}}{=} z^d, \quad \text{all } d \tag{4.20}$$

where z is called the *Bhattacharyya parameter* for the BSC.

■ **EXAMPLE 4.3**

Consider the BSC with $\epsilon = 0.01$. For this channel

$$2\sqrt{\epsilon(1-\epsilon)} \approx 0.2 \tag{4.21}$$

and

$$p_5 < 0.2^5 \approx 3.2 \cdot 10^{-4} \tag{4.22}$$

which is much less than the channel crossover probability $\epsilon = 0.01$.

Assuming Viterbi decoding, a necessary, but not sufficient, condition for a burst error to occur at the root is that the received sequence given an incorrect path diverging from the correct path at the root is more likely than the received sequence given the correct path. This condition is not sufficient since, even if it is fulfilled, the ultimately chosen path might begin with a correct branch.

Although the burst error probability is a code property, we will upper-bound it by considering paths diverging from the correct path at the root of the trellis and not returning to the allzero state until their termini. Such paths are encoder properties.

Let $\mathcal{E}^{(k)}$ be the event that a burst error at the root is caused by path k. Then we have

$$P_{\text{B}} \leq P(\cup \mathcal{E}^{(k)}) \leq \sum P(\mathcal{E}^{(k)}) \tag{4.23}$$

where the second inequality follows from the union bound, and the union and the sum are over all incorrect paths diverging from the correct path at the root.

Since a convolutional code is linear, we can without loss of generality assume that the correct path is the allzero sequence. Then, if the Hamming weight of the kth incorrect path is d, we have

$$P(\mathcal{E}^{(k)}) = p_d \tag{4.24}$$

where p_d is given in (4.18). Combining (4.23) and (4.24), we obtain

$$P_{\text{B}} \leq \sum_{d=d_{\text{free}}}^{\infty} n_d p_d \tag{4.25}$$

where n_d is the number of weight d paths, that is, the Viterbi weight spectrum of the convolutional encoder (cf. (3.35)). In order to obtain the best Viterbi weight spectrum for the convolutional code we use a minimal realization of a minimal encoding metric.

The number of weight d paths for $d = d_{\text{free}}, d_{\text{free}} + 1, \ldots$ for a minimal realization of a minimal encoding matrix is given by the path weight enumerator $T(W)$ discussed in Section 3.10:

$$T(W) = \sum_{d=d_{\text{free}}}^{\infty} n_d W^d \tag{4.26}$$

Thus, combining (4.20), (4.25), and (4.26), we get the following [Vit71]:

Theorem 4.2 (Viterbi) When using a convolutional code for communication over the BSC with crossover probability ϵ and maximum-likelihood decoding, the *burst error probability* is upper-bounded by

$$P_{\text{B}} < \sum_{d=d_{\text{free}}}^{\infty} n_d \left(2\sqrt{\epsilon(1-\epsilon)}\right)^d = T(W)\Big|_{W=2\sqrt{\epsilon(1-\epsilon)}} \tag{4.27}$$

where the path weight enumerator $T(W)$ is obtained for a minimal realization of a minimal encoding matrix.

■ **EXAMPLE 4.4**

Consider the BSC with $\epsilon = 0.01$ and the rate $R = 1/2$ convolutional code with the encoding matrix $G(D) = (1 + D + D^2 \quad 1 + D^2)$. The path weight enumerator for a minimal realization of this minimal encoding matrix is (3.230)

$$T(W) = \frac{W^5}{1 - 2W} \tag{4.28}$$

Viterbi's upper bound on the burst error probability is

$$\begin{aligned} P_{\text{B}} &< T\left(2\sqrt{\epsilon(1-\epsilon)}\right) \approx T(0.2) \\ &= \frac{0.2^5}{1 - 2 \cdot 0.2} \approx 5 \cdot 10^{-4} \end{aligned} \tag{4.29}$$

Van de Meeberg [Mee74] noticed that (see Problem 4.18)

$$p_{2i-1} = p_{2i}, \quad i \geq 1 \tag{4.30}$$

That is, for each path the decoding error probability when d is odd is the same as that for the case when d is increased by 1! Thus, when d is odd, the Bhattacharyya bound (4.20) can be replaced by

$$p_d < \left(2\sqrt{\epsilon(1-\epsilon)}\right)^{d+1}, \quad d \text{ odd} \tag{4.31}$$

Then from

$$\begin{aligned} &\sum_{d \text{ even}} n_d W^d + \sum_{d \text{ odd}} n_d W^{d+1} \\ &= \frac{1}{2}(T(W) + T(-W)) + \frac{W}{2}(T(W) - T(-W)) \end{aligned} \tag{4.32}$$

we have the next theorem.

Theorem 4.3 (Van de Meeberg) When using a convolutional code for communication over the BSC with crossover probability ϵ and maximum-likelihood decoding, the *burst error probability* is upper-bounded by

$$\begin{aligned} P_{\mathrm{B}} &< \left(\frac{1}{2}(T(W) + T(-W)) + \frac{W}{2}(T(W) - T(-W)) \right) \Big|_{W=2\sqrt{\epsilon(1-\epsilon)}} \\ &= \left(\frac{1+W}{2} T(W) + \frac{1-W}{2} T(-W) \right) \Big|_{W=2\sqrt{\epsilon(1-\epsilon)}} \end{aligned} \tag{4.33}$$

where the path weight enumerator $T(W)$ is obtained for a minimal realization of a minimal encoding matrix.

Remark: Actually, van de Meeberg [Mee74] derived a slightly tighter bound (Problem 4.19).

■ **EXAMPLE 4.5**

Consider the same channel and encoding matrix as in Example 4.4. Van de Meeberg's upper bound on the burst error probability is

$$\begin{aligned} P_{\mathrm{B}} &< \frac{1+2\sqrt{\epsilon(1-\epsilon)}}{2} T\left(2\sqrt{\epsilon(1-\epsilon)}\right) + \frac{1-2\sqrt{\epsilon(1-\epsilon)}}{2} T\left(-2\sqrt{\epsilon(1-\epsilon)}\right) \\ &\approx \frac{1+0.2}{2} T(0.2) + \frac{1-0.2}{2} T(-0.2) \approx 2 \cdot 10^{-4} \end{aligned} \tag{4.34}$$

which is an improvement by a factor 2.5 compared to Viterbi's bound.

In Fig. 4.10, simulations of the burst error probability for Viterbi decoding are compared with Viterbi's and van de Meeberg's upper bounds when the rate $R = 1/2$, memory $m = 2$ minimal encoder with encoding matrix $G(D) = (1 + D + D^2 \quad 1 + D^2)$ is used for communication over the BSC. Notice the close agreement between van de Meeberg's bound and the measurements, in particular for small ϵ. The perhaps somewhat surprisingly large difference between the two bounds is explained by the fact that this code has an odd free distance, viz., $d_{\text{free}} = 5$. In Fig. 4.11 we show the corresponding curves for Qualcomm's rate $R = 1/2$, memory $m = 6$ encoder with encoding matrix $G(D) = (1 + D + D^2 + D^3 + D^6 \quad 1 + D^2 + D^3 + D^5 + D^6)$ and $d_{\text{free}} = 10$.

We will now use the extended path enumerator and derive an upper bound on the bit error probability.

Suppose that the received sequence $\boldsymbol{r} = \boldsymbol{r}_0 \boldsymbol{r}_1 \ldots \boldsymbol{r}_{K-1}$ of length K c-tuples is decoded as $\widehat{\boldsymbol{v}} = \widehat{\boldsymbol{v}}_0 \widehat{\boldsymbol{v}}_1 \ldots \widehat{\boldsymbol{v}}_{K-1}$ and that the corresponding decision of the information sequence $\widehat{\boldsymbol{u}} = \widehat{\boldsymbol{u}}_0 \widehat{\boldsymbol{u}}_1 \ldots \widehat{\boldsymbol{u}}_{K-1}$ contains $I(K)$ errors.

Let I denote the number of erroneously decided information symbols in a burst of length L b-tuples and let N denote the number of b-tuples between two bursts. In particular, burst j, $j = 0, 1, 2, \ldots$, contains I_j erroneously decided information

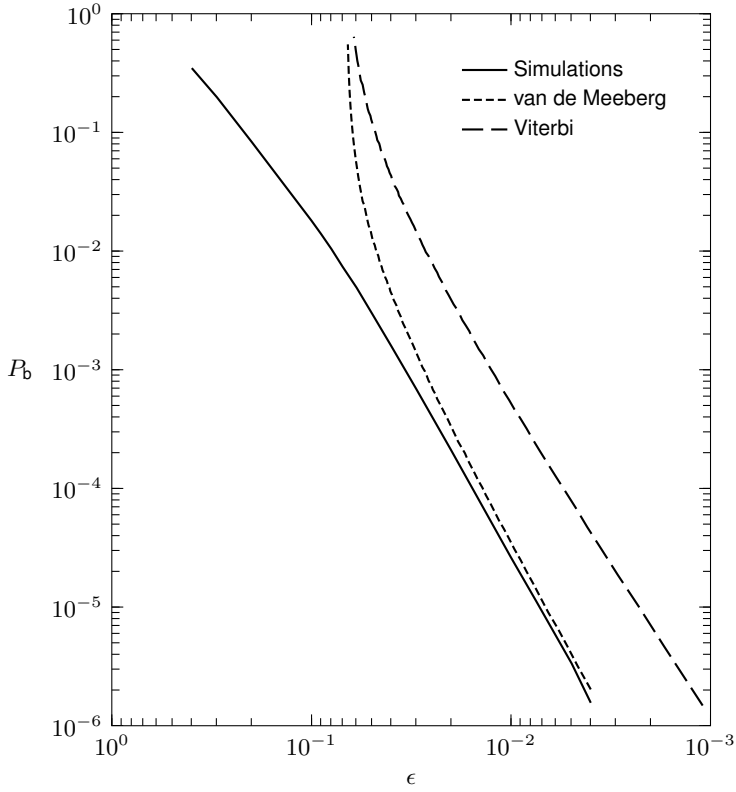


Figure 4.10 Viterbi's and van de Meeberg's upper bounds on the burst error probability for the minimal encoding matrix $G(D) = (1+D+D^2 \quad 1+D^2)$ realized in controller canonical form and the BSC with crossover probability ϵ.

symbols, is of length L_j b-tuples, and is separated by N_j b-tuples from the previous burst, where

$$N_j > m, \quad j = 0, 1, 2, \ldots \tag{4.35}$$

Erroneously decided information symbols separated by m or fewer b-tuples belong by definition to the same error burst.

We define the *bit error probability* P_b to be the ratio between the number of erroneously decoded information symbols and the total number of information symbols.

According to the law of large numbers, we have

$$\lim_{K\to\infty} P\left(\left|P_b - \frac{I(K)}{Kb}\right| > \epsilon\right) = 0 \tag{4.36}$$

for any $\epsilon > 0$ or, equivalently,

$$P_b = \lim_{K\to\infty} \frac{I(K)}{Kb} \quad \text{with probability 1} \tag{4.37}$$

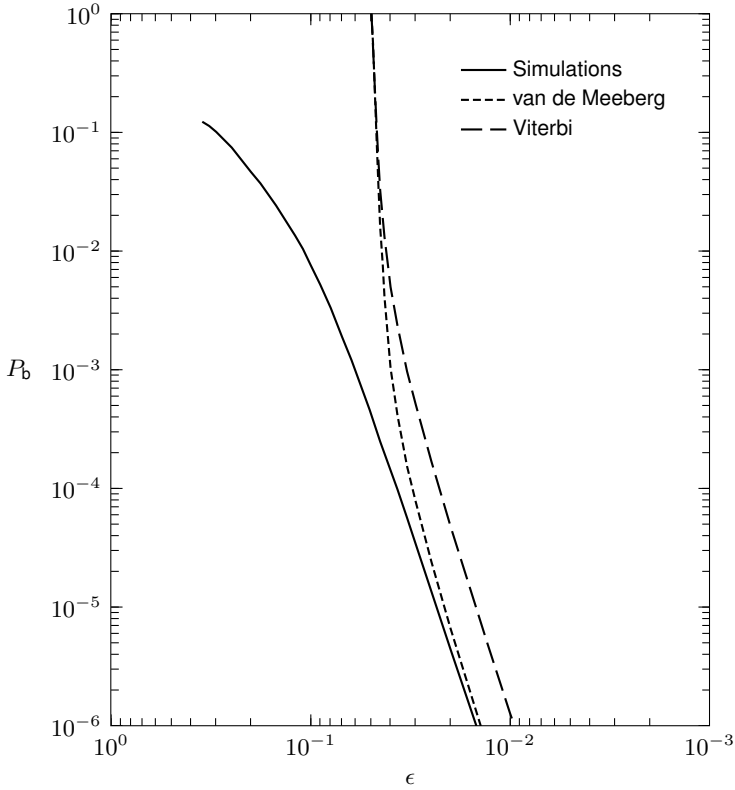


Figure 4.11 Viterbi's and van de Meeberg's upper bounds on the burst error probability for Qualcomm's $R = 1/2$, $m = 6$ minimal encoding matrix realization in controller canonical form and the BSC with crossover probability ϵ.

Assume that the $I(K)$ errors are distributed over J bursts with $I_0, I_1, \ldots, I_{J-1}$ errors, of lengths $L_0, L_1, \ldots, L_{J-1}$, and separated by error-free intervals of lengths $N_0, N_1, \ldots, N_{J-1}$. From (4.37) it follows that

$$\begin{aligned} P_b &= \lim_{J\to\infty} \frac{\sum_{j=0}^{J-1} I_j}{b\sum_{j=0}^{J-1}(N_j + L_j)} \\ &\leq \lim_{J\to\infty} \frac{\sum_{j=0}^{J-1} I_j}{b\sum_{j=0}^{J-1} N_j} \quad \text{with probability 1} \end{aligned} \tag{4.38}$$

and that

$$P_b \leq \frac{\lim_{J\to\infty} \frac{1}{J}\sum_{j=0}^{J-1} I_j}{b\lim_{J\to\infty} \frac{1}{J}\sum_{j=0}^{J-1} N_j} \quad \text{with probability 1} \tag{4.39}$$

According to the law of large numbers the limit in the numerator is equal to the expected value of the number I of bit errors in a burst, that is,

$$E\left[I \mid \text{burst}\right] = \lim_{J\to\infty} \frac{1}{J} \sum_{j=0}^{J-1} I_j \quad \text{with probability 1} \tag{4.40}$$

and the limit in the denominator is equal to the expected value of the length (in branches) of the error-free interval, that is,

$$E[N] = \lim_{J\to\infty} \frac{1}{J} \sum_{j=0}^{J-1} N_j \quad \text{with probability 1} \tag{4.41}$$

By combining (4.39), (4.40), and (4.41), we obtain

$$P_{\text{b}} \leq \frac{E\left[I \mid \text{burst}\right]}{bE[N]} \tag{4.42}$$

The probability, P_{B}, that an error burst starts at a given node can be defined as the limit when $J \to \infty$ of the number of bursts J divided by the number of instances where a burst could have started. This latter number is less than $\sum_{j=0}^{J-1} N_j$. (In fact, it is less than or equal to $\sum_{j=0}^{J-1}(N_j - m)$.) Hence, we have

$$\begin{aligned} P_{\text{B}} &\geq \lim_{J\to\infty} \frac{J}{\sum_{j=0}^{J-1} N_j} \\ &= \frac{1}{\lim_{J\to\infty} \frac{1}{J} \sum_{j=0}^{J-1} N_j} = \frac{1}{E[N]} \quad \text{with probability 1} \end{aligned} \tag{4.43}$$

From the law of large numbers and from inequalities (4.42) and (4.43), we deduce that

$$P_{\text{b}} \leq \frac{1}{b} E\left[I \mid \text{burst}\right] P_{\text{B}} \tag{4.44}$$

Let $p(i)$ be the probability of a burst introducing i errors in the decided information sequence. Then we have

$$P_{\text{B}} = \sum_{i=1}^{\infty} p(i) \tag{4.45}$$

or, equivalently,

$$\sum_{i=1}^{\infty} p\left(i \mid \text{burst}\right) = 1 \tag{4.46}$$

where

$$p\left(i \mid \text{burst}\right) = \frac{p(i)}{P_{\text{B}}} \tag{4.47}$$

and

$$E\left[I \mid \text{burst}\right] = \sum_{i=1}^{\infty} ip\left(i \mid \text{burst}\right) = \frac{\sum_{i=1}^{\infty} ip(i)}{P_{\text{B}}} \tag{4.48}$$

From (4.44) and (4.48) we conclude that

$$P_{\mathrm{b}} \leq \frac{1}{b} \sum_{i=1}^{\infty} i p(i) \tag{4.49}$$

Let $n(w, \ell, i)$ be the number of weight w paths of length ℓ introducing i errors in the decided information sequence. From the Bhattacharyya bound (4.20) it follows that the probability that a path of weight w causes a decoding error is upper-bounded by

$$\left(2\sqrt{\epsilon(1-\epsilon)}\right)^w$$

where ϵ is the crossover probability of the BSC. Then, by applying the union bound, we obtain

$$p(i) < \sum_{w=d_{\text{free}}}^{\infty} \sum_{\ell=\nu_{\min}+1}^{\infty} n(w, \ell, i) \left(2\sqrt{\epsilon(1-\epsilon)}\right)^w \tag{4.50}$$

where $\nu_{\min} = \min_i\{\nu_i\}$. The numbers $n(w, \ell, i)$ are given by the extended path enumerator $T(W, L, I)$ discussed in Section 3.10:

$$T(W, L, I) = \sum_w \sum_\ell \sum_i n(w, \ell, i) W^w L^\ell I^i \tag{4.51}$$

Combining (4.49), (4.50), and (4.51), we obtain the next theorem [Vit71].

Theorem 4.4 (Viterbi) When using a convolutional code with generator matrix $G(D)$ for communication over the BSC with crossover probability ϵ and maximum-likelihood decoding, the *bit error probability* is upper-bounded by

$$\begin{aligned} P_{\mathrm{b}} &< \frac{1}{b} \sum_{w=d_{\text{free}}}^{\infty} \sum_{\ell=\nu_{\min}+1}^{\infty} \sum_{i=1}^{\infty} i n(w, \ell, i) \left(2\sqrt{\epsilon(1-\epsilon)}\right)^w \\ &= \frac{1}{b} \frac{\partial T(W, L, I)}{\partial I} \Bigg|_{\substack{W=2\sqrt{\epsilon(1-\epsilon)} \\ L=1 \\ I=1}} \end{aligned} \tag{4.52}$$

where the extended path enumerator $T(W, L, I)$ is obtained for a minimal encoder.

■ **EXAMPLE 4.6**

Consider the BSC with $\epsilon = 0.01$ and the rate $R = 1/2$ convolutional code with the encoding matrix $G(D) = (1 + D + D^2 \quad 1 + D^2)$. Its extended path enumerator when realized in controller canonical form is (3.237)

$$T(W, L, I) = \frac{W^5 L^3 I}{1 - WL(1 + L)I} \tag{4.53}$$

Since

$$\frac{\partial T(W,L,I)}{\partial I} = \frac{W^5 L^3}{(1 - WL(1+L)I)^2} \tag{4.54}$$

Viterbi's upper bound on the bit error probability is

$$P_\mathrm{b} < \frac{\left(2\sqrt{\epsilon(1-\epsilon)}\right)^5}{\left(1 - 2\left(2\sqrt{\epsilon(1-\epsilon)}\right)\right)^2} \approx \frac{0.2^5}{(1 - 2 \cdot 0.2)^2} \approx 0.9 \cdot 10^{-3} \tag{4.55}$$

which shows that even by using this simple coding we can achieve an improvement of at least a factor of 10 over the raw error probability of the channel.

As a counterpart to van de Meeberg's tightening of Viterbi's bound on the burst error probability, we have the following (Problem 4.22):

Theorem 4.5 (Van de Meeberg) When using a convolutional code with generator matrix $G(D)$ for communication over the BSC with crossover probability ϵ and maximum-likelihood decoding, the *bit error probability* is upper-bounded by

$$P_\mathrm{b} < \frac{1}{b}\left(\frac{1+W}{2}\frac{\partial T(W,L,I)}{\partial I} + \frac{1-W}{2}\frac{\partial T(-W,L,I)}{\partial I}\right)\Bigg|_{\substack{W=2\sqrt{\epsilon(1-\epsilon)} \\ L=1 \\ I=1}} \tag{4.56}$$

where the extended path enumerator $T(W, L, I)$ is obtained for a minimal encoder.

From (4.56) it follows that the bit error probability P_b for the convolutional code with encoding matrix $G(D) = (1 + D + D^2 \quad 1 + D^2)$ realized in controller canonical form is upper-bounded by $320\epsilon^3 + O(\epsilon^4)$. In Section 4.4 we shall show that $P_\mathrm{b} = 44\epsilon^3 + O(\epsilon^4)$.

In Figs. 4.12 and 4.13 we compare Viterbi's and van de Meeberg's upper bounds on the bit error probability with simulations for the encoding matrix $G(D) = (1 + D + D^2 \quad 1 + D^2)$ and Qualcomm's $R = 1/2$, $m = 6$ encoder, respectively, both realized in controller canonical form.

Now suppose that a rate R convolutional code is used to communicate over the additive white Gaussian noise (AWGN) channel with BPSK modulation at the signal-to-noise ratio (SNR) E_s/N_0, where the *energy per symbol* E_s is related to the *energy per information bit* through the rate R:

$$E_\mathrm{s} = RE_\mathrm{b} \tag{4.57}$$

We assume that the allzero codeword $\mathbf{0}$ is transmitted and that there is a codeword $\boldsymbol{v}$ with $w_\mathrm{H}(\boldsymbol{v}) = d$. To simplify notations, we assume without essential loss of generality that the first d symbols of the codeword $\boldsymbol{v}$ are nonzero.

What is the probability, p_d, that the Viterbi decoder selects $\boldsymbol{v}$ in favor of the transmitted allzero codeword?

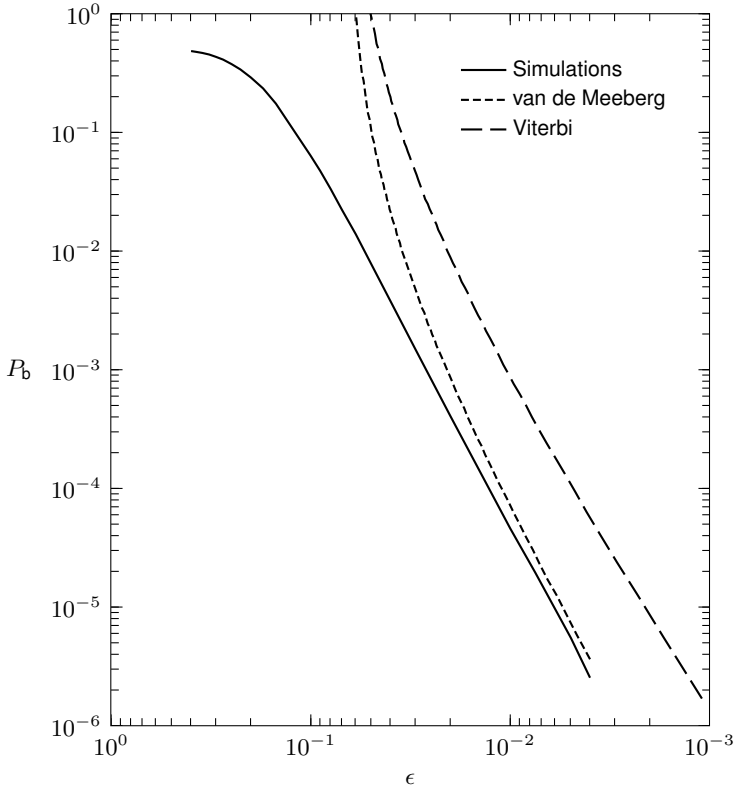


Figure 4.12 Viterbi's and van de Meeberg's upper bounds on the bit error probability for the minimal encoding matrix $G(D) = (1 + D + D^2 \quad 1 + D^2)$ realized in controller canonical form and the BSC with crossover probability ϵ.

The probability for this event equals the probability that the ratio of the conditional density functions is greater than 1, that is,

$$p_d = P\left(\frac{p\left(\boldsymbol{r} \mid \boldsymbol{v}\right)}{p\left(\boldsymbol{r} \mid \boldsymbol{0}\right)} > 1\right) = P\left(\prod_t \frac{p\left(r_t \mid v_t\right)}{p\left(r_t \mid 0\right)} > 1\right) \tag{4.58}$$

where

$$p\left(r_t \mid 0\right) = \frac{1}{\sqrt{\pi N_0}} e^{-\frac{\left(r_t + \sqrt{E_s}\right)^2}{N_0}} \tag{4.59}$$

and

$$p\left(r_t \mid 1\right) = \frac{1}{\sqrt{\pi N_0}} e^{-\frac{\left(r_t - \sqrt{E_s}\right)^2}{N_0}} \tag{4.60}$$

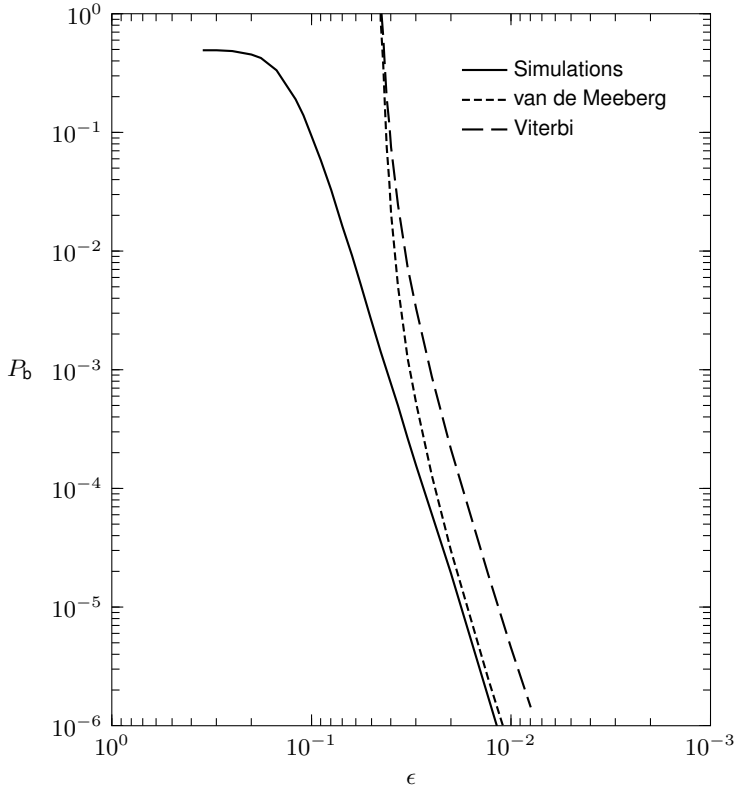


Figure 4.13 Viterbi's and van de Meeberg's upper bounds on the bit error probability for Qualcomm's $R = 1/2$, $m = 6$ minimal encoding matrix realized in controller canonical form and the BSC with crossover probability ϵ.

From (4.58) it follows that

$$\begin{aligned} p_d &= P\left(\prod_{j=1}^{d} \frac{e^{-\frac{\left(r_j-\sqrt{E_s}\right)^2}{N_0}}}{e^{-\frac{\left(r_j+\sqrt{E_s}\right)^2}{N_0}}} > 1\right) \\ &= P\left(\frac{1}{N_0}\sum_{j=1}^{d}\left(-\left(r_j-\sqrt{E_s}\right)^2+\left(r_j+\sqrt{E_s}\right)^2\right) > 0\right) \\ &= P\left(\sum_{j=1}^{d} r_j > 0\right) \end{aligned} \tag{4.61}$$

Since the channel noise values at different sample times are uncorrelated, it follows that $\sum_{j=1}^{d} r_j$ is the sum of d independent Gaussian random variables and, hence, that

$\sum_{j=1}^{d} r_j \in N(-d\sqrt{E_s}, dN_0/2)$. Thus, we have

$$\begin{aligned} p_d &= \frac{1}{\sqrt{\pi dN_0}} \int_0^\infty e^{-\frac{\left(r+d\sqrt{E_s}\right)^2}{dN_0}} dr \\ &= \frac{1}{\sqrt{2\pi}} \int_{\sqrt{2dE_s/N_o}}^\infty e^{-y^2/2} dy = Q\left(\sqrt{2dE_s/N_0}\right) \\ &= Q\left(\sqrt{2dRE_b/N_0}\right) \end{aligned} \tag{4.62}$$

where $Q(\ )$ is the complementary error function (1.13).

From (4.25) it follows that the burst error probability when communicating over the AWGN channel and exploiting soft decisions is upper-bounded by

$$P_B \leq \sum_{d=d_{free}}^{\infty} n_d p_d \tag{4.63}$$

where n_d is the number of weight d paths and p_d is given by (4.62).

Lemma 4.6 For $x > 0$,

$$Q(x) = \frac{1}{\sqrt{2\pi}} \int_x^\infty e^{-y^2/2} dy < e^{-x^2/2} \tag{4.64}$$

Proof: For any $\lambda > 0$, we have

$$\begin{aligned} Q(x) &< \frac{1}{\sqrt{2\pi}} \int_x^\infty e^{-\frac{y^2}{2}+\lambda(y-x)} dy \\ &< e^{-\lambda x+\frac{\lambda^2}{2}} \frac{1}{\sqrt{2\pi}} \int_{-\infty}^\infty e^{-\frac{(y-\lambda)^2}{2}} dy = e^{-\lambda x+\frac{\lambda^2}{2}} \end{aligned} \tag{4.65}$$

Choosing $\lambda = x$ completes the proof. ∎

By combining (4.62) and Lemma 4.6, we obtain the following:

Theorem 4.7 (Viterbi) When using a rate R convolutional code with generator matrix $G(D)$ for communication over the AWGN channel with BPSK modulation at signal-to-noise ratio E_b/N_0 and maximum-likelihood decoding, the *burst error probability* is upper-bounded by

$$P_B < \sum_{d=d_{free}}^{\infty} n_d e^{-dRE_b/N_0} = T(W)\,|_{W=e^{-RE_b/N_0}} \tag{4.66}$$

where the path weight enumerator $T(W)$ is obtained for a minimal encoder.

Viterbi's bound (4.66) can be tightened [ViO79] by using the following inequality:

Lemma 4.8 For $x > 0$, $z \geq 0$,

$$Q\left(\sqrt{x+z}\right) \leq Q\left(\sqrt{x}\right) e^{-z/2} \tag{4.67}$$

with equality if and only if $z = 0$.

Proof:

$$\begin{aligned} & Q\left(\sqrt{x+z}\right) - Q\left(\sqrt{x}\right) e^{-z/2} \\ &= \frac{1}{\sqrt{2\pi}} \int_{\sqrt{x+z}}^{\infty} e^{-y^2/2} dy - e^{-z/2} \frac{1}{\sqrt{2\pi}} \int_{\sqrt{x}}^{\infty} e^{-y^2/2} dy \\ &= \frac{1}{\sqrt{2\pi}} \int_{0}^{\infty} \left(e^{-\left(y+\sqrt{x+z}\right)^2/2} - e^{-z/2} e^{-\left(y+\sqrt{x}\right)^2/2} \right) dy \\ &= \frac{1}{\sqrt{2\pi}} \int_{0}^{\infty} e^{-\frac{y^2}{2} - \frac{x+z}{2}} \left(e^{-y\sqrt{x+z}} - e^{-y\sqrt{x}} \right) dy \leq 0 \end{aligned} \tag{4.68}$$

where the inequality follows from the fact that

$$e^{-y\sqrt{x+z}} \leq e^{-y\sqrt{x}} \tag{4.69}$$

We have equality in (4.68) and (4.69) if and only if $z = 0$ and the proof is complete. ■

Theorem 4.9 When using a rate R convolutional code with generator matrix $G(D)$ for communication over the AWGN channel with BPSK modulation at signal-to-noise ratio E_b/N_0 and maximum-likelihood decoding, the *burst error probability* is upper-bounded by

$$\begin{aligned} P_\mathrm{B} &< Q\left(\sqrt{2d_\mathrm{free} R E_\mathrm{b}/N_0}\right) e^{d_\mathrm{free} R E_\mathrm{b}/N_0} \sum_{d=d_\mathrm{free}}^{\infty} n_d e^{-dRE_\mathrm{b}/N_0} \\ &= Q\left(\sqrt{2d_\mathrm{free} R E_\mathrm{b}/N_0}\right) e^{d_\mathrm{free} R E_\mathrm{b}/N_0} T(W)\,|_{W=e^{-RE_\mathrm{b}/N_0}} \end{aligned} \tag{4.70}$$

where the path weight enumerator $T(W)$ is obtained for a minimal encoder.

Proof: Let $x = d_\mathrm{free}$ and $z = d - d_\mathrm{free}$. Then by combining (4.62) and (4.67) we obtain

$$p_d < Q\left(\sqrt{2d_\mathrm{free} R E_\mathrm{b}/N_0}\right) e^{-(d-d_\mathrm{free})RE_\mathrm{b}/N_0} \tag{4.71}$$

Inserting (4.71) into (4.63) completes the proof. ■

In order to upper-bound the bit error probability, we combine the union bound (4.62) and Lemma 4.6. Thus, we have

$$p(i) < \sum_{w=d_\mathrm{free}}^{\infty} \sum_{\ell=\nu_\mathrm{min}+1}^{\infty} n(w,\ell,i) \left(e^{-RE_\mathrm{b}/N_0}\right)^w \tag{4.72}$$

From (4.49) and (4.72) the next theorem follows.

Theorem 4.10 (Viterbi) When using a rate R convolutional code with generator matrix $G(D)$ for communication over the AWGN channel with BPSK modulation at

signal-to-noise ratio E_b/N_0 and maximum-likelihood decoding, the *bit error probability* is upper-bounded by

$$\begin{aligned} P_\mathrm{b} &< \frac{1}{b} \sum_{w=d_\mathrm{free}}^{\infty} \sum_{\ell=\nu_\mathrm{min}+1}^{\infty} \sum_{i=1}^{\infty} i n(w,\ell,i) e^{-wRE_\mathrm{b}/N_0} \\ &= \frac{1}{b} \frac{\partial T(W,L,I)}{\partial I} \Bigg|_{\substack{W=e^{-RE_\mathrm{b}/N_0} \\ L=1 \\ I=1}} \end{aligned} \tag{4.73}$$

where the extended path enumerator $T(W,L,I)$ is obtained for a minimal encoder.

As a counterpart to Theorem 4.9 we have the following [ViO79]:

Theorem 4.11 When using a rate R convolutional code with generator matrix $G(D)$ for communication over the AWGN channel with BPSK modulation at signal-to-noise ratio E_b/N_0 and maximum-likelihood decoding, the *bit error probability* is upper-bounded by

$$\begin{aligned} P_\mathrm{b} &< \frac{1}{b} Q\left(\sqrt{2d_\mathrm{free}RE_\mathrm{b}/N_0}\right) e^{d_\mathrm{free}RE_\mathrm{b}/N_0} \\ &\quad \times \left(\sum_{w=d_\mathrm{free}}^{\infty} \sum_{\ell=\nu_\mathrm{min}+1}^{\infty} \sum_{i=1}^{\infty} i n(w,\ell,i) e^{-wRE_\mathrm{b}/N_0} \right) \\ &= \frac{1}{b} Q\left(\sqrt{2d_\mathrm{free}RE_\mathrm{b}/N_0}\right) e^{d_\mathrm{free}RE_\mathrm{b}/N_0} \frac{\partial T(W,L,I)}{\partial I} \Bigg|_{\substack{W=e^{-RE_\mathrm{b}/N_0} \\ L=1 \\ I=1}} \end{aligned} \tag{4.74}$$

where the extended path enumerator $T(W,L,I)$ is obtained for a minimal encoder.

Proof: Follows immediately from (4.49), (4.67), and (4.72). ∎

It is interesting to compare the bit error probability bounds for hard decisions (BSC) and soft decisions (AWGN channel). For high signal-to-noise ratios, the terms at distance d_free dominate the bounds. Thus, the dominating term in (4.52) is

$$\frac{1}{b}\left(2\sqrt{\epsilon(1-\epsilon)}\right)^{d_\mathrm{free}} \approx \frac{1}{b} 2^{d_\mathrm{free}} \epsilon^{d_\mathrm{free}/2} \tag{4.75}$$

From (1.12) and (4.57) it follows that

$$\epsilon = Q\left(\sqrt{2RE_\mathrm{b}/N_0}\right) \tag{4.76}$$

Hence, we have

$$\begin{aligned} \frac{1}{b} 2^{d_\mathrm{free}} \epsilon^{d_\mathrm{free}/2} &= \frac{1}{b} 2^{d_\mathrm{free}} \left(Q\left(\sqrt{2RE_\mathrm{b}/N_0}\right)\right)^{d_\mathrm{free}/2} \\ &\le \frac{1}{b} 2^{d_\mathrm{free}} e^{-\frac{1}{2} d_\mathrm{free} RE_\mathrm{b}/N_0} \end{aligned} \tag{4.77}$$

where the inequality follows from Lemma 4.6. Thus, for the BSC we have asymptotically

$$P_{\mathrm{b}} \lesssim \frac{1}{b} 2^{d_{\mathrm{free}}} e^{-\frac{1}{2} d_{\mathrm{free}} R E_{\mathrm{b}}/N_0} \left(\sum_{\ell=\nu_{\min}+1}^{\infty} \sum_{i=1}^{\infty} i n(d_{\mathrm{free}}, \ell, i) \right) \tag{4.78}$$

For the AWGN channel we have asymptotically

$$\begin{aligned} P_{\mathrm{b}} &\lesssim \frac{1}{b} Q\left(\sqrt{2 d_{\mathrm{free}} R E_{\mathrm{b}}/N_0}\right) \left(\sum_{\ell=\nu_{\min}+1}^{\infty} \sum_{i=1}^{\infty} i n(d_{\mathrm{free}}, \ell, i) \right) \\ &\leq \frac{1}{b} e^{-d_{\mathrm{free}} R E_{\mathrm{b}}/N_0} \left(\sum_{\ell=\nu_{\min}+1}^{\infty} \sum_{i=1}^{\infty} i n(d_{\mathrm{free}}, \ell, i) \right) \end{aligned} \tag{4.79}$$

By comparing (4.78) with (4.79) we see that the exponent of (4.79) is larger by a factor of 2. Thus, we have a 3 dB energy advantage for the AWGN channel over the BSC.

In the Viterbi *bounds* we pick up asymptotically (high signal-to-noise ratios) a 3 dB gain by using soft decisions instead of hard decisions. In Chapter 1 (Fig. 1.7) we used Shannon's channel capacity theorem (low signal-to-noise ratios) to show that soft decisions are about 2 dB more efficient than hard decisions.

For high signal-to-noise ratios the bit error probability is determined by the exponent $d_{\mathrm{free}} R E_{\mathrm{b}}/N_0$ in (4.79). In the uncoded case we have $d_{\mathrm{free}} = R = 1$. Hence, we define the asymptotical *coding gain* to be

$$\gamma = 10 \log_{10}(d_{\mathrm{free}} R) \ \text{ dB} \tag{4.80}$$

The asymptotical coding gain is a rough estimate of the performance of a coded system representing the potential increase due to coding.

For a rate $R = 1/2$, memory $m = 2$ convolutional code with $d_{\mathrm{free}} = 5$, we have $\gamma = 10 \log_{10}(5 \cdot \frac{1}{2}) = 4$ dB. The true coding gain at $P_{\mathrm{b}} = 10^{-5}$ is approximately 3.5 dB [Wil96]. The difference at this bit error probability level is due to the nonneglible contribution of the other terms in the extended path enumerator.

Less powerful codes approach their asymptotic coding gain much faster than more powerful codes. For example, the rate $R = 1/2$, memory $m = 6$ convolutional code with $d_{\mathrm{free}} = 10$ has an asymptotic coding gain $\gamma = 10 \log_{10}(10 \cdot \frac{1}{2}) = 7$ dB, but at $P_{\mathrm{b}} = 10^{-5}$ the coding gain is only 5 dB [Wil96].

4.3 TIGHTER ERROR BOUNDS FOR TIME-INVARIANT CONVOLUTIONAL CODES

Most of the bounds we derived in the previous section are quite tight when few channel symbols are in error, that is, for high signal-to-noise ratios. However, they become trivial (> 1) when we have many errors among the channel symbols (that is, for low signal-to-noise ratios). In this section, we will derive an essentially tighter bound for the burst error probability for the BSC [CJZ84a]. Since the bound will not

depend on the starting point for the error burst, we will without loss of generality consider bursts starting at the root.

Let us separate the error event into two disjoint events corresponding to "few" ($\mathcal{F}$) and "many" ($\mathcal{M}$) errors, respectively. If we let $\mathcal{E}$ denote the burst error event (that is, the event that the first information symbol is erroneously decoded by maximum-likelihood decoding on the BSC), we have

$$\begin{aligned} P_{\mathrm{B}} = P(\mathcal{E}) &= P(\mathcal{E} \mid \mathcal{F})\, P(\mathcal{F}) + P(\mathcal{E} \mid \mathcal{M})\, P(\mathcal{M}) \\ &\leq P(\mathcal{E} \mid \mathcal{F})\, P(\mathcal{F}) + P(\mathcal{M}) \\ &= P(\mathcal{E}, \mathcal{F}) + P(\mathcal{M}) \end{aligned} \tag{4.81}$$

To obtain an upper bound on the probability that we have many channel errors, $P(\mathcal{M})$, we use a random-walk argument that is similar to what we used in Section 3.4 when we derived a lower bound on the distance profile.

Let us accrue a metric α when the channel symbol is correctly received and a metric β when we have a channel error. Then the cumulative metric along the correct path is a random walk $0, S_0, S_1, S_2, \ldots$, where

$$S_t = \sum_{i=0}^{t} Z_i \tag{4.82}$$

for $t \geq 0$. The *branch metrics* Z_i, $i = 0, 1, 2, \ldots$, are independent, identically distributed random variables and can be written

$$Z_i = \sum_{\ell=1}^{c} Y_{i\ell} \tag{4.83}$$

where the $Y_{i\ell}$'s are independent, identically distributed random variables,

$$Y_{i\ell} = \begin{cases} \alpha > 0 & \text{with probability } 1 - \epsilon \\ \beta < 0 & \text{with probability } \epsilon \end{cases} \tag{4.84}$$

where ϵ is the crossover probability of the BSC. Clearly (cf. Example B.1),

$$P(Z_i = k\alpha + (c-k)\beta) = \binom{c}{k} (1-\epsilon)^k \epsilon^{c-k} \tag{4.85}$$

Let us choose the metrics

$$\alpha = \log \frac{1-\epsilon}{1-a} \tag{4.86}$$

and

$$\beta = \log \frac{\epsilon}{a} \tag{4.87}$$

where $\epsilon < a < 1$ is a parameter to be chosen later.

Suppose we have w_t errors among the first $(t+1)c$ channel symbols. Then,

$$S_t = w_t \beta + ((t+1)c - w_t)\alpha \tag{4.88}$$

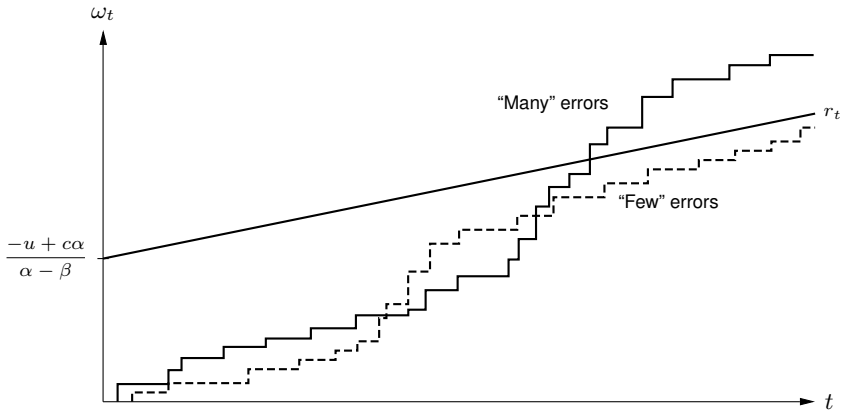


Figure 4.14 Barrier at r_t which separates paths with many and few errors.

Now we can more precisely state what we mean by "few" and "many" errors. Those error patterns for which S_t stays above the barrier at $u < 0$ contain few errors, and those error patterns for which the cumulative metric hits or crosses the barrier contain many errors. Few errors, that is,

$$S_{\min} = \min_t \{S_t\} > u \tag{4.89}$$

is equivalent to

$$w_t\beta + ((t+1)c - w_t)\alpha > u, \;\; t \geq 0 \tag{4.90}$$

or

$$w_t < \frac{-u}{\alpha-\beta} + (t+1)c\frac{\alpha}{\alpha-\beta} \stackrel{\text{def}}{=} r_t, \;\; t \geq 0 \tag{4.91}$$

In Fig. 4.14 we illustrate inequality (4.91).

From Wald's identity (Corollary B.6) we have

$$P(\mathcal{M}) = P(S_{\min} \leq u) \leq 2^{-\lambda_0 u} \tag{4.92}$$

where u is a parameter to be chosen later and $\lambda_0 < 0$ is the root of the following equation (see Example B.1):

$$g(\lambda) \stackrel{\text{def}}{=} \left((1-\epsilon)2^{\lambda\alpha} + \epsilon 2^{\lambda\beta}\right)^c = 1 \tag{4.93}$$

that is,

$$\lambda_0 = -1 \tag{4.94}$$

To upper-bound the probability that a burst starts at the root and that we have few channel symbol errors, $P(\mathcal{E}, \mathcal{F})$, we use the union bound and obtain

$$P(\mathcal{E}, \mathcal{F}) \leq \sum_k P(\mathcal{E}^{(k)}, \mathcal{F}) \tag{4.95}$$

where $\mathcal{E}^{(k)}$ is the event that a burst error starting at the root is caused by path k.

A path has few channel errors only if it stays below the barrier in Fig. 4.14 for *all* $t \geq 0$. If we take all paths of length $t+1$ with $w_t < r_t$ channel errors we will get all paths with few channel errors together with the paths with many channel errors that take one or more detours above the barrier. Hence, for path k of length $j+1$ we have

$$P(\mathcal{E}^{(k)}, \mathcal{F}) \leq P(\mathcal{E}^{(k)}, w_j < r_j) \tag{4.96}$$

where

$$P(\mathcal{E}^{(k)}, w_j < r_j) = \sum_{w_j < r_j} \binom{(j+1)c}{w_j} (1-\epsilon)^{(j+1)c-w_j} \epsilon^{w_j} P\left(\mathcal{E}^{(k)} \;\middle|\; w_j\right) \tag{4.97}$$

If we multiply the terms in (4.97) by $\eta^{-(r_j - w_j)}$, $0 < \eta \leq 1$, we obtain the inequality

$$\begin{aligned} &P(\mathcal{E}^{(k)}, w_j < r_j) \\ &\quad \leq \eta^{-r_j} \sum_{w_j < r_j} \binom{(j+1)c}{w_j} (\eta\epsilon)^{w_j} (1-\epsilon)^{(j+1)c-w_j} P\left(\mathcal{E}^{(k)} \;\middle|\; w_j\right) \end{aligned} \tag{4.98}$$

Let us introduce

$$\epsilon_0 \stackrel{\text{def}}{=} \frac{\eta\epsilon}{\eta\epsilon + 1 - \epsilon} \leq \epsilon \tag{4.99}$$

and

$$1 - \epsilon_0 \stackrel{\text{def}}{=} \frac{1-\epsilon}{\eta\epsilon + 1 - \epsilon} \geq 1 - \epsilon \tag{4.100}$$

Substituting (4.99) and (4.100) into (4.98) and rearranging (4.98), we get

$$\begin{aligned} P(\mathcal{E}^{(k)}, w_j < r_j) \leq &\left(\frac{\epsilon(1-\epsilon_0)}{\epsilon_0(1-\epsilon)}\right)^{r_j} \left(\frac{1-\epsilon}{1-\epsilon_0}\right)^{(j+1)c} \\ &\times \sum_{w_j < r_j} \binom{(j+1)c}{w_j} \epsilon_0^{w_j} (1-\epsilon_0)^{(j+1)c-w_j} P\left(\mathcal{E}^{(k)} \;\middle|\; w_j\right) \end{aligned} \tag{4.101}$$

Overbounding by summing over all $0 \leq w_j \leq (j+1)c$, we have

$$P(\mathcal{E}^{(k)}, w_j < r_j) \leq \left(\frac{\epsilon(1-\epsilon_0)}{\epsilon_0(1-\epsilon)}\right)^{r_j} \left(\frac{1-\epsilon}{1-\epsilon_0}\right)^{(j+1)c} P(\mathcal{E}^{(k)})_{d,j+1,\epsilon_0} \tag{4.102}$$

where $P(\mathcal{E}^{(k)})_{d,j+1,\epsilon_0}$ is the probability that a decoding error is caused by the kth path of weight d and length $j+1$ on an improved BSC with crossover probability ϵ_0.

Using the Bhattacharyya bound (4.20), we obtain from (4.102)

$$P(\mathcal{E}^{(k)}, w_j < r_j) < \left(\frac{\epsilon(1-\epsilon_0)}{\epsilon_0(1-\epsilon)}\right)^{r_j} \left(\frac{1-\epsilon}{1-\epsilon_0}\right)^{(j+1)c} \left(2\sqrt{\epsilon_0(1-\epsilon_0)}\right)^d \tag{4.103}$$

Using the definition of r_j given in (4.91) and rearranging (4.103), we obtain

$$P(\mathcal{E}^{(k)}, w_j < r_j) < \left(\frac{\epsilon(1-\epsilon_0)}{\epsilon_0(1-\epsilon)}\right)^{\frac{-u}{\alpha-\beta}} W^d L^{j+1} \tag{4.104}$$

where

$$W = 2\sqrt{\epsilon_0(1-\epsilon_0)} \tag{4.105}$$

and

$$L = \left(\left(\frac{1-\epsilon}{1-\epsilon_0} \right) \left(\frac{\epsilon(1-\epsilon_0)}{\epsilon_0(1-\epsilon)} \right)^{\frac{\alpha}{\alpha-\beta}} \right)^c \tag{4.106}$$

Finally, we combine (4.95) and (4.96) with (4.104) and obtain the following upper bound on the probability of having few channel symbol errors and making an error when decoding the first information symbol:

$$\begin{aligned} P(\mathcal{E}, \mathcal{F}) &< \left(\frac{\epsilon(1-\epsilon_0)}{\epsilon_0(1-\epsilon)} \right)^{\frac{-u}{\alpha-\beta}} \sum_d \sum_j n(d, j+1) W^d L^{j+1} \\ &= \left(\frac{\epsilon(1-\epsilon_0)}{\epsilon_0(1-\epsilon)} \right)^{\frac{-u}{\alpha-\beta}} T(W, L) \end{aligned} \tag{4.107}$$

where $n(d, j+1)$ is the number of paths of weight d and length $j+1$ stemming from the root and

$$T(W, L) \stackrel{\text{def}}{=} T(W, L, I)|_{I=1} \tag{4.108}$$

where $T(W, L, I)$ is the extended path enumerator.

We now combine the bounds (4.81), (4.92), and (4.107) to obtain the following upper bound on the burst error probability:

$$P_{\text{B}} = P(\mathcal{E}) < \left(\frac{\epsilon(1-\epsilon_0)}{\epsilon_0(1-\epsilon)} \right)^{\frac{-u}{\alpha-\beta}} T(W, L) + 2^u \tag{4.109}$$

This bound is valid for all u. By taking the derivative of the right side of (4.109), we find that its minimum is obtained for

$$u_0 = -\frac{(\alpha-\beta)\left(\log(\alpha-\beta) - \log T(W, L) - \log\log \frac{\epsilon(1-\epsilon_0)}{\epsilon_0(1-\epsilon)} \right)}{\log \frac{\epsilon(1-\epsilon_0)}{\epsilon_0(1-\epsilon)} + \alpha - \beta} \tag{4.110}$$

Inserting (4.110) and rearranging (4.109) give the upper bound

$$P(\mathcal{E}) \leq 2^{h(\gamma)} T(W, L)^{\gamma} \tag{4.111}$$

where $h(\cdot)$ is the binary entropy function (1.22) and

$$\gamma^{-1} = 1 + \frac{\log \frac{\epsilon(1-\epsilon_0)}{\epsilon_0(1-\epsilon)}}{\alpha - \beta} \tag{4.112}$$

Finally, we use (4.86) and (4.87) and obtain a Viterbi-type bound:

Theorem 4.12 When using a convolutional code for communication over the BSC with crossover probability ϵ and maximum-likelihood decoding, the *burst error probability* is upper-bounded by

$$P_{\text{B}} \leq \inf_{0<\epsilon_0\leq\epsilon} \inf_{\epsilon<a<1} \left\{ 2^{h(\gamma)} T(W, L)^{\gamma} \right\} \tag{4.113}$$

where

$$\gamma^{-1} = 1 + \frac{\log \frac{\epsilon(1-\epsilon_0)}{\epsilon_0(1-\epsilon)}}{\log \frac{(1-\epsilon)a}{\epsilon(1-a)}} \tag{4.114}$$

the extended path enumerator $T(W)$ is obtained for a minimal encoder, and W and L are given by (4.105) and (4.106), respectively.

The bound (4.113) is significantly better than Viterbi's bound (4.27) for low signal-to-noise ratios. The latter bound can be obtained by choosing $\epsilon_0 = \epsilon$ rather than minimizing over ϵ_0 in (4.113).

Van de Meeberg's strengthening of Viterbi's bound (4.33) is also valid here, and we obtain the following:

Theorem 4.13 When using a convolutional code for communication over the BSC with crossover probability ϵ and maximum-likelihood, the *burst error probability* is upper-bounded by

$$P_{\mathrm{B}} < \inf_{0<\epsilon_0<\epsilon} \inf_{\epsilon<a<1} \left\{ 2^{h(\gamma)} \times \left(\frac{1+W}{2} T(W,L) + \frac{1-W}{2} T(-W,L) \right)^{\gamma} \right\} \tag{4.115}$$

where the extended path enumerator $T(W, L)$ is obtained for a minimal encoder and γ, W, and L are given by (4.114), (4.105), and (4.106), respectively.

In Fig. 4.15 we compare our tightened van de Meeberg-type bound (4.115) with van de Meeberg's bound (4.33) and simulations for Qualcomm's $R = 1/2, m = 6$ encoder.

Remark: In the next chapter we introduce a parameter for the BSC called the computational cutoff rate $R_0 = 1 - \log(1 + 2\sqrt{\epsilon(1-\epsilon)})$. For a BSC with crossover probability $\epsilon = 0.045$, we have $R_0 = 1/2$. We notice that in Fig. 4.15 van de Meeberg's upper bound is nontrivial only for rates $0 \leq R < R_0$ while our tightened bound is nontrivial for rates $0 \leq R < C$, where the channel capacity for the BSC is $C = 1 - h(\epsilon)$, that is, $\epsilon = 0.11$ corresponds to $C = 1/2$.

4.4 EXACT BIT ERROR PROBABILITY FOR VITERBI DECODING

In this section, we shall calculate the exact bit error probability for Viterbi decoding of minimal convolutional encoders when used to communicate over the BSC. For simplicity we consider only the rate $R = 1/2$ minimal feed-forward encoder with memory $m = 1$ and encoding matrix $G(D) = (1 \quad 1 + D)$ in detail. The generalizations to other memories and rates as well as realizations with feedback and the quantized AWGN channel shall be discussed briefly.

Assume that the allzero sequence is transmitted over the BSC. Let $W_t(\sigma)$ denote the weight of the information sequence corresponding to the code sequence decided

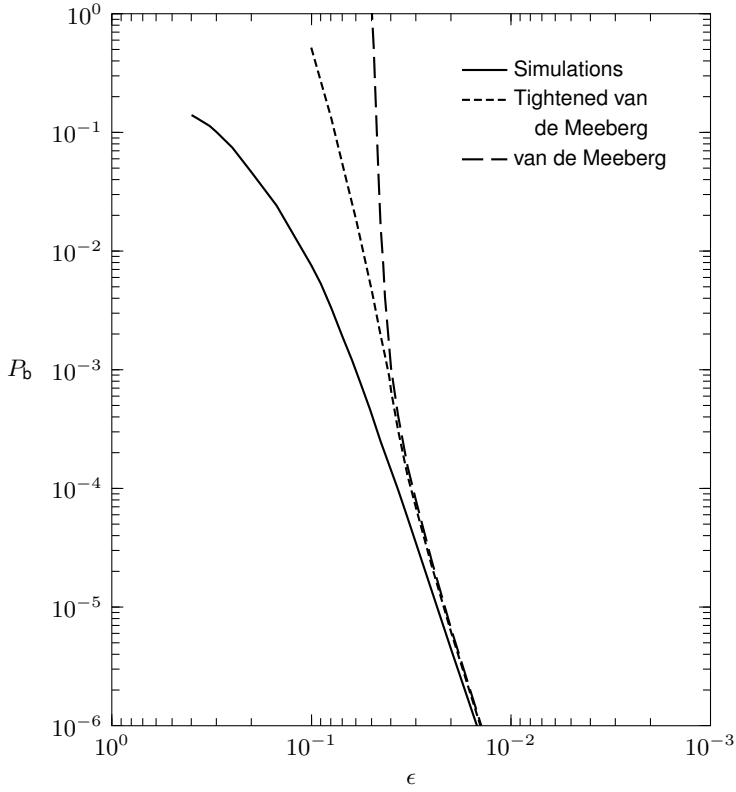


Figure 4.15 Tightened van de Meeberg-type bound compared with van de Meeberg's bound on the burst error probability for Qualcomm's $R = 1/2$, $m = 6$ minimal encoder and the BSC with crossover probability ϵ.

by the Viterbi decoder at state σ at time t. If the initial values $W_0(\sigma)$ are known, then the random process $W_t(\sigma)$ is a function of the random process of the received tuples $\boldsymbol{r}_i, i = 0, 1, \ldots, t-1$. Thus, the ensemble $\{\boldsymbol{r}_i, i = 0, 1, \ldots, t-1\}$ determines the ensemble $\{W_i(\sigma), i = 1, 2, \ldots, t\}$.

Our aim is to determine the average of the random variable $W_t(\sigma)$ over this ensemble, since for minimal rate $R = b/c$ convolutional encoders the bit error probability can be computed as the limit

$$P_b = \lim_{t \to \infty} \frac{E[W_t(\sigma = 0)]}{tb} \tag{4.116}$$

assuming that this limit exists.

When we consider realizations in controller canonical form, the states can be represented by the m-tuples of the inputs of the b shift registers, that is, $\boldsymbol{\sigma}_t = \boldsymbol{u}_{t-1}\, \boldsymbol{u}_{t-2} \ldots \boldsymbol{u}_{t-m}$. In the sequel we usually denote these encoder states σ, $\sigma \in \{0, 1, \ldots, 2^{bm} - 1\}$. During the decoding step at time $t + 1$ the Viterbi decoder

computes the cumulative Viterbi branch metric tuple

$$\boldsymbol{\mu}_{t+1} = (\mu_{t+1}(0)\ \mu_{t+1}(1)\ \ldots \mu_{t+1}(2^{bm}-1)) \tag{4.117}$$

at time $t+1$ using the tuple $\boldsymbol{\mu}_t$ at time t and the received c-tuple $\boldsymbol{r}_t$. In order to simplify our analysis we normalize the metrics such that the cumulative metrics at every zero state will be zero, that is, we subtract the value $\mu_t(0)$ from $\mu_t(1), \mu_t(2), \ldots, \mu_t(2^{bm}-1)$ and obtain the cumulative normalized branch metric tuple

$$\begin{aligned}\boldsymbol{\phi}_t &= \big(\mu_t(1)-\mu_t(0)\,\mu_t(2)-\mu_t(0)\ \ldots \mu_t(2^{bm}-1)-\mu_t(0)\big)\\ &= \big(\phi_t(1)\,\phi_t(2)\ \ldots \phi_t(2^{bm}-1)\big)\end{aligned} \tag{4.118}$$

For rate $R = 1/2$, memory $m = 1$ (2-state) encoders the cumulative normalized branch metric will be a scalar

$$\boldsymbol{\phi}_t = \phi_t(1) = \mu_t(1) - \mu_t(0) \tag{4.119}$$

First we consider the rate $R = 1/2$, memory $m = 1$, minimal (2-state) encoder with generator matrix $G(D) = (1 \quad 1+D)$ and $d_{\text{free}} = 3$. In Fig. 4.16 we show the 20 different trellis sections corresponding to the $M = 5$ different normalized cumulative metrics $\phi_t = -2, -1, 0, 1, 2$ and the four different received tuples $\boldsymbol{r}_t = 00, 01, 10, 11$. In order to find the $M = 5$ different metrics ϕ_t we start with $\phi_t = 0$. Then we obtain (the third row in Fig. 4.16) that ϕ_{t+1} is -1, -1, 1, 1 for the four different received tuples, respectively. Letting ϕ_t be -1 and 1, we obtain additionally that ϕ_{t+1} can be -2 and 2. Letting ϕ_t be -2 and 2, it is easily verified that we already found all $M = 5$ values of the cumulative normalized branch metrics.

The bold branches in Fig. 4.16 correspond to the branches decided by the Viterbi decoder at time $t + 1$. When we have two branches entering the same state with the same metric, we have a tie that we, in our analysis, resolve by coin flipping.

The sequence of cumulative normalized metrics ϕ_t forms a 5-state Markov chain with transition probability matrix $\Phi = (\phi_{jk})$, where

$$\phi_{jk} = P(\phi_{t+1} = \phi^{(k)} | \phi_t = \phi^{(j)}) \tag{4.120}$$

From the trellis sections in Fig. 4.16 we obtain the following transition probability matrix:

$$\Phi = \begin{array}{c} \\ -2 \\ -1 \\ 0 \\ 1 \\ 2 \\ \phi^{(j)} \end{array} \begin{array}{c} \begin{array}{ccccc} -2 & -1 & 0 & 1 & 2 \end{array} \\ \begin{pmatrix} (1-\epsilon)^2 & 0 & 2\epsilon(1-\epsilon) & 0 & \epsilon^2 \\ (1-\epsilon)^2 & 0 & 2\epsilon(1-\epsilon) & 0 & \epsilon^2 \\ 0 & 1-\epsilon & 0 & \epsilon & 0 \\ \epsilon(1-\epsilon) & 0 & \epsilon^2+(1-\epsilon)^2 & 0 & \epsilon(1-\epsilon) \\ \epsilon(1-\epsilon) & 0 & \epsilon^2+(1-\epsilon)^2 & 0 & \epsilon(1-\epsilon) \end{pmatrix} \\ \phi^{(k)} \end{array} \tag{4.121}$$

The state transition diagram for the Markov chain is shown in Fig. 4.17.

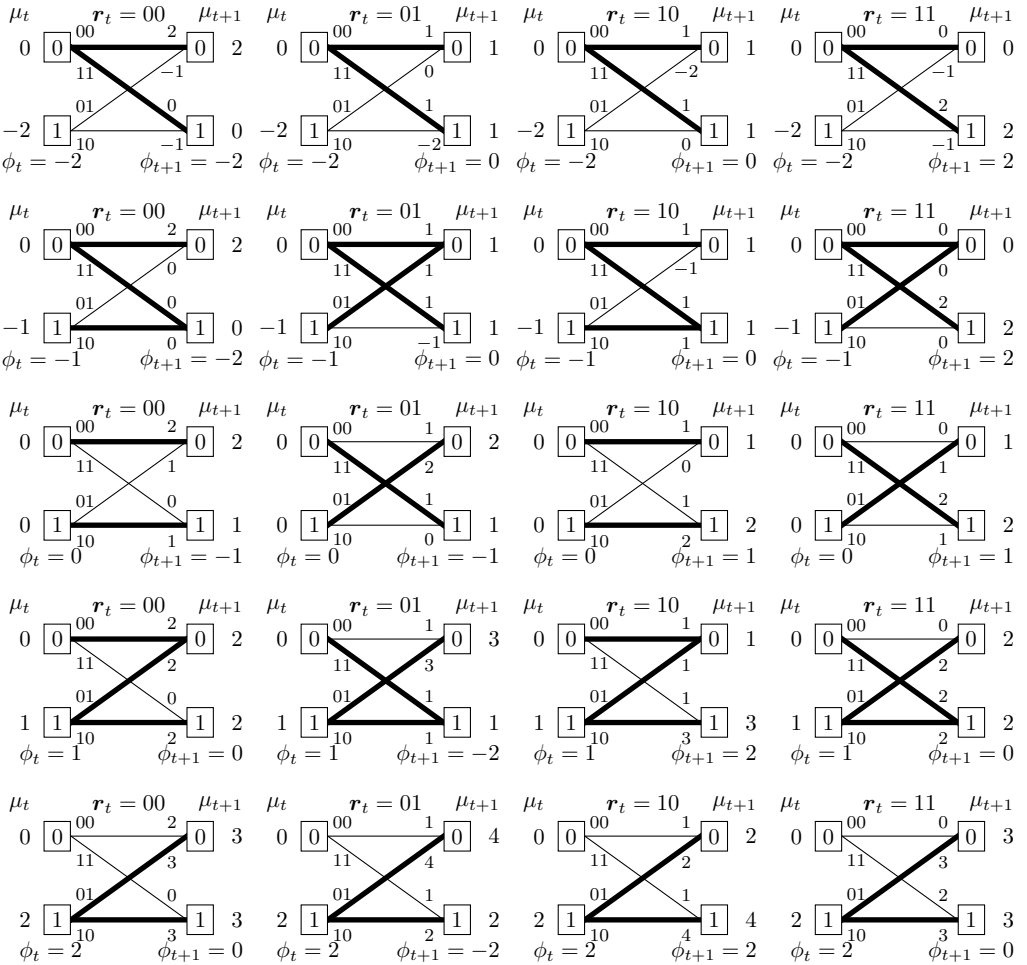


Figure 4.16 All 20 different trellis sections for the $G(D) = (1 \quad 1 + D)$ generator matrix.

Let $\boldsymbol{p}_\infty = (p_\infty^{(1)}\, p_\infty^{(2)}\, \ldots\, p_\infty^{(5)})$ denote the stationary distribution of the cumulative normalized metrics ϕ_t. It is determined as the solution of, for example, the first $M - 1 = 4$ equations of

$$\boldsymbol{p}_\infty \Phi = \boldsymbol{p}_\infty \tag{4.122}$$

and

$$\sum_{i=1}^{M} p_\infty^{(i)} = 1 \tag{4.123}$$

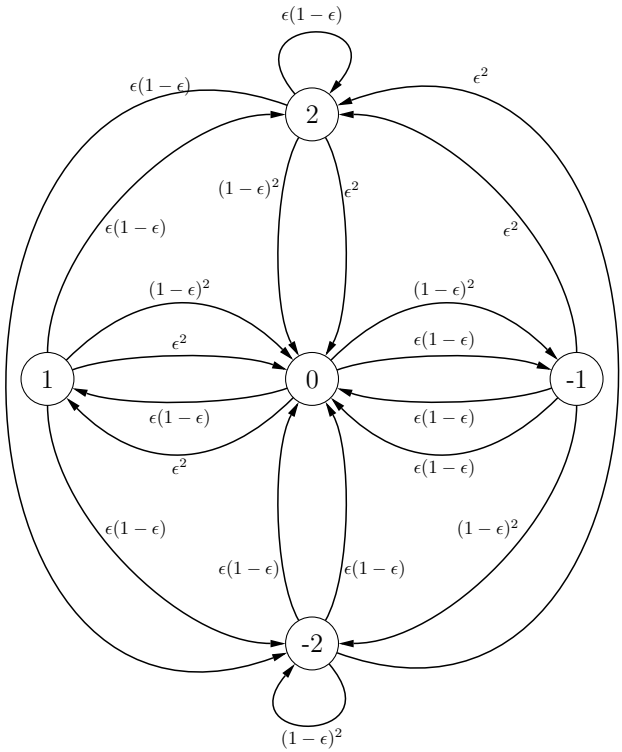


Figure 4.17 Illustration of the 5-state Markov chain formed by the sequences of cumulative normalized metrics ϕ_t.

For our memory $m = 1$ encoder we obtain

$$\boldsymbol{p}_\infty^{\mathrm{T}} = \frac{1}{1+3\epsilon^2-2\epsilon^3} \begin{pmatrix} 1-4\epsilon+8\epsilon^2-7\epsilon^3+2\epsilon^4 \\ 2\epsilon-5\epsilon^2+5\epsilon^3-2\epsilon^4 \\ 2\epsilon-3\epsilon^2+3\epsilon^3 \\ 2\epsilon^2-3\epsilon^3+2\epsilon^4 \\ \epsilon^2+\epsilon^3-2\epsilon^4 \end{pmatrix} \tag{4.124}$$

Now we return to the information weight $W_t(\sigma)$. From the trellis sections in Fig. 4.16 it is easily seen how the information weights are transformed during one step of the Viterbi decoding. Transitions from state 0 or state 1 to state 0 decided by the Viterbi decoder without tie breaking do not cause an increment of the information weights; we simply copy the information weight from the state at the root of the branch to the state at the termini of the branch since such a transmission corresponds to $\widehat{u}_t = 0$. Having a transition from state 0 to state 1 decided by the Viterbi decoder without tie breaking, we obtain the information weight at state 1 and time $t+1$ by incrementing the information weight at state 0 and time t since such a transition corresponds to $\widehat{u}_t = 1$. Similarly, coming from state 1 we obtain the information

weight at state 1 and time $t+1$ by incrementing the information weight at state 1 and time t. If we have tie breaking, we use the arithmetic average of the information weights at the two states 0 and 1 at time t in our updating procedure.

Let $\boldsymbol{w}_t$ be the tuple of the information weights at time t split both on the two states σ_t and the five metric values ϕ_t, that is, we can write $\boldsymbol{w}_t$ as a vector of 10 entries:

$$\begin{aligned}\boldsymbol{w}_t = \Big(&w_t(\phi^{(1)}, \sigma = 0) \ \ldots \ \ w_t(\phi^{(5)}, \sigma = 0) \\ &w_t(\phi^{(1)}, \sigma = 1) \ \ldots \ w_t(\phi^{(5)}, \sigma = 1)\Big)\end{aligned} \tag{4.125}$$

The tuple $\boldsymbol{w}_t$ describes the dynamics of the information weights when we proceed along the trellis.

In general, for rate $R = b/c$, memory m convolutional encoders realized in controller canonical form, $\boldsymbol{w}_t$ has $M2^{bm}$ entries and satisfies the recurrent equation

$$\begin{aligned}\boldsymbol{w}_{t+1} &= \boldsymbol{w}_t A + \boldsymbol{b}_t B \\ \boldsymbol{b}_{t+1} &= \boldsymbol{b}_t \Pi\end{aligned} \tag{4.126}$$

where A and B are $M2^{bm} \times M2^{bm}$ nonnegative matrices and Π is an $M2^{bm} \times M2^{bm}$ stochastic matrix. The matrix A is the linear part of the affine transformation of the information weights and it can be determined from the $M2^{bm}$ trellis sections as illustrated in Fig. 4.16. The matrix B describes the increments of the information weights. The $M2^{bm}$-tuple $\boldsymbol{b}_t$ is the concatenation of 2^{bm} stochastic M-tuples $\boldsymbol{p}_i$. Hence, the matrix Π is given by

$$\Pi = \begin{pmatrix} \Phi & 0 & \ldots & 0 \\ 0 & \Phi & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & \Phi \end{pmatrix} \tag{4.127}$$

For simplicity, we choose the initial values

$$\begin{aligned}\boldsymbol{w}_0 &= \boldsymbol{0} \\ \boldsymbol{b}_0 &= \boldsymbol{b}_\infty = (\boldsymbol{p}_\infty \ \boldsymbol{p}_\infty \ \ldots \ \boldsymbol{p}_\infty)\end{aligned} \tag{4.128}$$

where $\boldsymbol{p}_\infty$ is the stationary probability distribution of the normalized cumulative metrics ϕ_t.

From (4.122) it follows that

$$\boldsymbol{b}_\infty = \boldsymbol{b}_\infty \Pi \tag{4.129}$$

Thus, (4.126) and (4.128) can be simplified to

$$\begin{aligned}\boldsymbol{w}_{t+1} &= \boldsymbol{w}_t A + \boldsymbol{b}_\infty B \\ \boldsymbol{w}_0 &= \boldsymbol{0}\end{aligned} \tag{4.130}$$

Next we shall return to our 2-state encoder and illustrate how the matrix A can be obtained from the trellis sections in Fig. 4.16.

Consider first a situation without tie breaking; for example, the trellis section in the upper left corner, where we have $\phi_t = -2, \phi_{t+1} = -2$, and $\boldsymbol{r}_t = 00$. Following the bold branches, we first copy with probability $\Pr(\boldsymbol{r}_t = 00) = (1-\epsilon)^2$ the information weight from state $\sigma_t = 0$ to state $\sigma_{t+1} = 0$, and then obtain the information weight at $\sigma_{t+1} = 1$ as the information weight at $\sigma_t = 0$ plus 1 (since $\widehat{u}_t = 1$ for this branch). We have now determined four of the entries in A, namely, the two entries for $\sigma_t = 0$, $\phi_t = -2$, and $\phi_{t+1} = -2$, which both are $(1-\epsilon)^2$, and the two entries for $\sigma_t = 1$, $\phi_t = -2$, and $\phi_{t+1} = -2$, which both are 0. Notice that, when we determine the entry for $\phi_{t+1} = 0$, we have to add the probabilities for the two trellis sections with $\phi_{t+1} = 0$.

Next we include tie breaking and choose as an example the trellis section with $\phi_t = -1, \phi_{t+1} = -2$, and $\boldsymbol{r}_t = 00$. Here we have to resolve ties at $\sigma_{t+1} = 1$. By following the bold branch from $\sigma_t = 0$ to $\sigma_{t+1} = 0$ we conclude that the information weight at state $\sigma_{t+1} = 0$ is a copy of the information weight at state $\sigma_t = 0$. Then we follow the two bold branches to state $\sigma_{t+1} = 1$ where the information weight is the arithmetic average of the information weights at states $\sigma_t = 0$ and $\sigma_t = 1$ plus 1. We have now determined another four entries of A, namely, the entry for $\sigma_t = 0$, $\phi_t = -1, \phi_{t+1} = -2$, and $\sigma_{t+1} = 0$, which is $(1-\epsilon)^2$; the two entries for $\phi_t = -1$, $\phi_{t+1} = -2$, and $\sigma_{t+1} = 1$, which are both $(1-\epsilon)^2/2$ (the tie is resolved by coin flipping); and, finally, the entry for $\sigma_t = 1$, $\phi_t = -2$, $\phi_{t+1} = -2$, and $\sigma_{t+1} = 0$, which is 0 since there is no bold branch between $\sigma_t = 1$ and $\sigma_{t+1} = 0$ in this trellis section.

Proceeding in this manner yields the matrix A for the memory $m = 1$ (2-state) convolutional encoder with generator matrix $G(D) = (1 \;\; 1+D)$:

$$A = \begin{array}{c} \\ \sigma_t = 0 \\ \sigma_t = 1 \end{array} \begin{array}{c} \begin{array}{cc} \sigma_{t+1} = 0 & \sigma_{t+1} = 1 \end{array} \\ \begin{pmatrix} A_{00} & A_{01} \\ A_{10} & A_{11} \end{pmatrix} \end{array}$$

$$= \begin{array}{c} \\ -2 \\ -1 \\ 0 \\ 1 \\ 2 \\ \hline -2 \\ -1 \\ 0 \\ 1 \\ 2 \end{array} \left(\begin{array}{ccccc|ccccc} \begin{array}{c} -2 \\ (1-\epsilon)^2 \\ (1-\epsilon)^2 \\ 0 \\ 0 \\ 0 \\ \hline 0 \\ 0 \\ 0 \\ \epsilon(1-\epsilon) \\ \epsilon(1-\epsilon) \end{array} & \begin{array}{c} -1 \\ 0 \\ 0 \\ (1-\epsilon)^2 \\ 0 \\ 0 \\ \hline 0 \\ 0 \\ \epsilon(1-\epsilon) \\ 0 \\ 0 \end{array} & \begin{array}{c} 0 \\ 2\epsilon(1-\epsilon) \\ \frac{3}{2}\epsilon(1-\epsilon) \\ 0 \\ \frac{1}{2}(1-\epsilon)^2 \\ 0 \\ \hline 0 \\ \frac{1}{2}\epsilon(1-\epsilon) \\ 0 \\ \epsilon^2 \\ \frac{1}{2}(1-\epsilon)^2 + \epsilon^2 \end{array} & \begin{array}{c} 1 \\ 0 \\ \frac{1}{2}\epsilon^2 \\ \epsilon(1-\epsilon) \\ 0 \\ \frac{1}{2}(1-\epsilon)\epsilon \\ \hline 0 \\ \frac{1}{2}\epsilon^2 \\ \epsilon^2 \\ 0 \\ \frac{1}{2}\epsilon(1-\epsilon) \end{array} & \begin{array}{c} 2 \\ \epsilon^2 \\ \frac{1}{2}\epsilon^2 \\ 0 \\ 0 \\ \frac{1}{2}\epsilon(1-\epsilon) \\ \hline 0 \\ \frac{1}{2}\epsilon^2 \\ 0 \\ 0 \\ \epsilon(1-\epsilon) \end{array} & \begin{array}{c} -2 \\ (1-\epsilon)^2 \\ \frac{1}{2}(1-\epsilon)^2 \\ \epsilon(1-\epsilon) \\ 0 \\ \frac{1}{2}\epsilon(1-\epsilon) \\ \hline 0 \\ \frac{1}{2}(1-\epsilon)^2 \\ 0 \\ (1-\epsilon)^2 \\ \epsilon(1-\epsilon) \end{array} & \begin{array}{c} -1 \\ 0 \\ 0 \\ \epsilon(1-\epsilon) \\ 0 \\ 0 \\ \hline 0 \\ 0 \\ \frac{1}{2}\epsilon(1-\epsilon) \\ 0 \\ 0 \end{array} & \begin{array}{c} 0 \\ 2\epsilon(1-\epsilon) \\ \frac{3}{2}\epsilon(1-\epsilon) \\ 0 \\ \frac{1}{2}\epsilon^2 \\ 0 \\ \hline 0 \\ \frac{1}{2}\epsilon(1-\epsilon) \\ 0 \\ (1-\epsilon)^2 + \frac{1}{2}\epsilon^2 \\ (1-\epsilon)^2 + \epsilon^2 \end{array} & \begin{array}{c} 1 \\ 0 \\ \frac{1}{2}\epsilon^2 \\ \epsilon^2 \\ 0 \\ \frac{1}{2}\epsilon(1-\epsilon) \\ \hline 0 \\ \frac{1}{2}\epsilon^2 \\ \epsilon(1-\epsilon) \\ 0 \\ \epsilon(1-\epsilon) \end{array} & \begin{array}{c} 2 \\ \epsilon^2 \\ \frac{1}{2}\epsilon^2 \\ 0 \\ 0 \\ \epsilon(1-\epsilon) \\ \hline 0 \\ \frac{1}{2}\epsilon^2 \\ 0 \\ 0 \\ \epsilon(1-\epsilon) \end{array} \end{array}\right) \tag{4.131}$$

For rate $R = 1/c$ convolutional feed-forward encoders realized in controller canonical form we only have increments when entering the states σ_{t+1}, when written

as an m-tuple, have a 1 as the first digit. For example, for our 2-state encoder we have

$$B = \begin{pmatrix} \mathbf{0}_{5,5} & A_{01} \\ \mathbf{0}_{5,5} & A_{11} \end{pmatrix} \tag{4.132}$$

Since every encoder state is reachable with probability 1, we have

$$\sum_{i=0}^{2^{bm}-1} A_{ij} = \Phi, \quad j = 0, 1, \ldots, 2^{bm} - 1 \tag{4.133}$$

In order to solve the recurrent matrix equation (4.130) we iterate it using the initial condition. Then we obtain

$$\boldsymbol{w}_{t+1} = \boldsymbol{b}_\infty BA^t + \boldsymbol{b}_\infty BA^{t-1} + \cdots + \boldsymbol{b}_\infty B \tag{4.134}$$

From (4.134) it follows that

$$\begin{aligned} \lim_{t\to\infty} \frac{\boldsymbol{w}_t}{t} &= \lim_{t\to\infty} \frac{\boldsymbol{w}_{t+1}}{t} = \lim_{t\to\infty} \frac{1}{t} \sum_{j=0}^{t} \boldsymbol{b}_\infty BA^{t-j} \\ &= \boldsymbol{b}_\infty BA^\infty \end{aligned} \tag{4.135}$$

where A^∞ denotes the limit of the sequence A^t when t tends to infinity and we used that, if a sequence is convergent to a finite limit, then it is Cesàro-summable to the same limit.

From (4.133) it follows that

$$\boldsymbol{e}_{\mathrm{L}} = (\boldsymbol{p}_\infty \ \ \boldsymbol{p}_\infty \cdots \boldsymbol{p}_\infty) \tag{4.136}$$

satisfies

$$\boldsymbol{e}_{\mathrm{L}} A = \boldsymbol{e}_{\mathrm{L}} \tag{4.137}$$

and, hence, $\boldsymbol{e}_{\mathrm{L}}$ is a left eigenvector of A with eigenvalue $\lambda = 1$.

From the nonnegativity of A it follows (Corollary 8.1.30 [HoJ90]) that $\lambda = 1$ is a maximal eigenvalue of A. Let $\boldsymbol{e}_{\mathrm{R}}$ be the right eigenvector corresponding to the eigenvalue $\lambda = 1$ and let $\boldsymbol{e}_{\mathrm{R}}$ be normalized such that $\boldsymbol{e}_{\mathrm{L}}\boldsymbol{e}_{\mathrm{R}} = 1$. Since $\boldsymbol{e}_{\mathrm{L}}$ is unique (up to normalization) it follows (Lemma 8.2.7, statement (i) [HoJ90]) that

$$A^\infty = \boldsymbol{e}_{\mathrm{R}}\boldsymbol{e}_{\mathrm{L}} \tag{4.138}$$

Combining (4.135), (4.136), and (4.138) yields

$$\lim_{t\to\infty} \frac{\boldsymbol{w}_t}{t} = \boldsymbol{b}_\infty B\boldsymbol{e}_{\mathrm{R}}(\boldsymbol{p}_\infty \ \ \boldsymbol{p}_\infty \cdots \boldsymbol{p}_\infty) \tag{4.139}$$

From (4.116) it follows that the expression for the exact bit error probability can be written as

$$\begin{aligned} P_{\mathrm{b}} &= \lim_{t\to\infty} \frac{E[W_t(\sigma = 0)]}{tb} = \lim_{t\to\infty} \frac{\sum_{i=1}^{M} \boldsymbol{w}_t(\phi^{(i)}, \sigma = 0)}{tb} \\ &= \lim_{t\to\infty} \frac{\boldsymbol{w}_t(\sigma = 0)\mathbf{1}_{1,M}^{\mathrm{T}}}{tb} \end{aligned} \tag{4.140}$$

where $\mathbf{1}_{1,M}$ is the all-one row vector of length M. In other words, to get the expression for P_b we sum up the first M components of the vector on the right-hand side of (4.139), or, equivalently, we multiply this vector by the vector $(\mathbf{1}_{1,M}\ \ \mathbf{0}_{1,M}\ldots\mathbf{0}_{1,M})^\mathrm{T}$. Then we obtain the following closed-form expression for the exact bit error probability:

$$P_\mathrm{b} = \boldsymbol{b}_\infty B \boldsymbol{e}_\mathrm{R}/b \tag{4.141}$$

In summary, for rate $R = b/c$ minimal convolutional encoders we can determine the exact bit error probability P_b for Viterbi decoding, when communicating over the BSC, as follows:

- Construct the set of metric states and find the stationary probability distribution $\boldsymbol{p}_\infty$.
- Construct the matrices A and B analogously to the memory $m = 1$ encoder given above and compute the right eigenvector $\boldsymbol{e}_\mathrm{R}$ normalized according to $(\boldsymbol{p}_\infty\ \ \boldsymbol{p}_\infty\ldots\boldsymbol{p}_\infty)\boldsymbol{e}_\mathrm{R} = 1$.
- Compute P_b using (4.141).

■ **EXAMPLE 4.7**

The rate $R = 1/2$, memory $m = 1$ (2-state) convolutional encoder with generator matrix $G(D) = (1\ \ 1+D)$ and $d_\mathrm{free} = 3$ has the set of metric states $\{-2,-1,0,1,2\}$ and the stationary probability distribution $\boldsymbol{p}_\infty$ is given by (4.124).

From the trellis sections in Fig. 4.16 we obtain the matrix A (4.131) with normalized right eigenvector

$$\boldsymbol{e}_\mathrm{R} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \dfrac{\epsilon(1-\epsilon)}{2} \\ \dfrac{4\epsilon(1-\epsilon)}{2-\epsilon+4\epsilon^2-4\epsilon^3} \\ \dfrac{(2+7\epsilon-12\epsilon^2+13\epsilon^3-12\epsilon^4+4\epsilon^5)}{2(2-\epsilon+4\epsilon^2-4\epsilon^3)} \\ 1 \end{pmatrix} \tag{4.142}$$

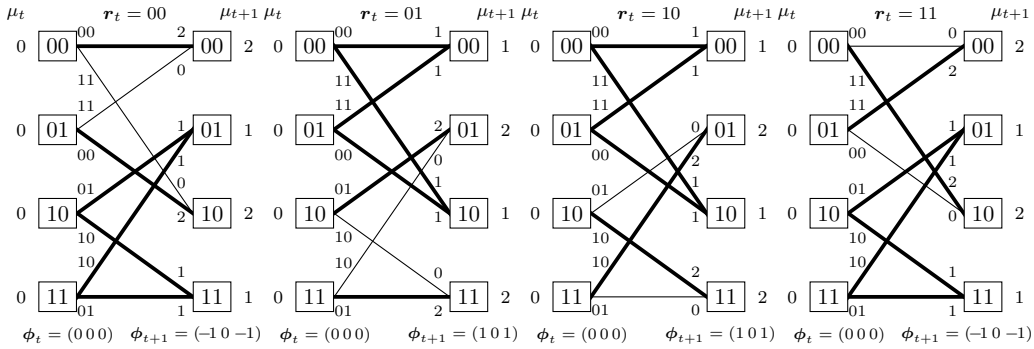


Figure 4.18 Four of in total 124 different trellis sections for the $G(D) = (1 + D^2 \quad 1 + D + D^2)$ encoding matrix.

Finally, inserting (4.124), (4.132), and (4.142) into (4.141) yields the exact bit error probability

$$\begin{aligned} P_{\mathrm{b}} &= \frac{14\epsilon^2 - 23\epsilon^3 + 16\epsilon^4 + 2\epsilon^5 - 16\epsilon^6 + 8\epsilon^7}{(1 + 3\epsilon^2 - 2\epsilon^3)(2 - \epsilon + 4\epsilon^2 - 4\epsilon^3)} \\ &= 7\epsilon^2 - 8\epsilon^3 - 31\epsilon^4 + 64\epsilon^5 + 86\epsilon^6 \\ &\quad - \frac{635}{2}\epsilon^7 - \frac{511}{4}\epsilon^8 + \frac{10165}{8}\epsilon^9 - \frac{4963}{16}\epsilon^{10} - \cdots \end{aligned} \tag{4.143}$$

which coincides with the bit error probability formula in [BBL95].

■ EXAMPLE 4.8

Consider the rate $R = 1/2$, memory $m = 2$ (4-state) convolutional minimal feed-forward encoder with encoding matrix $G(D) = (1 + D^2 \quad 1 + D + D^2)$ and $d_{\text{free}} = 5$ realized in controller canonical form. In Fig. 4.18 we show the four trellis sections for the normalized cumulative states $\boldsymbol{\phi}_t = (0 \quad 0 \quad 0)$. It follows from this figure that at time $t + 1$ we have two different normalized cumulative metric states, namely, $\boldsymbol{\phi}_{t+1} = (-1 \quad 0 \quad -1)$ and $\boldsymbol{\phi}_{t+1} = (1 \quad 0 \quad 1)$.

If we consider all different trellis sections we find that there exist $M = 31$ different normalized cumulative metric states. Hence, we have $2^m M = 4 \times 31 = 124$ different trellis sections. The matrix A consists of eight different nontrivial blocks corresponding to the eight branches in each trellis section. We obtain the 124×124 matrix

$$A = \begin{pmatrix} A_{00} & \mathbf{0}_{31,31} & A_{02} & \mathbf{0}_{31,31} \\ A_{10} & \mathbf{0}_{31,31} & A_{12} & \mathbf{0}_{31,31} \\ \mathbf{0}_{31,31} & A_{21} & \mathbf{0}_{31,31} & A_{23} \\ \mathbf{0}_{31,31} & A_{31} & \mathbf{0}_{31,31} & A_{33} \end{pmatrix} \tag{4.144}$$

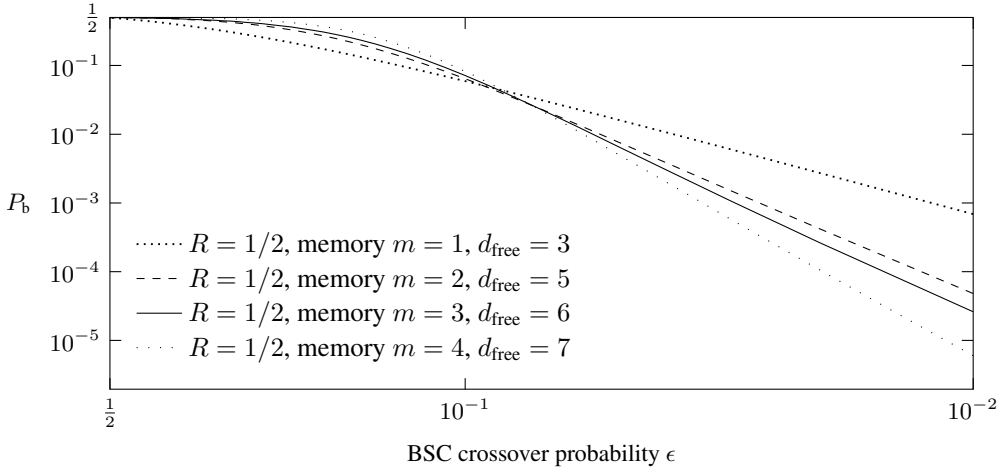


Figure 4.19 Exact bit error probability for the rate $R = 1/2$, memory $m = 1, 2, 3, 4$ encoding matrices.

Analogously to (4.132) we have

$$B = \begin{pmatrix} \mathbf{0}_{31,31} & \mathbf{0}_{31,31} & A_{02} & \mathbf{0}_{31,31} \\ \mathbf{0}_{31,31} & \mathbf{0}_{31,31} & A_{12} & \mathbf{0}_{31,31} \\ \mathbf{0}_{31,31} & \mathbf{0}_{31,31} & \mathbf{0}_{31,31} & A_{23} \\ \mathbf{0}_{31,31} & \mathbf{0}_{31,31} & \mathbf{0}_{31,31} & A_{33} \end{pmatrix} \tag{4.145}$$

and from (4.128) if follows that

$$\boldsymbol{b}_\infty = (\boldsymbol{p}_\infty \quad \boldsymbol{p}_\infty \quad \boldsymbol{p}_\infty \quad \boldsymbol{p}_\infty) \tag{4.146}$$

Solving for $\boldsymbol{e}_\mathrm{R}$ and inserting it together with B and $\boldsymbol{b}_\infty$ into (4.141) yields the following expression for the exact bit error probability:

$$\begin{aligned} P_\mathrm{b} = {} & 44\epsilon^3 + \frac{3519}{8}\epsilon^4 - \frac{14351}{32}\epsilon^5 - \frac{1267079}{64}\epsilon^6 \\ & - \frac{31646405}{512}\epsilon^7 + \frac{978265739}{2048}\epsilon^8 \\ & + \frac{3931764263}{1024}\epsilon^9 - \frac{48978857681}{32768}\epsilon^{10} + \cdots \end{aligned} \tag{4.147}$$

which coincides with the bit error probability formula in [LTZ04].

In Fig. 4.19 we show the exact bit error probability for rate $R = 1/2$ convolutional minimal feed-forward encoders with memory $m = 1, 2, 3, 4$. For the memory $m = 3$ (8-state) convolutional encoder with encoding matrix $G(D) = (1 + D^2 + D^3 \quad 1 + D + D^2 + D^3)$ we have $M = 433$ normalized cumulative metric states and the A and B

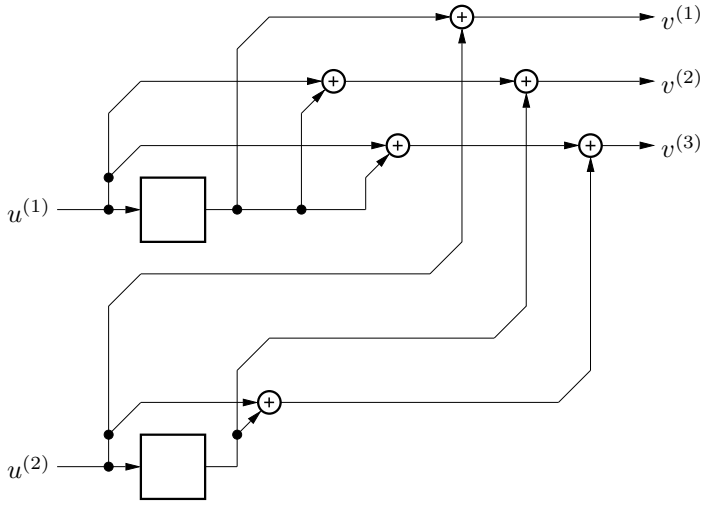


Figure 4.20 Realization of a rate $R = 2/3$, overall constraint length $\nu = 2$ convolutional encoder.

matrices are of size $433 \cdot 2^3 \times 433 \cdot 2^3$. For the memory $m = 4$ (16-state) convolutional encoder with encoding matrix $G(D) = (1 + D^2 + D^3 + D^4 \quad 1 + D + D^4)$ we have as many as $M = 188687$ normalized cumulative metric states.

It is not feasible to calculate the number of metric states for the rate $R = 1/2$, memory $m = 5$ (32-state) convolutional encoder with generator matrix $G(D) = (1 + D + D^2 + D^3 + D^4 + D^5 \quad 1 + D^3 + D^5)$ and $d_{\text{free}} = 8$; the number of metric states exceeds 4130000.

Next we shall illustrate how to extend the calculations of the exact bit error probability to high-rate minimal feed-forward convolutional encoders realized in controller canonical form.

■ **EXAMPLE 4.9**

Consider the rate $R = 2/3$, memory $m = 1$, overall constraint length $\nu = 2$, 4-state convolutional encoding matrix

$$G(D) = \begin{pmatrix} D & 1+D & 1+D \\ 1 & D & 1+D \end{pmatrix} \tag{4.148}$$

In Fig. 4.20 we show its realization in controller canonical form. Since the number of normalized metric states is $M = 19$, we have $2^c M = 8 \cdot 19 = 512$ trellis sections. One of these is shown in Fig. 4.21. Considering all trellis sections

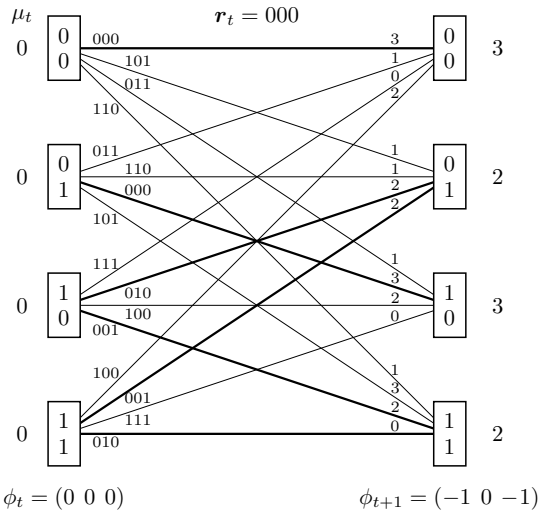


Figure 4.21 One typical trellis section of a rate $R = 2/3$, overall constraint length $\nu = 2$ convolutional encoder.

we obtain the $M2^{bm} \times M2^{bm} = 19 \cdot 4 \times 19 \cdot 4 = 76 \times 76$ matrices

$$A = \begin{pmatrix} A_{00} & A_{01} & A_{02} & A_{03} \\ A_{10} & A_{11} & A_{12} & A_{13} \\ A_{20} & A_{21} & A_{22} & A_{23} \\ A_{30} & A_{31} & A_{32} & A_{33} \end{pmatrix} \tag{4.149}$$

$$B = \begin{pmatrix} \mathbf{0} & A_{01} & A_{02} & 2A_{03} \\ \mathbf{0} & A_{11} & A_{12} & 2A_{13} \\ \mathbf{0} & A_{21} & A_{22} & 2A_{23} \\ \mathbf{0} & A_{31} & A_{32} & 2A_{33} \end{pmatrix} \tag{4.150}$$

and the row vector

$$\boldsymbol{b}_\infty = \begin{pmatrix} \boldsymbol{p}_\infty & \boldsymbol{p}_\infty & \boldsymbol{p}_\infty & \boldsymbol{p}_\infty \end{pmatrix} \tag{4.151}$$

The four states at time $t+1$ correspond to information weight increments of $0, 1, 1$, and 2, respectively, as shown in matrix B.

Then we compute the normalized eigenvector $\boldsymbol{e}_{\mathrm{R}}$ and use (4.151) to calculate the exact bit error probability for this rate $R = 2/3$, overall constraint length $\nu = 2$ convolutional encoder. The result is shown in Fig. 4.22 together with the corresponding curves for the rate $R = 2/3$, overall constraint lengths $\nu = 3$ and $\nu = 4$ convolutional encoders with encoding matrices

$$G(D) = \begin{pmatrix} 1+D & D & 1 \\ D^2 & 1 & 1+D+D^2 \end{pmatrix} \tag{4.152}$$

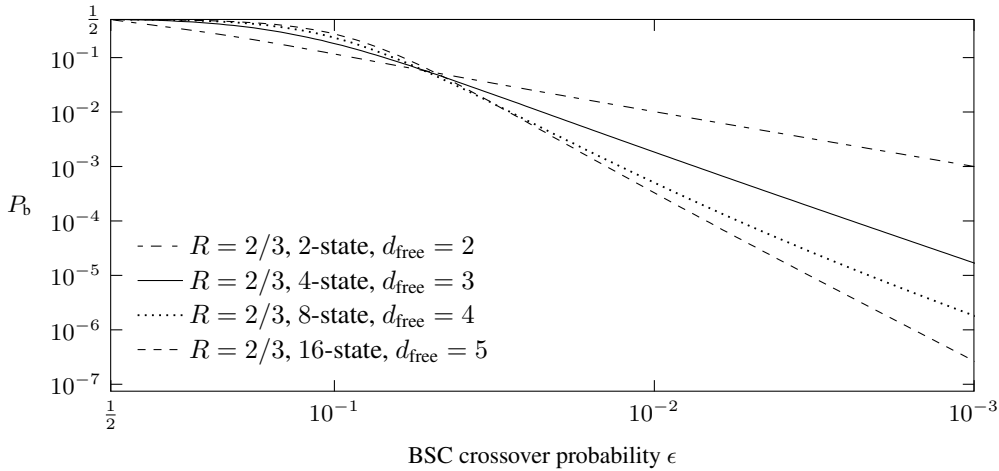


Figure 4.22 Exact bit error probability for the rate $R = 2/3$, overall constraint length $\nu = 2, 3$, and 4 (4-state, 8-state, and 16-state, respectively) encoding matrices in Example 4.9 and the rate $R = 2/3$ 2-state encoding matrix in Example 4.10.

with 8 encoder states and $M = 347$ metric states and

$$G(D) = \begin{pmatrix} D + D^2 & 1 & 1 + D^2 \\ 1 & D + D^2 & 1 + D + D^2 \end{pmatrix} \tag{4.153}$$

with 16 encoder states and $M = 15867$ metric states, respectively.

The extension to high-rate convolutional encoders is more subtle when the minimal realizations require realizations that are not in controller canonical form. We illustrate this by the following example.

■ **EXAMPLE 4.10**

Consider the following rate $R = 2/3$, overall constraint length $\nu = 2$ encoding matrix:

$$G(D) = \begin{pmatrix} 1 & 0 & 1 + D \\ 0 & 1 & 1 + D \end{pmatrix} \tag{4.154}$$

It has a 2-state minimal realization in observer canonical form as shown in Fig. 4.23.

Since our encoder is 2-state we have four blocks in the matrices A and B, that is,

$$A = \begin{pmatrix} A_{00} & A_{01} \\ A_{10} & A_{11} \end{pmatrix} \tag{4.155}$$

and

$$B = \begin{pmatrix} B_{00} & B_{01} \\ B_{10} & B_{11} \end{pmatrix} \tag{4.156}$$

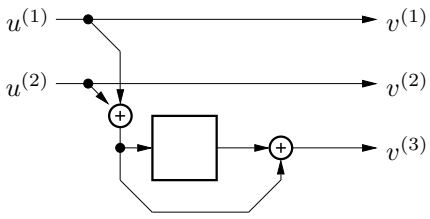


Figure 4.23 A minimal encoder for the encoding matrix given in (4.154).

By drawing all trellis sections we obtain the $M = 5$ normalized cumulative metrics: -2, -1, 0, 1, 2. In Fig. 4.24 we show four of the $M2^c = 5 \cdot 2^3 = 40$ different trellis sections.

Consider the trellis section for $\phi_t = 0$ and the triple $\boldsymbol{r}_t = 011$. Assuming that the allzero sequence is transmitted, this trellis section occurs with probability $(1-\epsilon)\epsilon^2$. At state $\sigma_{t+1} = 0$ we use coin flipping to resolve the tie between the two bold branches stemming from state $\sigma_t = 1$. Here we notice that the upper branch with the code triple $\boldsymbol{v}_t = 001$ corresponds to the information tuple $\boldsymbol{u}_t =$ 00, that is, no increment of the corresponding information weight entry at state $\sigma_{t+1} = 0$. However, the lower branch with the code triple $\boldsymbol{v}_t = 111$ corresponds to the information tuple $\boldsymbol{u}_t = 11$; thus, if this branch is decided by the Viterbi decoder, we have to add 2 to the corresponding information weight entry at state $\sigma_{t+1} = 0$.

The contributions from this trellis section to the entries of the matrices A and B determined by $(\sigma_t = 0, \phi_t = 0)$ and $(\sigma_{t+1} = 0, \phi_{t+1} = 1)$ are 0 for both matrices since the transition $\sigma_t = 0$ to $\sigma_{t+1} = 1$ is not decided by the Viterbi decoder for these normalized cumulative metrics. The same holds for $\boldsymbol{r}_t = 101$. But for both $\boldsymbol{r}_t = 010$ and $\boldsymbol{r}_t = 100$ we get for these normalized cumulative matrices a tie for the transitions between $\sigma_t = 0$ and $\sigma_{t+1} = 0$. Hence, the corresponding entry in submatrix A_{00} is $2\epsilon^2(1-\epsilon)$. For the submatrix B_{00} this entry is also $2\epsilon^2(1-\epsilon)$; there we get a contribution only for the lower branch but this transition corresponds to 2 information bits. Next we consider the transitions from $(\sigma_t = 0, \phi_t = 0)$ to $(\sigma_{t+1} = 1, \phi_{t+1} = 1)$. Then for $\boldsymbol{r}_t = 010$ and $\boldsymbol{r}_t = 100$ there are no branches decided by the Viterbi decoder and, hence, no contributions to A_{01} and B_{01}. For $\boldsymbol{r}_t = 011$ and $\boldsymbol{r}_t = 101$ we obtain the contributions $\epsilon^2(1-\epsilon)$ for both received triples and, hence, $2\epsilon^2(1-\epsilon)$ in total to A_{01}. For matrix B_{01} we have one erroneous information bit for both received triples and, hence, we get the same value as for A_{01}.

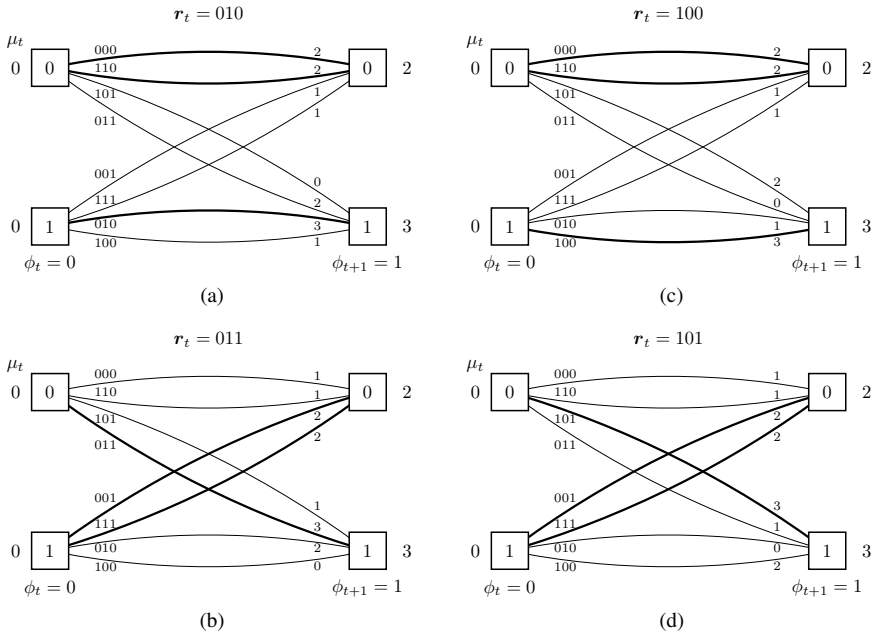


Figure 4.24 Four trellis sections for the rate $R = 2/3$, 2-state encoder in Fig. 4.23.

Exploiting all 40 trellis sections in a similar way yields

$$A_{00} = \begin{array}{c} \\ \text{-2} \\ \text{-1} \\ 0 \\ 1 \\ 2 \end{array} \begin{array}{c} \begin{array}{ccccc} \text{-2} & \text{-1} & 0 & 1 & 2 \end{array} \\ \begin{pmatrix} (1-\epsilon)^3+\epsilon^2(1-\epsilon) & 0 & \epsilon^3+3\epsilon(1-\epsilon)^2 & 0 & 2\epsilon^2(1-\epsilon) \\ (1-\epsilon)^3+\epsilon^2(1-\epsilon) & 0 & \frac{1}{2}\epsilon^3+\frac{5}{2}\epsilon(1-\epsilon)^2 & 0 & \epsilon^2(1-\epsilon) \\ 0 & (1-\epsilon)^3+\epsilon^2(1-\epsilon) & 0 & 2\epsilon(1-\epsilon)^2 & 0 \\ 0 & 0 & \frac{1}{2}(1-\epsilon)^3+\frac{1}{2}\epsilon^2(1-\epsilon) & 0 & \epsilon(1-\epsilon)^2 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

and

$$B_{00} = \begin{array}{c} \\ \text{-2} \\ \text{-1} \\ 0 \\ 1 \\ 2 \end{array} \begin{array}{c} \begin{array}{ccccc} \text{-2} & \text{-1} & 0 & 1 & 2 \end{array} \\ \begin{pmatrix} 2\epsilon^2(1-\epsilon) & 0 & 2\epsilon^3+2\epsilon(1-\epsilon)^2 & 0 & 2\epsilon^2(1-\epsilon) \\ 2\epsilon^2(1-\epsilon) & 0 & \epsilon^3+2\epsilon(1-\epsilon)^2 & 0 & \epsilon^2(1-\epsilon) \\ 0 & 2\epsilon^2(1-\epsilon) & 0 & 2\epsilon(1-\epsilon)^2 & 0 \\ 0 & 0 & \epsilon^2(1-\epsilon) & 0 & \epsilon(1-\epsilon)^2 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

Determining the remaining blocks is left as an exercise (cf. Problem 4.20 and Problem 4.21).

After having determined the right eigenvector of A, that is, $\boldsymbol{e}_{\mathrm{R}}$, we exploit the 40 trellis sections and obtain the normalized cumulative metric transition matrix Φ, which yields the steady state probability vector $\boldsymbol{p}_\infty$ and hence we have determined $\boldsymbol{b}_\infty$.

Finally, we insert $\boldsymbol{b}_\infty$, B, $\boldsymbol{e}_\mathrm{R}$, and $b = 2$ into (4.141) and obtain the following expression for the exact bit error probability for our $R = 2/3$, 2-state minimal convolutional encoder realized in observer canonical form, illustrated in Fig. 4.22,

$$\begin{aligned} P_\mathrm{b} &= \Big(-4\epsilon+2\epsilon^2-67\epsilon^3+320\epsilon^4-818\epsilon^5+936\epsilon^6+884\epsilon^7-5592\epsilon^8+11232\epsilon^9 \\ &\quad -13680\epsilon^{10}+11008\epsilon^{11}-5760\epsilon^{12}+1792\epsilon^{13}+6\epsilon^{14}\Big) / \Big(-2+5\epsilon-41\epsilon^2+128\epsilon^3 \\ &\quad -360\epsilon^4+892\epsilon^5-1600\epsilon^6+1904\epsilon^7-1440\epsilon^8+640\epsilon^9-128\epsilon^{10}\Big) \\ &= 2\epsilon + 4\epsilon^2 + \frac{5}{2}\epsilon^3 - \frac{431}{4}\epsilon^4 - \frac{125}{8}\epsilon^5 + \frac{32541}{16}\epsilon^6 - \frac{70373}{32}\epsilon^7 \\ &\quad - \frac{1675587}{64}\epsilon^8 + \frac{7590667}{128}\epsilon^9 + \frac{67672493}{256}\epsilon^{10} - \ldots \end{aligned} \tag{4.157}$$

In this example, the increments depend not only on the states σ_{t+1} but also on the received triple $\boldsymbol{r}_t$ and the resolving of ties.

Determining the exact bit error probability for encoders with feedback follows the steps outlined in Example 4.10.

If we replace the BSC with a quantized AWGN channel, the calculations follow the methods described in the previous examples, but the computational complexity increases drastically as illustrated by the following example. The quantization levels are determined by optimizing the computational cutoff rate as described in Section 5.6.

■ **EXAMPLE 4.11**

For the encoding matrix $G(D) = (1 + D^2 \quad 1 + D + D^2)$ realized in controller canonical form we have $M = 31$ normalized cumulative metric states for the BSC, but as many as $M = 16639$ for the AWGN channel with eight quantization levels. Moreover, for the AWGN channel we have to repeat all calculations for each value of the SNR. In Fig. 4.25 we show the exact bit error probability for this encoder and the AWGN channel. For comparison we also give the exact bit error probabilities for the same encoder and its equivalents $G(D) = \big(1 \quad (1 + D^2)/(1 + D + D^2)\big)$, $M = 31$, $G(D) = \big(1 \quad (1 + D + D^2)/(1 + D^2)\big)$, $M = 31$, when used to communicate over the BSC channel.

4.5 THE BCJR ALGORITHM FOR APP DECODING

The BCJR (*two-way*) algorithm is the most celebrated algorithm for APP decoding of *terminated* convolutional codes.

Suppose that a binary, rate $R = b/c$, convolutional code of memory m is used to communicate over the binary-input, q-ary output DMC (introduced in Section 1.1). As before, let $\boldsymbol{v}$ denote the code sequence, $\boldsymbol{r}$ the received sequence, and $P(\boldsymbol{r} \mid \boldsymbol{v})$ the channel-transition probabilities, and use the zero-tail (ZT) method to terminate the

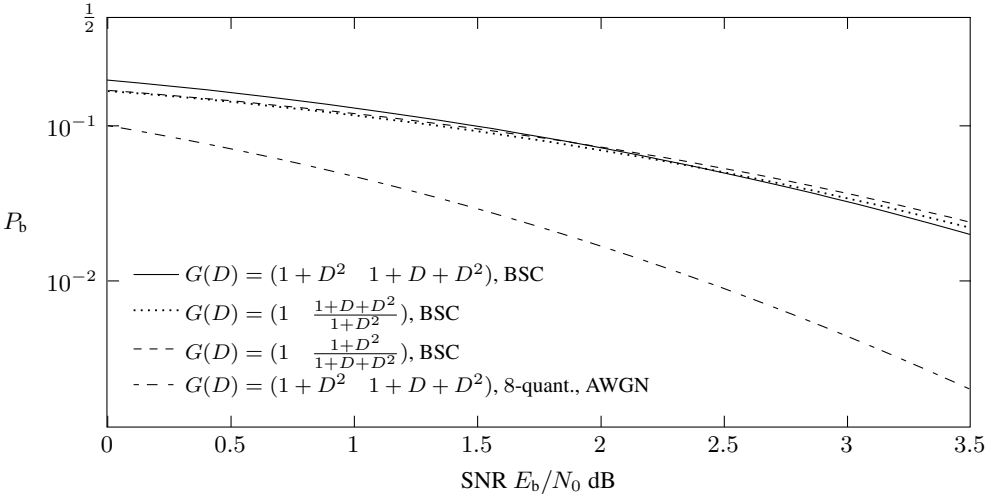


Figure 4.25 Exact bit error probability for the rate $R = 1/2$, memory $m = 2$ encoding matrix $G(D) = (1 + D^2 \quad 1 + D + D^2)$ for BSC and quantized AWGN channel.

convolutional code into a block code of block length $N = (n + m)c$ code symbols. Furthermore, let

$$\begin{aligned}\boldsymbol{u}_{[0,n+m)} &= \boldsymbol{u}_0\, \boldsymbol{u}_1 \ldots \boldsymbol{u}_{n+m-1} \\ &= u_0^{(1)} u_0^{(2)} \ldots u_0^{(b)} u_1^{(1)} u_1^{(2)} \ldots u_1^{(b)} \ldots u_{n+m-1}^{(1)} u_{n+m-1}^{(2)} \ldots u_{n+m-1}^{(b)}\end{aligned} \tag{4.158}$$

denote the sequence of bn information symbols followed by a tail of bm dummy symbols which terminates the convolutional code into a block code. We let $P(u_i^{(k)} = 0)$ denote the *a priori* probability that information symbol $u_i^{(k)} = 0, i = 0, 1, \ldots, n-1$, $k = 1, 2, \ldots, b$. In general, $P(u_i^{(k)} = 0) \neq 1/2$. Let $P\left(u_i^{(k)} = 0 \,\middle|\, \boldsymbol{r}_{[0,n+m)}\right)$ denote the *a posteriori* probability that $u_i^{(k)} = 0$ given the received sequence $\boldsymbol{r}_{[0,n+m)}$, where

$$\begin{aligned}\boldsymbol{r}_{[0,n+m)} &= \boldsymbol{r}_0\, \boldsymbol{r}_1 \ldots \boldsymbol{r}_{n+m-1} \\ &= r_0^{(1)} r_0^{(2)} \ldots r_0^{(c)} r_1^{(1)} r_1^{(2)} \ldots r_1^{(c)} \ldots r_{n+m-1}^{(1)} r_{n+m-1}^{(2)} \ldots r_{n+m-1}^{(c)}\end{aligned} \tag{4.159}$$

Let $\mathcal{U}_{[0,n)}$ denote the set of all information sequences $\boldsymbol{u}_0\boldsymbol{u}_1 \ldots \boldsymbol{u}_{n-1}$, and let $\mathcal{U}_{[0,n)i}^{(k)}$ denote the set of information sequences $\boldsymbol{u}_0\boldsymbol{u}_1 \ldots \boldsymbol{u}_{n-1}$ such that $u_i^{(k)} = 0$. Then we have the following expression for the *a posteriori* probability that $u_i^{(k)} = 0$ given the

received sequence $\boldsymbol{r}_{[0,n+m)}$:

$$P\left(u_i^{(k)}=0 \;\middle|\; \boldsymbol{r}_{[0,n+m)}\right) = \frac{P(\boldsymbol{r}_{[0,n+m)}, u_i^{(k)}=0)}{P(\boldsymbol{r}_{[0,n+m)})} = \frac{\sum_{\boldsymbol{u}_{[0,n)}\in\mathcal{U}_{[0,n)i}^{(k)}} P\left(\boldsymbol{r}_{[0,n+m)} \mid \boldsymbol{u}_{[0,n)}\right) P(\boldsymbol{u}_{[0,n)})}{\sum_{\boldsymbol{u}_{[0,n)}\in\mathcal{U}_{[0,n)}} P\left(\boldsymbol{r}_{[0,n+m)} \mid \boldsymbol{u}_{[0,n)}\right) P(\boldsymbol{u}_{[0,n)})} \tag{4.160}$$

where $P\left(\boldsymbol{r}_{[0,n+m)} \mid \boldsymbol{u}_{[0,n)}\right)$ is the probability that $\boldsymbol{r}_{[0,n+m)}$ is received given that the code sequence corresponding to the information sequence $\boldsymbol{u}_{[0,n)}$ followed by a tail of m dummy zeros is transmitted and $P(\boldsymbol{u}_{[0,n)})$ is the *a priori* probability of the information sequence $\boldsymbol{u}_{[0,n)}$. Our goal is to compute the sequence of *a posteriori* probabilities

$$\begin{aligned}
&P\left(u_0^{(1)}=0 \;\middle|\; \boldsymbol{r}_{[0,n+m)}\right), P\left(u_0^{(2)}=0 \;\middle|\; \boldsymbol{r}_{[0,n+m)}\right), \ldots, P\left(u_0^{(b)}=0 \;\middle|\; \boldsymbol{r}_{[0,n+m)}\right),\\
&P\left(u_1^{(1)}=0 \;\middle|\; \boldsymbol{r}_{[0,n+m)}\right), P\left(u_1^{(2)}=0 \;\middle|\; \boldsymbol{r}_{[0,n+m)}\right), \ldots, P\left(u_1^{(b)}=0 \;\middle|\; \boldsymbol{r}_{[0,n+m)}\right), \ldots,\\
&P\left(u_{n-1}^{(1)}=0 \;\middle|\; \boldsymbol{r}_{[0,n+m)}\right), P\left(u_{n-1}^{(2)}=0 \;\middle|\; \boldsymbol{r}_{[0,n+m)}\right), \ldots, P\left(u_{n-1}^{(b)}=0 \;\middle|\; \boldsymbol{r}_{[0,n+m)}\right)
\end{aligned}$$

For an encoding matrix of overall constraint length ν, we denote for simplicity the encoder state at depth t by σ_t, where $\sigma_t = \sigma$, $\sigma = 0, 1, \ldots, 2^\nu - 1$. We denote by $\mathcal{S}_{[0,n+m]}$ the set of state sequences $\boldsymbol{\sigma}_{[0,n+m]}$ such that $\sigma_0 = \sigma_{n+m} = 0$, that is,

$$\mathcal{S}_{[0,n+m]} \stackrel{\text{def}}{=} \left\{\boldsymbol{\sigma}_{[0,n+m]} = \sigma_0\sigma_1\ldots\sigma_{n+m} \;\middle|\; \sigma_0 = \sigma_{n+m} = 0\right\} \tag{4.161}$$

and by $\mathcal{S}_{[0,n+m]i}^{(k)}$ the set of state sequences $\boldsymbol{\sigma}_{[0,n+m]}$ such that $\sigma_0 = \sigma_{n+m} = 0$ and that the transition from state σ_i at depth i to state σ_{i+1} at depth $i+1$ implies that $u_i^{(k)} = 0$, that is,

$$\mathcal{S}_{[0,n+m]i}^{(k)} \stackrel{\text{def}}{=} \left\{\boldsymbol{\sigma}_{[0,n+m]} = \sigma_0\sigma_1\ldots\sigma_{n+m} \;\middle|\; \sigma_0 = \sigma_{n+m} = 0 \text{ and } \sigma_i \to \sigma_{i+1} \Rightarrow u_i^{(k)} = 0\right\} \tag{4.162}$$

We always start and end the encoding in the zero state, and we have a one-to-one correspondence between the information sequence $\boldsymbol{u}_{[0,n)}$ and the sequence of encoder states $\boldsymbol{\sigma}_{[0,n+m]} = \sigma_0\sigma_1\ldots\sigma_{n+m}$, where $\sigma_0 = \sigma_{n+m} = 0$. Hence, we can rewrite (4.160) as

$$P\left(u_i^{(k)}=0 \;\middle|\; \boldsymbol{r}_{[0,n+m)}\right) = \frac{\sum_{\boldsymbol{\sigma}_{[0,n+m]}\in\mathcal{S}_{[0,n+m]i}^{(k)}} P\left(\boldsymbol{r}_{[0,n+m)} \mid \boldsymbol{\sigma}_{[0,n+m]}\right) P\left(\boldsymbol{\sigma}_{[0,n+m]}\right)}{\sum_{\boldsymbol{\sigma}_{[0,n+m]}\in\mathcal{S}_{[0,n+m]}} P\left(\boldsymbol{r}_{[0,n+m)} \mid \boldsymbol{\sigma}_{[0,n+m]}\right) P\left(\boldsymbol{\sigma}_{[0,n+m]}\right)} \tag{4.163}$$

where $P\left(\boldsymbol{r}_{[0,n+m)} \mid \boldsymbol{\sigma}_{[0,n+m]}\right)$ is the probability that $\boldsymbol{r}_{[0,n+m)}$ is received given that the code sequence corresponding to the state sequence $\boldsymbol{\sigma}_{[0,n+m]}$ is transmitted and $P(\boldsymbol{\sigma}_{[0,n+m]})$ is the *a priori* probability of the state sequence $\boldsymbol{\sigma}_{[0,n+m]}$.

Let $P(\boldsymbol{r}_t, \sigma_{t+1} = \sigma' \mid \sigma_t = \sigma)$, where $\sigma, \sigma' \in \{0, 1, \ldots, 2^\nu - 1\}$, be the conditional probability that at depth t we receive the c-tuple $\boldsymbol{r}_t$ and that the encoder makes the state transition to $\sigma_{t+1} = \sigma'$ at depth $t+1$, given that it is at $\sigma_t = \sigma$ at depth t, and let $P(\sigma_{t+1} = \sigma' \mid \sigma_t = \sigma)$ be the probability of the same state transition. Next, we introduce the $2^\nu \times 2^\nu$ state-transition matrix

$$P_t = (p_t(\sigma, \sigma'))_{\sigma,\sigma'}, \quad 0 \le \sigma < 2^\nu, \quad 0 \le \sigma' < 2^\nu \tag{4.164}$$

where

$$\begin{aligned} p_t(\sigma, \sigma') &= P(\boldsymbol{r}_t, \sigma_{t+1} = \sigma' \mid \sigma_t = \sigma) \\ &= P(\boldsymbol{r}_t \mid \sigma_{t+1} = \sigma', \sigma_t = \sigma)\, P(\sigma_{t+1} = \sigma' \mid \sigma_t = \sigma) \end{aligned} \tag{4.165}$$

and $\sigma, \sigma' \in \{0, 1, \ldots, 2^\nu - 1\}$. The matrix P_t is a sparse matrix; in each row and each column, only 2^b elements are nonzero.

Let

$$\boldsymbol{e}_i = (0 \ldots 0 \underset{i}{1} 0 \ldots 0), \quad 0 \le i < 2^\nu \tag{4.166}$$

Consider the product

$$\boldsymbol{e}_0 P_0 P_1 \ldots P_{n+m-1} = (\gamma\, 0 \ldots 0) \tag{4.167}$$

where the equalities to 0 in the last $2^\nu - 1$ entries follow from the fact that the tail of m 0-state driving b-tuples causes the state transitions $\sigma_n \to \sigma_{n+1} \to \cdots \to \sigma_{n+m}$ to terminate in $\sigma_{n+m} = 0$. The value γ obtained by (4.167) is the conditional probability that we receive $\boldsymbol{r}_{[0,n+m)}$ given that a code sequence is transmitted, that is,

$$\gamma = \sum_{\boldsymbol{\sigma}_{[0,n+m]} \in \mathcal{S}_{[0,n+m]}} P\left(\boldsymbol{r}_{[0,n+m)} \mid \boldsymbol{\sigma}_{[0,n+m]}\right) P\left(\boldsymbol{\sigma}_{[0,n+m]}\right) \tag{4.168}$$

which is the denominator of (4.163).

In order to calculate the numerator of (4.163), we introduce, as a counterpart to P_t, the state-transition matrix

$$P_t^{(k)} = \left(p_t^{(k)}(\sigma, \sigma')\right)_{\sigma,\sigma'}, \quad 0 \le \sigma < 2^\nu, \quad 0 \le \sigma' < 2^\nu \tag{4.169}$$

where

$$\begin{aligned} p_t^{(k)}(\sigma, \sigma') &= P\left(\boldsymbol{r}_t, \sigma_{t+1} = \sigma', u_t^{(k)} = 0 \,\middle|\, \sigma_t = \sigma\right) \\ &= P\left(\boldsymbol{r}_t \,\middle|\, \sigma_{t+1} = \sigma', \sigma_t = \sigma, u_t^{(k)} = 0\right) \\ &\quad \times P\left(\sigma_{t+1} = \sigma' \,\middle|\, \sigma_t = \sigma, u_t^{(k)} = 0\right) P\left(u_t^{(k)} = 0\right) \end{aligned} \tag{4.170}$$

and $\sigma, \sigma' \in \{0, 1, \ldots, 2^\nu - 1\}$. The matrix element $p_t^{(k)}(\sigma, \sigma')$ is the conditional probability that at depth t we receive the c-tuple $\boldsymbol{r}_t$, that the encoder makes the state transition to $\sigma_{t+1} = \sigma'$ at time $t+1$, and that the kth information symbol at depth t is $u_t^{(k)} = 0$, given that it is at $\sigma_t = \sigma$ at depth t.

As a counterpart to (4.167), we have

$$\boldsymbol{e}_0 P_0 P_1 \dots P_{i-1} P_i^{(k)} P_{i+1} \dots P_{n+m-1} = \left(\gamma_i^{(k)} 0 \dots 0\right) \tag{4.171}$$

where $\gamma_i^{(k)}$ is the conditional probability that we receive $\boldsymbol{r}_{[0,n+m)}$, given that a code sequence corresponding to $u_i^{(k)} = 0$ is transmitted, that is,

$$\gamma_i^{(k)} = \sum_{\boldsymbol{\sigma}_{[n+m]} \in \mathcal{S}_{[0,n+m]i}^{(k)}} P\left(\boldsymbol{r}_{[0,n+m)} \mid \boldsymbol{\sigma}_{[0,n+m]}\right) P\left(\boldsymbol{\sigma}_{[0,n+m]}\right) \tag{4.172}$$

which is the numerator of (4.163). Hence, we can rewrite (4.163) as

$$P\left(u_i^{(k)} = 0 \,\middle|\, \boldsymbol{r}_{[0,n+m)}\right) = \gamma_i^{(k)} / \gamma \tag{4.173}$$

where γ and $\gamma_i^{(k)}$ are given by (4.167) and (4.171), respectively.

The most crucial part of the BCJR algorithm is the calculation of $\gamma_i^{(k)}$ for $i = 0, 1, \dots, n-1$ and $k = 1, 2, \dots, b$. First, we start at the root and go forward and calculate

$$\boldsymbol{\alpha}_i = \left(\alpha_i(0)\, \alpha_i(1) \dots \alpha_i(2^\nu - 1)\right) \stackrel{\text{def}}{=} \boldsymbol{e}_0 P_0 P_1 \dots P_{i-1}, \quad 1 \le i \le n+m \tag{4.174}$$

By convention, we have $\boldsymbol{\alpha}_0 = \boldsymbol{e}_0$. For each depth i, $i = 1, \dots, n+m$, the components of $\boldsymbol{\alpha}_i$ are stored at the corresponding states. Then we start at the terminal node at depth $n+m$, go backward, and calculate

$$\begin{aligned} \boldsymbol{\beta}_i^{(k)} &= \left(\beta_i^{(k)}(0)\, \beta_i^{(k)}(1) \dots \beta_i^{(k)}(2^\nu - 1)\right) \\ &\stackrel{\text{def}}{=} \boldsymbol{e}_0 P_{n+m-1}^{\mathrm{T}} P_{n+m-2}^{\mathrm{T}} \dots P_{i+1}^{\mathrm{T}} \left(P_i^{(k)}\right)^{\mathrm{T}}, \quad 0 \le i < n, 1 \le k \le b \end{aligned} \tag{4.175}$$

By combining (4.172) with the definitions of $\boldsymbol{\alpha}_i$ and $\boldsymbol{\beta}_i^{(k)}$ we obtain

$$\gamma_i^{(k)} = \sum_{\sigma=0}^{2^\nu - 1} \alpha_i(\sigma)\, \beta_i^{(k)}(\sigma), \quad 0 \le i < n \tag{4.176}$$

From (4.168) and the definition of $\boldsymbol{\alpha}_i(\sigma)$ it follows that

$$\gamma = \alpha_{n+m}(0) \tag{4.177}$$

Hence, combining (4.173), (4.176), and (4.177) yields

$$P\left(u_i^{(k)} = 0 \,\middle|\, \boldsymbol{r}_{[n+m)}\right) = \frac{\sum_{\sigma=0}^{2^\nu - 1} \alpha_i(\sigma) \beta_i^{(k)}(\sigma)}{\alpha_{n+m}(0)}, \quad 0 \le i < n, 1 \le k \le b \tag{4.178}$$

Since the matrix P_t is sparse, it is efficient to compute $\gamma_i^{(k)}$ by trellis searches. For each state we will calculate both forward and backward multiplicative metrics.

In the forward direction, we start at depth $t = 0$ with the metric

$$\mu_0(\sigma) = \begin{cases} 1, & \sigma = 0 \\ 0, & \text{otherwise} \end{cases} \tag{4.179}$$

Then for $t = 1, 2, \ldots, n + m$ we calculate the *forward metric* $\mu_t(\sigma)$ as

$$\mu_t(\sigma') = \sum_{\sigma=0}^{2^\nu - 1} \mu_{t-1}(\sigma) p_{t-1}(\sigma, \sigma'), \quad 0 \le \sigma' < 2^\nu \tag{4.180}$$

where $p_{t-1}(\sigma, \sigma')$ is given by (4.165). The sum has only 2^b nonzero terms. The forward metrics $\mu_t(\sigma')$ are stored at the corresponding states. It is easily verified that

$$\mu_i(\sigma) = \alpha_i(\sigma), \quad 0 \le i \le n + m, 0 \le \sigma < 2^\nu \tag{4.181}$$

where $\alpha_i(\sigma)$ is given by (4.174).

In the backward direction, we start at depth $n + m$ with the metric

$$\tilde{\mu}_{n+m}(\sigma') = \begin{cases} 1, & \sigma' = 0 \\ 0, & \text{otherwise} \end{cases} \tag{4.182}$$

Then for $t = n + m - 1, n + m - 2, \ldots, 1$ we calculate the *backward metric*

$$\tilde{\mu}_t(\sigma) = \sum_{\sigma'=0}^{2^\nu - 1} \tilde{\mu}_{t+1}(\sigma') p_t(\sigma, \sigma'), \quad 0 \le \sigma < 2^\nu \tag{4.183}$$

where $p_t(\sigma, \sigma')$ is given by (4.165). It is easily shown that

$$\sum_{\sigma'=0}^{2^\nu - 1} p_i^{(k)}(\sigma, \sigma') \tilde{\mu}_{i+1}(\sigma') = \beta_i^{(k)}(\sigma), \quad 0 \le i < n, 1 \le k \le b, 0 \le \sigma < 2^\nu \tag{4.184}$$

where $p_i^{(k)}(\sigma, \sigma')$ and $\beta_i^{(k)}(\sigma)$ are given by (4.170) and (4.175), respectively.

Finally, we obtain

$$\gamma_i^{(k)} = \sum_{\sigma=0}^{2^\nu - 1} \sum_{\sigma'=0}^{2^\nu - 1} \mu_i(\sigma) p_i^{(k)}(\sigma, \sigma') \tilde{\mu}_{i+1}(\sigma') \tag{4.185}$$

The BCJR algorithm can be summarized as follows:

Algorithm BCJR (The BCJR algorithm for APP decoding)

BCJR1. Initialize $\mu_0(0) = \tilde{\mu}_{n+m}(0) = 1$, $\mu_0(\sigma) = \tilde{\mu}_{n+m}(\sigma) = 0$ for all nonzero states ($\sigma \neq 0$).

BCJR2. For $t = 1, 2, \ldots, n + m$ calculate the forward metric

$$\mu_t(\sigma') = \sum_{\sigma=0}^{2^\nu - 1} \mu_{t-1}(\sigma) p_{t-1}(\sigma, \sigma'), \quad 0 \le \sigma' < 2^\nu$$

BCJR3. For $t = n+m-1, n+m-2, \ldots, 1$ calculate the backward metric

$$\tilde{\mu}_t(\sigma) = \sum_{\sigma'=0}^{2^\nu-1} \tilde{\mu}_{t+1}(\sigma') p_t(\sigma, \sigma'), \quad 0 \le \sigma < 2^\nu$$

BCJR4. For $i = 0, 1, \ldots, n-1$ and $k = 1, 2, \ldots, b$ calculate

$$\gamma_i^{(k)} = \sum_{\sigma=0}^{2^\nu-1} \sum_{\sigma'=0}^{2^\nu-1} \mu_i(\sigma) p_i^{(k)}(\sigma, \sigma') \tilde{\mu}_{i+1}(\sigma')$$

and output

$$P\left(u_i^{(k)} = 0 \,\middle|\, \boldsymbol{r}_{[0,n+m)}\right) = \gamma_i^{(k)} / \mu_{n+m}(0)$$

In iterative decoding, for example, we use the *a posteriori* probabilities that are calculated by the BCJR algorithm as *a priori* probabilities in the following step of the iteration. In *maximum a posteriori probability* (MAP) decoding the *a posteriori* probabilities are used to obtain decisions about the information symbols; we simply use the rule

$$\widehat{u}_i^{(k)} = \begin{cases} 0 & \text{if } P\left(u_i^{(k)} = 0 \,\middle|\, \boldsymbol{r}_{[n+m)}\right) \ge 1/2 \\ 1 & \text{otherwise} \end{cases} \tag{4.186}$$

■ **EXAMPLE 4.12**

Consider the binary-input, 8-ary output DMC shown in Fig. 4.6 with transition probabilities $P(r \mid v)$ given by the following table:

		r							
		0_4	0_3	0_2	0_1	1_1	1_2	1_3	1_4
v	0	0.434	0.197	0.167	0.111	0.058	0.023	0.008	0.002
	1	0.002	0.008	0.023	0.058	0.111	0.167	0.197	0.434

Suppose that the same encoding matrix $G(D) = (1 + D + D^2 \quad 1 + D^2)$ as in Example 4.2 is used and that four information symbols followed by two dummy zeros are encoded. Assume that the *a priori* probabilities for the information symbols are $P(u_t = 0) = 2/3$, $t = 0, 1, 2, 3$. For the dummy zeros we have $P(u_t = 0) = 1$, $t = 4, 5$. Let $\boldsymbol{r} = 1_1 0_4 \; 0_1 1_2 \; 1_1 0_1 \; 0_1 1_1 \; 0_1 1_3 \; 0_4 0_3$ be the received sequence. The trellis is shown in Fig. 4.26.

Next we will use the BCJR algorithm to obtain the *a posteriori* probabilities $P\left(u_t = 0 \,\middle|\, \boldsymbol{r}_{[0,6)}\right)$, $t = 0, 1, 2, 3$. First, we calculate the probabilities $p_t(\sigma, \sigma')$ and $p_t^{(1)}(\sigma, \sigma')$. Then we calculate the forward metrics $\mu_t(\sigma')$ according to (4.179) and (4.180) and write the values next to the corresponding states in Fig. 4.27 (**BCJR2**).

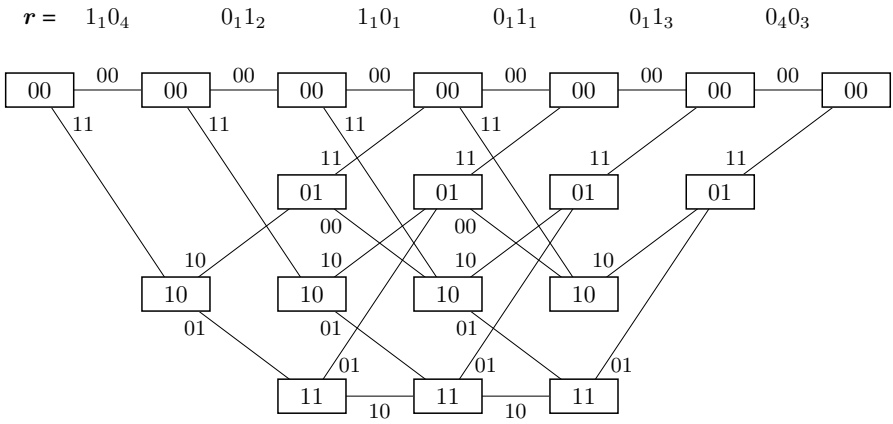


Figure 4.26 The trellis used in Example 4.7.

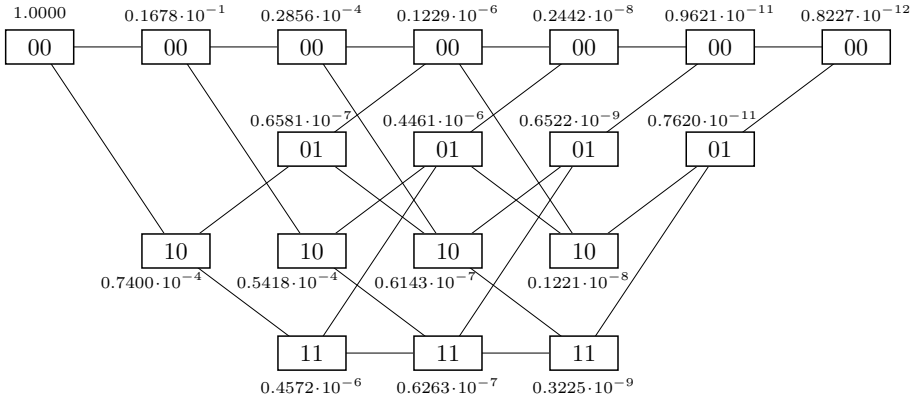


Figure 4.27 The forward metrics $\mu_t(\sigma')$ are written next to the corresponding states.

The backward metrics $\tilde{\mu}_t(\sigma)$ are calculated according to (4.182) and (4.183) and their values are written next to the corresponding states in Fig. 4.28 (**BCJR3**).

We have now reached step **BCJR4** and calculate $\gamma_i^{(1)}$ according to (4.185). Then we obtain

$$\gamma_0^{(1)} = 0.8069 \cdot 10^{-12}$$
$$\gamma_1^{(1)} = 0.1747 \cdot 10^{-12}$$

$$\gamma_2^{(1)} = 0.1854 \cdot 10^{-12}$$
$$\gamma_3^{(1)} = 0.8226 \cdot 10^{-12}$$

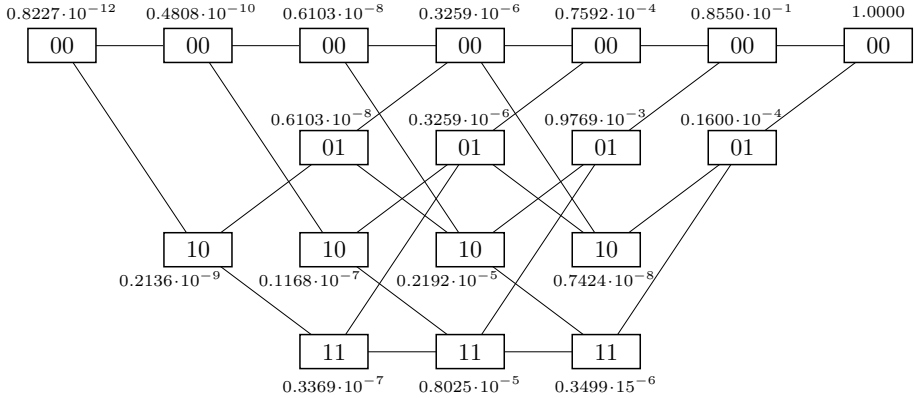


Figure 4.28 The backward metrics $\tilde{\mu}_t(\sigma)$ are written next to the corresponding states.

corresponding to the four information symbols and (to check our calculations)

$$\gamma_4^{(1)} = 0.8227 \cdot 10^{-12}$$
$$\gamma_5^{(1)} = 0.8227 \cdot 10^{-12}$$

corresponding to the two dummy zeros in the tail. Since $\mu_6(0) = 0.8227 \cdot 10^{-12}$ (see Fig. 4.27), we have the *a posteriori* probabilities

$$P\left(u_0^{(1)} = 0 \,\middle|\, \boldsymbol{r}_{[0,6)}\right) = 0.9808$$
$$P\left(u_1^{(1)} = 0 \,\middle|\, \boldsymbol{r}_{[0,6)}\right) = 0.2124$$

$$P\left(u_2^{(1)} = 0 \,\middle|\, \boldsymbol{r}_{[0,6)}\right) = 0.2254$$
$$P\left(u_3^{(1)} = 0 \,\middle|\, \boldsymbol{r}_{[0,6)}\right) = 0.9999$$

and for the two dummy zeros, as expected,

$$P\left(u_4^{(1)} = 0 \,\middle|\, \boldsymbol{r}_{[0,6)}\right) = 1.0000$$
$$P\left(u_5^{(1)} = 0 \,\middle|\, \boldsymbol{r}_{[0,6)}\right) = 1.0000$$

Using (4.186), we obtain the decision for information symbols $\widehat{\boldsymbol{u}}_{[0,4)}^{(1)} = \widehat{u}_0^{(1)}\widehat{u}_1^{(1)}\widehat{u}_2^{(1)}\widehat{u}_3^{(1)} = 0110$. It is interesting to notice that the maximum-likelihood decision for the information sequence obtained by the Viterbi algorithm in Example 4.2 is the same.

In Fig. 4.29, we show the bit error probabilities when the BCJR algorithm is used to communicate over the AWGN channel. Simulation results are shown for

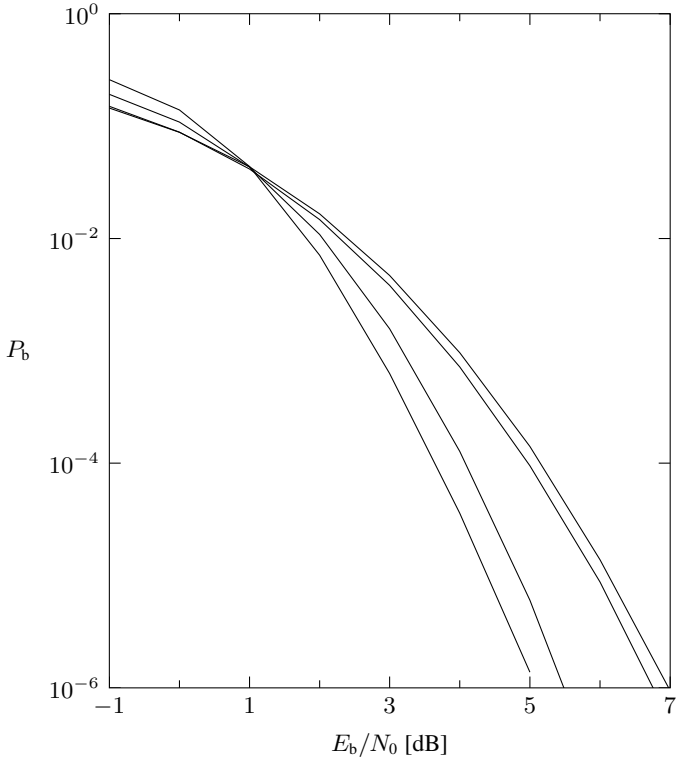


Figure 4.29 Bit error probabilities for $G_1(D) = (1 + D + D^2 \quad 1 + D^2)$ and $G_2(D) = (1 + D + D^2 + D^3 + D^6 \quad 1 + D^2 + D^3 + D^5 + D^6)$ for the BCJR algorithm and the AWGN channel. The curves from left to right are $G_2(D)$ with $L = 100$ information symbols, $G_2(D)$ with $L = 30$ information symbols, $G_1(D)$ with $L = 100$ information symbols, and $G_1(D)$ with $L = 30$ information symbols.

the two rate $R = 1/2$ encoding matrices $G_1(D) = (1 + D + D^2 \quad 1 + D^2)$ and $G_2(D) = (1 + D + D^2 + D^3 + D^6 \quad 1 + D^2 + D^3 + D^5 + D^6)$ for both $L = 30$ and $L = 100$ information symbols, followed by a tail of m dummy zeros.

Next we assume that the encoder is systematic, that is,

$$v_i^{(k)} = u_i^{(k)}, \quad 1 \le k \le b \tag{4.187}$$

From (4.173) we obtain

$$P\left(u_i^{(k)} = 1 \;\middle|\; \boldsymbol{r}_{[0,n+m)}\right) = 1 - \frac{\gamma_i^{(k)}}{\gamma} = \frac{\gamma - \gamma_i^{(k)}}{\gamma} \tag{4.188}$$

Let $z_i^{(k)}$ denote the log-likelihood ratio of the *a posteriori* probabilities for the kth information symbol in the ith b-tuple, that is,

$$z_i^{(k)} \stackrel{\text{def}}{=} \log \frac{P\left(u_i^{(k)} = 0 \;\middle|\; \boldsymbol{r}_{[0,n+m)}\right)}{P\left(u_i^{(k)} = 1 \;\middle|\; \boldsymbol{r}_{[0,n+m)}\right)} \tag{4.189}$$

Combining (4.173) and (4.188) yields

$$z_i^{(k)} = \log \frac{\gamma_i^{(k)}}{\gamma - \gamma_i^{(k)}} \tag{4.190}$$

Now we shall split $z_i^{(k)}$ into summands representing the intrinsic information about the information symbol $u_i^{(k)}$, the log-likelihood ratio of *a priori* probabilities of $u_i^{(k)}$, and the extrinsic log-likelihood ratio.

Let $\boldsymbol{r}_t^{(\not k)}$ denote the tth received c-tuple *except* its kth symbol $r_t^{(k)}$, that is,

$$\boldsymbol{r}_t^{(\not k)} = r_t^{(1)} r_t^{(2)} \ldots r_t^{(k-1)} r_t^{(k+1)} \ldots r_t^{(b)} \tag{4.191}$$

Since the channel is memoryless we can rewrite (4.170) as follows:

$$\begin{aligned}
p_i^{(k)}(\sigma, \sigma') &= P\left(\boldsymbol{r}_i \;\middle|\; \sigma_{i+1} = \sigma', \sigma_i = \sigma, u_i^{(k)} = 0\right) \\
&\quad \times P\left(\sigma_{i+1} = \sigma' \;\middle|\; \sigma_i = \sigma, u_i^{(k)} = 0\right) P(u_i^{(k)} = 0) \\
&= P\left(r_i^{(k)} \;\middle|\; \sigma_{i+1} = \sigma', \sigma_i = \sigma, u_i^{(k)} = 0\right) \\
&\quad \times P\left(\boldsymbol{r}_i^{(\not k)} \;\middle|\; \sigma_{i+1} = \sigma', \sigma_i = \sigma, u_i^{(k)} = 0\right) \\
&\quad \times P\left(\sigma_{i+1} = \sigma' \;\middle|\; \sigma_i = \sigma, u_i^{(k)} = 0\right) P\left(u_i^{(k)} = 0\right) \\
&= P\left(r_i^{(k)} \;\middle|\; u_i^{(k)} = 0\right) P\left(u_i^{(k)} = 0\right) \\
&\quad \times P\left(\boldsymbol{r}_i^{(\not k)} \;\middle|\; \sigma_{i+1} = \sigma', \sigma_i = \sigma, u_i^{(k)} = 0\right) \\
&\quad \times P\left(\sigma_{i+1} = \sigma' \;\middle|\; \sigma_i = \sigma, u_i^{(k)} = 0\right)
\end{aligned} \tag{4.192}$$

where the last equality follows from the systematicity of the encoder ($r_i^{(k)}$ depends only on $u_i^{(k)}$ and not on the state transition).

Let us introduce

$$\begin{aligned}
p_i^{(\not k)}(\sigma, \sigma') &\stackrel{\text{def}}{=} P\left(\boldsymbol{r}_i^{(\not k)}, \sigma_{i+1} = \sigma' \;\middle|\; \sigma_i = \sigma, u_i^{(k)} = 0\right) \\
&= P\left(\boldsymbol{r}_i^{(\not k)} \;\middle|\; \sigma_{i+1} = \sigma', \sigma_i = \sigma, u_i^{(k)} = 0\right) \\
&\quad \times P\left(\sigma_{i+1} = \sigma' \;\middle|\; \sigma_i = \sigma, u_i^{(k)} = 0\right)
\end{aligned} \tag{4.193}$$

Then, combining (4.192) and (4.193) yields

$$p_i^{(k)}(\sigma,\sigma') = P\left(r_i^{(k)} \;\middle|\; u_i^{(k)}=0\right) P\left(u_i^{(k)}=0\right) p_i^{(\not k)}(\sigma,\sigma') \tag{4.194}$$

or, equivalently, in matrix form

$$P_i^{(k)} = P\left(r_i^{(k)} \;\middle|\; u_i^{(k)}=0\right) P\left(u_i^{(k)}=0\right) P_i^{(\not k)} \tag{4.195}$$

where

$$P_i^{(\not k)} = \left(p_i^{(\not k)}(\sigma,\sigma')\right)_{\sigma,\sigma'} \tag{4.196}$$

Analogously to (4.192) we have

$$\begin{aligned} p_i(\sigma,\sigma') - p_i^{(k)}(\sigma,\sigma') = {} & P\left(r_i^{(k)} \;\middle|\; u_i^{(k)}=1\right) P\left(u_i^{(k)}=1\right) \\ & \times P\left(\boldsymbol{r}_i^{(\not k)} \;\middle|\; \sigma_{i+1}=\sigma', \sigma_i=\sigma, u_i^{(k)}=1\right) \\ & \times P\left(\sigma_{i+1}=\sigma' \;\middle|\; \sigma_i=\sigma, u_i^{(k)}=1\right) \end{aligned} \tag{4.197}$$

or, equivalently, in matrix form

$$P_i - P_i^{(k)} = P\left(r_i^{(k)} \;\middle|\; u_i^{(k)}=1\right) P\left(u_i^{(k)}=1\right) P_i^{c(\not k)} \tag{4.198}$$

where the "complementary" matrix (corresponding to $u_i^{(k)}=1$)

$$P_i^{c(\not k)} = \left(p_i^{c(\not k)}(\sigma,\sigma')\right)_{\sigma,\sigma'} \tag{4.199}$$

and where

$$\begin{aligned} p_i^{c(\not k)}(\sigma,\sigma') \stackrel{\text{def}}{=} {} & P\left(\boldsymbol{r}_i^{(\not k)}, \sigma_{i+1}=\sigma' \;\middle|\; \sigma_i=\sigma, u_i^{(k)}=1\right) \\ = {} & P\left(\boldsymbol{r}_i^{(\not k)} \;\middle|\; \sigma_{i+1}=\sigma', \sigma_i=\sigma, u_i^{(k)}=1\right) \\ & \times P\left(\sigma_{i+1}=\sigma' \;\middle|\; \sigma_i=\sigma, u_i^{(k)}=1\right) \end{aligned} \tag{4.200}$$

Analogously to (4.173) we obtain the following expression for the log-likelihood ratio of the *a posteriori* probabilities for $u_i^{(k)}$:

$$\begin{aligned} z_i^{(k)} &= \log \frac{P\left(u_i^{(k)}=0 \;\middle|\; \boldsymbol{r}_{[0,n+m)}\right)}{P\left(u_i^{(k)}=1 \;\middle|\; \boldsymbol{r}_{[0,n+m)}\right)} \\ &= \log \frac{P\left(u_i^{(k)}=0\right)}{P\left(u_i^{(k)}=1\right)} \frac{P\left(r_i^{(k)} \;\middle|\; u_i^{(k)}=0\right)}{P\left(r_i^{(k)} \;\middle|\; u_i^{(k)}=1\right)} \frac{\gamma_i^{(\not k)}}{\gamma_i^{c(\not k)}} \end{aligned} \tag{4.201}$$

where $\gamma_i^{(\not k)}$ and $\gamma_i^{c(\not k)}$ are defined by

$$\boldsymbol{e}_0 P_0 P_1 \ldots P_{i-1} P_i^{(\not k)} P_{i+1} \ldots P_{n+m-1} \stackrel{\text{def}}{=} (\gamma_i^{(\not k)} 0 \ldots 0) \tag{4.202}$$

and

$$\boldsymbol{e}_0 P_0 P_1 \dots P_{i-1} P_i^{c(k)} P_{i+1} \dots P_{n+m-1} \stackrel{\text{def}}{=} (\gamma_i^{c(k)} 0 \dots 0) \tag{4.203}$$

respectively.

The first factor in (4.201) is the ratio of the *a priori* probabilities of the information symbol $u_i^{(k)}$, the second is the intrinsic likelihood ratio, and the third is the extrinsic likelihood ratio.

4.6 THE ONE-WAY ALGORITHM FOR APP DECODING

The BCJR algorithm is only applicable to terminated convolutional codes. In this section we consider the *one-way algorithm*, which is a forward-only algorithm for *a posteriori* decoding of convolutional codes. It uses a sliding window and can be considered as the APP decoding counterpart to the Viterbi algorithm with a finite back-search limit to be considered in Section 5.5.

The one-way algorithm calculates the *a posteriori* probability for $u_i^{(k)} = 0$ given that the receiver has reached depth $i+\tau$, i.e., based on the received sequence $\boldsymbol{r}_{[0,i+\tau)}$. Hence, analogously to (4.163) we have

$$\begin{aligned} P\left(u_i^{(k)} = 0 \mid \boldsymbol{r}_{[0,i+\tau)}\right) &= \frac{P(\boldsymbol{r}_{[0,i+\tau)}, u_i^{(k)} = 0)}{P(\boldsymbol{r}_{[0,i+\tau)})} \\ &= \frac{\sum_{\boldsymbol{\sigma}_{[0,i+\tau]} \in \mathcal{S}_{[0,i+\tau]i}^{(k)}} P\left(\boldsymbol{r}_{[0,i+\tau)} \mid \boldsymbol{\sigma}_{[0,i+\tau]}\right) P(\boldsymbol{\sigma}_{[0,i+\tau]})}{\sum_{\boldsymbol{\sigma}_{[0,i+\tau]} \in \mathcal{S}_{[0,i+\tau]}} P\left(\boldsymbol{r}_{[0,i+\tau)} \mid \boldsymbol{\sigma}_{[0,i+\tau]}\right) P(\boldsymbol{\sigma}_{[0,i+\tau]})}, \quad 1 \le k \le b \end{aligned} \tag{4.204}$$

where $\mathcal{S}_{[0,i+\tau]}$ and $\mathcal{S}_{[0,i+\tau]i}^{(k)}$ are given by

$$\mathcal{S}_{[0,i+\tau]} \stackrel{\text{def}}{=} \left\{\boldsymbol{\sigma}_{[0,i+\tau]} = \sigma_0 \sigma_1 \dots \sigma_{i+\tau} \;\middle|\; \sigma_0 = 0\right\} \tag{4.205}$$

and

$$\mathcal{S}_{[0,i+\tau]}^{(k)} \stackrel{\text{def}}{=} \left\{\boldsymbol{\sigma}_{[0,i+\tau]} = \sigma_0 \sigma_1 \dots \sigma_{i+\tau} \;\middle|\; \sigma_0 = 0 \,\&\, \sigma_i \to \sigma_{i+1} \Rightarrow u_i^{(k)} = 0\right\} \tag{4.206}$$

respectively. Let

$$\gamma_{i+\tau} \stackrel{\text{def}}{=} \sum_{\sigma=0}^{2^\nu - 1} \alpha_{i+\tau}(\sigma) \tag{4.207}$$

where $\alpha_{i+\tau}(\sigma)$ is given by (4.174). Then it follows that $\gamma_{i+\tau}$ equals the denominator of (4.204).

Let

$$\begin{aligned} \boldsymbol{\alpha}_{i+\tau}^{(k)} &= \left(\alpha_{i+\tau}^{(k)}(0)\, \alpha_{i+\tau}^{(k)}(1) \dots \alpha_{i+\tau}^{(k)}(2^\nu - 1)\right) \\ &\stackrel{\text{def}}{=} \boldsymbol{e}_0 P_0 P_1 \dots P_{i-1} P_i^{(k)} P_{i+1} \dots P_{i+\tau-1} \end{aligned} \tag{4.208}$$

where P_i and $P_i^{(k)}$ are given by (4.164) and (4.169), respectively. Then let

$$\gamma_{i+\tau}^{(k)} \stackrel{\text{def}}{=} \sum_{\sigma=0}^{2^\nu-1} \alpha_{i+\tau}^{(k)}(\sigma) \tag{4.209}$$

We conclude that the numerator of (4.204) equals $\gamma_{i+\tau}^{(k)}$. Hence, we can rewrite (4.204) as

$$P\left(u_i^{(k)} = 0 \;\middle|\; \boldsymbol{r}_{[0,i+\tau)}\right) = \gamma_{i+\tau}^{(k)}/\gamma_{i+\tau} \tag{4.210}$$

Both $\gamma_{i+\tau}^{(k)}$ and $\gamma_{i+\tau}$ can be calculated recursively. First we consider $\gamma_{i+\tau}$. For $\gamma_{i+\tau}$ it follows immediately from (4.174) that

$$\begin{aligned} \boldsymbol{\alpha}_0 &= \boldsymbol{e}_0 \\ \boldsymbol{\alpha}_{i+\tau+1} &= \boldsymbol{\alpha}_{i+\tau} P_t \end{aligned} \tag{4.211}$$

which together with (4.207) yields $\gamma_{i+\tau+1}$.

In order to calculate $\gamma_{i+\tau}^{(k)}$ we introduce

$$\begin{aligned} \boldsymbol{\alpha}_{ij}^{(k)} &= \left(\alpha_{ij}^{(k)}(0)\, \alpha_{ij}^{(k)}(1) \ldots \alpha_{ij}^{(k)}(2^\nu - 1)\right) \\ &\stackrel{\text{def}}{=} \boldsymbol{e}_0 P_0 P_1 \ldots P_{i-1} P_i^{(k)} P_{i+1} \ldots P_{j-1}, \quad j - \tau < i \le j-1, \;\; 1 \le k \le b \end{aligned}$$

Let

$$A_{ij} = \begin{pmatrix} \boldsymbol{\alpha}_{ij}^{(1)} \\ \boldsymbol{\alpha}_{ij}^{(2)} \\ \vdots \\ \boldsymbol{\alpha}_{ij}^{(b)} \end{pmatrix}, \quad j - \tau < i \le j - 1 \tag{4.212}$$

be a $b \times 2^\nu$ matrix and let $\mathbb{A}_t$ be a $(b\tau + 1) \times 2^\nu$ matrix whose first τ entries are the matrices A_{it}, $t - \tau + 1 \le i \le t$, and the last entry is the vector $\boldsymbol{\alpha}_t$ given by (4.174), that is,

$$\mathbb{A}_t = \begin{pmatrix} A_{t-\tau+1,t} \\ A_{t-\tau+2,t} \\ \vdots \\ A_{t,t} \\ \boldsymbol{\alpha}_t \end{pmatrix} \tag{4.213}$$

It is easily shown that

$$\mathbb{A}_t P_t = \begin{pmatrix} A_{t-\tau+1,t+1} \\ A_{t-\tau+2,t+1} \\ \vdots \\ A_{t,t+1} \\ \boldsymbol{\alpha}_{t+1} \end{pmatrix} \tag{4.214}$$

If we delete the top matrix $A_{t-\tau+1,t+1}$ from $\mathbb{A}_t P_t$, shift all matrices $A_{t-i+1,t+1}$, $1 \le i < \tau$, up one position, and replace the matrix $A_{t,t+1}$ by the matrix

$$A_{t+1,t+1} = \begin{pmatrix} \boldsymbol{\alpha}_{t+1,t+1}^{(1)} \\ \boldsymbol{\alpha}_{t+1,t+1}^{(2)} \\ \vdots \\ \boldsymbol{\alpha}_{t+1,t+1}^{(b)} \end{pmatrix} = \begin{pmatrix} \boldsymbol{\alpha}_t P_t^{(1)} \\ \boldsymbol{\alpha}_t P_t^{(2)} \\ \vdots \\ \boldsymbol{\alpha}_t P_t^{(b)} \end{pmatrix} \tag{4.215}$$

then we obtain

$$\mathbb{A}_{t+1} = \begin{pmatrix} A_{t-\tau+2,t+1} \\ A_{t-\tau+3,t+1} \\ \vdots \\ A_{t+1,t+1} \\ \boldsymbol{\alpha}_{t+1} \end{pmatrix} \tag{4.216}$$

The rows of the deleted top matrix $A_{t-\tau+1,t+1}$ are the vectors $\boldsymbol{\alpha}_{t+1}^{(k)}$, $1 \le k \le b$, defined by (4.208). They are used to calculate the probabilities $P(u_{t-\tau+1}^{(k)} = 0, \boldsymbol{r}_{[0,t+\tau)})$ according to (4.210).

The sparseness of the matrices P_t and $P_t^{(k)}$ can be exploited to simplify the calculation of the elements of the matrix $\mathbb{A}_t$. Assign to each of 2^ν states at depth t of the trellis a $(b\tau+1)$-dimensional column vector metric

$$\begin{aligned}\boldsymbol{\mu}_t(\sigma) = (&\mu_{t-\tau+1,t}^{(1)}(\sigma)\, \mu_{t-\tau+1,t}^{(2)}(\sigma) \ldots \mu_{t-\tau+1,t}^{(b)}(\sigma)\, \mu_{t-\tau+2,t}^{(1)}(\sigma)\, \mu_{t-\tau+2,t}^{(2)}(\sigma) \\ &\ldots \mu_{t-\tau+2,t}^{(b)}(\sigma) \ldots \mu_{t,t}^{(1)}(\sigma)\, \mu_{t,t}^{(2)}(\sigma) \ldots \mu_{t,t}^{(b)}(\sigma)\, \mu_t(\sigma))^{\mathrm{T}},\ t = 0, 1, \ldots, 0 \le \sigma < 2^\nu\end{aligned}$$

such that

$$\boldsymbol{\mu}_0(\sigma) = \begin{cases} (00\ldots01)^{\mathrm{T}}, & \text{if } \sigma = 0 \\ (00\ldots00)^{\mathrm{T}}, & \text{otherwise} \end{cases} \tag{4.217}$$

For $t = 1, 2, \ldots$ we first calculate

$$\widehat{\boldsymbol{\mu}}_t(\sigma') = \sum_{\sigma=0}^{2^\nu-1} \boldsymbol{\mu}_{t-1}(\sigma) p_{t-1}(\sigma, \sigma') \tag{4.218}$$

then we exclude the first b components of (4.218) (for $t \ge \tau - 1$ they are used for calculating the *a posteriori* probabilities $P\left(u_{t-\tau}^{(k)} \,\middle|\, \boldsymbol{r}_{[0,t)}\right)$, $1 \le k \le b$), shift all components except the last one b positions up, and replace the following entries $b(\tau-1)+1$, $b(\tau-1)+2, \ldots$, and $b\tau$ by

$$\begin{pmatrix} \mu_{t,t}^{(1)}(\sigma') \\ \mu_{t,t}^{(2)}(\sigma') \\ \vdots \\ \mu_{t,t}^{(b)}(\sigma') \end{pmatrix} \stackrel{\text{def}}{=} \sum_{\sigma=0}^{2^\nu-1} \mu_{t-1}(\sigma) \begin{pmatrix} p_{t-1}^{(1)}(\sigma, \sigma') \\ p_{t-1}^{(2)}(\sigma, \sigma') \\ \vdots \\ p_{t-1}^{(b)}(\sigma, \sigma') \end{pmatrix} \tag{4.219}$$

Then we obtain the metric $\boldsymbol{\mu}_{t+1}(\sigma')$.

As mentioned above, the first b entries of the vectors $\widehat{\boldsymbol{\mu}}_t(\sigma')$, $0 \le \sigma' < 2^\nu$, viz.,

$$\widehat{\mu}^{(k)}_{t-\tau,t}(\sigma') = \sum_{\sigma=0}^{2^\nu-1} \mu^{(k)}_{t-\tau,t-1} p_{t-1}(\sigma,\sigma'), \quad 1 \le k \le b \tag{4.220}$$

are used together with $\mu_t(\sigma')$, $0 \le \sigma' < 2^\nu$, to calculate the *a posteriori* probability

$$P\left(u^{(k)}_{i-\tau} = 0 \mid \boldsymbol{r}_{[0,t)}\right) = \gamma^{(k)}_i/\gamma_t, \quad 1 \le k \le b \tag{4.221}$$

where

$$\gamma^{(k)}_t = \sum_{\sigma'=0}^{2^\nu-1} \widehat{\mu}^{(k)}_{t-\tau,t}(\sigma'), \quad 1 \le k \le b \tag{4.222}$$

and

$$\gamma_t = \sum_{\sigma=0}^{2^\nu-1} \mu_t(\sigma) \tag{4.223}$$

Finally, we obtain the *a posteriori* probabilities from (4.210).

The sliding window algorithm requires more memory than the Viterbi algorithm but less than the BCJR algorithm. The decoding delay is much less than that of the BCJR algorithm.

■ **EXAMPLE 4.13**

Consider the binary-input, 8-ary output DMC used in Example 4.2. Suppose that the encoding matrix $G(D) = (1 + D + D^2 \quad 1 + D^2)$ is used and assume that the *a priori* probabilities for the information symbols are $P(u_t = 0) = 2/3$, $t = 0, 1, \ldots$. Let $\boldsymbol{r} = 1_1 0_4\ 0_1 1_2\ 1_1 0_1\ 0_1 1_1\ 0_1 1_3\ 0_4 0_3\ 0_4 1_1\ 1_2 0_1\ 0_1 1_1\ 1_2 0_2\ 0_4 0_1\ 0_3 0_2\ 1_1 0_1\ 0_1 1_1 \ldots$ be the received sequence.

For the sliding window algorithm we obtain the following results:

$\tau = 6$

i	$\gamma^{(1)}_{i+\tau}$	$\gamma_{i+\tau}$	$P\left(u^{(1)}_i = 0 \mid \boldsymbol{r}_{[0,i+\tau)}\right)$	$\widehat{u}_i$
0	$0.5376 \cdot 10^{-12}$	$0.5458 \cdot 10^{-12}$	0.9849	0
1	$0.3166 \cdot 10^{-14}$	$0.9089 \cdot 10^{-14}$	0.3483	1
2	$0.1342 \cdot 10^{-16}$	$0.5249 \cdot 10^{-16}$	0.2557	1
3	$0.2137 \cdot 10^{-18}$	$0.3780 \cdot 10^{-18}$	0.5652	0
4	$0.2299 \cdot 10^{-20}$	$0.2599 \cdot 10^{-20}$	0.8844	0
5	$0.3132 \cdot 10^{-22}$	$0.5014 \cdot 10^{-22}$	0.6246	0
6	$0.3536 \cdot 10^{-24}$	$0.6478 \cdot 10^{-24}$	0.5458	0
7	$0.1108 \cdot 10^{-26}$	$0.4668 \cdot 10^{-26}$	0.2374	0
8	$0.1841 \cdot 10^{-28}$	$0.3076 \cdot 10^{-28}$	0.5984	1

$\tau = 12$

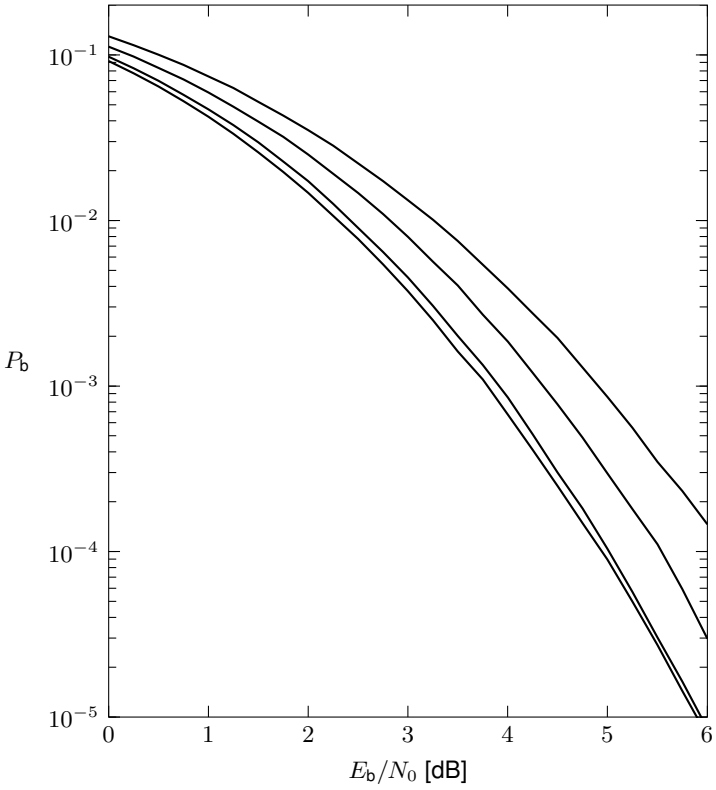


Figure 4.30 Bit error probabilities for $G(D) = (1 + D + D^2 \quad 1 + D^2)$ for the one-way algorithm and the AWGN channel. The curves from left to right are for $\tau = 4,\ 6,\ 10, 20$.

i	$\gamma_{i+\tau}^{(1)}$	$\gamma_{i+\tau}$	$P\left(u_i^{(1)} = 0 \mid \boldsymbol{r}_{[0,i+\tau)}\right)$	$\widehat{u}_i$
0	$0.6393 \cdot 10^{-24}$	$0.6478 \cdot 10^{-24}$	0.9868	0
1	$0.1944 \cdot 10^{-26}$	$0.4668 \cdot 10^{-26}$	0.4164	1
2	$0.6923 \cdot 10^{-29}$	$0.3076 \cdot 10^{-28}$	0.2251	1

In Figs. 4.30 and 4.31 we show the bit error probabilities for the AWGN channel when the sliding window APP decoding algorithm is used for maximum *a posteriori* probability (MAP) decoding for rate $R = 1/2$ convolutional codes encoded by the encoding matrices $G(D) = (1 + D + D^2 \quad 1 + D^2)$ and $G(D) = (1 + D + D^2 + D^3 + D^6 \quad 1 + D^2 + D^3 + D^5 + D^6)$ (Qualcomm's memory $m = 6$ encoder), respectively.

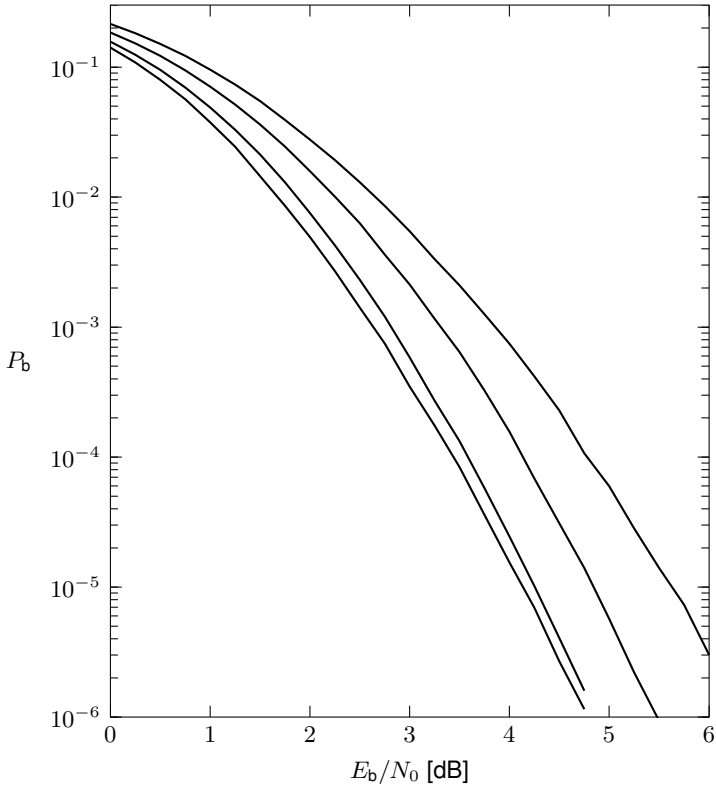


Figure 4.31 Bit error probabilities for $G(D) = (1 + D + D^2 + D^3 + D^6 \quad 1 + D^2 + D^3 + D^5 + D^6)$ for the one-way algorithm and the AWGN channel. The curves from left to right are for $\tau = 12, 18, 30, 60$.

4.7 A SIMPLE UPPER BOUND ON THE BIT ERROR PROBABILITY FOR EXTREMELY NOISY CHANNELS

When concatenated schemes (see Section 1.5) operate near the channel capacity (cf. Chapter 9) their component encoders may operate above capacity. In this section, we shall investigate the decoding bit error probability of convolutional codes near and above the channel capacity.

In particular, we shall show that, at extremely low signal-to-noise ratios, a systematic feedback encoder results in a lower bit error probability than does a nonsystematic feedforward encoder for the same convolutional code. We showed in Section 4.2 that the free distance is the principal determiner of the burst error probability for large signal-to-noise ratios when maximum-likelihood decoding is used. Since the burst error probability, as well as the free distance, is a code property, the burst error probability is the same whether the convolutional code was encoded by a systematic

feedback encoder or by a nonsystematic feedforward encoder. The decoding bit error probability, however, depends on the encoder (cf. Section 4.4).

Consider a rate $R = b/c$ generator matrix $G(D)$ of a convolutional code with a right pseudoinverse $\widetilde{G^{-1}}(D)$. Let the number of *taps* in a right pseudoinverse be the total number of nonzero coefficients in the power series that are entries in the $c \times b$ matrix $\widetilde{G^{-1}}(D)$. Then we have the following:

Definition A *tap-minimal right pseudoinverse* $\widetilde{G^{-1}}_{\text{tm}}(D)$ of the generator matrix $G(D)$ is a right pseudoinverse of $G(D)$ with the minimum number of taps among all right pseudoinverses of $G(D)$.

■ **EXAMPLE 4.14**

Consider the rate $R = 1/2$, memory $m = 5$, convolutional encoding matrix $G(D) = (1 + D^2 + D^5 \quad 1 + D + D^2 + D^4 + D^5)$. It has the tap-minimal right pseudoinverse

$$\widetilde{G^{-1}}_{\text{tm}}(D) = \begin{pmatrix} 1 + D \\ D \end{pmatrix} \tag{4.224}$$

with three taps. It is easily verified that $G(D)\widetilde{G^{-1}}_{\text{tm}}(D) = 1$. Hence, we conclude that (4.224) is not only a right pseudoinverse but also a right inverse of $G(D)$. Moreover, since $G(D)$ is not systematic and since it clearly has no right inverse with only two taps; we conclude that (4.224) is tap-minimal.

Next we define the *pseudoinverse decoder* (π-decoder) for convolutional codes. Assume that we use a convolutional code $\mathcal{C}$ encoded by the generator matrix $G(D)$ for communication over the BSC with crossover probability ϵ. Our decoding technique is as simple as it gets: We feed the received sequence $\boldsymbol{r}$ directly to a tap-minimal right pseudoinverse of $G(D)$ whose output is the decoded information sequence. At a first glance this sounds too simple to be useful, but let us analyze this decoder.

Assume without loss of generality that we transmit the allzero sequence. Let $r_1, r_2, \ldots, r_{n_i}$ be a sequence of n_i received statistically independent random variables for which $P(r_j = 1) = 1 - P(r_j = 0) = \epsilon$. Let n_i^{odd} and n_i^{even} denote the largest odd and even integers, respectively, that are $\leq n_i$. Then the probability that an odd number of the random variables r_j are 1s is

$$\begin{aligned} P_i^{\text{odd}} = &\binom{n_i}{1}\epsilon^1(1-\epsilon)^{n_i-1} + \binom{n_i}{3}\epsilon^3(1-\epsilon)^{n_i-3} + \cdots \\ &+ \binom{n_i}{n_i^{\text{odd}}}\epsilon^{n_i^{\text{odd}}}(1-\epsilon)^{n_i-n_i^{\text{odd}}} \end{aligned} \tag{4.225}$$

and the probability that an even number of the random variables r_j are 1s is

$$P_i^{\text{even}} = \binom{n_i}{0}\epsilon^0(1-\epsilon)^{n_i} + \binom{n_i}{2}\epsilon^2(1-\epsilon)^{n_i-2} + \cdots + \binom{n_i}{n_i^{\text{even}}}\epsilon^{n_i^{\text{even}}}(1-\epsilon)^{n_i-n_i^{\text{even}}} \tag{4.226}$$

From (4.225) and (4.226) it follows that

$$P_i^{\text{odd}} + P_i^{\text{even}} = (\epsilon + (1-\epsilon))^{n_i} = 1 \tag{4.227}$$

$$-P_i^{\text{odd}} + P_i^{\text{even}} = (-\epsilon + (1-\epsilon))^{n_i} = (1-2\epsilon)^{n_i} \tag{4.228}$$

Solving for P_i^{odd} yields

$$P_i^{\text{odd}} = \frac{1}{2}\left(1 - (1-2\epsilon)^{n_i}\right) \tag{4.229}$$

From this result we conclude that the *exact bit error probability* using the π-decoder is

$$P_{\text{b}} = \frac{1}{b}\sum_{i=1}^{b}\frac{1}{2}\left(1 - (1-2\epsilon)^{n_i}\right) \tag{4.230}$$

where n_i is the number of taps for the ith output of the right pseudoinverse. Clearly, the righthand side of (4.230) is an upper bound on the decoding bit error probability with BCJR decoding. We call it the π-*bound*. For crossover probabilities $\epsilon < 0.5$, the bound given by the right-hand side of (4.230) suggests that systematic encoders, which have the fewest taps, namely b, in their tap-minimal right pseudoinverse, give lower decoding bit error probability than nonsystematic ones. In Fig. 4.32 we compare the π-bound with the simulated bit error probability for the BCJR decoder for various generator matrices.[10] We conclude that for extremely noisy BSCs the π-bound is very tight and on such channels the superiority of systematic encoding is confirmed.

Consider the *binary erasure channel (BEC)*, which is shown in Fig. 1.22. When we transmit over the BEC, the binary input symbol is received correctly with probability $1-\delta$ and erased, that is, received as the symbol Δ, with probability δ. In order to use our π-decoder we assume that either a zero or a one are assigned randomly with equal probability to the erased symbols in the channel output sequence. This binary sequence, in which 1s occur with probability $\frac{1}{2}\delta$, is then fed into the π-decoder, which yields the bit error probability

$$P_{\text{b}} = \frac{1}{b}\sum_{i=1}^{b}\frac{1}{2}(1 - (1-\delta)^{n_i}) \tag{4.231}$$

[10]To obtain a compact notation for the generator matrices, we use the octal notation illustrated by the following memory $m = 6$ polynomial: $1 + D + D^4 + D^5 + D^6$ with coefficients (110 011 1) which are collected in groups of three starting from the *left* yielding 634. (If $1 + m$ is not a multiple of three, padding is used at the *right-hand* side.)

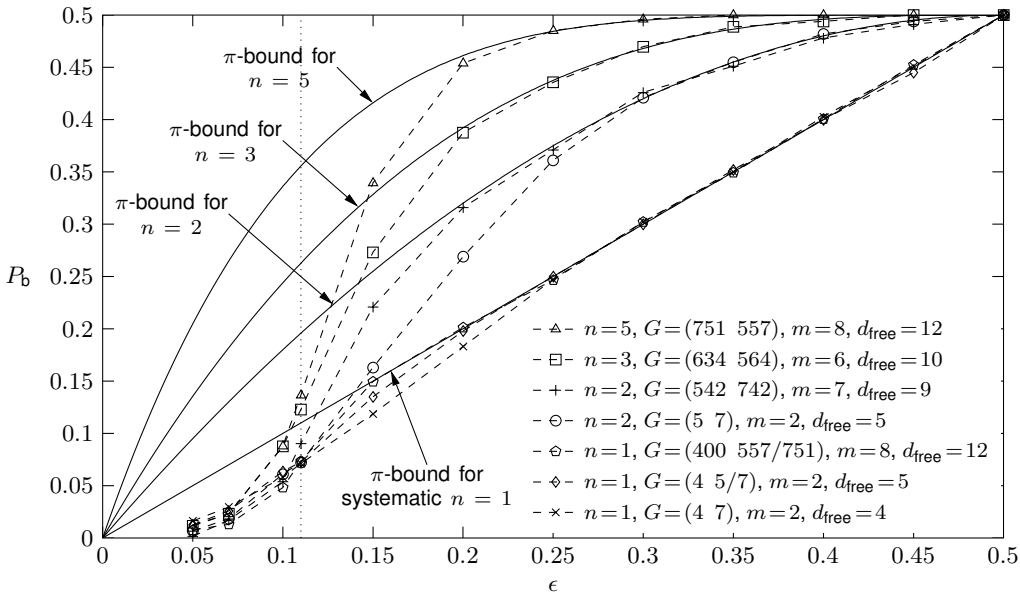


Figure 4.32 π-Bounds and simulated BCJR-decoding bit error probability on a BSC of various rate $R = 1/2$ convolutional encoders.

Again this P_b is an upper bound on the minimal bit error probability decoding.

Ancheta's linear source coding bound for the binary symmetric source (BSS)[11] states that the minimum rate required to achieve a bit error probability P_b in the source reconstruction for *linear* source coding of a BSS is

$$R_\mathrm{L}(P_\mathrm{b}) = 1 - 2P_\mathrm{b} \text{ binary digits/source letter} \tag{4.232}$$

Moreover, when used as a linear source encoder with n and r input and output symbols, respectively, any $n \times r$ binary matrix of rank r achieves this bound if and only if it has $n - r$ allzero rows [Anc77].

Using Ancheta's bound for linear source coding, we can show (see Problem 4.26) that the minimum decoding bit error probability that can be achieved with rate R *linear* encoding for a BEC is

$$P_\mathrm{b} = (1 - C/R)/2 \tag{4.233}$$

where $C = 1 - \delta$ is the channel capacity of the BEC.

In Fig. 4.33 we show the π-bounds and simulated decoding bit error probabilities for BEC for various generator matrices together with the lower bound (4.233). Also

[11] A BSS is a source that outputs statistically independent equiprobable binary digits.

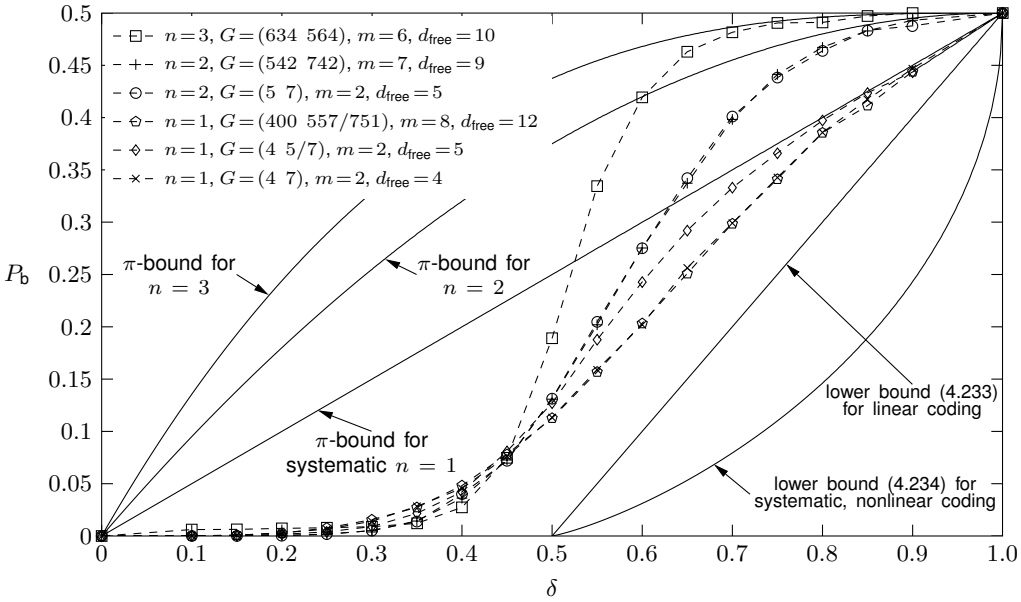


Figure 4.33 π-Bounds and simulated BCJR-decoding bit error probability on a BEC of various rate $R = 1/2$ convolutional encoders.

for extremely noisy BECs, the π-bound is very tight and again on such channels the superiority of systematic encoding is confirmed.

The best decoding bit error probability P_{b} obtainable with linear coding on a BSC with crossover probability ϵ $(0 \leq \epsilon \leq 1/2)$ is *at least* as great as that obtainable with linear coding on BEC with erasure probability $\delta = 2\epsilon$ (see Problem 4.27).

Finally, we remark that Shamai et al. [SVZ98] have given a general formulation for the minimum code rate required to approach a specified bit error probability, showing that nonsystematic encoding is inherently superior to systematic encoding. For systematic encoding on the BEC, their minimum code rate can be explicitly written as

$$R = \frac{C}{1 - (1 - C)h(\frac{P_{\text{b}}}{1-C})} \tag{4.234}$$

for all bit error probabilities P_{b} with $0 < P_{\text{b}} \leq (1 - C)/2$. This lower bound is also shown in Fig. 4.32. From the two lower bounds in Fig. 4.32 we conclude that it is *impossible* with *linear* encoding to obtain with systematic encoding the performance promised by (4.234). The inherent superiority of nonsystematic encoding over systematic encoding appears to be limited to the case where *nonlinear* codes are used, which is the atypical case in practice.

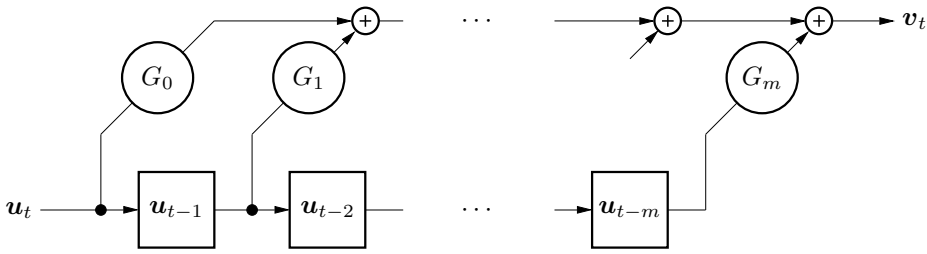


Figure 4.34 A polynomial generator matrix $G(D)$ realized in controller canonical form.

4.8 TAILBITING TRELLISES

So far we have used the zero-tail (ZT) method to terminate convolutional codes into block codes. If the trellis is short, the rate loss due to the terminating m dummy b-tuples (zeros when the generator matrix is polynomial and realized in controller canonical form) might not be acceptable. In this section, we will briefly describe a method to terminate convolutional codes into block codes without any rate loss. This method is called *tailbiting* and can be used to construct powerful trellis representations of block codes.

Consider a rate $R = b/c$ convolutional code $\mathcal{C}$ encoded by a polynomial generator matrix

$$G(D) = G_0 + G_1 D + \cdots + G_m D^m \tag{4.235}$$

of memory m realized in controller canonical form as shown in Fig. 4.34.

Let us truncate the causal codewords after L c-tuples, where $L > m$. Then, assuming that the encoder state is allzero at time $t = 0$, we have

$$\boldsymbol{v}_t = \boldsymbol{u}_t G_0 + \boldsymbol{u}_{t-1} G_1 + \cdots + \boldsymbol{u}_{t-m} G_m, \quad 0 \le t < L \tag{4.236}$$

or, equivalently,

$$\boldsymbol{v}_{[0,L)} = \boldsymbol{u}_{[0,L)} \boldsymbol{G}_L \tag{4.237}$$

where

$$\boldsymbol{G}_L = \begin{pmatrix} G_0 & G_1 & \dots & & G_m & & & \\ & G_0 & G_1 & \dots & & G_m & & \\ & & \ddots & \ddots & & & \ddots & \\ & & & G_0 & G_1 & \dots & & G_m \\ & & & & G_0 & G_1 & \dots & G_{m-1} \\ & & & & & \ddots & \ddots & \vdots \\ & & & & & & \ddots & G_1 \\ & & & & & & & G_0 \end{pmatrix} \tag{4.238}$$

is an $L \times L$ matrix and G_i, $0 \le i \le m$, are $b \times c$ matrices.

Let $\boldsymbol{\sigma}_L = (\boldsymbol{\sigma}_{L1}\, \boldsymbol{\sigma}_{L2} \ldots \boldsymbol{\sigma}_{Lm})$ denote the encoder state at time $t = L$. Clearly, we have

$$\boldsymbol{\sigma}_L = (\boldsymbol{u}_{L-1}\, \boldsymbol{u}_{L-2} \ldots \boldsymbol{u}_{L-m}) \tag{4.239}$$

Now we assume that we have the encoder state at time $t = 0$ equal to the encoder state at time L, that is, $\boldsymbol{\sigma}_0 = \boldsymbol{\sigma}_L$, and that the input is the allzero sequence. If we let $\boldsymbol{v}^{zi}_{[0,L)}$ denote the output sequence corresponding to initial encoder state $\boldsymbol{\sigma}_L$ and allzero input sequence, then $\boldsymbol{v}^{zi}_{[0,L)}$ can be expressed as

$$\boldsymbol{v}^{zi}_{[0,L)} = \boldsymbol{u}^{zi}_{[0,L)} \boldsymbol{G}^{zi}_L \tag{4.240}$$

where

$$\boldsymbol{u}^{zi}_{[0,L)} = (0\, 0 \ldots 0\, \boldsymbol{u}_{L-m}\, \boldsymbol{u}_{L-m+1}\, \ldots\, \boldsymbol{u}_{L-1}) \tag{4.241}$$

and

$$\boldsymbol{G}^{zi}_L = \begin{pmatrix} & & & & & & & \\ & & & & & & & \\ & & & & & & & \\ & & & & & & & \\ G_m & & & & & & & \\ G_{m-1} & G_m & & & & & & \\ \vdots & & \ddots & & & & & \\ G_1 & G_2 & \ldots & G_m & & & & \end{pmatrix} \tag{4.242}$$

is an $L \times L$ matrix. Hence, we conclude the codewords of the tailbiting representation of the block code $\mathcal{B}_{\text{tb}}$ obtained from the convolutional code $\mathcal{C}$ encoded by generator matrix $G(D)$ by using the reciprocal of the last m input b-tuples as the initial encoder state can be compactly written as

$$\boldsymbol{v}_{[0,L)} = \boldsymbol{u}_{[0,L)} \boldsymbol{G}^{\text{tb}}_L \tag{4.243}$$

where

$$\begin{aligned} \boldsymbol{G}^{\text{tb}}_L &= \boldsymbol{G}_L + \boldsymbol{G}^{zi}_L \\ &= \begin{pmatrix} G_0 & G_1 & \ldots & & G_m & & & \\ & G_0 & G_1 & \ldots & & G_m & & \\ & & \ddots & \ddots & & & \ddots & \\ & & & G_0 & G_1 & \ldots & & G_m \\ G_m & & & & G_0 & G_1 & \ldots & G_{m-1} \\ G_{m-1} & G_m & & & & \ddots & \ddots & \vdots \\ \vdots & & \ddots & & & & \ddots & G_1 \\ G_1 & G_2 & \ldots & G_m & & & & G_0 \end{pmatrix} \end{aligned} \tag{4.244}$$

is the $L \times L$ generator matrix for the tailbiting representation of the block code $\mathcal{B}_{\text{tb}}$.

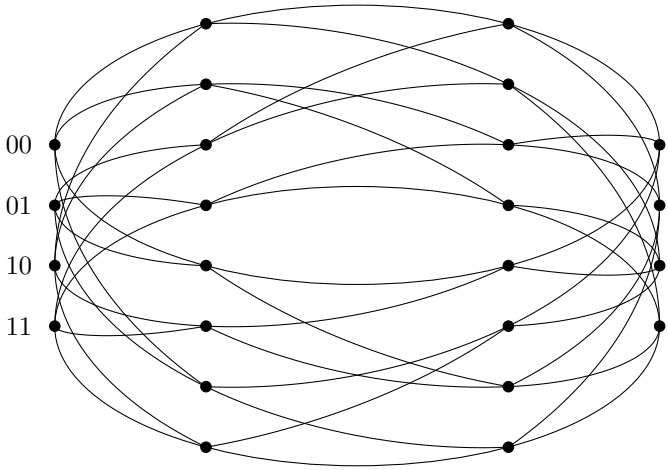


Figure 4.35 Circular trellis of length $L = 6$ for a 4-state tailbiting representation of a block code.

Since we require that we have the same state at the beginning as at the end, we can use a *circular trellis* for the tailbiting representation of a block code. In Fig. 4.35 we show as an example a circular trellis of length $L = 6$ for a 4-state tailbiting representation.

A circular trellis of length L corresponds to a total of $K = bL$ information symbols, c code symbols per branch, block length $N = Lc$, 2^b branches per trellis node; the number of codewords is

$$M = 2^K = 2^{bL} \tag{4.245}$$

and its rate is

$$R = K/N = b/c \tag{4.246}$$

The block code is the union of 2^{bm} subsets corresponding to the paths that go through each of the 2^{bm} states at time 0. These are 2^{bm} cosets of the $(N, K - bm)$ zero-tail terminated convolutional code corresponding to the paths that go through the allzero state at time 0.

It remains to find the minimum distance $d_{\min}$ of the block code. Since it is linear, it is enough to find the minimum weight of the nonzero codewords. The nonzero (?) block code trellis paths fall into two cases, which are illustrated in Fig. 4.36.

Case (i): The neighbor path touches the allzero path at least once; all paths considered are within the subset of paths leaving the allzero state. By the tailbiting condition they will sooner or later remerge with the allzero path. Finding the minimum weight among such paths is the same as finding the minimum weight path for a convolutional code. By the symmetry of the circular trellis, the behavior of paths out of one allzero node is the same as for all the other allzero nodes. The

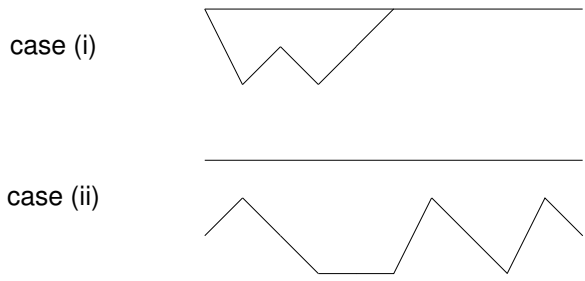


Figure 4.36 Two different types of paths.

minimum weight of the paths within this subset is the $(L-m-1)$th order row distance, but we call it the *intra minimum distance* of the tailbiting trellis representation of the block code and denote it d_{intra}. If L is long enough, d_{intra} is equal to the free distance of the corresponding convolutional code, but in general we have $d_{\text{intra}} \geq d_{\text{free}}$.

Case (ii): The neighbor path never touches the allzero path. This case is unique to block codes. For each nonzero starting state, we have to find the minimum weight path and then take the minimum over all outcomes. We call the minimum distance between the code subsets, that is, between the subset considered in case (i) and its cosets, the *inter minimum distance* of the tailbiting trellis representation of the block code and denote it d_{inter}.

The minimum distance of the block code is

$$d_{\min} = \min\{d_{\text{intra}}, d_{\text{inter}}\} \tag{4.247}$$

If the tailbiting circle is long enough, case (i) paths lead to the minimum distance, but for short circles case (ii) codewords may lead to the minimum weight path, that is, d_{inter} might be less than d_{intra}. Hence, short tailbiting representations of block codes have quite different optimal (largest distance) generators than do zero-tail terminated convolutional codes.

So far, we have constructed tailbiting representations of block codes from convolutional codes by first truncating the semi-infinite generator matrix $\boldsymbol{G}$ for the convolutional code,

$$\boldsymbol{G} = \begin{pmatrix} G_0 & G_1 & \dots & G_m & & \\ & G_0 & G_1 & \dots & G_m & \\ & & \ddots & \ddots & & \ddots \end{pmatrix} \tag{4.248}$$

after L rows,

$$\begin{pmatrix} G_0 & G_1 & \dots & G_m & & & & & & \\ & G_0 & G_1 & \dots & G_m & & & & & \\ & & \ddots & \ddots & & \ddots & & & & \\ & & & G_0 & G_1 & \dots & G_m & & & \\ & & & & G_0 & & G_{m-1} & G_m & & \\ & & & & & \ddots & \vdots & & \ddots & \\ & & & & & & G_0 & G_1 & \dots & G_m \end{pmatrix} \tag{4.249}$$

which is an $L \times (L+m)$ matrix. Then we "wraparound" the part consisting of the last m columns and obtain $\boldsymbol{G}_L^{\text{tb}}$ as given in (4.244).

We can also construct convolutional codes by "unwrapping" a tailbiting generator matrix of the type $\boldsymbol{G}_L^{\text{tb}}$ and then extending the "unwrapped" matrix to a semi-infinite generator matrix for the corresponding convolutional code.

■ **EXAMPLE 4.15**

Consider the rate $R = 1/2$ convolutional code with encoding matrix $G(D) = (1+D^2+D^3 \quad 1+D)$ of memory $m = 3$ and with $d_{\text{free}} = 5$. By the "wraparound" technique, we obtain the following generator matrix for a tailbiting representation of the corresponding $(12, 6)$ block code:

$$\boldsymbol{G}_6^{\text{tb}} = \begin{pmatrix} 11 & 01 & 10 & 10 & 00 & 00 \\ 00 & 11 & 01 & 10 & 10 & 00 \\ 00 & 00 & 11 & 01 & 10 & 10 \\ 10 & 00 & 00 & 11 & 01 & 10 \\ 10 & 10 & 00 & 00 & 11 & 01 \\ 01 & 10 & 10 & 00 & 00 & 11 \end{pmatrix} \tag{4.250}$$

It can be verified that $d_{\text{intra}} = d_{\text{free}} = 5$, $d_{\text{inter}} = 4$, and $d_{\min} = 4$. (Adding rows 3, 4, 5, and 6 gives the codeword 01 00 01 10 00 10 of Hamming weight 4 which upper-bounds $d_{\min}$.) By increasing the number of information symbols by two to $K = 8$, we obtain a $(16, 8)$ block code with $d_{\min} = d_{\text{free}} = 5$.

The $(24, 12)$ extended Golay code $\mathcal{B}_{24}$ with $d_{\min} = 8$ is a quite remarkable block code that has often been used as a benchmark in the studies of code structure and decoding algorithms. Surprisingly enough, there exists a rate $R = 1/2$ tailbiting representation of $\mathcal{B}_{24}$ that requires only 16 states [CFV99]. It has the following

generator matrix:

$$\boldsymbol{G}_{12}^{\text{tb}} = \begin{pmatrix} 11 & 01 & 11 & 01 & 11 & 00 & 00 & 00 & 00 & 00 & 00 & 00 \\ 00 & 11 & 11 & 10 & 01 & 11 & 00 & 00 & 00 & 00 & 00 & 00 \\ 00 & 00 & 11 & 01 & 10 & 11 & 11 & 00 & 00 & 00 & 00 & 00 \\ 00 & 00 & 00 & 11 & 01 & 11 & 01 & 11 & 00 & 00 & 00 & 00 \\ 00 & 00 & 00 & 00 & 11 & 01 & 11 & 01 & 11 & 00 & 00 & 00 \\ 00 & 00 & 00 & 00 & 00 & 11 & 11 & 10 & 01 & 11 & 00 & 00 \\ 00 & 00 & 00 & 00 & 00 & 00 & 11 & 01 & 10 & 11 & 11 & 00 \\ 00 & 00 & 00 & 00 & 00 & 00 & 00 & 11 & 01 & 11 & 01 & 11 \\ 11 & 00 & 00 & 00 & 00 & 00 & 00 & 00 & 11 & 01 & 11 & 01 \\ 01 & 11 & 00 & 00 & 00 & 00 & 00 & 00 & 00 & 11 & 11 & 10 \\ 10 & 11 & 11 & 00 & 00 & 00 & 00 & 00 & 00 & 00 & 11 & 01 \\ 01 & 11 & 01 & 11 & 00 & 00 & 00 & 00 & 00 & 00 & 00 & 11 \end{pmatrix} \tag{4.251}$$

We notice that if we regard $\boldsymbol{G}_{12}^{\text{tb}}$ as a rate $R = 1/2$ generator matrix, the corresponding generator matrix for the convolutional code is time varying with period $T = 4$ and of memory $m = 4$. That is, we need a $2^{bm} = 2^{1\cdot 4} = 16$-state trellis. However, $\boldsymbol{G}_{12}^{\text{tb}}$ can, for example, also be regarded as a rate $R = 4/8$ generator matrix with a corresponding time-invariant generator matrix of memory $m = 1$, that is, $2^{bm} = 2^{4\cdot 1} = 16$ states.

■ **EXAMPLE 4.16**

By "unwrapping" $\boldsymbol{G}_{12}^{\text{tb}}$ we obtain, for example, the rate $R = 1/2$ Golay convolutional code (GCC) with the time varying with period $T = 4$ encoding matrix of memory $m = 4$,

$$G_t^{\text{GCC}}(D) = \begin{cases} \begin{pmatrix} 1 + D^2 + D^4 & 1 + D + D^2 + D^3 + D^4 \end{pmatrix}, & t = 0, 4, \ldots \\ \begin{pmatrix} 1 + D + D^2 + D^4 & 1 + D + D^3 + D^4 \end{pmatrix}, & t = 1, 5, \ldots \\ \begin{pmatrix} 1 + D^2 + D^3 + D^4 & 1 + D + D^3 + D^4 \end{pmatrix}, & t = 2, 6, \ldots \\ \begin{pmatrix} 1 + D^2 + D^4 & 1 + D + D^2 + D^3 + D^4 \end{pmatrix}, & t = 3, 7, \ldots \end{cases}$$

(Notice that $G_t(D) = G_{t+3}(D)$.) Alternatively, we can obtain the $R = 4/8$ Golay convolutional code with the time-invariant encoding matrix of memory $m = 1$,

$$G^{\text{GCC}}(D) = \begin{pmatrix} 1 + D & 0 & 1 & 0 & 1 + D & 1 & 1 & 1 \\ 0 & 1 + D & 1 & 1 & D & 1 + D & 1 & 0 \\ D & D & 1 + D & 0 & 0 & D & 1 + D & 1 \\ 0 & D & 0 & 1 + D & D & D & D & 1 + D \end{pmatrix}$$

The rate $R = 4/8$ Golay convolutional code has $d_{\text{free}} = 8$ and the following path weight enumerator:

$$\begin{aligned} T(W) &= \frac{W^8\left(49 - 20W^4 - 168W^8 + 434W^{12} - 560W^{16} + 448W^{20} - 224W^{24} + 64W^{28} - 8W^{32}\right)}{1 - 28W^4 - 17W^8 + 118W^{12} - 204W^{16} + 204W^{20} - 128W^{24} + 48W^{28} - 8W^{32}} \\ &= 49W^8 + 1352W^{12} + 38521W^{16} + 1096224W^{20} + \cdots \end{aligned} \tag{4.252}$$

When we consider the GCC as a time varying rate $R = 1/2$ convolutional code it is reasonable to average the spectra for the four different phases. With this convention we have

$$T(W) = 12.25W^8 + 338W^{12} + 9455.25W^{16} + 264376W^{20} + \cdots \tag{4.253}$$

If we multiply the values n_8 and n_{12} in (4.253) by 4, then we obtain the numbers given by (4.252) but

$$4n_{16} = 4 \cdot 9455.25 = 37821 \tag{4.254}$$

which is 700 less than the corresponding number for the rate $R = 4/8$ GCC. The time varying rate $R = 1/2$ GCC has memory $m = 4$. Hence, its shortest detour from the allzero sequence is of length $(1 + m)c = (1 + 4)2 = 10$ code symbols. The rate $R = 4/8$ GCC has memory $m = 1$ and, then, a shortest detour of $(1+m)c = (1+1)8 = 16$. The information sequence $100001000000\ldots$ is encoded as 11 01 11 01 11 11 11 10 01 11 00 00 ... and 11011101 11111110 01110000 ..., respectively. The first code sequence corresponds to two consecutive detours, each of length 10 code symbols and of weight 8. The second code sequence corresponds to a single detour of length $(2 + 1)8 = 24$ code symbols and of weight 16. The path weight enumerator counts only single detours. Hence, this code sequence is counted when we consider the GCC as a rate $R = 4/8$ convolutional code but not when we consider it as a rate $R = 1/2$ convolutional code. This phenomenon explains the discrepancy in the numbers of weight 16 code sequences.

The GCC is *doubly-even*, that is, the weights of the codewords grow in steps of 4, and it can easily be verified that its codewords are self-orthogonal, that is, the Golay convolutional code is self-dual. A code that is both doubly-even and self-dual is called *Type II*.

■ **EXAMPLE 4.17**

The rate $R = 4/8$, time-invariant, memory $m = 1$ (16-state), convolutional code encoded by encoding matrix [JSW00]

$$G(D) = \begin{pmatrix} 0 & 0 & 1 & D & 1+D & 1+D & 1 & D \\ D & 0 & 0 & 1 & 1 & 1+D & 1+D & 1 \\ 1+D & D & 1+D & D & D & 0 & 1 & 0 \\ 1+D & 1+D & 0 & 0 & D & 1 & D & 1 \end{pmatrix} \tag{4.255}$$

is also Type II and has free distance $d_{\text{free}} = 8$, but its path weight enumerator

$$\begin{aligned} T(W) &= \frac{W^8\left(33-6W^4+8W^8-138W^{12}+260W^{16}-226W^{20}+112W^{24}-32W^{28}+4W^{32}\right)}{1-30W^4-W^8+20W^{12}+32W^{16}-74W^{20}+56W^{24}-22W^{28}+4W^{32}} \\ &= 33W^8 + 984W^{12} + 29561W^{16} + 886644W^{20} + \cdots \end{aligned} \tag{4.256}$$

is better than that of the GCC (4.252).

■ **EXAMPLE 4.18**

Through the "wraparound" technique, we can from the rate $R = 4/8$ encoding matrix in Example 4.12 obtain a tailbiting representation of a rate $R = 16/32$ block code with $d_{\min} = 8$:

$$G = \begin{pmatrix}
00101110 & 00011101 & 00000000 & 00000000 \\
00011111 & 10000110 & 00000000 & 00000000 \\
10100010 & 11111000 & 00000000 & 00000000 \\
11000101 & 11001010 & 00000000 & 00000000 \\
00000000 & 00101110 & 00011101 & 00000000 \\
00000000 & 00011111 & 10000110 & 00000000 \\
00000000 & 10100010 & 11111000 & 00000000 \\
00000000 & 11000101 & 11001010 & 00000000 \\
00000000 & 00000000 & 00101110 & 00011101 \\
00000000 & 00000000 & 00011111 & 10000110 \\
00000000 & 00000000 & 10100010 & 11111000 \\
00000000 & 00000000 & 11000101 & 11001010 \\
00011101 & 00000000 & 00000000 & 00101110 \\
10000110 & 00000000 & 00000000 & 00011111 \\
11111000 & 00000000 & 00000000 & 10100010 \\
11001010 & 00000000 & 00000000 & 11000101
\end{pmatrix} \tag{4.257}$$

The tailbiting termination does not work for certain encoders at certain tailbiting lengths. For encoders with feedback and for catastrophic encoders it can happen that we do not have a one-to-one mapping between the information sequences and the codewords.

Consider the systematic rate $R = 1/2$ encoding matrix

$$G(D) = \begin{pmatrix} 1 & \dfrac{1+D^2}{1+D+D^2} \end{pmatrix} \tag{4.258}$$

whose minimal realization is shown in Fig. 2.4. Assume that we start the encoder in the state $\boldsymbol{\sigma}_0 = (01)$ and feed it with zeros. Then we have the following sequence of encoder states: (01) (10) (11) $(01)\ldots$, that is, we return to the state (01) at every third time instant. The corresponding code sequence is $00\,01\,01\,00\ldots$. Clearly, when the tailbiting length L is a multiple of three, we have two codewords, one of them is a multiple of $00\,01\,01$ and the other is the allzero codeword, that both correspond to the allzero information sequence. Hence, tailbiting fails for the encoding matrix if the tailbiting length is a multiple of three.

Tailbiting will also fail if we have two information sequences corresponding to the same codeword. As an example of this situation we consider the catastrophic encoding matrix

$$G(D) = \begin{pmatrix} 1+D^2 & 1+D+D^2+D^3 \end{pmatrix} \tag{4.259}$$

If we start in state $\boldsymbol{\sigma}_0 = (111)$, then the codeword corresponding to the information sequence $\boldsymbol{u} = (1\,1\ldots1)$ is the allzero codeword. For a linear encoder the allzero

information sequence always corresponds to the allzero codeword; thus, we have two information sequences that correspond to the allzero codeword and we conclude that for the catastrophic encoder matrix (4.259) tailbiting fails for all tailbiting lengths.

The following theorem states when the tailbiting technique works.

Theorem 4.14 Consider a rate $R = b/c$ convolutional generator matrix $G(D)$. Let $\gamma_b(D)/q(D)$ denote the bth invariant factor of $G(D)$, where $q(D)$ is the least common multiple (lcm) of all denominators of $G(D)$. Tailbiting works for tailbiting length L if and only if both $\gamma_b(D)$ and $q(D)$ are relatively prime to the polynomial $1 + D^L$.

Proof: Tailbiting works if and only if there exists a one-to-one mapping between $\boldsymbol{u}(D)$ and $\boldsymbol{v}(D)$. Write $G(D)$ as its invariant factor decomposition

$$G(D) = A(D)\Gamma(D)B(D) \tag{4.260}$$

and let

$$\begin{aligned}\boldsymbol{v}'(D) &= (v'_1(D)\, v'_2(D) \ldots v'_c(D)) \\ &= \boldsymbol{v}(D)B^{-1}(D) \pmod{(1 + D^L)}\end{aligned} \tag{4.261}$$

and

$$\begin{aligned}\boldsymbol{u}'(D) &= (u'_1(D)\, u'_2(D) \ldots u'_b(D)) \\ &= \boldsymbol{u}(D)A^{-1}(D) \pmod{(1 + D^L)}\end{aligned} \tag{4.262}$$

Since $A(D)$ and $B(D)$ have unit determinants they are invertible modulo $1 + D^L$ and it follows that $\boldsymbol{v}'(D)$ and $\boldsymbol{u}'(D)$ are in one-to-one correspondence with $\boldsymbol{v}(D)$ and $\boldsymbol{u}(D)$, respectively. We have

$$\begin{aligned}\boldsymbol{v}'(D) = \boldsymbol{u}(D)G(D)B^{-1}(D) &= \boldsymbol{u}(D)A(D)\Gamma(D) \\ &= \boldsymbol{u}'(D)\Gamma(D) \pmod{(1 + D^L)}\end{aligned} \tag{4.263}$$

We exploit (2.60) and rewrite (4.263) as

$$v'_i(D) = \begin{cases} \dfrac{\gamma_i(D)}{q(D)} u'_i(D) \pmod{(1 + D^L)}, & 1 \le i \le b \\ 0, & b < i \le c \end{cases} \tag{4.264}$$

Hence, tailbiting works if and only if there exists a one-to-one mapping between $v'_i(D)$ and $u'_i(D)$ modulo $1 + D^L$ for all i, that is, if and only if $\gamma_i(D)$, $i \le i \le b$, and $q(D)$ are all invertible modulo $1 + D^L$. It is well known that $\gamma_i(D)$ and $q(D)$ are invertible if and only if $\gcd(\gamma_i(D), 1 + D^L) = 1$ and $\gcd(q(D), 1 + D^L) = 1$. If $\gcd(\gamma_b(D), 1+D^L) = 1$, it follows, since $\gamma_i(D) \mid \gamma_b(D)$, that $\gcd(\gamma_i(D), 1+D^L) = 1$, $1 \le i \le b$, and we conclude that tailbiting works if and only if both $q(D)$ and $\gamma_b(D)$ are relatively prime to $1 + D^L$. ∎

The following example shows that tailbiting can also work with catastrophic encoders.

■ **EXAMPLE 4.19**

Consider the rate $R = 2/3$ polynomial, catastrophic generator matrix $G(D)$ ($q(D) = 1$) and its Smith form decomposition

$$G(D) = \begin{pmatrix} 1 & 1+D & D \\ D^2 & D & 1+D+D^2+D^3 \end{pmatrix}$$
$$= \begin{pmatrix} 1 & 0 \\ D^2 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1+D+D^2 & 0 \end{pmatrix} \begin{pmatrix} 1 & 1+D & D \\ 0 & D & 1 \\ 0 & 1 & 0 \end{pmatrix} \quad (4.265)$$

If we consider tailbiting length $L = 5$, then, since $\gamma_2(D) = 1 + D + D^2$ and $\gcd(1 + D + D^2, 1 + D^5) = 1$, it follows from Theorem 4.14 that tailbiting will work although $G(D)$ is catastrophic. However, if we have tailbiting length $L = 6$, then, since $\gcd(1 + D + D^2, 1 + D^6) = 1 + D + D^2$, tailbiting fails.

Corollary 4.15 For noncatastrophic feedforward convolutional encoders tailbiting works for all tailbiting lengths L.

Proof: For a feedforward encoder we have $q(D) = 1$ and, hence, $\gcd(q(D), 1 + D^L) = 1$ for all L. From Corollary 2.11 we know that a polynomial generator matrix is noncatastrophic if and only if $\gamma_b(D) = D^s$ for some integer $s \geq 0$. Since $\gcd(D^s, 1 + D^L) = 1$ for all $L \geq 0$ and all $s \geq 0$ it follows from Theorem 4.14 that tailbiting works for all lengths L. ■

4.9 DECODING OF TAILBITING CODES

An obvious decoding method for tailbiting trellises is to use the Viterbi algorithm for each of the 2^ν subcodes where ν is the overall constraint length and a subcode consists of the codewords going through a given state. This procedure leads to 2^ν candidate codewords, and the best one is chosen as the decoder output.

A simpler but suboptimal decoding method is to initialize the metrics for all states at $t = 0$ to zero and then decode with the Viterbi algorithm going around the cyclic trellis a few times. Stop the Viterbi algorithm after a preset number of cycles n. Trace the winning path backward to determine whether it contains a cycle of length L that starts and ends in the same state. If such a cycle exists, it is chosen as the decoder's decision; otherwise the result is an erasure. Experiments have shown that a few decoding cycles often suffice [ZiC89]. However, the decoding performance may be significantly affected by "pseudo-codewords" corresponding to trellis paths of more than one cycle that do not pass through the same state at any integer multiple of the cycle length other than the pseudo-codeword length.

In Fig. 4.37 we compare the bit error probabilities when these two algorithms are used to decode the 16-state tailbiting representation of the extended Golay code when it is used to communicate over the BSC.

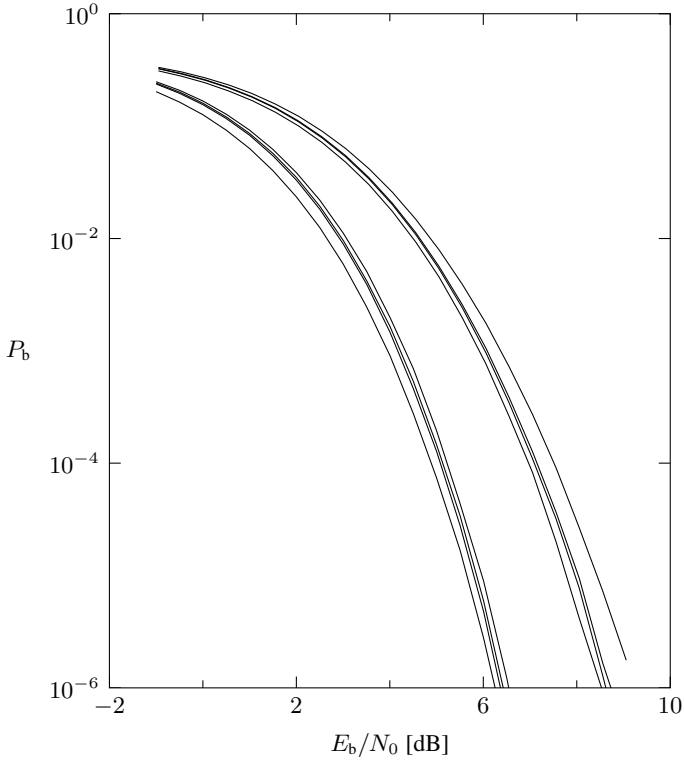


Figure 4.37 Comparison of the bit error probabilities for the 16-state extended Golay code. The bundles of curves correspond to soft decisions (left) and hard decisions (right). In each bundle the curves correspond to (from left to right) ML-decoding and suboptimal decoding with 10, 5, and 3 cycles, respectively.

Next we consider *a posteriori* probability (APP) decoding of rate $R = b/c$ tailbiting block codes of block length $N = Lc$ code symbols, where L is the block length in branches. We assume that the underlying rate $R = b/c$ convolutional code has memory m and overall constraint length ν. Let σ_t denote the encoder state at depth t where $\sigma_t \in \{0, 1, \ldots, 2^\nu - 1\}$.

We impose the tailbiting condition that only those paths that start and end at the same state are valid. Then as a counterpart to $\mathcal{S}_{[0,n+m]}$ in (4.161) we let $\mathcal{S}_{[0,L]}(\sigma)$ denote the set of all state sequences $\boldsymbol{\sigma}_{[0,L]}$ such that $\sigma_0 = \sigma_L = \sigma$, that is,

$$\mathcal{S}_{[0,L]}(\sigma) \stackrel{\text{def}}{=} \left\{ \boldsymbol{\sigma}_{[0,L]} = \sigma_0\, \sigma_1 \ldots \sigma_L \;\middle|\; \sigma_0 = \sigma_L = \sigma \right\}, \quad \sigma \in \{0, 1, \ldots, 2^\nu - 1\} \tag{4.266}$$

As a counterpart to $\mathcal{S}^{(k)}_{[0,n+m]i}$ in (4.162) we let $\mathcal{S}^{(k)}_{[0,L]i}(\sigma)$ denote the set of state sequences $\boldsymbol{\sigma}_{[0,L]}$ such that $\sigma_0 = \sigma_L = \sigma$ and the transition from state σ_i at depth i

to state σ_{i+1} at depth $i+1$ implies that $u_i^{(k)} = 0$, that is,

$$\mathcal{S}_{[0,L]i}^{(k)}(\sigma) \stackrel{\text{def}}{=} \left\{ \boldsymbol{\sigma}_{[0,L]} = \sigma_0\, \sigma_1 \ldots \sigma_L \;\middle|\; \sigma_0 = \sigma_L = \sigma \;\&\; \sigma_i \to \sigma_{i+1} \Rightarrow u_i^{(k)} = 0 \right\}, \quad \sigma \in \{0, 1, \ldots, 2^\nu - 1\} \tag{4.267}$$

To obtain the *a posteriori* probability that $u_i^{(k)} = 0$ given the received sequence $\boldsymbol{r}_{[0,L]}$, we have to sum both the numerator and denominator of (4.163) over all starting and ending states such that $\sigma_0 = \sigma_L = \sigma$. Then we obtain the following expression:

$$\begin{aligned} P\left(u_i^{(k)} = 0 \;\middle|\; \boldsymbol{r}_{[0,L)}\right) &= \frac{P(\boldsymbol{r}_{[0,L)}, u_i^{(k)} = 0)}{P(\boldsymbol{r}_{[0,L)})} \\ &= \frac{\sum_{\sigma=0}^{2^\nu-1} \sum_{\boldsymbol{\sigma}_{[0,L]} \in \mathcal{S}_{[0,L]i}^{(k)}(\sigma)} P\left(\boldsymbol{r}_{[0,L)} \mid \boldsymbol{\sigma}_{[0,L]}\right) P(\boldsymbol{\sigma}_{[0,L]})}{\sum_{\sigma=0}^{2^\nu-1} \sum_{\boldsymbol{\sigma}_{[0,L]} \in \mathcal{S}_{[0,L]}(\sigma)} P\left(\boldsymbol{r}_{[0,L)} \mid \boldsymbol{\sigma}_{[0,L]}\right) P(\boldsymbol{\sigma}_{[0,L]})}, \\ & \qquad 0 \le i < L, 1 \le k \le b \end{aligned} \tag{4.268}$$

The tailbiting counterparts to the vectors $\boldsymbol{\alpha}_i$ and $\boldsymbol{\beta}_i$ defined in (4.174) and (4.175), respectively, are $2^\nu \times 2^\nu$ dimensional matrices, viz.,

$$A_i = P_0 P_1 \ldots P_{i-1}, \quad 1 \le i \le L \tag{4.269}$$

and

$$B_i^{(k)} = P_{L-1}^{\mathrm{T}} P_{L-2}^{\mathrm{T}} \ldots P_{i+1}^{\mathrm{T}} \left(P_i^{(k)}\right)^{\mathrm{T}}, \quad 0 \le i < L, \quad 1 \le k \le b \tag{4.270}$$

where P_i and $P_i^{(k)}$ are given by (4.164) and (4.169), respectively. By convention

$$A_0 = I_{2^\nu} \tag{4.271}$$

where I_{2^ν} is the $2^\nu \times 2^\nu$ identity matrix.

As a tailbiting counterpart to $\gamma_i^{(k)}$ (cf. (4.172)), we have the $2^\nu \times 2^\nu$ dimensional matrix

$$\begin{aligned} C_i^{(k)} &= A_i \left(B_i^{(k)}\right)^{\mathrm{T}} \\ &= P_0 P_1 \ldots P_{i-1} \left(P_i^{(k)}\right) P_{i+1} \ldots P_{L-1}, \quad 0 \le i < L, 1 \le k \le b \end{aligned} \tag{4.272}$$

Because of the tailbiting condition, we are interested in the sum of the diagonal elements of the matrices A_L and $C_i^{(k)}$. Along the diagonals of A_L and $C_i^{(k)}$ we have

$$\sum_{\boldsymbol{\sigma}_{[0,L]} \in \mathcal{S}_{[0,L]}(\sigma)} P\left(\boldsymbol{r}_{[0,L)} \mid \boldsymbol{\sigma}_{[0,L]}\right) P(\boldsymbol{\sigma}_{[0,L]}), \quad 0 \le \sigma < 2^\nu$$

and

$$\sum_{\boldsymbol{\sigma}_{[0,L]} \in \mathcal{S}_{[0,L]i}^{(k)}(\sigma)} P\left(\boldsymbol{r}_{[0,L)} \mid \boldsymbol{\sigma}_{[0,L]}\right) P(\boldsymbol{\sigma}_{[0,L]}), \quad 0 \le \sigma < 2^\nu$$

respectively. Hence, we can rewrite (4.268) as

$$P\left(u_i^{(k)} = 0 \;\middle|\; \boldsymbol{r}_{[0,L)}\right) = \frac{\operatorname{Tr} C_i^{(k)}}{\operatorname{Tr} A_L}, \quad 0 \le i < L, \quad 1 \le k \le b \tag{4.273}$$

where $\operatorname{Tr} M = \sum_i m_{ii}$ is called the *trace* of the matrix $M = (m_{ij})_{ij}$.

Also in the tailbiting version of the BCJR algorithm we exploit the sparseness of the matrix P_t and compute the *a posteriori* probabilities $P\left(u_i^{(k)} = 0 \;\middle|\; \boldsymbol{r}_{[0,L)}\right)$ by trellis searches.

Let us introduce the forward (vector) metric

$$\boldsymbol{\mu}_t(\sigma') = \left(\mu_{t0}(\sigma')\, \mu_{t1}(\sigma') \ldots \mu_{t(2^\nu-1)}(\sigma')\right) \tag{4.274}$$

In the forward direction, we start at depth $t = 0$ with the (vector) metric

$$\boldsymbol{\mu}_0(\sigma) = \boldsymbol{e}_\sigma, \quad 0 \le \sigma < 2^\nu \tag{4.275}$$

Then for $t = 1, 2, \ldots, L$ we calculate the forward (vector) metric

$$\boldsymbol{\mu}_t(\sigma') = \sum_{\sigma=0}^{2^\nu-1} \boldsymbol{\mu}_{t-1}(\sigma) p_{t-1}(\sigma, \sigma'), \quad 0 \le \sigma' < 2^\nu \tag{4.276}$$

where $p_{t-1}(\sigma, \sigma')$ is given by (4.165). The vectors $\boldsymbol{\mu}_t(\sigma')$, $0 \le \sigma' < 2^\nu$, are stored at the corresponding states.

Analogously to the forward (vector) metric $\boldsymbol{\mu}_t(\sigma')$, we introduce the backward (vector) metrics

$$\tilde{\boldsymbol{\mu}}_t(\sigma) = \left(\tilde{\mu}_{t0}(\sigma)\, \tilde{\mu}_{t1}(\sigma) \ldots \tilde{\mu}_{t(2^\nu-1)}(\sigma)\right) \tag{4.277}$$

and

$$\tilde{\boldsymbol{\mu}}_t^{(k)}(\sigma) = \left(\tilde{\mu}_{t0}^{(k)}(\sigma)\, \tilde{\mu}_{t1}^{(k)}(\sigma) \ldots \tilde{\mu}_{t(2^\nu-1)}^{(k)}(\sigma)\right) \tag{4.278}$$

In the backward direction we start at depth L with the (vector) metric

$$\tilde{\boldsymbol{\mu}}_L(\sigma) = \boldsymbol{e}_\sigma, \quad 0 \le \sigma < 2^\nu \tag{4.279}$$

Then for $t = L-1, L-2, \ldots, 0$ we calculate the backward metrics

$$\tilde{\boldsymbol{\mu}}_t(\sigma) = \sum_{\sigma'=0}^{2^\nu-1} \tilde{\boldsymbol{\mu}}_{t+1}(\sigma') p_t(\sigma, \sigma'), \quad 0 \le \sigma < 2^\nu \tag{4.280}$$

and

$$\tilde{\boldsymbol{\mu}}_t^{(k)}(\sigma) = \sum_{\sigma'=0}^{2^\nu-1} \tilde{\boldsymbol{\mu}}_{t+1}(\sigma') p_t^{(k)}(\sigma, \sigma'), \quad 0 \le \sigma < 2^\nu, \quad 1 \le k \le b \tag{4.281}$$

where $p_t(\sigma, \sigma')$ and $p_t^{(k)}(\sigma, \sigma')$ are given by (4.165) and (4.170), respectively.

Finally, we have the (scalar) metrics

$$\mu_{i\sigma}^{(k)} \stackrel{\text{def}}{=} \sum_{\sigma'=0}^{2^\nu-1} \mu_{i\sigma}(\sigma')\tilde{\mu}_{i\sigma}^{(k)}(\sigma'), \quad 0 \le i < L,\, 0 \le \sigma < 2^\nu,\, 1 \le k \le b \tag{4.282}$$

Since

$$\sum_{\sigma=0}^{2^\nu-1} \mu_{L\sigma}(\sigma) = \operatorname{Tr} A_L \tag{4.283}$$

and

$$\sum_{\sigma=0}^{2^\nu-1} \mu_{i\sigma}^{(k)} = \operatorname{Tr} C_i^{(k)}, \quad 0 \le i < L, \quad 1 \le k \le b \tag{4.284}$$

we have the following BCJR algorithm for *a posteriori* probability decoding of tailbiting trellises.

Algorithm BCJRTB (BCJR algorithm for APP decoding of tailbiting trellises)

BCJRTB1. Initialize $\boldsymbol{\mu}_0(\sigma) = \tilde{\boldsymbol{\mu}}_L(\sigma) = \boldsymbol{e}_\sigma, \quad 0 \le \sigma < 2^\nu$.

BCJRTB2. For $t = 1, 2, \ldots, L$ calculate

$$\boldsymbol{\mu}_t(\sigma') = \sum_{\sigma=0}^{2^\nu-1} \boldsymbol{\mu}_{t-1}(\sigma)p_{t-1}(\sigma, \sigma'), \quad 0 \le \sigma' < 2^\nu$$

BCJRTB3. For $t = L-1, L-2, \ldots, 0$ calculate

$$\tilde{\boldsymbol{\mu}}_t(\sigma) = \sum_{\sigma'=0}^{2^\nu-1} \tilde{\boldsymbol{\mu}}_{t+1}(\sigma')p_t(\sigma, \sigma'), \quad 0 \le \sigma < 2^\nu$$

and

$$\tilde{\boldsymbol{\mu}}_t^{(k)}(\sigma) = \sum_{\sigma'=0}^{2^\nu-1} \tilde{\boldsymbol{\mu}}_{t+1}(\sigma')p_t^{(k)}(\sigma, \sigma'), \quad 0 \le \sigma < 2^\nu, \quad 1 \le k \le b$$

BCJRTB4. For $i = 0, 1, \ldots, L-1$ and $k = 1, \ldots, b$ calculate

$$\mu_{i\sigma}^{(k)} = \sum_{\sigma'=0}^{2^\nu-1} \mu_{i\sigma}(\sigma')\tilde{\mu}_{i\sigma}^{(k)}(\sigma'), \quad 0 \le i < L, \quad 0 \le \sigma < 2^\nu, \quad 1 \le k \le b$$

and output

$$P\left(u_i^{(k)} = 0 \,\middle|\, \boldsymbol{r}_{[0,L)}\right) = \frac{\sum_{\sigma=0}^{2^\nu-1} \mu_{i\sigma}^{(k)}}{\sum_{\sigma=0}^{2^\nu-1} \mu_{L\sigma}(\sigma)}$$

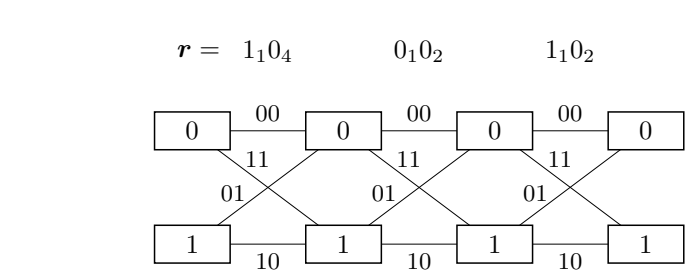


Figure 4.38 The tailbiting trellis used in Example 4.7.

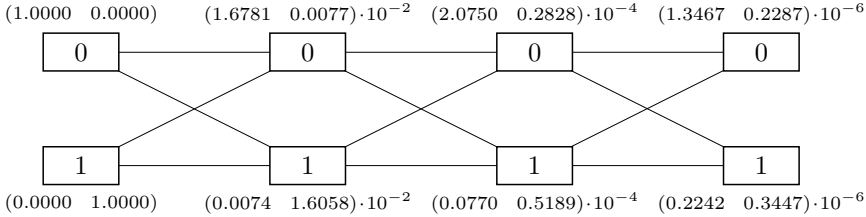


Figure 4.39 The forward (vector) metrics $\boldsymbol{\mu}_t(\sigma')$ are written next to the corresponding states.

■ EXAMPLE 4.20

Consider the same channel as in Example 4.2 and suppose that the encoding matrix $G(D) = (1 \quad 1+D)$ is used to encode a tailbiting representation of a block code of block length $N = 6$ code symbols. Assume that the *a priori* probabilities for the $K = 3$ information symbols and $P(u_t = 0) = 2/3$, $t = 0, 1, 2$. The trellis together with the received sequence is shown in Fig. 4.38.

We will use the BCJR tailbiting algorithm to obtain the *a posteriori* probabilities $P\left(u_t = 0 \,\middle|\, \boldsymbol{r}_{[0,3)}\right)$, $t = 0, 1, 2$.

First we calculate the probabilities $p_t(\sigma, \sigma')$ and $p_t^{(1)}(\sigma, \sigma')$. Then we calculate the forward (vector) metrics $\boldsymbol{\mu}_t(\sigma')$ according to (4.275) and (4.276) and write the values next to the corresponding states in Fig. 4.39 (**BCJRTB2**).

The backward (vector) metrics $\widetilde{\boldsymbol{\mu}}_t(\sigma)$ and $\widetilde{\boldsymbol{\mu}}_t^{(k)}(\sigma)$ are calculated according to (4.280) and (4.281), and their values are written next to the corresponding states in Figs. 4.40 and 4.41, respectively (**BCJRTB3**).

We have now reached step **BCJRTB4** and calculate $\mu_{i\sigma}^{(1)}$ according to (4.282). Then we obtain

$$\begin{aligned}
\mu_{00}^{(1)} + \mu_{01}^{(1)} &= 1.3467 \cdot 10^{-6} \\
\mu_{10}^{(1)} + \mu_{11}^{(1)} &= 1.3640 \cdot 10^{-6} \\
\mu_{20}^{(1)} + \mu_{21}^{(1)} &= 1.3467 \cdot 10^{-6}
\end{aligned}$$

corresponding to the three information symbols. Since

$$\mu_{L0}(0) + \mu_{L1}(1) = 1.6914 \cdot 10^{-6}$$

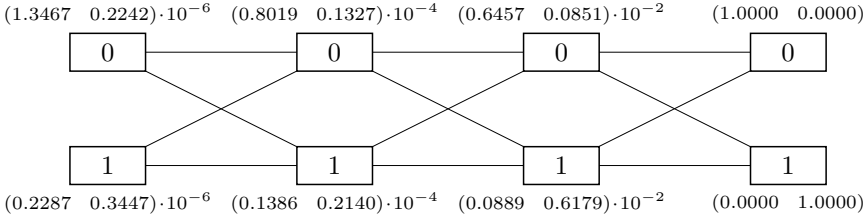


Figure 4.40 The backward (vector) metrics $\widetilde{\boldsymbol{\mu}}_t(\sigma)$ are written next to the corresponding states.

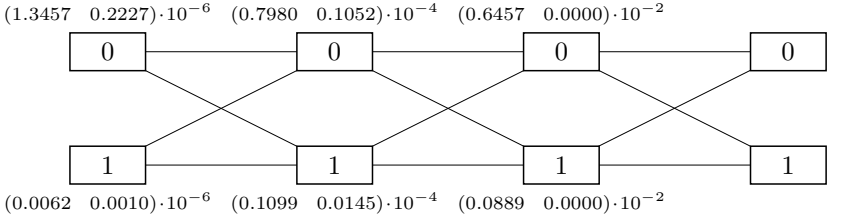


Figure 4.41 The backward (vector) metrics $\widetilde{\boldsymbol{\mu}}_t^{(1)}(\sigma)$ are written next to the corresponding states.

we have the *a posteriori* probabilities

$$P\left(u_0^{(1)} = 0 \,\middle|\, \boldsymbol{r}_{[0,3)}\right) = 0.7962$$
$$P\left(u_1^{(1)} = 0 \,\middle|\, \boldsymbol{r}_{[0,3)}\right) = 0.8065$$
$$P\left(u_2^{(1)} = 0 \,\middle|\, \boldsymbol{r}_{[0,3)}\right) = 0.7962$$

Hence, the maximum *a posteriori* decision for the information symbols is $\widehat{\boldsymbol{u}} = 000$.

In Fig. 4.42 we compare the bit error probabilities when the BCJR tailbiting algorithm is used to decode the 16-state and 64-state tailbiting representations of the extended Golay code when they are used to communicate over the BSC. The discrepancy between the two curves is due to the different mappings between the information symbols and the codewords for the two representations.

4.10 BEAST DECODING OF TAILBITING CODES

In this section, we will describe a Bidirectional Efficient Algorithm for Searching Trees (BEAST) [BHJ04, BJK04a]. It was developed for determining distance spectra of convolutional encoders (see Chapter 10) but turned out to be very powerful for maximum-likelihood decoding of block codes in general and tailbiting block codes in particular.

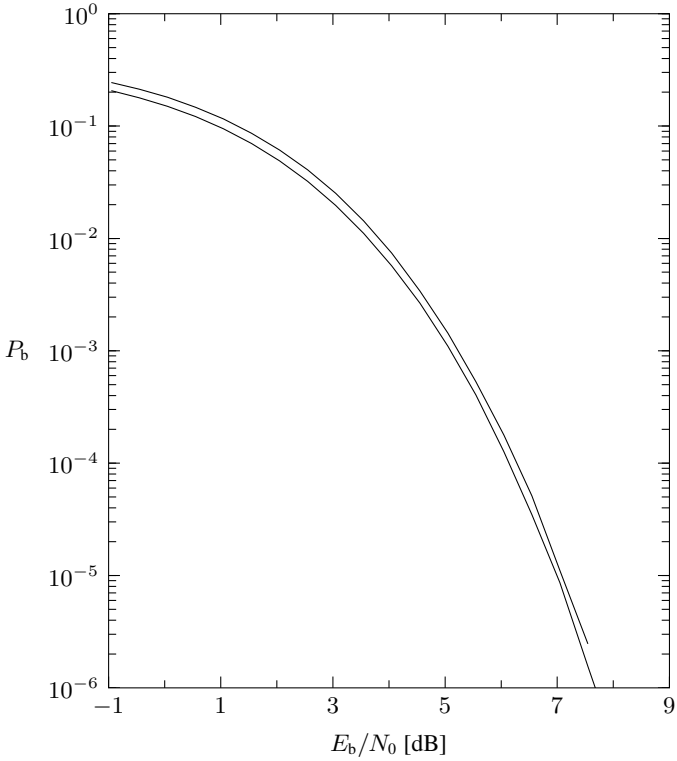


Figure 4.42 Bit error probabilities for the 16-state (left curve) and 64-state (right curve) tailbiting representations of the extended Golay code $\mathcal{B}_{24}$.

Consider as a simple example the rate $R = 1/2$, memory $m = 2$ convolutional encoding matrix with free distance $d_{\text{free}} = 4$,

$$G(D) = \begin{pmatrix} 1 & 1 + D + D^2 \end{pmatrix} \tag{4.285}$$

Its semi-infinite generator matrix G is

$$\boldsymbol{G} = \begin{pmatrix} 11 & 01 & 01 & & \\ & 11 & 01 & 01 & \\ & & \ddots & \ddots & \ddots \end{pmatrix} \tag{4.286}$$

By choosing tailbiting length $L = 4$ we obtain

$$G^{\text{eH}} = \begin{pmatrix} 11 & 01 & 01 & 00 \\ 00 & 11 & 01 & 01 \\ 01 & 00 & 11 & 01 \\ 01 & 01 & 00 & 11 \end{pmatrix} \tag{4.287}$$

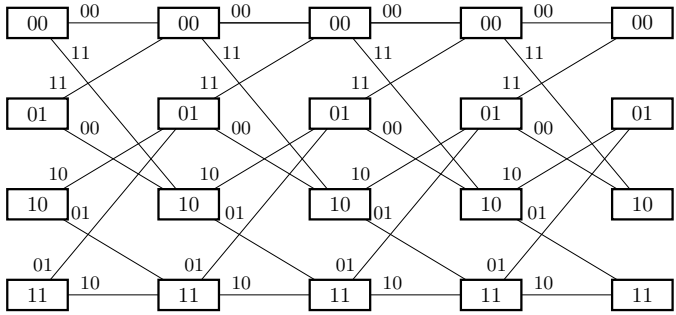


Figure 4.43 A sectionalized ($c = 2$) tailbiting trellis for the $(8, 4, 4)$ extended Hamming code.

which is a generator matrix for an $(8, 4, 4)$ extended Hamming code (cf. Problem 1.22). In Fig. 4.43 we show its *sectionalized (tailbiting) trellis*; we have $c = 2$ code symbols per branch, or, equivalently, per trellis section.

Next we shall aim at *unsectionalized (tailbiting) trellises*, that is, we shall consider trellises with only one code symbol per branch. Then we start by rewriting the generator matrix (4.287) as

$$G^{\mathrm{eH}} = \begin{pmatrix} 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 \end{pmatrix} \tag{4.288}$$

In Section 2.6 we defined the span of a Laurent series. Similarly, when we consider unsectionalized trellises and the row vectors of the corresponding generator matrices, we define the *span of a vector* $\boldsymbol{x}$ of finite length to be the interval $[\mathrm{start}(\boldsymbol{x}), \mathrm{end}(\boldsymbol{x})]$ from the index of the first nonzero component of $\boldsymbol{x}$, that is, $\mathrm{start}(\boldsymbol{x})$, to the index of the last nonzero component, that is, $\mathrm{end}(\boldsymbol{x})$, where $1 \le \mathrm{start}(\boldsymbol{x}) \le \mathrm{end}(\boldsymbol{x}) \le N$.

For tailbiting trellises the definition of the span is slightly more subtle since the time axis is circular. Thus, it is possible to have $\mathrm{start}(\boldsymbol{x}) > \mathrm{end}(\boldsymbol{x})$. The first nonzero component of $\boldsymbol{x}$ may be chosen as any nonzero component of $\boldsymbol{x}$. The index of the "last nonzero component" of $\boldsymbol{x}$ is the *least* integer $\mathrm{end}(\boldsymbol{x})$ such that $[\mathrm{start}(\boldsymbol{x}), \mathrm{end}(\boldsymbol{x}) + N](\mathrm{modulo} N)$ covers all nonzero components of $\boldsymbol{x}$.

We say that a vector $\boldsymbol{x}$ is *active* during the interval $[\mathrm{start}(\boldsymbol{x}), \mathrm{end}(\boldsymbol{x}) - 1] = [\mathrm{start}(\boldsymbol{x}), \mathrm{end}(\boldsymbol{x}))$, which we call the *active interval* of $\boldsymbol{x}$.

The *circular active interval* is

$$[\mathrm{start}(\boldsymbol{x}), \mathrm{end}(\boldsymbol{x})] \stackrel{\mathrm{def}}{=} [\mathrm{start}(\boldsymbol{x}), \mathrm{end}(\boldsymbol{x}) + N)(\mathrm{modulo}\ N) \tag{4.289}$$

and its length is

$$\begin{cases} \mathrm{end}(\boldsymbol{x}) - \mathrm{start}(\boldsymbol{x}) & \text{if } \mathrm{end}(\boldsymbol{x}) \ge \mathrm{start}(\boldsymbol{x}) \\ N - \mathrm{start}(\boldsymbol{x}) + \mathrm{end}(\boldsymbol{x}) & \text{otherwise} \end{cases} \tag{4.290}$$

Let μ_n be the *state complexity* of the (tailbiting) trellis for the (tailbiting) block code $\mathcal{B}$ at position n, that is,

$$\mu_n = \log_n |\mathcal{S}_n|, \quad 0 \le n \le L \tag{4.291}$$

where $|\mathcal{S}_n|$ denotes the cardinality of the state set $\mathcal{S}_n$ at position n. The $(N+1)$-tuple

$$\boldsymbol{\mu} = (\mu_0\, \mu_1 \ldots \mu_N) \tag{4.292}$$

is called the *state complexity profile* of the trellis. The *maximal state complexity* or *μ-state complexity* of the trellis is defined as [CFV99]

$$\mu_{\max} = \max_{n=0,1,\ldots,L} \{\mu_n\} \tag{4.293}$$

The maximal state complexity of a minimal trellis of any (N, K) linear block code is upper-bounded by the *Wolf bound* [Wol78]

$$\mu_{\max} \le \min\{K, N-K\} \tag{4.294}$$

If we sum the μ-state complexities over all positions we obtain the *π-state (product) complexity* [CFV99]

$$\pi = \sum_{n=0}^{L-1} \mu_n \tag{4.295}$$

Definition Let $\boldsymbol{g}_j, j = 0, 1, \ldots, K-1$, be the jth row of the generator matrix G of an $(N, K, d_{\min})$ block code. Then the matrix G is said to be in *minimal-span form* if, for every $j \ne k$,

$$\begin{aligned} \text{start}(\boldsymbol{g}_j) &\ne \text{start}(\boldsymbol{g}_k) \\ \text{end}(\boldsymbol{g}_j) &\ne \text{end}(\boldsymbol{g}_k) \end{aligned} \tag{4.296}$$

that is, there are not two rows in G that start or end in the same position.

The importance of the minimal-span form follows from the following theorem [For88, KsS95, McE96], which we give without a proof:

Theorem 4.16 Let the generator matrix G be in minimal-span form. Then the corresponding trellis has both minimal maximal state complexity and minimal product complexity over all conventional trellises for the given code.

A given generator matrix can be transformed into its minimal-span form by using Gaussian elimination in a two-step procedure. First, the matrix is reduced to row echelon form, which determines unique starting positions for every row. Then unique ending positions of the rows are obtained by the "cancellation above" procedure, starting from the last row.

Consider row $\boldsymbol{g}_j$. When it starts, its information bit u_j enters the encoder and influences the output code symbols during the active interval of the row $\boldsymbol{g}_j$. Thus,

the number of active rows at a certain position n determines the corresponding state complexity of the *minimal trellis*.

We summarize the most important properties of the minimal trellises:

- There are at most two branches arriving at or leaving each state; they carry the opposite code symbols.
- If a row starts (becomes active) at position n, $n = 0, 1, \ldots, N-1$, then all nodes at position n branch into two children nodes at position $n+1$.
- If a row ends at position n, $n = 0, 1, \ldots, N-1$, then pairs of nodes at position n merge into one child at position $n+1$.
- If no row starts or ends at position n, then every node at position n is connected to exactly one child (the same state) at position $n+1$.
- If a row starts and ends at the same position n (that is, its active interval is empty), then each state at position n is connected to its child (the same state) at position $n+1$ by two parallel branches, carrying code symbols 0 and 1.

The state complexity of a tailbiting trellis for a linear code $\mathcal{C}$ with generator matrix G at each position n is equal to the number of rows a_n in G that are active, that is, $\mu_n = a_n$ [CFV99, KoV98]. We have the following theorem [BJK02], which we give without a proof:

Theorem 4.17 For any linear code $\mathcal{C}$ defined by the generator matrix G corresponding to a linear unsectionalized tailbiting trellis with maximal state complexity $\mu_{\max}$ and product state complexity π, there exists an equivalent generator matrix G^{tsf} in tailbiting span form corresponding to a linear unsectionalized tailbiting trellis with maximal state complexity $\mu^{\text{tsf}} \leq \mu_{\max}$ and product state complexity $\pi^{\text{tsf}} \leq \pi$.

The minimal trellis yields the smallest state complexity when we assume a fixed ordering of the code symbols. By permuting the columns of the generator matrix we obtain an equivalent code which might have a reduced trellis state complexity. A minimal trellis with the smallest maximal state complexity $\mu_{\max}$ among all equivalent codes is called the *absolute minimal* trellis. Finding a permutation that yields the absolute minimal trellis is in general NP-hard.

Now we return to the generator matrix G^{eH} given in (4.288). By elementary row operations we can obtain (left as an exercise!) the equivalent minimal-span form

$$G^{\text{eH}}_{\text{MS}} = \begin{pmatrix} 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \end{pmatrix} \tag{4.297}$$

The complexity profile of its trellis is $\boldsymbol{\mu} = (012343210)$ with maximal state complexity $\mu_{\max} = 4$, or equivalently 16 states in the middle of the trellis.

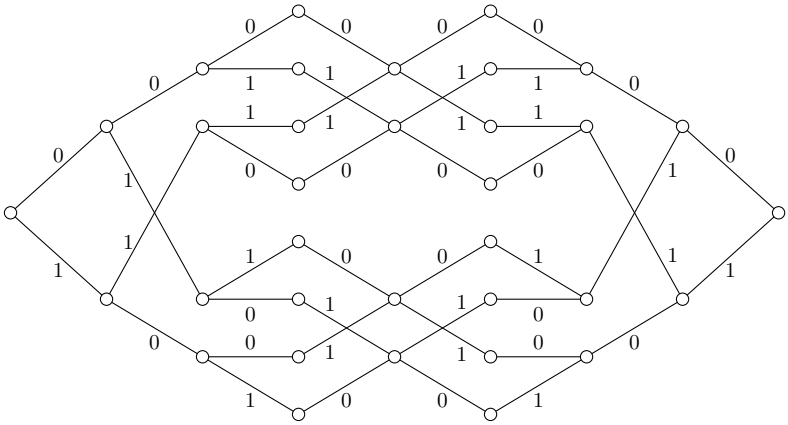


Figure 4.44 An absolute minimal trellis for the $(8, 4, 4)$ extended Hamming code.

By permuting the columns and perfoming elementary row operations of $G_{\rm MS}^{\rm eH}$ we can obtain (left as an exercise!) the *absolute minimal-span form* (for an equivalent code) [Mas78, For88, Mud88, BeB93]

$$G_{\rm AMS}^{\rm eH} = \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix} \tag{4.298}$$

Its complexity profile is $\boldsymbol{\mu} = (012323210)$ with maximal state complexity $\mu_{\max} = 3$. An absolute minimal trellis is shown in Fig. 4.44.

For an AWGN channel with BPSK modulation at the signal-to-noise ratio $E_{\rm s}/N_0$ the conditional density is

$$p\left(\boldsymbol{r} \mid \boldsymbol{v}\right) = \frac{1}{(\sqrt{\pi N_0})^N} e^{-\frac{1}{N_0}\|\boldsymbol{r}-\boldsymbol{x}\|^2} \tag{4.299}$$

where $\boldsymbol{x} = x_0 x_1 \ldots x_{N-1}$ is the signal sequence transmitted over the channel. The relation between the bipolar signal sequence $\boldsymbol{x}$ and the codeword $\boldsymbol{v}$ is

$$x_k = 2v_k - 1 = \begin{cases} -1, & v_k = 0 \\ +1, & v_k = 1 \end{cases} \tag{4.300}$$

The *squared Euclidean distances* between $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ is defined as

$$\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2 \stackrel{\rm def}{=} \sum_{k=0}^{N-1} (x_{ik} - x_{jk})^2 \tag{4.301}$$

which we denote as $d_{\rm E}^2(\boldsymbol{v}_i, \boldsymbol{v}_j)$. From (4.300) follows that

$$d_{\rm E}^2(\boldsymbol{v}_i, \boldsymbol{v}_j) = 4d_{\rm H}(\boldsymbol{v}_i, \boldsymbol{v}_j) \tag{4.302}$$

From (4.299) we conclude that for transmission over the AWGN channel with BPSK

$$\log p\left(\boldsymbol{r} \mid \boldsymbol{v}\right) = -N \log \sqrt{\pi N_0} - \frac{1}{N_0}\left(\|\boldsymbol{r}\|^2 + \|\boldsymbol{x}\|^2\right) + \frac{2\sqrt{E_\mathrm{s}}}{N_0} \sum_{i=0}^{N-1} |r_i| - \frac{4\sqrt{E_\mathrm{s}}}{N_0} \lambda(\boldsymbol{x}, \boldsymbol{r}) \tag{4.303}$$

where the *correlation discrepancy* $\lambda(\boldsymbol{x}, \boldsymbol{r})$ between the signal sequence $\boldsymbol{x}$ and the received sequence $\boldsymbol{r}$ is defined as

$$\lambda(\boldsymbol{x}, \boldsymbol{r}) = \sum_{i=0}^{N-1} \lambda(x_i, r_i) \tag{4.304}$$

where

$$\lambda(x_i, r_i) = \begin{cases} |r_i|, & \text{sign}(r_i) \neq \text{sign}(x_i) \\ 0, & \text{sign}(r_i) = \text{sign}(x_i) \end{cases} \tag{4.305}$$

Since BPSK is equi-energy signaling we can choose A and $f_k^{(\ell)}$ in (4.3) such that minimizing $\mu(\boldsymbol{v}, \boldsymbol{r})$ is equivalent to minimizing $\lambda(\boldsymbol{x}, \boldsymbol{r})$. In other words, an algorithm that minimizes $\lambda(\boldsymbol{x}, \boldsymbol{r})$ performs soft-decision ML decoding.

The correlation discrepancy is a generalization of the Hamming distance metric (4.9) for the BSC to soft-decision ML decoding. The *weighted Hamming distance* between the the binary N-tuples $\boldsymbol{a} = a_0 a_1 \ldots a_{N-1}$ and $\boldsymbol{b} = b_0 b_1 \ldots b_{N-1}$ is defined as [Kab91]

$$d_\mathrm{wH}(\boldsymbol{a}, \boldsymbol{b}) = \sum_{i=0}^{N-1} \begin{cases} \omega_i, & a_i \neq b_i \\ 0, & a_i = b_i \end{cases} \tag{4.306}$$

where $\omega_i \geq 0$ are any weights that can be reliabilities of symbols.

Consider the ith received symbol r_i and let h_i denote its hard decision, that is,

$$h_i \stackrel{\text{def}}{=} \begin{cases} 0, & r_i > 0 \\ 1, & r_i < 0 \end{cases} \tag{4.307}$$

The reliability of h_i is

$$L_\mathrm{h}(v_i) = \log \frac{p\left(r_i \mid x_i = +\sqrt{E_\mathrm{s}}\right)}{p\left(r_i \mid x_i = -\sqrt{E_\mathrm{s}}\right)} = \frac{4\sqrt{E_\mathrm{s}}}{N_0} r_i \tag{4.308}$$

Hence, the weighted Hamming distance $d_\mathrm{wH}\left(\boldsymbol{v}, \boldsymbol{h}\right)$ between the codeword $\boldsymbol{v}$ and the hard-decision received sequence $\boldsymbol{h} = h_0 h_1 \ldots h_{N-1}$, with reliability weights $\omega_i = |L_\mathrm{h}(v_i)|, i = 0, 1, \ldots, N-1$, is equal to the correlation discrepancy $\lambda(\boldsymbol{x}, \boldsymbol{r})$ scaled by the factor $4\sqrt{E_s}/N_0$), that is,

$$d_\mathrm{wH}\left(\boldsymbol{v}, \boldsymbol{h}\right) = \sum_{i=0}^{N-1} \begin{cases} |L_\mathrm{h}(v_i)|, & v_i \neq h_i \\ 0, & v_i = h_i \end{cases} \tag{4.309}$$

We are now well-prepared to introduce the Bidirectional Efficient Algorithm for Searching Trees (BEAST)

We can represent any block code of block length N by a trellis with N sections. In Fig. 4.44 we have shown the absolute minimal trellis for the $(8, 4, 4)$ extended Hamming code. The idea behind BEAST is that when decoding we grow a forward tree starting from the root at depth 0 and a backward tree starting from the toor at depth N. These two trees are embedded in the trellis as will be illustrated in Example 4.21.

Let ξ denote a node in the code tree and let ξ^{p} be its parent node. Every node is characterized by the following three parameters: state $\boldsymbol{\sigma}(\xi)$, depth $l(\xi)$, and distance $d(\xi)$. In the forward tree we use subscript F for the depth $l_{\mathrm{F}}(\xi)$ and the distance $d_{\mathrm{F}}(\xi)$. The depth $l_{\mathrm{F}}(\xi)$ is the length (in branches) and the distance $d_{\mathrm{F}}(\xi)$ is the weighted Hamming distance $d_{\mathrm{wH}}\left(\boldsymbol{r}_{[0,l_{\mathrm{F}}(\xi))}, \boldsymbol{v}_{[0,l_{\mathrm{F}}(\xi))}\right)$, where $\boldsymbol{v}_{[0,l_{\mathrm{F}}(\xi))}$ is the code sequence along the path $\xi_{\mathrm{root}} \to \xi$.

In the backward tree we use the subscript B and obtain the depth $l_{\mathrm{B}}(\xi)$ and the distance $d_{\mathrm{B}}(\xi)$, where $d_{\mathrm{B}}(\xi)$ is the weighted Hamming distance $d_{\mathrm{wH}}(\boldsymbol{r}_{[N-l_{\mathrm{B}}(\xi),N)}, \boldsymbol{v}_{[N-l_{\mathrm{B}}(\xi),N)})$, where $\boldsymbol{v}_{[N-l_{\mathrm{B}}(\xi))}$ is the code sequence along the path $\xi \to \xi_{\mathrm{root}}$. Finally, we have $\boldsymbol{\sigma}(\xi_{\mathrm{root}}) = \boldsymbol{\sigma}(\xi_{\mathrm{toor}}) = \mathbf{0}$.

BEAST needs a target metric threshold T; then it will find all codewords whose path metrics are below or at this threshold. If there are no paths below or at the threshold, we must increase T. If there are several paths below or at T, then we simply pick the one with the smallest metric; that path is our ML decision! Clearly, for BEAST to be efficient, how to choose the threshold is crucial. We shall return to this problem after having described the algorithm.

For hard-decision decoding, the path metric is the Hamming distance and the threshold increments are $\delta_i = 1$, all i. A good choice for the initial threshold is $T_1 = \lceil d_{\min}/2 \rceil$, where $d_{\min}$ is the minimum Hamming distance of the code.

Remark: This choice of T_1 is motivated by the fact that the *covering radius* of good codes is roughly half the minimum distance. The cover radius is the smallest integer ρ such that for any received sequence $\boldsymbol{r}$ there exists at least one codeword within Hamming distance ρ from $\boldsymbol{r}$. In other words, ρ is the smallest radius of Hamming spheres, centered at each codeword, such that they cover the entire space.

For the AWGN channel we use the weighted Hamming distance and the threshold increments $\delta_1, \delta + 2, \ldots, \delta_N$ are chosen to be $|L_{\mathrm{h}}(v_i)| = (4\sqrt{E}_{\mathrm{s}}/N_0)|r_i|$ sorted in increasing order. Clearly, we can eschew the factor $4\sqrt{E}_{\mathrm{s}}/N_0$ and simply use the sorted absolute values $|r_i|$. The threshold is initialized as

$$T_1 = \sum_{i=1}^{\lceil d_{\min}/2 \rceil} \delta_i \tag{4.310}$$

Algorithm B (BEAST for ML and List decoding)

B1. *Initialization:* Initialize the metric threshold T with a starting value $T = T_1$. Initialize the forward tree with the zero-state root, which has $w_{\mathrm{F}} = 0$, $\ell_{\mathrm{F}} = 0$.

Initialize the backward tree with the zero-state toor, which has $w_{\mathrm{B}} = 0$, $\ell_{\mathrm{B}} = 0$. Initialize the codeword list as $\mathcal{L} = \emptyset$.

B2. *Forward search:* Extend the forward tree to find the set of nodes

$$\mathcal{F} = \left\{ \xi \;\middle|\; w_{\mathrm{F}}(\xi) \geq T/2,\ w_{\mathrm{F}}(\xi^{\mathrm{p}}) < T/2,\ \ell_{\mathrm{F}}(\xi) \leq N - \min_{\xi' \in \mathcal{B}} \{\ell_{\mathrm{B}}(\xi')\} \right\}$$

Hence, every node at depth below $N - \min_{\xi' \in \mathcal{B}} \{\ell_{\mathrm{B}}(\xi')\}$, whose weight is below $T/2$, is extended.[12] Every extended node needs to be stored so that it remembers its parent. If a child node has reached or exceeded the target weight $T/2$, it is placed in the set $\mathcal{F}$ and not extended further. Hence, $\mathcal{F}$ contains the *leaves* of the partially explored forward tree. For each leaf in $\mathcal{F}$, its state, depth, weight, and parent node are stored.

B3. *Backward search:* Extend the backward tree to find the set of nodes

$$\mathcal{B} = \left\{ \xi \;\middle|\; w_{\mathrm{B}}(\xi) \leq T/2,\ \ell_{\mathrm{B}}(\xi) \leq N - \min_{\xi' \in \mathcal{F}} \{\ell_{\mathrm{F}}(\xi')\} \right\}$$

In the backward direction, every node whose weight does not exceed $T/2$ is extended and included in the set $\mathcal{B}$. Nodes with weights larger than $T/2$ are neither extended nor stored. Thus, the set $\mathcal{B}$ contains *all interior nodes* of the partially explored backward tree. For each node in $\mathcal{B}$, its state, depth, weight, and parent node are stored.

B4. *Matching:* Find all pairs of nodes $(\xi, \xi') \in \mathcal{F} \times \mathcal{B}$ such that $\boldsymbol{\sigma}(\xi) = \boldsymbol{\sigma}(\xi')$ and $\ell_{\mathrm{F}}(\xi) + \ell_{\mathrm{B}}(\xi') = N$. Each such match uniquely describes a codeword with the metric $w = w_{\mathrm{F}}(\xi) + w_{\mathrm{B}}(\xi')$. Discard all candidates that do not fulfill $w \leq T$ (without this condition the chosen codewords are not necessarily the globally best candidates; see Example 4.21 below).

B5. For each match with metric $w \leq T$, perform backtracing from the matching node to the root/toor to obtain the corresponding codeword. Add the so-found codeword to the list $\mathcal{L}$ in a sorted manner, according to its metric w. The best codeword has the smallest w. If the number of codewords on the updated list is $\geq L$, output the L best ones together with their metrics and stop[13] (for ML decoding, $L = 1$). Otherwise, increment i, increment the threshold $T \leftarrow T + \delta_i$, and go to step **B2**.

The following example illustrates the efficiency of BEAST.

[12] Since the nodes in the sets $\mathcal{F}$ and $\mathcal{B}$ are stored in the order of increasing depth, then $\min_{\xi' \in \mathcal{B}} \{\ell_{\mathrm{B}}(\xi')\}$ is simply the depth of the first node in $\mathcal{B}$. The same holds for the set $\mathcal{F}$.

[13] If the list of L best codewords is not unique (e.g., for the BSC), the BEAST outputs one of the possible lists.

■ **EXAMPLE 4.21**

Assume that the $(8, 4, 4)$ extended Hamming code with the generator matrix (4.298) is used to communicate over the AWGN channel with BPSK signalling and that the received sequence is

$$\boldsymbol{r} = (\,0.18\ -0.56\ \ 0.91\ \ 0.60\ -0.02\ \ 1.60\ \ 2.80\ \ 0.34\,)$$

Then the hard-decision received sequence is

$$\boldsymbol{h}_{\text{ch}} = (0\,1\,0\,0\,1\,0\,0\,0)$$

The correlation discrepancy will be used as a decoding metric. Hence, the threshold increments are

$$\{\delta_i\}_{i=1}^{N} = \{0.02,\ 0.18,\ 0.34,\ 0.56,\ 0.60,\ 0.91,\ 1.6,\ 2.8\} \tag{4.311}$$

and the initial threshold is $T_1 = \sum_{i=1}^{2} \delta_i = 0.02 + 0.18 = 0.2$. First, we grow a forward tree with the target metric $T_1/2 = 0.1$. The tree is shown in Fig. 4.45. The extended nodes are marked with unfilled (white) circles. The leaves of the forward tree, marked with filled (black) circles, are nodes whose metrics are $w_{\mathsf{F}}(\xi) \geq T_1/2 = 0.1$. Their states, weights, and depths are stored in the set $\mathcal{F}$:

$$\begin{aligned}
\mathcal{F}_{\boldsymbol{\sigma}} &= \{1000,\ 0000,\ 0100,\ 0110\} \\
\mathcal{F}_{w} &= \{0.18,\ 0.56,\ 0.91,\ 0.6\} \\
\mathcal{F}_{\ell} &= \{1,\ 2,\ 3,\ 4\}
\end{aligned}$$

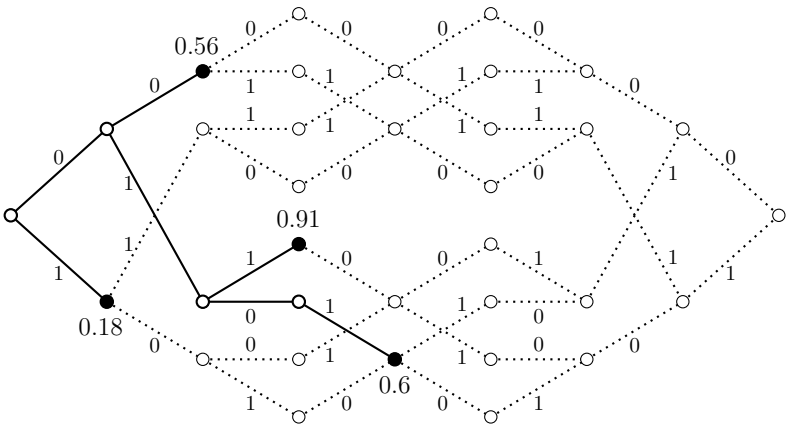


Figure 4.45 Forward tree for $T_1/2 = 0.1$: filled nodes are leaves of the partially explored tree whose metrics exceed $T_1/2$; their states, metrics, and depths are stored in the list $\mathcal{F}$. Interior nodes are visited, but not stored.

The next step is to grow a backward tree to find all paths with metrics $w_{\mathsf{B}} \leq T_1/2 = 0.1$. The backward tree is shown in Fig. 4.46. All visited nodes have weights $w_{\mathsf{B}}(\xi) \leq 0.1$ and are stored in the set $\mathcal{B}$:

$$\mathcal{B}_{\boldsymbol{\sigma}} = \{0000,\ 0000,\ 0000,\ 0000,\ 0000,\ 0000,\ 0000\}$$
$$\mathcal{B}_{w} = \{0.0,\ 0.0,\ 0.0,\ 0.0,\ 0.02,\ 0.02,\ 0.02\}$$
$$\mathcal{B}_{\ell} = \{0,\ 1,\ 2,\ 3,\ 4,\ 5,\ 6\}$$

Matching of the sets $\mathcal{F}$ and $\mathcal{B}$ is illustrated in Fig. 4.47, and it yields one matching node: $\mathcal{F}_{\boldsymbol{\sigma}} \cap \mathcal{B}_{\boldsymbol{\sigma}} = \{0000\}$ at depth $\ell_{\mathsf{F}} = 2$, with the total weight $w = 0.56 + 0.02 = 0.58 > T_1$. This match corresponds to the allzero codeword. Since its metric is larger than the current threshold, this needs not be the best codeword and, hence, we do not accept it as the solution. Instead, we increase the threshold and continue the search.

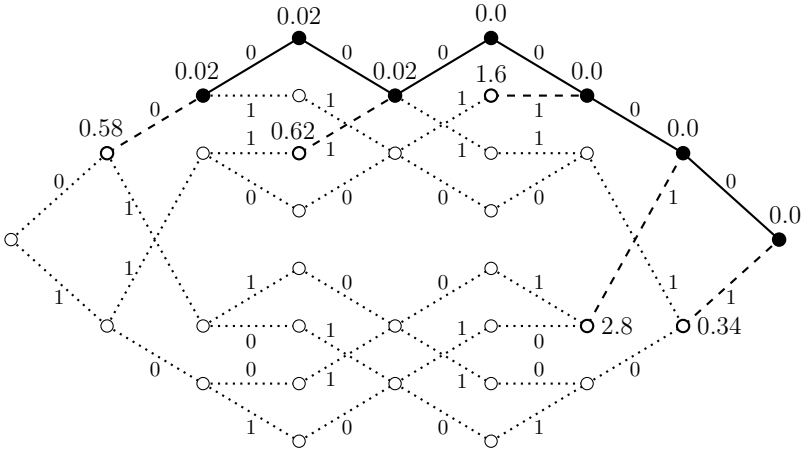


Figure 4.46 Backward tree for $T_1/2 = 0.1$: filled nodes are interior nodes of the partially explored tree, with metrics not larger than $T_1/2$. Their states, weights, and depths are stored in the set $\mathcal{B}$. The leaves of the backward tree (nodes connected via dashed branches to the interior nodes) are checked to have metrics exceeding $T_1/2$ and are neither extended nor stored.

In the next step, we increase the threshold by the next smallest absolute reliability value, $T_2 = T_1 + 0.34 = 0.54$, and we extend the forward tree from the previous step, using $T_2/2 = 0.27$ as the target metric value. Fig. 4.48 shows the obtained forward tree. The stored nodes are:

$$\mathcal{F}_{\boldsymbol{\sigma}} = \{0000,\ 1100,\ 1000,\ 0100,\ 0110,\ 0000,\ 0001\}$$
$$\mathcal{F}_{w} = \{0.56,\ 0.74,\ 1.09,\ 0.91,\ 0.6,\ 1.78,\ 3.0\}$$
$$\mathcal{F}_{\ell} = \{2,\ 2,\ 3,\ 3,\ 4,\ 6,\ 7\}$$

The backward tree with the target metric $w_{\mathsf{B}} \leq T_2/2 = 0.27$ is the same as in the previous step (Fig. 4.46), that is, no node can be extended. The matching of

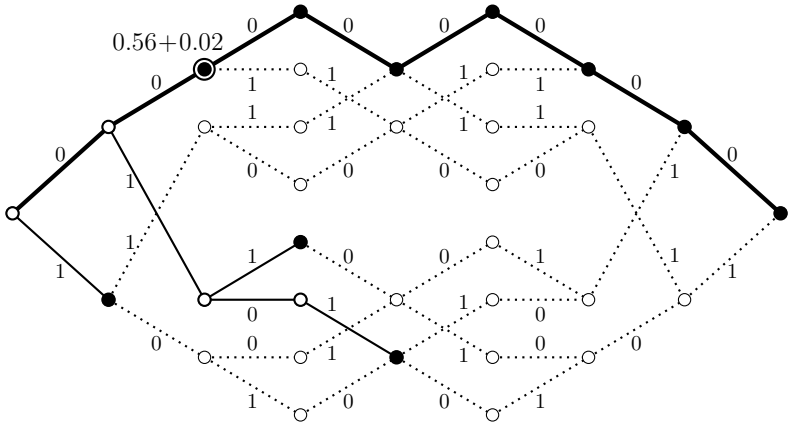


Figure 4.47 Matching the forward and the backward tree in the first iteration: there is one matching node (highlighted) corresponding to the allzero codeword. Its metric is above the current threshold and is thus not accepted.

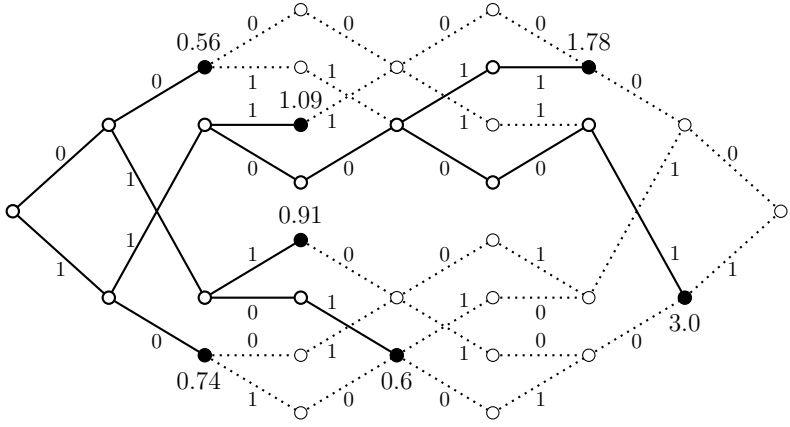


Figure 4.48 Forward tree in the second iteration, with threshold $T_2 = 0.54$.

the two trees is shown in Fig. 4.49. There are now two matching nodes, with the total metrics 0.58 and 1.78, respectively. Both values are above T_2 and we need to increase the threshold again and continue the search.

In the third step, the threshold is $T_3 = T_2 + 0.56 = 1.1$. Using $T_3/2 = 0.55$ as the target metric in the forward direction does not change the forward tree from the previous step. The backward tree gets extended as shown in Fig. 4.50. The matching of the sets $\mathcal{F}$ and $\mathcal{B}$ is shown in Fig. 4.51. There are four matching

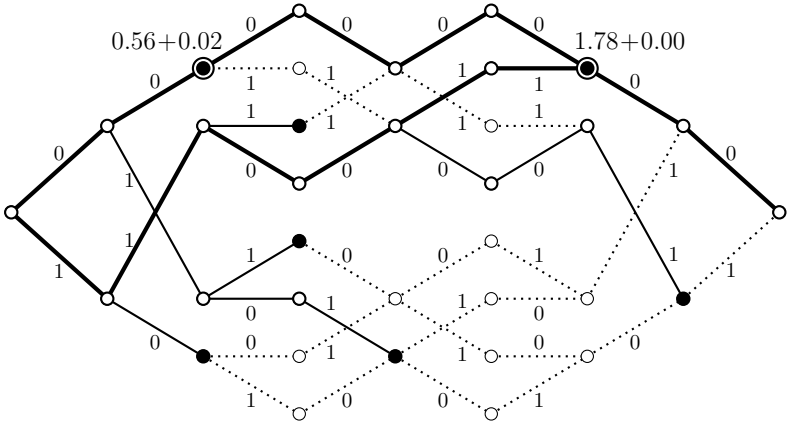


Figure 4.49 Matching the forward and the backward tree in the second iteration: there are two matching nodes (highlighted), but both have total metrics above the current threshold and are thus not accepted.

nodes:

$$\mathcal{M}_{\boldsymbol{\sigma}} = \mathcal{F}_{\boldsymbol{\sigma}} \cap \mathcal{B}_{\boldsymbol{\sigma}} = \{0000,\, 0100,\, 0000,\, 0001\}$$
$$\mathcal{M}_{w} = \{0.58,\, 1.25,\, 1.78,\, 3.32\}$$
$$\mathcal{M}_{\ell} = \{2,\, 3,\, 6,\, 7\}$$

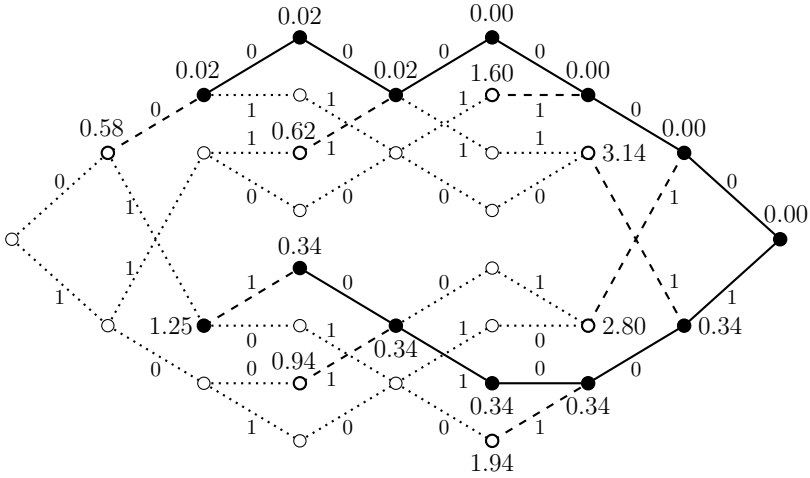


Figure 4.50 Backward tree in the third iteration, with threshold $T_3 = 1.1$.

Since only the node at depth $\ell_{\mathsf{F}} = 2$ with $\boldsymbol{\sigma}(\xi) = 0000$ has total metric below $T_3 = 1.1$ (in general, there may be more such nodes, in which case we pick the one

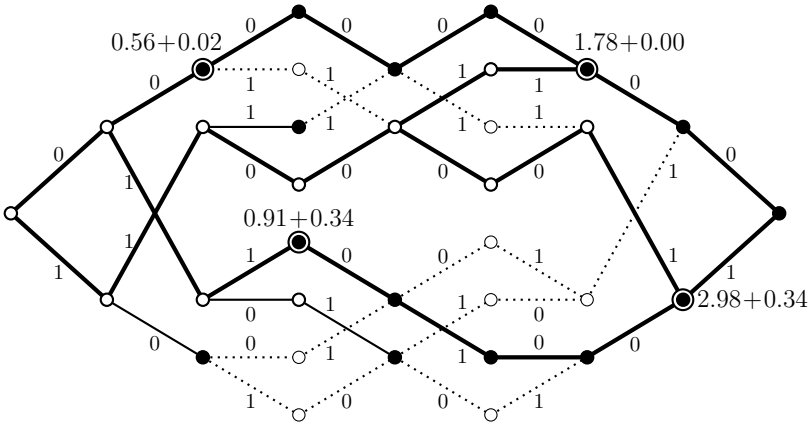


Figure 4.51 Matching the forward and the backward tree in the third iteration: there are four matching nodes (highlighted). The one with the smallest metric below the threshold determines the ML path.

with the minimal metric) this node determines the ML path: $\widehat{\boldsymbol{v}}_{\mathrm{ML}} = 00000000$, with the corresponding information sequence $\widehat{\boldsymbol{u}}_{\mathrm{ML}} = 0000$.

If we were interested in finding the $L = 2$ best codewords, we would need to increase the threshold once again and extend the trees. We would obtain that the second-best codeword is $\boldsymbol{v}_2 = 01101001$ with the metric 1.25.

Note that the ML codeword was already found in the first decoding step, but it was not accepted as a final decision since its metric was above the current threshold. This ensures that we pick the codeword with the smallest metric of *all* codewords. This also holds for a list of L best codewords; for example, accepting the match we found in the second step (codeword 11001100, with the metric 1.78) as a second best codeword would be erroneous, since in the next step we found the codeword 01101001 with a smaller metric.

The total number of nodes visited by BEAST during decoding is 23 (11 in the forward and 12 in the backward tree). ML decoding with the Viterbi algorithm requires visiting all nodes in the trellis, which amounts to 34 nodes.

Since BEAST operates on trees, it does not take into account merging properties of the code trellis, as the Viterbi algorithm does. One might suspect that BEAST visits many "duplicate" nodes and, hence, that it is less efficient than the Viterbi algorithm from a complexity point of view. However, it can be shown that when performing ML decoding for the BSC, BEAST will not visit more nodes than the Viterbi algorithm. Similarly, when performing bounded distance decoding up to $d_{\min}$ for the AWGN channel, BEAST will not visit more nodes than the Viterbi algorithm [BJK04a].

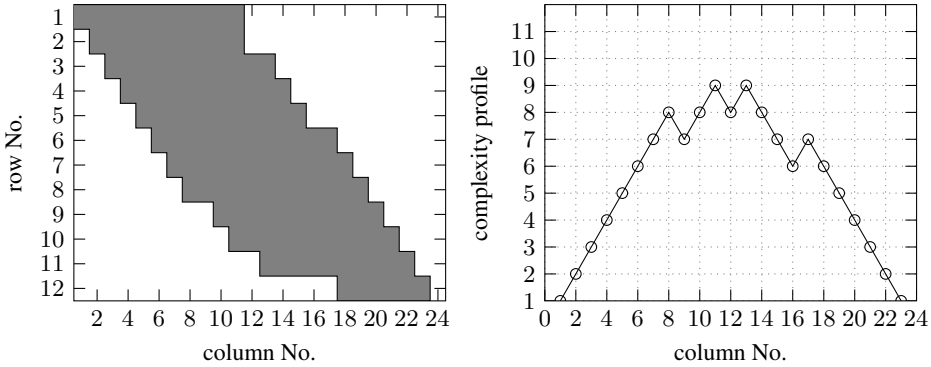


Figure 4.52 Active rows (left) and state complexity profile $\boldsymbol{\mu}$ (right) for the $(24, 12, 8)$ extended Golay code with generator matrix (4.312).

■ **EXAMPLE 4.22**

Consider the $(24, 12, 8)$ extended Golay code with generator matrix

$$G = \begin{pmatrix}
1\,1\,1\,1\,0\,1\,1\,1\,1\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0 \\
0\,1\,0\,0\,1\,0\,1\,1\,1\,1\,1\,1\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0 \\
0\,0\,1\,1\,1\,0\,0\,1\,1\,0\,1\,0\,1\,1\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0 \\
0\,0\,0\,1\,1\,1\,1\,1\,0\,1\,0\,0\,1\,0\,1\,0\,0\,0\,0\,0\,0\,0\,0\,0 \\
0\,0\,0\,0\,1\,0\,0\,0\,0\,1\,1\,1\,1\,1\,1\,1\,0\,0\,0\,0\,0\,0\,0\,0 \\
0\,0\,0\,0\,0\,1\,1\,1\,1\,1\,1\,1\,1\,0\,1\,1\,1\,1\,0\,0\,0\,0\,0\,0 \\
0\,0\,0\,0\,0\,0\,1\,1\,0\,1\,1\,0\,1\,1\,0\,0\,0\,1\,1\,0\,0\,0\,0\,0 \\
0\,0\,0\,0\,0\,0\,0\,1\,1\,0\,1\,1\,1\,0\,0\,1\,0\,0\,1\,1\,0\,0\,0\,0 \\
0\,0\,0\,0\,0\,0\,0\,0\,0\,1\,1\,0\,1\,0\,0\,1\,1\,1\,0\,1\,1\,0\,0\,0 \\
0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,1\,1\,1\,0\,1\,0\,1\,1\,1\,0\,0\,1\,0\,0 \\
0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,1\,1\,1\,0\,1\,1\,1\,0\,0\,1\,0\,1\,0 \\
0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,1\,1\,1\,1\,1\,1\,1\,1\,1
\end{pmatrix} \tag{4.312}$$

The active rows and the complexity profile of G are given in Fig. 4.52. The maximal state complexity, $\mu_{\max} = 9$, determines the complexity of the Viterbi ML decoder.

The bit error probability for BEAST soft-decision ML decoding is shown in Fig. 4.53. The maximum and average number of visited nodes are illustrated in Fig. 4.54 for both BEAST and the Viterbi algorithm. The average complexity of BEAST at low SNR is about $2^4 = 16$ times lower than the complexity of the Viterbi algorithm.

Asymptotically BEAST-ML decoding is more efficient than Viterbi-ML decoding. So-called double-zero-tail terminated codes [BJK04b] have the lowest upper bound on the BEAST decoding complexity. For high code rates, this bound is an improvement on previously known bounds on the ML decoding complexity (see [BHJ05]).

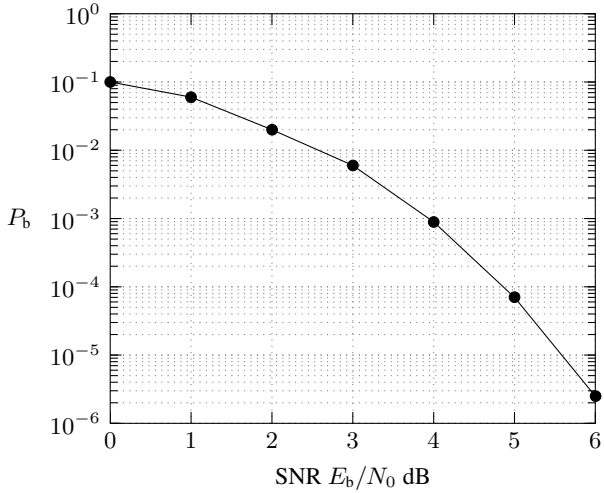


Figure 4.53 Bit error probability for the ML-decoded $(24, 12, 8)$ extended Golay code with generator matrix (4.312).

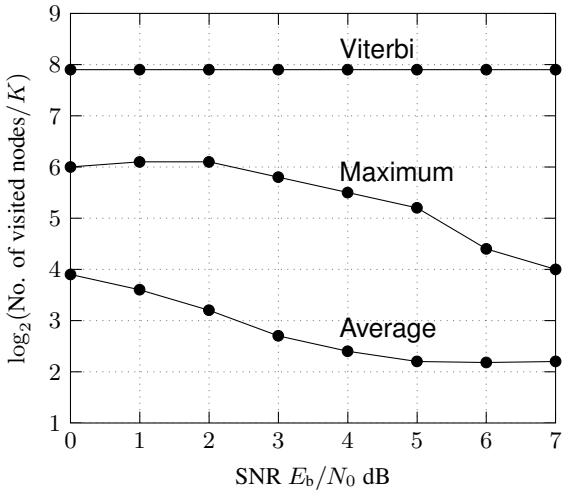


Figure 4.54 Average and maximum number of visited nodes per bit (in logarithmic scale) for BEAST-ML decoding for the $(24, 12, 8)$ extended Golay code with generator matrix (4.312).

4.11 COMMENTS

By presenting his decoding algorithm Viterbi gave a remarkable boost both to academic work on coding theory and to industrial applications. Using signal flowcharts for analyzing convolutional codes was introduced by Viterbi [Vit71]. The "few" and

"many" idea which we have exploited for bounding the error probability for fixed convolutional codes can be found in Fano's textbook [Fan61], where it is applied to random ensembles of block codes. For convolutional codes the "few" and "many" idea was first used in [CJZ84a].

Exact calculations of the bit error probability for rate $R = 1/2$ convolution codes encoded by minimal-basic encoding matrices realized in controller canonical form for memory $m = 1$ and $m = 2$ are given in [BBL95] and [LTZ04], respectively.

The closed-form expression for the exact bit error probability for Viterbi decoding in Section 4.4 is from [BHJ12] (see also Florian Hug's thesis [Hug12]).

The Bahl, Cocke, Jelinek, and Raviv (BCJR) algorithm [BCJ74] was introduced in 1974 as a decoding method for convolutional codes or block codes with a trellis structure. For a tutorial introduction to a general class of two-way algorithms, the reader is referred to [For97].

The simple upper bound presented in Section 4.7 grew out of the embryos [JMS00, JMS02, HJM01].

Tailbiting representations of block codes were introduced by Solomon and van Tilborg [Sov79]. See also [MaW86].

BEAST was introduced in [BHJ01, BHJ04] as an efficient algorithm for determining the spectra and decoding of block and convolutional codes (see also Marc Handlery's thesis [Han02]). In [BJK04a] decoding of block codes by BEAST was treated in depth and the superiority of BEAST was demonstrated (see also Maja Lončar's thesis [Lon07]). For tailbiting block codes it turns out that it was very efficient to first obtain the absolute minimal-span form and then decode with BEAST. In [BJK07] list decoding with BEAST was analyzed. Example 4.21 in Section 4.10 is from Maja Lončar's thesis [Lon07]. In Florian Hug's thesis [Hug12] the efficiency of BEAST for decoding LDPC codes and other codes on graphs was demonstrated.

PROBLEMS

4.1 Consider the rate $R = 1/2$, memory $m = 1$, systematic encoding matrix $G(D) = (1 \quad 1 + D)$.

- **a)** Draw the length $\ell = 5$ trellis.
- **b)** Suppose that the encoder is used to communicate over a BSC with crossover probability $\epsilon, 0 < \epsilon < 1/2$. Use the Viterbi algorithm to decode the received sequence $\boldsymbol{r} = 10\,01\,11\,11\,11\,11$.
- **c)** How many channel errors have occurred if the optimal path corresponds to the transmitted sequence?

4.2 Consider the rate $R = 1/2$, memory $m = 2$ encoding matrix $G(D) = (1 + D + D^2 \quad 1 + D^2)$.

- **a)** Draw the length $\ell = 4$ trellis.
- **b)** Suppose that the encoder is used to communicate over a BSC with crossover probability $\epsilon, 0 < \epsilon < 1/2$. Use the Viterbi algorithm to decode the received sequence $\boldsymbol{r} = 11\,11\,00\,11\,01\,11$.

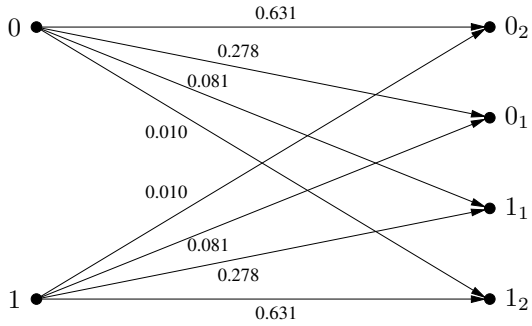


Figure 4.55 DMC used in Problems 4.8 and 4.28.

c) Suppose that the information sequence is $\boldsymbol{u} = 1011$. How many channel errors are corrected in (b)?

4.3 Repeat Problem 4.2 for $\boldsymbol{r} = 10\,11\,10\,10\,01\,00$ and $\boldsymbol{u} = 0100$.

4.4 Repeat Problem 4.2 for $\boldsymbol{r} = 00\,11\,10\,01\,00\,00$ and $\boldsymbol{u} = 0000$.

4.5 Consider a binary-input, 8-ary output DMC with transition probabilities $P(r \mid v)$ given in Example 4.2.

a) Suppose that the rate $R = 1/2$ encoder with encoding matrix $G(D) = (1 + D + D^2 \quad 1 + D^2)$ is used to communicate over the given channel. Use the Viterbi algorithm to decode $\boldsymbol{r} = 1_3 1_4\ 1_3 0_4\ 0_2 0_4\ 0_1 1_3\ 0_3 1_1\ 0_4 1_1$.

b) Connect the channel to a BSC by combining the soft-decision outputs $0_1, 0_2, 0_3, 0_4$ and $1_1, 1_2, 1_3, 1_4$ to hard-decision outputs 0 and 1, respectively. Use the Viterbi algorithm to decode the hard-decision version of the received sequence in (a).

4.6 Repeat Problem 4.5 for $\boldsymbol{r} = 0_4 0_3\ 1_3 0_1\ 0_1 1_1\ 1_1 0_1\ 0_1 1_2\ 1_1 0_4$.

4.7 Repeat Problem 4.5 for $\boldsymbol{r} = 1_1 0_4\ 0_1 1_2\ 1_1 0_1\ 0_1 1_1\ 1_3 0_1\ 0_4 0_3$.

4.8 Consider the binary-input, 4-ary output DMC with transition probabilities given in Fig. 4.55.

a) Suppose that the rate $R = 1/2$ encoder with encoding matrix $G(D) = (1 + D + D^2 \quad 1 + D^2)$ is used to communicate over the given channel. Use the Viterbi algorithm to decode $\boldsymbol{r} = 0_1 1_1\ 1_2 0_1\ 1_1 0_2\ 0_1 1_1\ 1_2 1_1\ 1_1 1_2$.

b) Connect the channel to a BSC by combining the soft-decision outputs $0_1, 0_2$ and $1_1, 1_2$ to hard-decision outputs 0 and 1, respectively. Use the Viterbi algorithm to decode the hard-decision version of the received sequence in (a).

4.9 Consider the binary-input, 8-ary output DMC shown in Fig. 4.6 with transition probabilities $P(r \mid v)$ given by the following table:

		r							
		0_4	0_3	0_2	0_1	1_1	1_2	1_3	1_4
v	0	0.1415	0.3193	0.2851	0.1659	0.0676	0.0180	0.0025	0.0001
	1	0.0001	0.0025	0.0180	0.0676	0.1659	0.2851	0.3193	0.1415

Suppose that the rate $R = 1/2$ encoder with encoding matrix $G(D) = (1 + D^2 \quad 1 + D + D^2)$ is used to communicate over the given channel.

After appropriate scaling and rounding of the metrics, use the Viterbi algorithm to decode $\boldsymbol{r} = 0_1 1_2 \; 1_2 1_3 \; 0_3 0_2 \; 0_1 0_3 \; 0_3 0_2 \; 0_2 0_1$.

4.10 Consider the binary-input, 8-ary output DMC shown in Fig. 4.6 with transition probabilities $P(r \mid v)$ given by the following table:

		r							
		0_4	0_3	0_2	0_1	1_1	1_2	1_3	1_4
v	0	0.2196	0.2556	0.2144	0.1521	0.0926	0.0463	0.0167	0.0027
	1	0.0027	0.0167	0.0463	0.0926	0.1521	0.2144	0.2556	0.2196

Repeat Problem 4.5 for $\boldsymbol{r} = 0_2 1_3 \; 1_2 0_2 \; 0_1 1_3 \; 1_1 1_1 \; 1_2 1_4 \; 1_1 1_1$.

4.11 Consider the communication system in Problem 4.5.

a) Which parameters determine the system error-correcting capability?

b) Find the values of the parameters in (a).

4.12 Consider a binary-input, q-ary output DMC with a high signal-to-noise ratio. Which of the following three convolutional encoding matrices will perform best in combination with ML decoding?

$$G_1(D) = \begin{pmatrix} 1 + D & 1 + D + D^2 \end{pmatrix}$$
$$G_2(D) = \begin{pmatrix} 1 + D + D^2 + D^3 & 1 + D^3 \end{pmatrix}$$
$$G_3(D) = \begin{pmatrix} 1 & 1 + D^2 + D^4 \end{pmatrix}$$

4.13 Suppose that we use the rate $R = 1/2$ convolutional encoding matrix $G(D) = (1 + D^2 \quad 1 + D)$ to encode 998 information bits followed by two dummy zeros. The codeword is transmitted over a BSC with crossover probability ϵ, $0 < \epsilon < \frac{1}{2}$.

a) What is the total number of possibly received sequences?

b) Find the total number of codewords.

c) Suppose that the received sequence $\boldsymbol{r} = (r_1 r_2 \ldots r_?)$ is given by

$$r_i = \begin{cases} 1, & i = 536, 537 \\ 0, & \text{otherwise} \end{cases}$$

Find the ML decision for the information sequence.

4.14 Use the periodic sequence $[111001]^\infty$ as puncturing sequence together with the encoding matrix $G(D) = (1 + D^2 \quad 1 + D + D^2)$ to communicate over a BSC with crossover probability ϵ.

a) Find the rate for the punctured code.

b) Use the Viterbi algorithm to decode $\boldsymbol{r} = 01010101$.

4.15 Consider the encoding matrix in Problem 3.9 and assume that it is realized in controller canonical form. Evaluate the upper bounds on the burst error probability for a BSC with crossover probability ϵ.

a) $\epsilon = 0.1$

b) $\epsilon = 0.01$

c) $\epsilon = 0.001$

4.16 Repeat Problem 4.15 for the encoding matrix in Problem 3.10.

4.17 Verify (4.20) for d even.

4.18 Verify (4.30).

4.19 Van de Meeberg [Mee74] derived a slightly tighter bound than (4.33).

a) Show that

$$p_{2i} \leq \binom{2\delta - 1}{\delta} 2^{-2\delta} \left(2\sqrt{\epsilon}\right)^{2i}$$

where

$$\delta = \left\lfloor \frac{d_\infty + 1}{2} \right\rfloor$$

Hint: $(1-\epsilon)^{2i-1} \sum_{e=i}^{2i-1} \left(\frac{\epsilon}{1-\epsilon}\right)^e = 2^{-2i} \left(2\sqrt{\epsilon}\right)^{2i} \frac{(1-\epsilon)^i - \epsilon^i}{1-2\epsilon}$.

b) Show that

$$P_{\mathrm{B}} < \binom{2\delta - 1}{\delta} 2^{-2\delta} \times \left(\frac{1}{2} \left(T(W) + T(-W)\right) + \frac{W}{2} \left(T(W) - T(-W)\right) \right) \Big|_{W = 2\sqrt{\epsilon}}$$

4.20 In Fig. 4.24 we illustrated four trellis sections for $\phi_t = 0$ for the encoder used in Example 4.10.

a) Draw the remaining four trellis sections for $\phi_t = 0$ and verify the entries in the rows corresponding to $\phi_t = 0$ for A_{00} and B_{00}.

b) Draw all eight trellis sections for $\phi_t = -2$ and verify the entries in the rows corresponding to $\phi_t = -2$ for A_{00} and B_{00}.

c) Repeat (b) for $\phi_t = -1$.

d) Repeat (b) for $\phi_t = 1$.

e) Repeat (b) for $\phi_t = 2$.

4.21 Consider the encoder in Example 4.10.

a) Determine A_{01} and B_{01}.

b) Determine A_{10} and B_{10}.

c) Determine A_{11} and B_{11}.

4.22 Prove van de Meeberg's bound (4.56) on the bit error probability.

4.23 Consider the rate $R = 1/2$, memory $m = 1$, systematic encoding matrix $G(D) = (1 \quad 1 + D)$. Suppose that the encoder is used to communicate over a BSC with crossover probability $\epsilon = 0.045$. Assume that the received sequence $\boldsymbol{r} = 10\ 01\ 11\ 11\ 11\ 11$.

Use the BCJR algorithm to decode the received sequence when

a) the information symbols *a priori* are equiprobable.

b) $P(u_t = 0) = 2/3$.

c) $P(u_t = 0) = 1/3$.

4.24 Repeat Problem 4.23 for the rate $R = 1/2$, memory $m = 2$ encoding matrix $G(D) = (1 + D + D^2 \quad 1 + D^2)$ and $\boldsymbol{r} = 11\ 11\ 00\ 11\ 01\ 11$.

4.25 Repeat Problem 4.23 for the rate $R = 2/3$, memory $m = 1$ encoding matrix

$$G(D) = \begin{pmatrix} 1 & 1 & D \\ D & 1+D & 1 \end{pmatrix}$$

and $\boldsymbol{r} = 010\ 111\ 000$.

4.26 Prove formula (4.233).

4.27 Show that the best decoding bit error probability P_{b} obtainable with linear coding on a BSC with crossover probability ϵ, $0 \leq \epsilon \leq 1/2$, is *at least* as great as that obtainable with linear coding on BEC with erasure probability $\delta = 2\epsilon$.

4.28 Consider the binary-input, 4-ary output DMC with transition probabilities given in Fig. 4.55.

Suppose that the rate $R = 1/2$ encoder with encoding matrix $G(D) = (1 + D + D^2 \quad 1 + D^2)$ is used to communicate over the given channel. Use the BCJR algorithm to decode $\boldsymbol{r} = 0_1 1_1\ 1_2 0_1\ 1_1 0_2\ 0_1 1_1\ 1_2 1_1\ 1_1 1_2$ when

a) the information symbols *a priori* are equiprobable.

b) $P(u_t = 0) = 2/3$.

c) $P(u_t = 0) = 1/3$.

4.29 Consider the binary-input, 8-ary output DMC shown in Fig. 4.6 with transition probabilities $P(r \mid v)$ given by the following table:

		r							
		0_4	0_3	0_2	0_1	1_1	1_2	1_3	1_4
v	0	0.1415	0.3193	0.2851	0.1659	0.0676	0.0180	0.0025	0.0001
	1	0.0001	0.0025	0.0180	0.0676	0.1659	0.2851	0.3193	0.1415

Repeat Problem 4.28 for $\boldsymbol{r} = 0_1 1_2\ 1_2 1_3\ 0_3 0_2\ 0_1 0_3\ 0_3 0_2\ 0_2 0_1$.

4.30 Consider the binary-input, 8-ary output DMC shown in Fig. 3.3 with transition probabilities $P(r \mid v)$ given by the following table:

		r							
		0_4	0_3	0_2	0_1	1_1	1_2	1_3	1_4
v	0	0.2196	0.2556	0.2144	0.1521	0.0926	0.0463	0.0167	0.0027
	1	0.0027	0.0167	0.0463	0.0926	0.1521	0.2144	0.2556	0.2196

Repeat Problem 4.28 for $\boldsymbol{r} = 0_2 1_3\ 1_2 0_2\ 0_1 1_3\ 1_1 1_1\ 1_2 1_4\ 1_1 1_1$.

4.31 Repeat Problem 4.23 for the one-way algorithm when $\tau = 2, 3, 4$. Decode only the first two information symbols.

4.32 Repeat Problem 4.24 for the one-way algorithm when $\tau = 4$. Decode only the first two information symbols.

4.33 Repeat Problem 4.28 for the one-way algorithm when $\tau = 4$. Decode only the first two information symbols.

4.34 Construct a 2-state $(8, 4)$ tailbiting representation by using the convolutional encoding matrix $G(D) = (1 \quad 1 + D)$. Suppose that this block code is used to communicate over the BSC with crossover probability $\epsilon = 0.045$. Assume that the received sequence $\boldsymbol{r} = 10\,01\,11\,11$.

Use the BCJR tailbiting algorithm to decode the received sequence when

a) the information symbols *a priori* are equiprobable.

b) $P(u_t = 0) = 2/3$.

c) $P(u_t = 0) = 1/3$.

4.35 Construct a 4-state $(4, 2)$ tailbiting representation by using the convolutional encoding matrix $G(D) = (1 + D + D^2 \quad 1 + D^2)$. Suppose that this block code is used to communicate over the BSC with crossover probability $\epsilon = 0.045$. Assume that the received sequence $\boldsymbol{r} = 11\,11$.

Use the BCJR tailbiting algorithm to decode the received sequence when

a) the information symbols *a priori* are equiprobable.

b) $P(u_t = 0) = 2/3$.

c) $P(u_t = 0) = 1/3$.

4.36 Use the 4-ary output DMC given in Problem 4.28 and repeat Problem 4.34 for $\boldsymbol{r} = 0_1 1_1\ 1_2 0_1\ 1_1 0_2\ 0_1 1_1$.

4.37 Use the 4-ary output DMC given in Problem 4.28 and repeat Problem 4.35 for $\boldsymbol{r} = 0_1 1_1\ 1_2 0_1$.

4.38 Use the 8-ary output DMC given in Problem 4.29 and repeat Problem 4.34 for $\boldsymbol{r} = 0_1 1_1\ 1_2 1_3\ 0_3 0_2\ 0_1 0_3$.

4.39 Use the 8-ary output DMC given in Problem 4.29 and repeat Problem 4.35 for $\boldsymbol{r} = 0_1 1_1\ 1_2 1_3$.

4.40 Assume that Gilbert-Varshamov's lower bound on the minimum distance for block codes (3.183) and Costello's lower bound on the free distance for convolutional codes (3.162) are asymptotically tight. Show for a rate R tailbiting trellis of length L encoded by a generator matrix of memory m that the best error-correcting capability

for a given decoding complexity is obtained for

$$\frac{L}{m} = \frac{R}{-\rho \log(2^{1-R} - 1)} + o(m)$$

where ρ is the Gilbert-Varshamov parameter.

4.41 Consider the tailbiting generator matrix

$$G^{\text{tb}} = \begin{pmatrix} 11 & 10 & 11 & 00 \\ 00 & 11 & 10 & 11 \\ 11 & 00 & 11 & 10 \\ 10 & 11 & 00 & 11 \end{pmatrix}$$

and encode $\boldsymbol{u} = 1001$.

a) Find the starting state $\boldsymbol{\sigma}_{\text{start}}$.

b) Find the codeword.

4.42 [Sta01] Consider the tailbiting of length $L = 4$ using the equivalent systematic encoder to the encoder defined in Problem 4.41.

a) Find the rational generator matrix.

b) Find the codeword corresponding to the information sequence $u(D) = 1 + D + D^3$.

4.43 [Sta01] Consider the rate $R = 2/3$ polynomial generator matrix

$$G(D) = \begin{pmatrix} 1 & 1+D & D \\ D^2 & D & 1+D+D^2+D^3 \end{pmatrix}$$

a) Find the Smith form decomposition of $G(D)$.

b) Is $G(D)$ noncatastrophic?

c) Consider tailbiting length $L = 5$. Does tailbiting work?

d) Consider tailbiting length $L = 6$. Does tailbiting work?

4.44 [Lon07] Consider the $(5, 3, 2)$ block code whose parity-check matrix is

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

a) Find its generator matrix.

b) Draw the trellis of the $(5, 3, 2)$ code.

c) Bring the generator matrix into minimal-span form.

d) Permute the columns of the minimal-span form in (c) and obtain a generator matrix with a slightly improved complexity profile.

e) Draw the trellis corresponding to the generator matrix in (d).

4.45 [Hug12] Consider the $(6, 3, 3)$ shortened Hamming code with generator matrix

$$G = \begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \end{pmatrix}$$

Find its minimal-span form which is used to communicate over an AWGN channel. The received sequence is

$$\boldsymbol{r} = (0.82 \quad -0.42 \quad 0.17 \quad 1.25 \quad 0.83 \quad 0.37)$$

Use BEAST to obtain the ML decision $\widehat{\boldsymbol{v}}$.

CHAPTER 5

RANDOM ENSEMBLE BOUNDS FOR DECODING ERROR PROBABILITY

In this chapter, we continue our analysis of the decoding error probability of the Viterbi decoding algorithm for convolutional codes but for the ensemble of time-varying convolutional codes and derive upper bounds for the burst and bit error probabilities that are exponentially decreasing with the memory of the convolutional code. A general lower bound for the decoding error probability of convolutional coding is proved and it is shown how the R_0-criterion can be used to quantize the channel outputs.

5.1 UPPER BOUNDS ON THE OUTPUT ERROR BURST LENGTHS

The errors in the output from the Viterbi decoder are grouped in error bursts. In this section, we will upper-bound the distribution of the lengths of these error events for the ensemble of periodically time-varying convolutional codes. In the next section we will use these upper bounds to obtain upper bounds on the burst error probability. For simplicity we will only consider the binary symmetric channel (BSC).

Consider the ensemble $\mathcal{E}(b, c, m, T)$ of binary, rate $R = b/c$, periodically time-varying convolutional codes encoded by polynomial, periodically time-varying generator matrices of memory m and period T which we introduced in Section 3.6. Let

Fundamentals of Convolutional Coding, 2. Edition. By Rolf Johannesson, Kamil Sh. Zigangirov

$\widehat{\boldsymbol{u}}$ and $\boldsymbol{e} = \boldsymbol{u} + \widehat{\boldsymbol{u}}$ denote the decision for the information sequence and the error sequence in the decoder output, respectively.

In Section 3.6 we also introduced the set $\mathcal{U}^r_{[t_1-m,t_2+m]}$ of information sequences $\boldsymbol{u}_{t_1-m}\boldsymbol{u}_{t_1-m+1}\ldots\boldsymbol{u}_{t_2+m}$ such that the first m and the last m subblocks are zero and such that they do not contain $m+1$ consecutive zero subblocks. We have the following:

Definition The decoder output error sequence $\boldsymbol{e}_{[t-m-1,t+j+m+1]}$ is called an *error burst* or *error event of length* $j+1$ starting at time t and ending at time $t+j+1$ if $\boldsymbol{e}_{t-m-1} = \boldsymbol{0}, \boldsymbol{e}_{[t-m,t+j+m]} \in \mathcal{U}^r_{[t-m,t+j+m]}$, and $\boldsymbol{e}_{t+j+m+1} = \boldsymbol{0}$.

In order to upper-bound the distribution of the length of an error burst starting at time t, we consider the block code $\mathcal{B}_t(j)$ given by

$$\begin{aligned}\mathcal{B}_t(j) = \Big\{\, \boldsymbol{v}_{[t,t+j+m]} \;\Big|\; & \boldsymbol{v}_{[t,t+j+m]} = \boldsymbol{0} \text{ or} \\ & \boldsymbol{v}_{[t,t+j+m]} = \boldsymbol{u}_{[t-m,t+j+m]}\boldsymbol{G}_{[t,t+j+m]}, \\ & \boldsymbol{u}_{[t-m,t+j+m]} \in \mathcal{U}^r_{[t-m,t+j+m]} \Big\}\end{aligned} \tag{5.1}$$

where $\boldsymbol{G}_{[t,t+j+m]}$ is given by (3.136). The rate of the block code $\mathcal{B}_t(j)$ is upper-bounded by

$$r(j) = \frac{j+1}{j+m+1}R \tag{5.2}$$

(This is an *upper bound* since we have imposed a restriction on $\boldsymbol{u}_{[t-m,t+j+m]}$.)

Assume that the transmitted sequence is the allzero sequence, and let $\mathcal{L}_t(j)$ denote the event that an error burst starting at depth t has length $j+1$. A necessary—but not sufficient—condition for $\mathcal{L}_t(j)$ to occur is that the block code $\mathcal{B}_t(j)$ will be erroneously decoded. Thus, we have

$$P(\mathcal{L}_t(j)) \le P(\mathcal{E}_t(j)) \tag{5.3}$$

where $\mathcal{E}_t(j)$ denotes the event that $\mathcal{B}_t(j)$ is erroneously decoded. Hence, we can obtain an upper bound on $P(\mathcal{L}_t(j))$ by upper-bounding the error probability of the block code $\mathcal{B}_t(j)$.

For a periodically time-varying with period T convolutional code, we define the probability of the event that an error burst has length $j+1$, $j \le T$, to be

$$P(\mathcal{L}(j)) \stackrel{\text{def}}{=} \max_{0\le t<T}\{P(\mathcal{L}_t(j))\} \tag{5.4}$$

The probability $P(\mathcal{L}(j))$ is upper-bounded by

$$P(\mathcal{L}(j)) \le \sum_{t=0}^{T-1} P(\mathcal{L}_t(j)) \tag{5.5}$$

Before we proceed we need the following:

Definition The *computational cutoff rate* R_0 for the BSC with crossover probability ϵ is given by

$$R_0 \stackrel{\text{def}}{=} 1 - \log\left(1 + 2\sqrt{\epsilon(1-\epsilon)}\right) \tag{5.6}$$

Using the Bhattacharyya parameter z (4.20), we can express R_0 as

$$R_0 = 1 - \log(1+z) \tag{5.7}$$

which can be rewritten as

$$2^{-R_0} = \frac{1+z}{2} \tag{5.8}$$

Now we have the following:

Lemma 5.1 (Random coding bound) Consider the ensemble $\mathcal{E}(b, c, m, T)$ of binary, rate $R = b/c$, periodically time-varying convolutional codes encoded by polynomial, periodically time-varying generator matrices of memory m and period T, where $T = O(m^2)$. For rate $r(j)$ given by (5.2) the average probability that an error burst has length $j+1$ when we communicate over the BSC and use maximum-likelihood decoding is upper-bounded by the inequality

$$E[P(\mathcal{L}(j))] \leq 2^{-(R_0 - r(j) + o(1))(j+m+1)c} \tag{5.9}$$

for $j \leq T$ and rates $0 \leq r(j) < R_0$, where R_0 is the computational cutoff rate.

Proof: From Theorem 3.25 and (4.20) it follows that over the ensemble $\mathcal{E}(b, c, m, T)$ the probability that each codeword $\boldsymbol{v}_{[t,t+j+m]} \neq \mathbf{0}$, $j \leq T$, causes an error is upper-bounded by

$$\begin{aligned}\sum_{i=0}^{(j+m+1)c} \binom{(j+m+1)c}{i} 2^{-(j+m+1)c} z^i &= \left(\frac{1+z}{2}\right)^{(j+m+1)c} \\ &= 2^{-R_0(j+m+1)c}\end{aligned} \tag{5.10}$$

where the last equality follows from (5.8).

We combine (5.10) with the upper bound $2^{r(j)(j+m+1)c}$ on the total number of codewords in $\mathcal{B}_t(j)$ and apply (5.3) and (5.5) and the lemma follows. ∎

Next we have the following definition:

Definition The *expurgation rate* R_{exp} for the BSC with crossover probability ϵ is given by

$$R_{\text{exp}} = 1 - h\left(\frac{2\sqrt{\epsilon(1-\epsilon)}}{1 + 2\sqrt{\epsilon(1-\epsilon)}}\right) \tag{5.11}$$

where $h(\cdot)$ is the binary entropy function (1.22).

For rates $r(j)$ less than the expurgation rate R_{exp}, we can obtain an essentially stronger bound. Consider the fraction of convolutional codes in the ensemble

$\mathcal{E}(b,c,m,T)$ whose active row distances satisfy the condition in Lemma 3.32. Since this fraction is larger than f, it follows analogously to the proof of Lemma 5.1 that the average error probability for this expurgated ensemble is upper-bounded by

$$\begin{aligned} E_{\text{exp}}[P(\mathcal{E}(j))] &= \max_{0\le t<T}\{E_{\text{exp}}[P(\mathcal{E}_t(j))]\} \\ &\le \frac{T}{f}\sum_{i=\widehat{a}_j^r+1}^{(j+m+1)c}\binom{(j+m+1)c}{i}2^{(r(j)-1)(j+m+1)c}z^i \qquad (5.12) \\ &\quad j_0<j<T \end{aligned}$$

where $\widehat{a}_j^r$ is the largest integer satisfying (3.174) and j_0 is the smallest integer satisfying (3.172).

For any $\lambda>0$, we can further upper-bound (5.12):

$$\begin{aligned} E_{\text{exp}}[P(\mathcal{E}(j))] &\le \frac{T}{f}\sum_{i=\widehat{a}_j^r+1}^{(j+m+1)c}\binom{(j+m+1)c}{i}2^{\lambda(i-\widehat{a}_j^r)}2^{(r(j)-1)(j+m+1)c}z^i \\ &\le \frac{T}{f}\,2^{(r(j)-1)(j+m+1)c}2^{-\lambda\widehat{a}_j^r}\sum_{i=0}^{(j+m+1)c}\binom{(j+m+1)c}{i}2^{\lambda i}z^i \\ &= \frac{T}{f}\,2^{(r(j)-1)(j+m+1)c}2^{-\lambda\widehat{a}_j^r}(1+2^\lambda z)^{j+m+1)c} \qquad (5.13) \\ &\quad j_0<j<T \end{aligned}$$

Let us choose

$$2^\lambda=\frac{\widehat{\rho}}{(1-\widehat{\rho})z},\quad \lambda>0 \qquad (5.14)$$

where

$$\widehat{\rho}=\frac{\widehat{a}_j^r}{(j+m+1)c}>\frac{z}{1+z} \qquad (5.15)$$

and where the inequality follows from $\lambda>0$. Then we can rewrite (5.13) as

$$\begin{aligned} E_{\text{exp}}[P(\mathcal{E}(j))] &\le \frac{T}{f}2^{(r(j)-1)(j+m+1)c}2^{h(\widehat{\rho})(j+m+1)c}z^{\widehat{\rho}(j+m+1)c} \\ &= \frac{T}{f}\,2^{(h(\widehat{\rho})+r(j)-1)(j+m+1)c}z^{\widehat{\rho}(j+m+1)c} \\ &\le \frac{1-f}{fT}z^{\widehat{\rho}(j+m+1)c},\quad j_0<j<T \qquad (5.16) \end{aligned}$$

where the last inequality follows from (3.174). Let us choose $f=1/2$; then we have

$$E_{\text{exp}}[P(\mathcal{E}(j))]\le\frac{1}{T}z^{\widehat{\rho}(j+m+1)c},\quad j_0<j<T \qquad (5.17)$$

From (3.174) it follows that

$$h(\widehat{\rho})\le 1-r(j)-\frac{\log(2T^2)}{(j+m+1)c} \qquad (5.18)$$

By choosing $T = O(m^2)$, we obtain that for large values of m we have $j_0 = 0$ (cf. Section 3.8). Furthermore,

$$h(\widehat{\rho}) \leq 1 - r(j) + O\left(\frac{\log m}{m}\right) \tag{5.19}$$

and, hence, for large values of m we can replace $\widehat{\rho}$ by $\rho + O\left(\frac{\log m}{m}\right)$, where ρ is the Gilbert-Varshamov parameter for the rate $r(j)$. Thus, we have

$$E_{\text{exp}}[P(\mathcal{E}(j))] \leq 2^{\rho(j+m+1)c\log z + O(m)} \tag{5.20}$$

where $j < T$ and (cf. (5.15))

$$\rho > \frac{z}{(1+z)} + O\left(\frac{\log m}{m}\right) \tag{5.21}$$

For the BSC we have the Bhattacharyya parameter (cf. (4.20))

$$z = 2\sqrt{\epsilon(1-\epsilon)} \tag{5.22}$$

and, finally, we obtain

$$E_{\text{exp}}[P(\mathcal{E}(j))] \leq 2^{\rho(j+m+1)c\log\left(2\sqrt{\epsilon(1-\epsilon)}\right) + O(\log m)} \tag{5.23}$$

where $j \leq T$ and

$$\rho > \frac{2\sqrt{\epsilon(1-\epsilon)}}{1 + 2\sqrt{\epsilon(1-\epsilon)}} + O\left(\frac{\log m}{m}\right) \tag{5.24}$$

or, equivalently,

$$r(j) < R_{\text{exp}} + O\left(\frac{\log m}{m}\right) \tag{5.25}$$

where the expurgation rate R_{exp} is given by (5.11).

From inequality (5.3) we obtain the next lemma.

Lemma 5.2 (Expurgation bound) Consider the ensemble $\mathcal{E}(b, c, m, T)$ of binary, rate $R = b/c$, periodically time-varying convolutional codes encoded by polynomial, periodically time-varying generator matrices of memory m and period T, where $T = O(m^2)$. There exists a subset containing at least half of the codes such that the average probability that an error burst has length $j + 1$ when used to communicate over the BSC with maximum-likelihood decoding is upper-bounded by the inequality

$$E_{\text{exp}}[P(\mathcal{L}(j))] \leq 2^{\rho(j+m+1)c\log\left(2\sqrt{\epsilon(1-\epsilon)}\right) + O(\log m)} \tag{5.26}$$

for $j < T$, $0 \leq r(j) < R_{\text{exp}} + O\left(\frac{\log m}{m}\right)$ and where $\rho = h^{-1}(1 - r(j))$ is the Gilbert-Varshamov parameter for the rate $r(j)$ and R_{exp} is the expurgation rate (5.11).

Next, we introduce another definition.

Definition The *critical rate* R_{crit} for the BSC with crossover probability ϵ is given by

$$R_{\text{crit}} = 1 - h\left(\frac{\sqrt{\epsilon}}{\sqrt{\epsilon}+\sqrt{1-\epsilon}}\right) \tag{5.27}$$

where $h(\cdot)$ is the binary entropy function (1.22).

We will now derive the so-called sphere-packing bound for rates $r(j) > R_{\text{crit}}$. We will exploit the idea of "few" ($\mathcal{F}$) and "many" ($\mathcal{M}$) errors introduced in Section 4.3.

Assume that the allzero sequence is transmitted over the BSC and that $\boldsymbol{r}_{[t,t+j+m]}$ is received and decoded by a maximum-likelihood decoder. Let the set $\mathcal{M}$ be the event of errors such that

$$w_{\text{H}}(\boldsymbol{r}_{[t,t+j+m]}) \geq \rho(j+m+1)c \tag{5.28}$$

where ρ is the Gilbert-Varshamov parameter for the rate $r(j)$. The set $\mathcal{F}$ is the complement of $\mathcal{M}$. Following the thread in Section 4.3, we upper-bound the average error probability for the ensemble $\mathcal{E}(b,c,m,T)$ by (cf. (4.81))

$$E[P(\mathcal{E}_t(j))] \leq E[P(\mathcal{E}_t(j),\mathcal{F})] + P(\mathcal{M}) \tag{5.29}$$

Let i_0 be the smallest integer larger than or equal to $\rho(j+m+1)c$, that is,

$$i_0 = \lceil \rho(j+m+1)c \rceil \tag{5.30}$$

Then from (5.28) it follows that

$$P(\mathcal{M}) = \sum_{i=i_0}^{(j+m+1)c} \binom{(j+m+1)c}{i} \epsilon^i (1-\epsilon)^{(j+m+1)c-i} \tag{5.31}$$

For any $\lambda > 0$ we have

$$\begin{aligned} P(\mathcal{M}) &\leq \sum_{i=i_0}^{(j+m+1)c} \binom{(j+m+1)c}{i} \epsilon^i (1-\epsilon)^{(j+m+1)c-i} 2^{\lambda(i-i_0)} \\ &\leq 2^{-\lambda\rho(j+m+1)c} \sum_{i=i_0}^{(j+m+1)c} \binom{(j+m+1)c}{i} (\epsilon 2^{\lambda})^i \\ &\quad \times (1-\epsilon)^{(j+m+1)c-i} \end{aligned} \tag{5.32}$$

Next, we upper-bound the right-hand side of (5.32) by extending the summation to $i = 0$. Then we obtain

$$\begin{aligned} P(\mathcal{M}) &< 2^{-\lambda\rho(j+m+1)c} \sum_{i=0}^{(j+m+1)c} \binom{(j+m+1)c}{i} (\epsilon 2^{\lambda})^i (1-\epsilon)^{(j+m+1)c-i} \\ &= 2^{-\lambda\rho(j+m+1)c} (1-\epsilon+2^{\lambda}\epsilon)^{(j+m+1)c} \end{aligned} \tag{5.33}$$

Its minimal value is achieved when

$$\lambda = \lambda_0 = \log \frac{\rho(1-\epsilon)}{(1-\rho)\epsilon} \tag{5.34}$$

The minimizing value $\lambda_0 > 0$ if and only if $\rho > \epsilon$, which corresponds to

$$r(j) < C = 1 - h(\epsilon) \tag{5.35}$$

where C is the channel capacity of the BSC.

Inserting λ_0 into (5.33) yields

$$\begin{aligned} P(\mathcal{M}) &< \left(\frac{\rho(1-\epsilon)}{(1-\rho)\epsilon}\right)^{-\rho(j+m+1)c} \left(1-\epsilon+\frac{\rho(1-\epsilon)}{1-\rho}\right)^{(j+m+1)c} \\ &= 2^{-\left(\rho \log \frac{\rho}{\epsilon} + (1-\rho)\log \frac{1-\rho}{1-\epsilon}\right)(j+m+1)c}, \rho > \epsilon \end{aligned} \tag{5.36}$$

Next we will upper-bound $E[P(\mathcal{E}_t(j), \mathcal{F})]$. Suppose that the word received over the BSC contains i errors where $i < i_0$, that is, we have a situation with "few" errors. Assuming maximum-likelihood decoding, we can upper-bound the error probability by the probability that a randomly chosen codeword is at distance at most i from the received word. Since these two events are independent, we have the following upper bound:

$$\begin{aligned} E[P(\mathcal{E}_t(j), \mathcal{F})] \leq \sum_{i=0}^{i_0-1} \sum_{k=0}^{i} & 2^{r(j)(j+m+1)c} \binom{(j+m+1)c}{k} 2^{-(j+m+1)c} \\ & \times \binom{(j+m+1)c}{i} \epsilon^i (1-\epsilon)^{(j+m+1)c-i} \end{aligned} \tag{5.37}$$

Now we introduce two parameters $\lambda' > 0$ and $\mu > 0$ such that we can extend the summation intervals without increasing the bound too much. Thus, we can further upper-bound (5.37) as follows:

$$\begin{aligned} E[P(\mathcal{E}_t(j), \mathcal{F})] &\leq \sum_{i=0}^{i_0-1} \sum_{k=0}^{(j+m+1)c} \binom{(j+m+1)c}{k} \binom{(j+m+1)c}{i} \\ &\quad \times 2^{(r(j)-1)(j+m+1)c} 2^{\mu(i-k)} 2^{\lambda'(i_0-1-i)} \epsilon^i (1-\epsilon)^{(j+m+1)c-i} \\ &\leq 2^{\lambda'(i_0-1)} 2^{(r(j)-1)(j+m+1)c} \left(\sum_{k=0}^{(j+m+1)c} \binom{(j+m+1)c}{k} 2^{-\mu k} \right) \\ &\quad \times \left(\sum_{i=0}^{i_0-1} \binom{(j+m+1)c}{i} (\epsilon 2^{\mu-\lambda'})^i (1-\epsilon)^{(j+m+1)c-i} \right) \\ &\leq 2^{\lambda' \rho (j+m+1)c} 2^{(r(j)-1)(j+m+1)c} (1+2^{-\mu})^{(j+m+1)c} \\ &\quad \times \left(\sum_{i=0}^{i_0-1} \binom{(j+m+1)c}{i} (\epsilon 2^{\mu-\lambda'})^i (1-\epsilon)^{(j+m+1)c-i} \right) \end{aligned} \tag{5.38}$$

where we have used (5.30) to obtain the last inequality. Extending the summation over i to $i = (j+m+1)c$, we obtain:

$$E[P(\mathcal{E}_t(j), \mathcal{F})] \leq 2^{\lambda' \rho (j+m+1)c} 2^{(r(j)-1)(j+m+1)c} (1+2^{-\mu})^{(j+m+1)c} \times (1-\epsilon+\epsilon 2^{\mu-\lambda'})^{(j+m+1)c} \tag{5.39}$$

It is straightforward (but tedious) to show that the upper bound (5.39) achieves its minimum value for

$$\mu = \mu_0 = \log \frac{1-\rho}{\rho} \tag{5.40}$$

and

$$\lambda' = \lambda'_0 = \log \frac{(1-\rho)^2 \epsilon}{\rho^2 (1-\epsilon)} \tag{5.41}$$

where ρ is the Gilbert-Varshamov parameter. The inequality $\lambda'_0 > 0$ is equivalent to

$$\rho \leq \frac{\sqrt{\epsilon}}{\sqrt{\epsilon} + \sqrt{1-\epsilon}} \tag{5.42}$$

which corresponds to

$$r(j) \geq 1 - h\left(\frac{\sqrt{\epsilon}}{\sqrt{\epsilon} + \sqrt{1-\epsilon}} \right) = R_{\text{crit}} \tag{5.43}$$

where R_{crit} is the critical rate (5.27).

By inserting (5.40) and (5.41) into (5.39), we obtain

$$E[P(\mathcal{E}_t(j), \mathcal{F})] \leq 2^{-\left(\rho \log \frac{\rho}{\epsilon} + (1-\rho) \log \frac{1-\rho}{1-\epsilon}\right)(j+m+1)c} \tag{5.44}$$

which holds for $j < T$ and $r(j) \geq R_{\text{crit}}$.

We can obtain an upper bound by combining (5.29), (5.36), and (5.44). Somewhat surprisingly, we can obtain a more elegant upper bound directly from (5.29), (5.32), and (5.38) by inserting the parameters $\lambda = \lambda_0$, $\lambda' = \lambda'_0$, and $\mu = \mu_0$:

$$\begin{aligned}
E[P(\mathcal{E}_t(j))] &\leq E[P(\mathcal{E}_t(j), \mathcal{F})] + P(\mathcal{M}) \\
&\leq \left(\frac{(1-\rho)^2 \epsilon}{\rho^2 (1-\epsilon)} \right)^{\rho(j+m+1)c} 2^{(r(j)-1)(j+m+1)c} (1-\rho)^{-(j+m+1)c} \\
&\quad \times \sum_{i=0}^{i_0-1} \binom{(j+m+1)c}{i} \left(\frac{(1-\epsilon)\rho}{1-\rho} \right)^i (1-\epsilon)^{(j+m+1)c-i} \\
&\quad + \left(\frac{(1-\rho)\epsilon}{\rho(1-\epsilon)} \right)^{\rho(j+m+1)c} \sum_{i=i_0}^{(j+m+1)c} \binom{(j+m+1)c}{i} \left(\frac{(1-\epsilon)\rho}{1-\rho} \right)^i \\
&\quad \times (1-\epsilon)^{(j+m+1)c-i} \\
&= \left(\frac{(1-\rho)\epsilon}{\rho(1-\epsilon)} \right)^{\rho(j+m+1)c} \sum_{i=0}^{(j+m+1)c} \binom{(j+m+1)c}{i} \left(\frac{(1-\epsilon)\rho}{1-\rho} \right)^i \\
&\quad \times (1-\epsilon)^{(j+m+1)c}
\end{aligned} \tag{5.45}$$

where the last equality follows from

$$r(j) = 1 - h(\rho) \tag{5.46}$$

By evaluating the sum in (5.45), we obtain

$$\begin{aligned} E[P(\mathcal{E}_t(j))] &\leq \left(\frac{(1-\rho)\epsilon}{\rho(1-\epsilon)}\right)^{\rho(j+m+1)c} \left(\frac{(1-\epsilon)\rho}{1-\rho} + 1 - \epsilon\right)^{(j+m+1)c} \\ &= \left(\left(\frac{\rho}{\epsilon}\right)^{-\rho} \left(\frac{1-\rho}{1-\epsilon}\right)^{-(1-\rho)}\right) \\ &= 2^{-\left(\rho \log \frac{\rho}{\epsilon} + (1-\rho)\log \frac{1-\rho}{1-\epsilon}\right)(j+m+1)c} \end{aligned} \tag{5.47}$$

which holds for rates $R_{\text{crit}} \leq r(j) < C$, where C is the channel capacity for the BSC.

Applying (5.3) and (5.5) completes the proof of the next lemma.

Lemma 5.3 (Sphere-packing bound) Consider the ensemble $\mathcal{E}(b, c, m, T)$ of binary, rate $R = b/c$, periodically time-varying convolutional codes encoded by polynomial, periodically time-varying generator matrices of memory m and period T, where $T = O(m^2)$. For rates $r(j)$ given by (5.2), the average probability that an error burst has length $j + 1$ when we communicate over the BSC and use maximum-likelihood decoding is upper-bounded by the inequality

$$E[P(\mathcal{L}(j))] \leq 2^{-\left(\rho \log \frac{\rho}{\epsilon} + (1-\rho)\log \frac{1-\rho}{1-\epsilon} + o(1)\right)(j+m+1)c} \tag{5.48}$$

for $j < T$, $R_{\text{crit}} \leq r(j) < C$, and where $\rho = h^{-1}(1 - r(j))$ is the Gilbert-Varshamov parameter for the rate $r(j)$ and R_{crit} is the critical rate (5.27).

From our derivations we can also deduce the following (cf. [Gal68]):

Theorem 5.4 There exists a binary block code $\mathcal{B}(j)$ of rate $r(j)$ and block length $(j + m + 1)c$ such that the event $\mathcal{E}(j)$ that $\mathcal{B}(j)$ is erroneously decoded, when it is used to communicate over the BSC together with maximum-likelihood decoding, is upper-bounded by

$$P(\mathcal{E}(j)) \leq 2^{-(E_{\mathcal{B}}(r(j)) + o(1))(j+m+1)c} \tag{5.49}$$

where $E_{\mathcal{B}}(\cdot)$ is the *block coding exponent*

$$E_{\mathcal{B}}(r) = \begin{cases} -\rho \log\left(2\sqrt{\epsilon(1-\epsilon)}\right), & 0 \leq r < R_{\text{exp}} \\ R_0 - r, & R_{\text{exp}} \leq r < R_{\text{crit}} \\ \rho \log \frac{\rho}{\epsilon} + (1-\rho)\log \frac{1-\rho}{1-\epsilon}, & R_{\text{crit}} \leq r < C \end{cases} \tag{5.50}$$

and where

$$\rho = h^{-1}(1 - r) \tag{5.51}$$

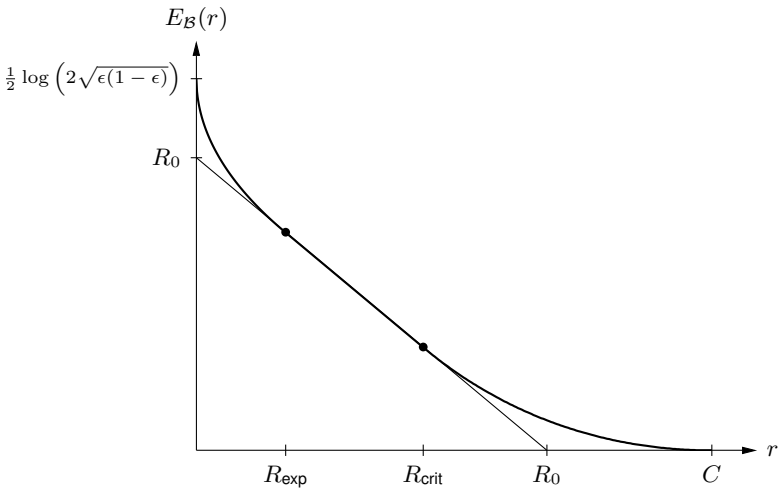


Figure 5.1 The block coding exponent $E_{\mathcal{B}}(r)$ for the BSC with crossover probability $\epsilon = 0.045$, that is, $R_0 = 1/2$.

is the Gilbert-Varshamov parameter, R_{exp} is the expurgation rate, and R_{crit} is the critical rate.

We call the interval $[0, R_{\text{exp}})$ the *expurgation region*, the interval $[R_{\text{exp}}, R_{\text{crit}})$ the *random coding region*, and the interval $[R_{\text{crit}}, C)$ the *sphere-packing region*. As we will see later, the main exponential term of the bound in the last interval coincides with the main exponential term of the lower bound for decoding error probability called the *sphere-packing bound*.

In Fig. 5.1 we show the block coding exponent $E_{\mathcal{B}}(r)$ for the BSC with crossover probability $\epsilon = 0.045$, which corresponds to $R_0 = 1/2$. The transition points between the expurgation, random coding, and sphere-packing regions are marked with $\bullet$.

From (5.3) it immediately follows that a convolutional code exists in the ensemble $\mathcal{E}(b, c, m, T)$ such that the probability of the event $\mathcal{L}_t(j)$ that an error burst starting at depth t, $t \geq 0$, has length $j + 1$, $0 \leq j < T$, is upper-bounded by the right side of inequality (5.49), that is,

$$P(\mathcal{L}(j)) \leq 2^{-(E_{\mathcal{B}}(r(j))+o(1))(j+m+1)c} \tag{5.52}$$

where $E_{\mathcal{B}}(r(j))$ and $r(j)$ are given by (5.50) and (5.2), respectively.

We define the *error burst length exponent* $L(\ell)$ to be

$$L(\ell) \stackrel{\text{def}}{=} \frac{E_{\mathcal{B}}(r(j))(j+m+1)}{m} \tag{5.53}$$

where

$$\ell = (j+1)/m \tag{5.54}$$

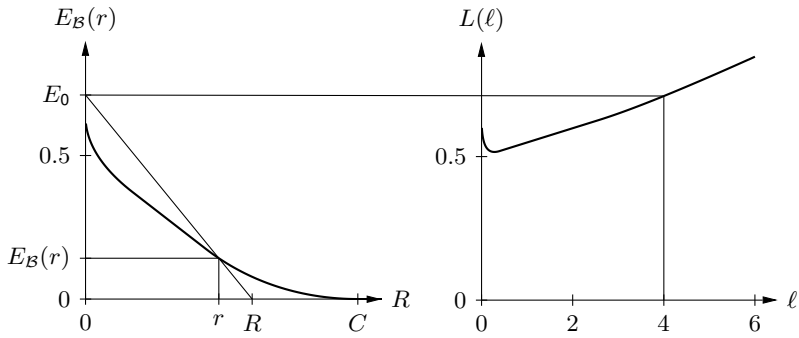


Figure 5.2 Geometrical interpretation of the relation between the block coding exponent $E_{\mathcal{B}}(r)$ and the error burst length exponent $L(\ell)$ for the BSC with crossover probability $\epsilon = 0.045$.

Then we have the following:

Theorem 5.5 There exists a binary, rate $R = b/c$, periodically time-varying convolutional code encoded by a polynomial, periodically time-varying generator matrix of memory m and period T, where $T = O(m^2)$, such that the probability of a length $j + 1$ error burst from a maximum-likelihood decoder when used to communicate over the BSC is upper-bounded by

$$P(\mathcal{L}(j)) \leq 2^{-(L(\ell)+o(1))mc} \tag{5.55}$$

where $j < T$, $\ell = (j + 1)/m$, and $L(\ell)$ is the error burst length exponent given by (5.53).

The error burst length exponent $L(\ell)$ can be constructed geometrically from the block coding exponent $E_{\mathcal{B}}(r(j))$. This construction is similar to Forney's inverse concatenated construction (cf. Fig. 3.12). Consider the block coding exponent $E_{\mathcal{B}}(r(j))$ as given in Fig. 5.2. Draw a straight line from the point $(R, 0)$ through $(r(j), E_{\mathcal{B}}(r(j)))$. This line intersects the $E_{\mathcal{B}}(r)$-axis in the point $(0, E_0)$. From Fig. 5.2 and (5.2) it follows that

$$\frac{E_{\mathcal{B}}(r(j))}{E_0} = \frac{R - r(j)}{R} = 1 - \frac{r(j)}{R} = \frac{m}{j + m + 1} = \frac{1}{\ell + 1} \tag{5.56}$$

Thus, by combining (5.53) and (5.56) we obtain

$$L(\ell) = E_0 \tag{5.57}$$

as illustrated in Fig. 5.2.

In Fig. 5.3 we show the error burst length exponent $L(\ell)$ for various rates for the BSC. The transition points between the expurgation, random coding, and sphere-packing regions are marked with •.

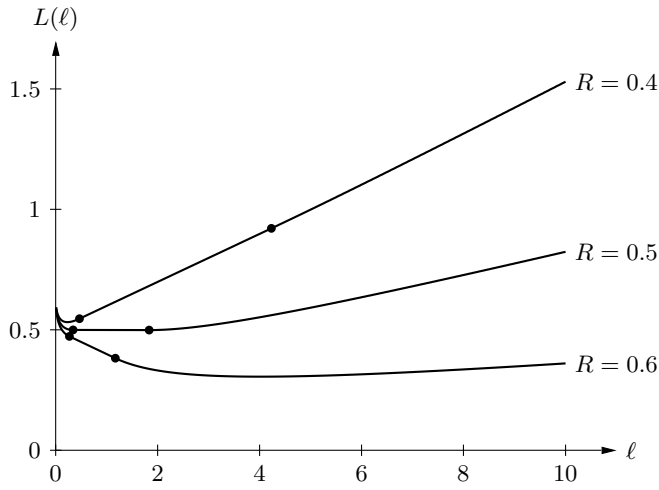


Figure 5.3 The error burst length exponent $L(\ell)$ for various rates for the BSC with crossover probability $\epsilon = 0.045$.

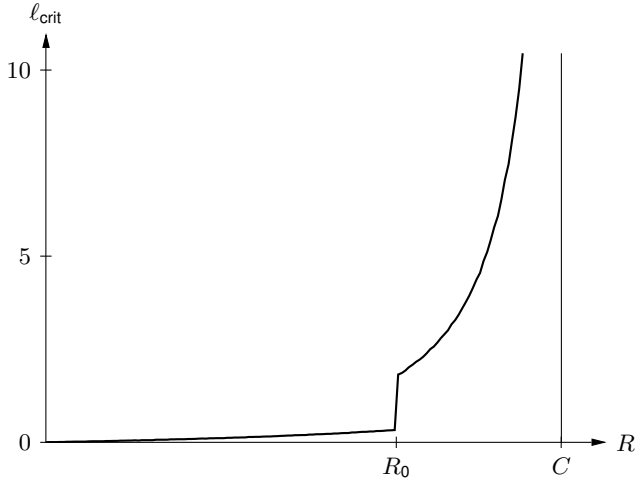


Figure 5.4 The critical length ℓ_{crit} for the BSC with crossover probability $\epsilon = 0.045$ corresponding to $R = R_0 = 1/2$.

The value of $\ell = (j+1)/m$ for which $L(\ell)$ achieves its minimum is called the *critical length* and denoted ℓ_{crit}. It is the most probable (normalized) length of an error burst. In Fig. 5.4 we show the critical length as a function of the rate R. We notice that ℓ_{crit} approaches infinity when R approaches the channel capacity. In other words, when we communicate at rates close to the channel capacity the typical error bursts are very long. Thus, we cannot draw any conclusions about the burst error

probability from the initial part of the path weight enumerator. The discontinuity at the computational cutoff rate R_0 is due to the straight line in the block coding exponent $E_{\mathcal{B}}(r(j))$ for rates $R_{\text{exp}} \le r(j) < R_{\text{crit}}$. For $R = R_0$, we have $E_0 = R_0$. Thus, the critical length makes a jump at R_0 from

$$\ell_{\text{crit}} = \frac{R_0}{E_{\mathcal{B}}(R_{\text{exp}})} - 1 = \frac{R_{\text{exp}}}{R_0 - R_{\text{exp}}} \quad \text{for } R = R_0 - \varepsilon \tag{5.58}$$

to

$$\ell_{\text{crit}} = \frac{R_0}{E_{\mathcal{B}}(R_{\text{crit}})} - 1 = \frac{R_{\text{crit}}}{R_0 - R_{\text{crit}}} \quad \text{for } R = R_0 + \varepsilon \tag{5.59}$$

Since the critical length is derived from the block coding exponent $E_{\mathcal{B}}(r)$, which is a lower bound on the "true" error exponent, it follows that the critical length shown in Fig. 5.4 is a lower bound on the "true" value. However, in Section 5.3 we show that in the sphere-packing region the block coding exponent is tight. Hence, ℓ_{crit} assumes the correct value for rates $R_0 < R < C$.

5.2 BOUNDS FOR PERIODICALLY TIME-VARYING CONVOLUTIONAL CODES

In order to upper-bound the burst error probability via the distribution of the error burst lengths for the ensemble $\mathcal{E}(b, c, m, T)$, we have to restrict the burst error probability to those error events caused by bursts of lengths that are at most T. Denote this restricted burst error probability by P_{B}^{T}. Then we use the union bound and obtain from (5.52) that

$$\begin{aligned} P_{\text{B}}^{\text{T}} &\le \sum_{j=0}^{T-1} P(\mathcal{L}(j)) \\ &\le T 2^{-(E_{\mathcal{B}}(r(j_{\min}))+o(1))(j_{\min}+m+1)c} \end{aligned} \tag{5.60}$$

where the real number $j_{\min}$ is the value of j (here we allow j to be any real number) that minimizes the exponent of the right-hand side of (5.52).

Let us introduce

$$f(j) = E_{\mathcal{B}}(r(j))(j + m + 1)c \tag{5.61}$$

Then we have

$$P_{\text{B}}^{\text{T}} \le T 2^{-f(j_{\min})+o(1)(j_{\min}+m+1)c} \tag{5.62}$$

In the expurgation region, that is, for rates $0 \le r(j) < R_{\text{exp}}$, we have

$$f(j) = -\rho(j + m + 1)c \log z \tag{5.63}$$

where z is the Bhattacharyya parameter (4.20) and ρ is the Gilbert-Varshamov parameter, that is,

$$\begin{aligned} r(j) &= \frac{j+1}{j+m+1} R = 1 - h(\rho) \\ &= 1 + \rho \log \rho + (1 - \rho) \log(1 - \rho) \end{aligned} \tag{5.64}$$

Regarding j as a continuous variable and taking the derivatives of (5.63) and (5.64), we obtain (notice that ρ is a function of j)

$$f'(j) = -\rho'(j+m+1)c\log z - \rho c\log z \tag{5.65}$$

and

$$r'(j) = \frac{m}{(j+m+1)^2}R = \rho' \log\frac{\rho}{1-\rho} \tag{5.66}$$

Since

$$f'(j_{\min}) = 0 \tag{5.67}$$

we obtain from (5.65) and (5.66) that

$$\begin{aligned}\rho &= -\rho'(j_{\min}+m+1)\\ &= -\frac{mR}{(j_{\min}+m+1)\log\frac{\rho}{1-\rho}}\end{aligned} \tag{5.68}$$

where

$$\rho = 1 - h^{-1}(r(j_{\min})) \tag{5.69}$$

Hence, by inserting (5.68) into (5.63) we obtain

$$\begin{aligned}f(j_{\min}) &= -\rho(j_{\min}+m+1)c\log z\\ &= \frac{mcR}{\log(\frac{\rho}{1-\rho})}\log z\end{aligned} \tag{5.70}$$

It is beneficial to introduce a parameter s, $1 < s < \infty$, such that

$$\rho = \frac{z^{1/s}}{1+z^{1/s}} \tag{5.71}$$

(If $0 < \rho < \frac{z}{1+z}$, then we can always find such an s.) Then we obtain from (5.68) that

$$\begin{aligned}\rho\log\frac{\rho}{1-\rho} &= -\frac{mR}{j_{\min}+m+1} = r(j_{\min}) - R\\ &= 1 + \rho\log\rho + (1-\rho)\log(1-\rho) - R\end{aligned} \tag{5.72}$$

where the last equality follows from (5.64). Using (5.71), we can rewrite (5.72) as

$$R = 1 + \log(1-\rho) = 1 - \log(1+z^{1/s}) \tag{5.73}$$

By inserting (5.71) into (5.70), we obtain

$$f(j_{\min})/mc = sR = s(1 - \log(1+z^{1/s})) \tag{5.74}$$

where the last equality follows from (5.73).

Next, we introduce the *expurgation function*

$$G_{\text{exp}}(s) \stackrel{\text{def}}{=} s(1 - \log(1+z^{1/s})) \tag{5.75}$$

where $1 < s < \infty$ and z is the Bhattacharyya parameter. Hence, in the expurgation region we have the parametric dependence

$$\begin{aligned} f(j_{\min})/mc &= G_{\text{exp}}(s) \\ R &= G_{\text{exp}}(s)/s \end{aligned} \tag{5.76}$$

where $1 < s < \infty$ or, equivalently, $0 < R < R_0$.

From (5.63), (5.71), and (5.76) it follows that

$$\begin{aligned} j_{\min} + m + 1 &= \frac{-G_{\text{exp}}(s)(1+z^{1/s})m}{z^{1/s}\log z} \\ &= -\frac{(1-\log(1+z^{1/s}))(1+z^{1/s})m}{z^{1/s}\log z^{1/s}} = -\frac{R(1+z^{1/s})m}{z^{1/s}\log z^{1/s}} \end{aligned} \tag{5.77}$$

From (5.75) and (5.76) we have

$$z^{1/s} = 2^{1-R} - 1 \tag{5.78}$$

which we insert into (5.77) and obtain

$$\frac{j_{\min}+1}{m} = -\frac{R2^{1-R} + (2^{1-R}-1)\log(2^{1-R}-1)}{(2^{1-R}-1)\log(2^{1-R}-1)}, \quad 0 < R < R_0 \tag{5.79}$$

In particular, when $R \to R_0$ or, equivalently, when $s \to 1$, we have

$$\lim_{s\to 1}\frac{j_{\min}+1}{m} = -\frac{(1+z)\log\frac{1+z}{2} + z\log z}{z\log z} = \frac{R_{\text{exp}}}{R_0 - R_{\text{exp}}} \tag{5.80}$$

which coincides with (5.58).

In the random coding region, that is, for rates $R_{\text{exp}} \le r(j) < R_0$, we have (cf. (5.50))

$$\begin{aligned} f(j) &= (R_0 - r(j))(j+m+1)c \\ &= R_0 mc - r(j)(j+m+1)c + (j+1)R_0 c \\ &= R_0 mc - (j+1)Rc + (j+1)R_0 c \\ &= R_0 mc + (j+1)(R_0 - R)c \end{aligned} \tag{5.81}$$

The minimum value of $f(j)$ is obtained for $j = 0$. Hence, for the random coding region we have

$$f(0)/mc = R_0 + (R_0 - R)/m \tag{5.82}$$

This bound is valid for the same region of rates R as the expurgation bound, viz., $0 \le R < R_0$, and since it is weaker than (5.76) it can be omitted.

Finally, we consider the sphere-packing region, that is, rates $R_0 \le r(j) < C$, where

$$f(j) = \left(\rho\log\frac{\rho}{\epsilon} + (1-\rho)\log\frac{1-\rho}{1-\epsilon}\right)(j+m+1)c \tag{5.83}$$

Then the minimizing value of j, viz., $j_{\min}$, is the root of

$$f'(j_{\min}) = \left(\rho'(j_{\min}+m+1)c\log\frac{\rho(1-\epsilon)}{(1-\rho)\epsilon} + \left(\rho\log\frac{\rho}{\epsilon}+(1-\rho)\log\frac{1-\rho}{1-\epsilon}\right)c\right)\ln 2 = 0 \tag{5.84}$$

where ρ' satisfies (5.66). Using (5.84) we can rewrite (5.83) as

$$\begin{aligned} f(j_{\min}) &= -\rho'(j_{\min}+m+1)^2 c\log\frac{\rho(1-\epsilon)}{(1-\rho)\epsilon} \\ &= -Rmc\frac{\log\frac{\rho(1-\epsilon)}{(1-\rho)\epsilon}}{\log\frac{\rho}{1-\rho}} \end{aligned} \tag{5.85}$$

where the last equality follows from (5.66).

Again it is beneficial to introduce a parameter s. Here we choose s, $0 \le s \le 1$, such that

$$\rho = \frac{\epsilon^{\frac{1}{1+s}}}{\epsilon^{\frac{1}{1+s}} + (1-\epsilon)^{\frac{1}{1+s}}} \tag{5.86}$$

or, equivalently,

$$\log\frac{\rho}{1-\rho} = \frac{1}{1+s}\log\frac{\epsilon}{1-\epsilon} \tag{5.87}$$

Then, from (5.85) we obtain

$$f(j_{\min})/mc = sR \tag{5.88}$$

From (5.84) it follows that

$$\rho'(j_{\min}+m+1) = -\frac{\rho\log\frac{\rho}{\epsilon}+(1-\rho)\log\frac{1-\rho}{1-\epsilon}}{\log\frac{\rho(1-\epsilon)}{(1-\rho)\epsilon}} \tag{5.89}$$

We insert (5.86) inside the logarithms in the numerator of (5.89), split the denominator, and obtain

$$\begin{aligned} \rho'(j_{\min}+m+1) &= -\frac{\rho\log\frac{\epsilon^{-\frac{s}{1+s}}}{\epsilon^{\frac{1}{1+s}}+(1-\epsilon)^{\frac{1}{1+s}}} + (1-\rho)\log\frac{(1-\epsilon)^{-\frac{s}{1+s}}}{\epsilon^{\frac{1}{1+s}}+(1-\epsilon)^{\frac{1}{1+s}}}}{\log\frac{\rho}{1-\rho}-\log\frac{\epsilon}{1-\epsilon}} \\ &= -\frac{-\frac{s}{1+s}(\rho\log\epsilon+(1-\rho)\log(1-\epsilon)) - \log\left(\epsilon^{\frac{1}{1+s}}+(1-\epsilon)^{\frac{1}{1+s}}\right)}{\log\frac{\rho}{1-\rho}-(1+s)\log\frac{\rho}{1-\rho}} \end{aligned} \tag{5.90}$$

where we used (5.87) in order to simplify the denominator. From (5.86) it follows that

$$\log\epsilon = (1+s)\left(\log\rho+\log\left(\epsilon^{\frac{1}{1+s}}+(1-\epsilon)^{\frac{1}{1+s}}\right)\right) \tag{5.91}$$

and

$$\log(1-\epsilon) = (1+s)\left(\log(1-\rho) + \log\left(\epsilon^{\frac{1}{1+s}} + (1-\epsilon)^{\frac{1}{1+s}}\right)\right) \tag{5.92}$$

Inserting (5.91) and (5.92) into (5.90) yields

$$\begin{aligned}
&\rho'(j_{\min}+m+1) \\
&\quad = \frac{-s(\rho\log\rho + (1-\rho)\log(1-\rho)) - (1+s)\log\left(\epsilon^{\frac{1}{1+s}} + (1-\epsilon)^{\frac{1}{1+s}}\right)}{s\log\frac{\rho}{1-\rho}} \\
&\quad = \frac{-(\rho\log\rho + (1-\rho)\log(1-\rho)) - \frac{1+s}{s}\log\left(\epsilon^{\frac{1}{1+s}} + (1-\epsilon)^{\frac{1}{1+s}}\right)}{\log\frac{\rho}{1-\rho}}
\end{aligned} \tag{5.93}$$

Equality (5.66) can be rewritten as

$$\rho'(j_{\min}+m+1) = \frac{mR}{(j_{\min}+m+1)\log\frac{\rho}{1-\rho}} \tag{5.94}$$

and from (5.64) it follows that

$$R - r(j) = \frac{mR}{j+m+1} \tag{5.95}$$

Combining (5.94) and (5.95) yields

$$\begin{aligned}
\rho'(j_{\min}+m+1) &= \frac{R-r(j)}{\log\frac{\rho}{1-\rho}} \\
&= \frac{R-1-\rho\log\rho-(1-\rho)\log(1-\rho)}{\log\frac{\rho}{1-\rho}}
\end{aligned} \tag{5.96}$$

where the last equality follows from (5.64). From (5.93) and (5.96) we conclude that

$$R = 1 - \frac{1+s}{s}\log\left(\epsilon^{\frac{1}{1+s}} + (1-\epsilon)^{\frac{1}{1+s}}\right) \tag{5.97}$$

Let us define the *Gallager function* for the BSC to be

$$G(s) = s - (1+s)\log\left(\epsilon^{\frac{1}{1+s}} + (1-\epsilon)^{\frac{1}{1+s}}\right) \tag{5.98}$$

Then, for the sphere-packing region we obtain from (5.88) and (5.97) the following parametric expression:

$$\begin{aligned}
f(j_{\min})/mc &= G(s) \\
R &= G(s)/s
\end{aligned} \tag{5.99}$$

where $0 < s \leq 1$. (The existence of a solution for all R, $0 \leq R < C$, of the second equation in (5.99) follows from the properties of $G(s)$ shown in Problem 5.1.)

From (5.83), (5.97), (5.98), and (5.99) it follows that

$$\begin{aligned} j_{\min} + m + 1 &= \frac{G(s)m}{\rho \log \frac{\rho}{\epsilon} + (1-\rho)\log\frac{1-\rho}{1-\epsilon}} \\ &= -\frac{sRm}{\rho \log \epsilon^{\frac{s}{1+s}} + (1-\rho)\log(1-\epsilon)^{\frac{s}{1+s}} + \log\left(\epsilon^{\frac{1}{1+s}} + (1-\epsilon)^{\frac{1}{1+s}}\right)} \\ &= -\frac{Rm}{\rho\log\rho + (1-\rho)\log(1-\rho) + \frac{1+s}{s}\log\left(\epsilon^{\frac{1}{1+s}} + (1-\epsilon)^{\frac{1}{1+s}}\right)} \end{aligned} \tag{5.100}$$

or, equivalently,

$$\frac{j_{\min}+1}{m} = -\frac{\rho\log\rho + (1-\rho)\log(1-\rho) + 1}{\rho\log\rho + (1-\rho)\log(1-\rho) + 1 - R} \tag{5.101}$$

where ρ is defined by (5.86). In particular, for $R = R_0$ we have

$$\frac{j_{\min}+1}{m} = \frac{R_{\text{crit}}}{R_0 - R_{\text{crit}}} \tag{5.102}$$

which coincides with (5.59).

From (5.60) and (5.61) it follows that the restricted burst error probability can be upper-bounded by

$$P_{\text{B}}^{\text{T}} \le T 2^{-f(j_{\min}) + o(1)(j+m+1)c} \tag{5.103}$$

By choosing $T = O(m^2)$, say, we can summarize our efforts in the following:

Theorem 5.6 There exists a binary rate $R = b/c$, periodically time-varying convolutional code encoded by a polynomial, periodically time-varying generator matrix of memory m and period T, where $T = O(m^2)$, such that the burst error probability due to error bursts of lengths at most T from a maximum-likelihood decoder when used to communicate over the BSC is upper-bounded by

$$P_{\text{B}}^{\text{T}} \le 2^{-(E_{\mathcal{C}}(R) + o(1))mc} \tag{5.104}$$

where

$$E_{\mathcal{C}}(R) = \begin{cases} \begin{cases} G_{\text{exp}}(s) \\ R = G_{\text{exp}}(s)/s, \end{cases} & 1 \le s < \infty, \quad 0 \le R \le R_0 \\ \begin{cases} G(s) \\ R = G(s)/s, \end{cases} & 0 < s \le 1, \quad R_0 \le R < C \end{cases} \tag{5.105}$$

is the *convolutional coding exponent* for the expurgation and sphere-packing regions, respectively, and where R_0 is the computational cutoff rate (5.6), $G_{\text{exp}}(s)$ is the expurgation function (5.75), $G(s)$ is the Gallager function (5.98), and C is the channel capacity for the BSC.

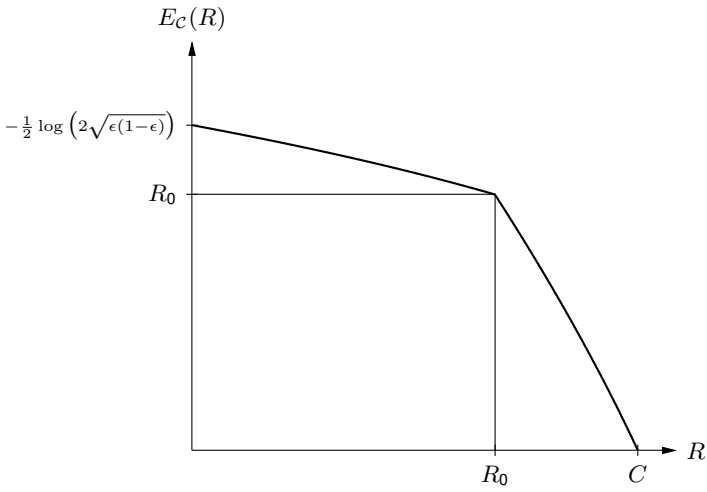


Figure 5.5 The convolutional coding exponent $E_{\mathcal{C}}(R)$ for the BSC with $\epsilon = 0.045$, that is, $R_0 = 1/2$.

From (5.73) it follows that $R \to 0$ when $s \to \infty$. Hence, from (5.75) we obtain

$$E_{\mathcal{C}}(0) = -\frac{1}{2}\log z \tag{5.106}$$

where z is the Bhattacharyya parameter (4.20). In Fig. 5.5 we show the convolutional coding exponent $E_{\mathcal{C}}(R)$ for the BSC with $\epsilon = 0.045$, which corresponds to $R_0 = 1/2$.

Forney [For74] showed that the convolutional coding exponent can easily be constructed from the block coding exponent and vice versa. Let r denote the rate of the block code $\mathcal{B}$ that gives the largest contribution to $P_{\mathrm{B}}^{\mathrm{T}}$, that is, $r = r(j_{\min})$. Then, for the expurgation region we combine (5.64), (5.71), (5.73), (5.75), and (5.76) and obtain

$$\begin{aligned} G_{\exp}(s)\left(1 - \frac{r}{R}\right) &= sR - (1 - h(\rho))s \\ &= \frac{\log z}{\log \frac{\rho}{1-\rho}}(\log(1-\rho) + h(\rho)) = -\rho \log z \\ &= E_{\mathcal{B}}(r) \end{aligned} \tag{5.107}$$

where $0 \le r < R_{\exp}$ and $0 \le R < R_0$.

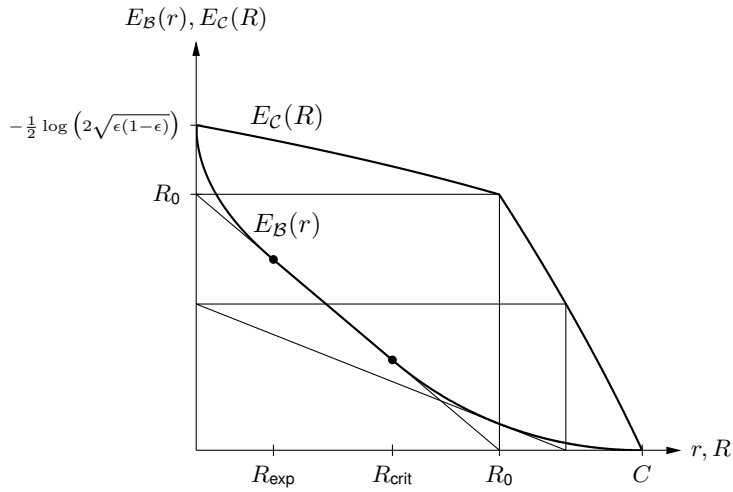


Figure 5.6 Geometrical construction of $E_{\mathcal{C}}(R)$ from $E_{\mathcal{B}}(r)$ and vice versa for the BSC with crossover probability $\epsilon = 0.045$.

Similarly, for the sphere-packing region we combine (5.51), (5.86), (5.87), (5.98), and (5.99) and obtain

$$\begin{aligned} G(s)\left(1-\frac{r}{R}\right) &= G(s)\left(1-\frac{(1-h(\rho))s}{G(s)}\right) \\ &= -(1+s)\log\left(\epsilon^{\frac{1}{1+s}}+(1-\epsilon)^{\frac{1}{1+s}}\right)+h(\rho)s \\ &= -\log\epsilon+(1+s)\log\rho+h(\rho)s \\ &= -\log\epsilon+\frac{\log\frac{\epsilon}{1-\epsilon}}{\log\frac{\rho}{1-\rho}}\log\rho+h(\rho)\left(\frac{\log\frac{\epsilon}{1-\epsilon}}{\log\frac{\rho}{1-\rho}}-1\right) \\ &= \rho\log\frac{\rho}{\epsilon}+(1-\rho)\log\frac{1-\rho}{1-\epsilon}=E_{\mathcal{B}}(r) \end{aligned} \tag{5.108}$$

where $R_{\text{crit}} \le r < C$ and $R_0 \le R < C$.

Hence, it follows from (5.107) and (5.108) that the convolutional and block coding exponents are related as

$$E_{\mathcal{C}}(R) = \frac{R}{R-r}E_{\mathcal{B}}(r) \tag{5.109}$$

The corresponding geometrical construction is shown in Fig. 5.6. Notice that the random coding region for the block coding exponent collapses into the point (R_0, R_0) on the $E_{\mathcal{C}}(R)$ curve.

In Section 4.2 we showed that the free distance is the principal determiner for the error probability when we use maximum-likelihood decoding at large signal-to-noise ratios, and in Section 3.5 we showed that the free distance for convolutional codes encoded by systematic, polynomial, encoding matrices is off by a factor $(1 - R)$

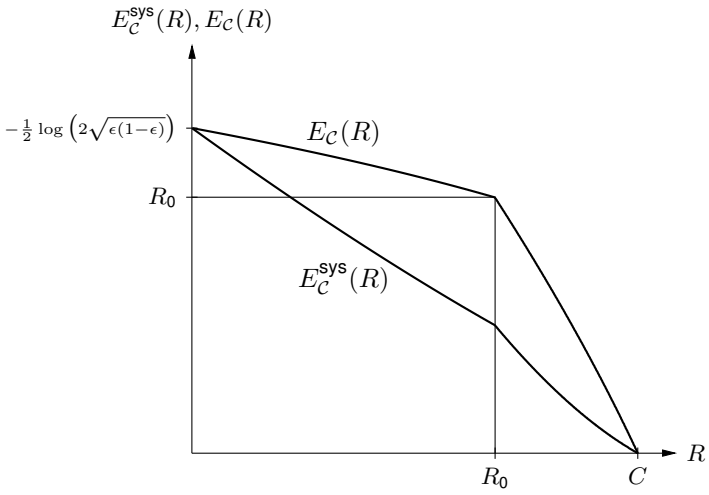


Figure 5.7 The convolutional coding exponents $E_{\mathcal{C}}^{\text{sys}}(R)$ and $E_{\mathcal{C}}(R)$ for the BSC with $\epsilon = 0.045$.

compared to the free distance for convolutional codes encoded by nonsystematic generator matrices. In the proof of Heller's bound (Theorem 3.22), the "effective" length when we used systematic, polynomial, encoding matrices was only $(m(1-R)+i)c$ instead of $(m+i)c$ for nonsystematic generator matrices. We have the same reduction in the derivation of the corresponding upper bound on the error probability. Hence, we have the following:

Theorem 5.7 There exists a binary rate $R = b/c$, periodically time-varying convolutional code encoded by a *systematic*, polynomial, periodically time-varying encoding matrix of memory m and period T, where $T = O(m^2)$, such that the burst error probability due to error bursts of lengths at most T from a maximum-likelihood decoder when used to communicate over the BSC is upper-bounded by

$$P_{\text{B}}^{\text{T}} \leq 2^{-(E_{\mathcal{C}}^{\text{sys}}(R)+o(1))mc}, \quad 0 \leq R < C \tag{5.110}$$

where

$$E_{\mathcal{C}}^{\text{sys}}(R) = E_{\mathcal{C}}(R)(1-R) \tag{5.111}$$

is the *convolutional coding exponent* for convolutional codes encoded by systematic, polynomial encoding matrices.

In Fig. 5.7 we compare the convolutional coding exponents $E_{\mathcal{C}}^{\text{sys}}(R)$ and $E_{\mathcal{C}}(R)$ for the BSC with $\epsilon = 0.045$ which corresponds to $R_0 = 1/2$.

We will conclude this section by upper-bounding the bit error probability via the distribution of the error burst lengths for the ensemble $\mathcal{E}(b, c, m, T)$. Thus, we have to restrict the bit error probability to those events caused by bursts of lengths that are at most T. We denote this restricted bit error probability by P_{B}^{T}.

Since an error burst of length $j+1$ can cause at most $(j+1)b$ bit errors, we have

$$P_{\mathrm{B}}^{\mathrm{T}} \leq \sum_{j=0}^{T-1} (j+1)b\, P(\mathcal{L}(j)) \tag{5.112}$$

Combining (5.112) and (5.49), we obtain

$$P_{\mathrm{B}}^{\mathrm{T}} \leq \sum_{j=0}^{T-1} (j+1)b\, 2^{-(E_{\mathcal{B}}(r(j))+o(1))(j+m+1)c} \tag{5.113}$$

Now we rewrite $(j+1)b$ as

$$(j+1)b = 2^{\log((j+1)b)} = 2^{o(1)(j+m+1)} \tag{5.114}$$

which is inserted into (5.113). Thus, the restricted bit error probability is upper-bounded by

$$\begin{aligned} P_{\mathrm{B}}^{\mathrm{T}} &\leq \sum_{j=0}^{T-1} (j+1)b\, 2^{-(E_{\mathcal{B}}(r(j))+o(1))(j+m+1)c} \\ &\leq T 2^{-(E_{\mathcal{B}}(r(j_{\min}))+o(1))(j_{\min}+m+1)c} \end{aligned} \tag{5.115}$$

which coincides with (5.60). Hence, for the restricted bit error probability we obtain the following as a counterpart to Theorem 5.6:

Theorem 5.8 There exists a binary rate $R = b/c$, periodically time-varying convolutional code encoded by a polynomial, periodically time-varying generator matrix of memory m and period T, where $T = O(m^2)$, such that the bit error probability due to error bursts of lengths at most T from a maximum-likelihood decoder when used to communicate over the BSC is upper-bounded by

$$P_{\mathrm{B}}^{\mathrm{T}} \leq 2^{-(E_{\mathcal{C}}(R)+o(1))mc}, \quad 0 \leq R < C \tag{5.116}$$

where the convolutional coding exponent $E_{\mathcal{C}}(R)$ is given by (5.105).

As a counterpart to Theorem 5.7 we have the following:

Theorem 5.9 There exists a binary rate $R = b/c$, periodically time-varying convolutional code encoded by a systematic, polynomial, periodically time-varying encoding matrix of memory m and period T, where $T = O(m^2)$, such that the bit error probability due to error bursts of lengths at most T from a maximum-likelihood decoder when used to communicate over the BSC is upper-bounded by

$$P_{\mathrm{B}}^{\mathrm{T}} \leq 2^{-(E_{\mathcal{C}}^{\mathrm{sys}}(R)+o(1))mc}, \quad 0 \leq R < C \tag{5.117}$$

where the convolutional coding exponent $E_{\mathcal{C}}^{\mathrm{sys}}(R)$ is given by (5.111).

Asymptotically, for increasing memory, the upper bound on the restricted bit error probability decreases exponentially with the same exponent as the upper bound on the restricted burst error probability.

5.3 LOWER ERROR PROBABILITY BOUNDS FOR CONVOLUTIONAL CODES

As a counterpart to our upper bounds on the burst and bit error probabilities, we will derive lower bounds; that is, we will be concerned with finding the minimum burst and bit error probabilities any rate R convolutional code of memory m must exceed. First, we need the corresponding bound for block codes. For simplicity we study only the BSC.

Consider an arbitrary rate r binary block code of block length N and assume that it is used to communicate over the BSC with crossover probability ϵ. The number of codewords is

$$M = 2^{rN} \tag{5.118}$$

Our goal is to obtain a lower bound on error probability $P(\mathcal{E})$, where

$$P(\mathcal{E}) = 1 - P(\overline{\mathcal{E}}) \stackrel{\text{def}}{=} 1 - \min_i \left\{ P\left(\overline{\mathcal{E}} \mid i\right) \right\} = \max_i \left\{ P\left(\mathcal{E} \mid i\right) \right\} \tag{5.119}$$

where $P\left(\overline{\mathcal{E}} \mid i\right)$ and $P\left(\mathcal{E} \mid i\right)$ denote the probability of correct and erroneous decoding, respectively, when the codeword $\boldsymbol{v}^{(i)}$, $i = 0, 1, \ldots, M-1$, is transmitted. Let $\mathcal{D}_i$ denote the *decoding region* for the ith codeword, that is,

$$\mathcal{D}_i \stackrel{\text{def}}{=} \left\{ \boldsymbol{r} \;\middle|\; \boldsymbol{r} \quad \text{is decoded as } \boldsymbol{v}^{(i)} \right\} \tag{5.120}$$

$i = 0, 1, \ldots, M-1$. Then it follows that

$$P(\overline{\mathcal{E}} \mid i) = \sum_{\boldsymbol{r} \in \mathcal{D}_i} P\left(\boldsymbol{r} \;\middle|\; \boldsymbol{v}^{(i)}\right) \tag{5.121}$$

Clearly,

$$\min_i \{|\mathcal{D}_i|\} \leq \frac{2^N}{M} = 2^{(1-r)N} \tag{5.122}$$

For the BSC we have

$$P\left(\boldsymbol{r} \;\middle|\; \boldsymbol{v}^{(i)}\right) = \epsilon^{d_{\mathrm{H}}\left(\boldsymbol{r} \mid \boldsymbol{v}^{(i)}\right)} (1-\epsilon)^{N - d_{\mathrm{H}}\left(\boldsymbol{r} \mid \boldsymbol{v}^{(i)}\right)} \tag{5.123}$$

where $d_{\mathrm{H}}(\boldsymbol{r}, \boldsymbol{v}^{(i)})$ is the Hamming distance between the received word $\boldsymbol{r}$ and the ith codeword $\boldsymbol{v}^{(i)}$.

The conditional probability of receiving $\boldsymbol{r}$ when the ith codeword is transmitted, $P\left(\boldsymbol{r} \;\middle|\; \boldsymbol{v}^{(i)}\right)$, is monotonically increasing with decreasing $d_{\mathrm{H}}(\boldsymbol{r}, \boldsymbol{v}^{(i)})$. Hence, in order to achieve its maximal value, the sum (5.121) should be taken over those received words $\boldsymbol{r}$ that are equal to $\boldsymbol{v}^{(i)}$, at Hamming distance 1 from $\boldsymbol{v}^{(i)}$, at Hamming distance 2 from $\boldsymbol{v}^{(i)}$, and so on until we have reached $|\mathcal{D}_i|$ terms.

Let k_0 denote the largest integer such that

$$\sum_{k=0}^{k_0 - 1} \binom{N}{k} \leq |\mathcal{D}_i| \tag{5.124}$$

and let

$$0 \le A_i = |\mathcal{D}_i| - \sum_{k=0}^{k_0-1} \binom{N}{k} < \binom{N}{k_0} \tag{5.125}$$

Then for a given value of $|\mathcal{D}_i|$, the maximum possible value of $P\left(\overline{\mathcal{E}} \mid i\right)$ will be

$$P\left(\overline{\mathcal{E}} \mid i\right) = \sum_{k=0}^{k_0-1} \binom{N}{k} \epsilon^k (1-\epsilon)^{N-k} + A_i \epsilon^{k_0} (1-\epsilon)^{N-k_0} \tag{5.126}$$

or, equivalently, since

$$\sum_{k=0}^{N} \binom{N}{k} \epsilon^k (1-\epsilon)^{N-k} = 1 \tag{5.127}$$

the smallest possible value of the error probability $P\left(\mathcal{E} \mid i\right)$ will be

$$\begin{aligned} P\left(\mathcal{E} \mid i\right) &= 1 - P\left(\overline{\mathcal{E}} \mid i\right) \\ &= \sum_{k=k_0}^{N} \binom{N}{k} \epsilon^k (1-\epsilon)^{N-k} - A_i \epsilon^{k_0} (1-\epsilon)^{N-k_0} \\ &> \binom{N}{k_0+1} \epsilon^{k_0+1} (1-\epsilon)^{N-k_0-1} \end{aligned} \tag{5.128}$$

where the inequality follows from (5.125).

From (5.122) and (5.125) we obtain

$$2^{(1-r)N} \ge \min_i \{|\mathcal{D}_i|\} \ge \sum_{k=0}^{k_0-1} \binom{N}{k} > \binom{N}{k_0-1} \tag{5.129}$$

By combining (5.119), (5.128), and (5.129) we obtain the following parametric lower bound of error probability for rate r block codes:

$$\begin{aligned} 2^{(1-r)N} &> \binom{N}{k_0-1} \\ P(\mathcal{E}) &> \binom{N}{k_0+1} \epsilon^{k_0+1} (1-\epsilon)^{N-k_0-1} \end{aligned} \tag{5.130}$$

Remark: The bound (5.130) is known as the *sphere-packing bound* for the following reason. The set of sequences at distance k_0 or less from a codeword can be interpreted as a *sphere* of radius k_0 around that codeword. If we could choose codewords such that the set of spheres of radius k_0 around the different codewords exhausted the space of binary N-tuples and intersected each other only on the outer shells of radius k_0, then the error probability would be lower-bounded by (5.130).

To obtain a more illuminative form of our lower bound, we need the following:

Lemma 5.10 The binomial coefficient $\binom{N}{k}$ satisfies the inequality

$$\binom{N}{k} \geq \sqrt{\frac{N}{8k(N-k)}} 2^{h\left(\frac{k}{N}\right)N} \tag{5.131}$$

where $h(\cdot)$ is the binary entropy function (1.22).

Proof: The lemma follows from a refinement of Stirling's formula [Fel68]:

$$\sqrt{2\pi n}\, n^n e^{-n} e^{(12n+1)^{-1}} < n! < \sqrt{2\pi n}\, n^n e^{-n} e^{(12n)^{-1}} \tag{5.132}$$

Hence, we have

$$\begin{aligned}\binom{N}{k} &= \frac{N!}{k!(N-k)!} \\ &> \sqrt{\frac{N}{2\pi k(N-k)}} \frac{N^N}{k^k (N-k)^{N-k}} e^{(12N+1)^{-1} - (12k)^{-1} - (12(N-k))^{-1}}\end{aligned} \tag{5.133}$$

We note that

$$-(12k)^{-1} - (12(N-k))^{-1} \geq -1/9, \tag{5.134}$$

except for $k = 1$, $N - k = 1$; $k = 1$, $N - k = 2$; and $k = 2$, $N - k = 1$. Thus, with these exceptions,

$$e^{(12N+1)^{-1} - (12k)^{-1} - (12(N-k))^{-1}} > e^{-1/9} > \sqrt{\frac{2\pi}{8}} \tag{5.135}$$

and

$$\begin{aligned}\binom{N}{k} &> \sqrt{\frac{N}{2\pi k(N-k)}} \sqrt{\frac{2\pi}{8}} 2^{\left(-\frac{k}{N}\log\frac{k}{N} - \left(1-\frac{k}{N}\right)\log\left(1-\frac{k}{N}\right)\right)N} \\ &= \sqrt{\frac{N}{8k(N-k)}} 2^{h\left(\frac{k}{N}\right)N}\end{aligned} \tag{5.136}$$

For the exceptions the inequality can easily be verified numerically; in fact, for $k = N - k = 1$ we have equality. ∎

We have $k_0 < N/2$, and, hence, we can rewrite (5.131) as

$$\binom{N}{k_0} \geq \frac{1}{\sqrt{8\rho(1-\rho)N}} 2^{h(\rho)N} \tag{5.137}$$

where $\rho = k_0/N < 1/2$. Then from (5.130) we obtain

$$\begin{aligned}1 - r &> h(\rho_{-1}) - \frac{1}{2N}\log(8\rho_{-1}(1-\rho_{-1})N) \\ &= h(\rho + o(1)) + o(1)\end{aligned} \tag{5.138}$$

where

$$\rho_{-1} = (k_0 - 1)/N = \rho - 1/N \tag{5.139}$$

and

$$\begin{aligned} -\frac{1}{N}\log P(\mathcal{E}) &< \rho_{+1}\log\frac{\rho_{+1}}{\epsilon} + (1-\rho_{+1})\log\frac{1-\rho_{+1}}{1-\epsilon} \\ &\quad + \frac{1}{2N}\log(8\rho_{+1}(1-\rho_{+1})N) \\ &= (\rho + o(1))\log\frac{\rho + o(1)}{\epsilon} + (1-\rho+o(1))\log\frac{1-\rho+o(1)}{1-\epsilon} + o(1) \end{aligned} \tag{5.140}$$

where

$$\rho_{+1} = (k_0 + 1)/N = \rho + 1/N \tag{5.141}$$

Both $h(\rho)$ and $\rho\log\frac{\rho}{\epsilon} + (1-\rho)\log\frac{1-\rho}{1-\epsilon}$ are continuously differentiable functions of ρ. It follows from (5.138) that

$$\rho > h^{-1}(1-r) + o(1) \tag{5.142}$$

When $N \to \infty$ we can always choose ρ to be arbitrarily close to $h^{-1}(1-r)$, and it follows that

$$-\frac{1}{N}P(\mathcal{E}) < \rho\log\frac{\rho}{\epsilon} + (1-\rho)\log\frac{1-\rho}{1-\epsilon} + o(1) \tag{5.143}$$

where

$$\rho = 1 - h^{-1}(r) \tag{5.144}$$

Thus, we have proved the next theorem.

Theorem 5.11 For any rate r block code $\mathcal{B}$ with block length N that is used to communicate over the BSC with crossover probability ϵ, the error probability is lower-bounded by

$$P(\mathcal{E}) > 2^{-\left(E_{\mathcal{B}}^{\text{sph}}(r) + o(1)\right)N} \tag{5.145}$$

where $E_{\mathcal{B}}^{\text{sph}}(r)$ is the *block sphere-packing exponent*

$$E_{\mathcal{B}}^{\text{sph}}(r) = \rho\log\frac{\rho}{\epsilon} + (1-\rho)\log\frac{1-\rho}{1-\epsilon} \tag{5.146}$$

and ρ is the Gilbert-Varshamov parameter (5.144).

We are now well prepared to derive the corresponding lower bound on the burst error probability for convolutional codes.

Lemma 5.12 For any rate $R = b/c$ convolutional code encoded by a generator matrix of memory m and overall constraint length $\nu = bm$ that is used to communicate over the BSC with crossover probability ϵ, the burst error probability is lower-bounded by

$$P_{\text{B}} > 2^{-\left(E_{\mathcal{C}}^{\text{sph}}(R) + o(1)\right)mc} \tag{5.147}$$

where $E_{\mathcal{C}}^{\text{sph}}(R)$ is the *convolutional sphere-packing exponent*

$$E_{\mathcal{C}}^{\text{sph}}(R) = G(s_0) \tag{5.148}$$

s_0 satisfies

$$R = G(s_0)/s_0 \tag{5.149}$$

and $0 \leq R < C$.

Proof: Theorem 5.11 states that for any $\varepsilon > 0$ there exists a block length N_ε such that for any $N > N_\varepsilon$ we have

$$P(\mathcal{E}) > 2^{-\left(E_{\mathcal{B}}^{\text{sph}}(r)+\varepsilon\right)N} \tag{5.150}$$

Analogously, Lemma 5.12 states that for any $\varepsilon > 0$ there exists a memory m_ε such that for any $m > m_\varepsilon$ we have

$$P_{\text{B}} > 2^{-\left(E_{\mathcal{C}}^{\text{sph}}(R)+\varepsilon\right)mc} \tag{5.151}$$

Now suppose that inequality (5.151) does not hold. Then as a consequence there exists a certain ε such that for any large enough m_ε there exists a memory $m > m_\varepsilon$ such that

$$P_{\text{B}} < 2^{-\left(E_{\mathcal{C}}^{\text{sph}}(R)+2\varepsilon\right)mc} \tag{5.152}$$

Then we terminate this convolutional code (with very good burst error probability according to (5.152)) into a block code $\mathcal{B}$ of rate r such that

$$E_{\mathcal{C}}^{\text{sph}}(R) = \frac{R}{R-r} E_{\mathcal{B}}^{\text{sph}}(r) \tag{5.153}$$

and that the block length is

$$N = \frac{E_{\mathcal{C}}^{\text{sph}}(R)}{E_{\mathcal{B}}^{\text{sph}}(r)} mc \tag{5.154}$$

It is easily shown that such a rate r exists for all R, $0 \leq R < C$.

Next we will show that this block code has such good error probability that it violates (5.150) and thus cannot exist.

Let us choose m_ε such that

$$\frac{E_{\mathcal{C}}^{\text{sph}}(R) - E_{\mathcal{B}}^{\text{sph}}(r)}{E_{\mathcal{B}}^{\text{sph}}(r)} m_\varepsilon < 2^{\varepsilon m_\varepsilon c} \tag{5.155}$$

Then for any $m > m_\varepsilon$ we have the number of information subblocks

$$j + 1 \stackrel{\text{def}}{=} \left(\frac{E_{\mathcal{C}}^{\text{sph}}(R)}{E_{\mathcal{B}}^{\text{sph}}(r)} - 1 \right) m < 2^{\varepsilon mc} \tag{5.156}$$

The error probability $P(\mathcal{E})$ for our block code $\mathcal{B}$ is equal to the probability that we will have an error burst starting in at least one of the $j+1$ positions. Thus, there exists an $m > m_\varepsilon$ such that

$$P(\mathcal{E}) < (j+1)P_{\mathrm{B}} < (j+1)2^{-\left(E_{\mathcal{C}}^{\mathrm{sph}}(R)+2\varepsilon\right)mc} \tag{5.157}$$

where the last inequality follows from (5.152). Combining (5.156) and (5.157) yields

$$P(\mathcal{E}) < 2^{-\left(E_{\mathcal{C}}^{\mathrm{sph}}(R)+\varepsilon\right)mc} \tag{5.158}$$

Inserting (5.154) into (5.158) gives

$$\begin{aligned} P(\mathcal{E}) &< 2^{-\left(E_{\mathcal{B}}^{\mathrm{sph}}(r)+\varepsilon E_{\mathcal{B}}^{\mathrm{sph}}(r)/E_{\mathcal{C}}^{\mathrm{sph}}(R)\right)N} \\ &= 2^{-\left(E_{\mathcal{B}}^{\mathrm{sph}}(r)+\varepsilon'\right)N} \end{aligned} \tag{5.159}$$

where

$$\varepsilon' = \frac{\varepsilon E_{\mathcal{B}}^{\mathrm{sph}}(r)}{E_{\mathcal{C}}^{\mathrm{sph}}(R)} \tag{5.160}$$

Thus, assuming that inequality (5.151) does not hold, we have constructed a block code whose error probability satisfies (5.159), which contradicts Theorem 5.11. Hence, we conclude that inequality (5.151) must hold and the proof is complete. ■

Next we will derive a simple lower bound that is much better than the bound in Lemma 5.12 for low rates.

From Heller's asymptotic bound (Corollary 3.23) it follows that for any codeword there always exists a codeword at distance

$$\begin{aligned} d &= \left(\frac{1}{2} + o(1)\right)(m+1)c \\ &= \left(\frac{1}{2} + o(1)\right)mc \end{aligned} \tag{5.161}$$

or less. Assume that such a codeword is transmitted over the BSC with crossover probability ϵ. Then, assuming that d is even, the burst error probability is lower-bounded by the probability that an error is caused by a codeword of weight d, that is,

$$\begin{aligned} P_{\mathrm{B}} &> \frac{1}{2}\binom{d}{d/2}\epsilon^{d/2}(1-\epsilon)^{d-d/2} + \sum_{i=d/2+1}^{d}\binom{d}{i}\epsilon^{i}(1-\epsilon)^{d-i} \\ &> \frac{1}{2}\binom{d}{d/2}\epsilon^{d/2}(1-\epsilon)^{d/2} \\ &> \frac{1}{2}\sqrt{\frac{d}{8(d/2)(d-d/2)}}2^{h\left(\frac{d/2}{d}\right)d}\epsilon^{d/2}(1-\epsilon)^{d/2} \\ &= \frac{1}{2\sqrt{2d}}\left(2\sqrt{\epsilon(1-\epsilon)}\right)^{d} = 2^{(\log z + o(1))d} \\ &= 2^{\left(\frac{1}{2}\log z + o(1)\right)mc} \end{aligned} \tag{5.162}$$

where the last inequality follows from Lemma 5.1 and z is the Bhattacharyya parameter (4.20).

It can be shown (Problem 5.2) that (5.162) also holds for d odd. Thus, we have the following:

Lemma 5.13 For any rate $R = b/c$ convolutional code $\mathcal{C}$ encoded by a generator matrix of memory m and overall constraint length $\nu = bm$ that is used to communicate over the BSC with crossover probability ϵ, the burst error probability is lower-bounded by

$$P_{\mathrm{B}} > 2^{\left(\frac{1}{2}\log z + o(1)\right)mc} \tag{5.163}$$

where z is the Bhattacharyya parameter (4.20).

In the sphere-packing region, $R_0 \le R < C$, in Section 5.2 we restricted the values of s to $0 < s \le 1$. Here we extend the permissible values of s to include $s > 1$. Then it follows from (5.86) and (5.97) that $\rho \to 1/2$ and $R \to 0$ when $s \to \infty$. Then, from (5.108) we obtain

$$E_{\mathcal{C}}^{\mathrm{sph}}(0) = -\log z \tag{5.164}$$

and, hence, at $R = 0$ the exponent $E_{\mathcal{C}}^{\mathrm{sph}}(0)$ is twice the exponent in Lemma 5.13.

We summarize our results in the following:

Theorem 5.14 For any rate $R = b/c$ convolutional code $\mathcal{C}$ encoded by a generator matrix of memory m and overall constraint length $\nu = bm$ that is used to communicate over the BSC with crossover probability ϵ, the burst error probability is lower-bounded by

$$P_{\mathrm{B}} > 2^{-\left(E_{\mathcal{C}}^{\mathrm{low}}(R) + o(1)\right)mc} \tag{5.165}$$

where $E_{\mathcal{C}}^{\mathrm{low}}(R)$ is the *convolutional lower-bound exponent*

$$E_{\mathcal{C}}^{\mathrm{low}}(R) = \min\left\{E_{\mathcal{C}}^{\mathrm{sph}}(R), -\frac{1}{2}\log\left(2\sqrt{\epsilon(1-\epsilon)}\right)\right\} \tag{5.166}$$

and $0 \le R < C$.

From (5.105) and (5.108) it follows that in the sphere-packing region $R_0 \le R < C$ the convolutional lower-bound exponent $E_{\mathcal{C}}^{\mathrm{low}}(R)$ is identical to the convolutional coding exponent $E_{\mathcal{C}}(R)$. Hence, in this region the convolutional coding exponent is asymptotically optimal. In Fig. 5.8 we show the convolutional coding and lower-bound exponents for our upper and lower bounds.

In order to lower-bound the bit error probability we return to (4.38), that is,

$$P_{\mathrm{b}} = \lim_{J\to\infty} \frac{\sum_{j=0}^{J-1} I_j}{b\sum_{j=0}^{J-1}(N_j + L_j)} \quad \text{with probability 1} \tag{5.167}$$

where the jth burst contains I_j errors, is of length L_j, and is separated from the previous burst by N_j b-tuples.

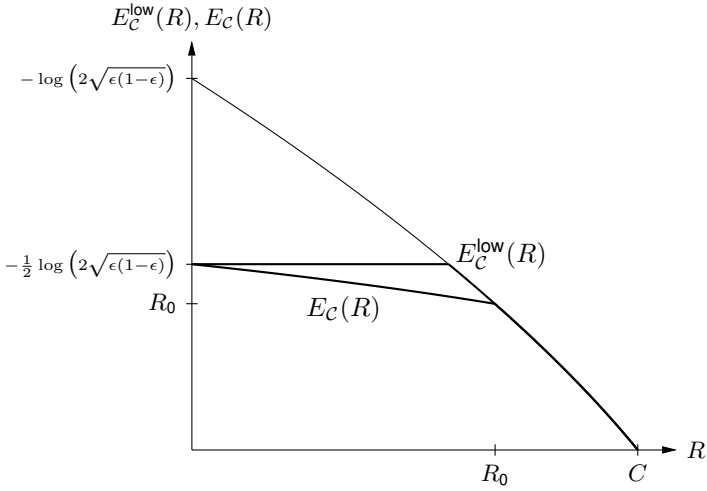


Figure 5.8 The convolutional coding exponent $E_{\mathcal{C}}(R)$ and the lower-bound exponent $E_{\mathcal{C}}^{\text{low}}$ for the BSC with crossover probability $\epsilon = 0.045$.

Since according to our definition an error burst cannot have more than m consecutive error-free b-tuples, it follows that

$$I_j \geq \frac{L_j}{m+1} \tag{5.168}$$

By combining (5.167) and (5.168), we can lower-bound the bit error probability as follows:

$$\begin{aligned} P_{\text{b}} &\geq \lim_{J\to\infty} \frac{\sum_{j=0}^{J-1} L_j}{b(m+1)\sum_{j=0}^{J-1}(N_j + L_j)} \\ &= \frac{\lim_{J\to\infty} \frac{1}{J}\sum_{j=0}^{J-1} L_j}{b(m+1)\left(\lim_{J\to\infty} \frac{1}{J}\sum_{j=0}^{J-1} N_j + \lim_{J\to\infty} \frac{1}{J}\sum_{j=0}^{J-1} L_j\right)} \\ &= \frac{E[L]}{b(m+1)(E[N]+E[L])} \\ &\geq \frac{1}{b(m+1)(E[N]+1)} \quad \text{with probability 1} \end{aligned} \tag{5.169}$$

where the last inequality follows from the fact that the error burst length is lower-bounded by 1.

Next we return to the burst error probability which can be expressed as the limit of the ratio between the number of error bursts and the number of nodes in which an

error burst could have started, that is (cf. (4.43)),

$$P_{\mathrm{b}} = \lim_{J\to\infty} \frac{J}{J + \sum_{j=0}^{J-1}(N_j - m)} = \frac{1}{1 + \lim_{J\to\infty} \frac{1}{J}\sum_{j=0}^{J-1} N_j - m}$$
$$= \frac{1}{E[N] + 1 - m} \quad \text{with probability 1} \tag{5.170}$$

Equation (5.170) can be rewritten as

$$\frac{1}{E[N]+1} = \frac{P_{\mathrm{B}}}{P_{\mathrm{B}}m + 1} \geq \frac{P_{\mathrm{B}}}{m+1} \tag{5.171}$$

where we used $P_{\mathrm{B}} \leq 1$ to obtain the inequality.

Combining (5.169) and (5.171) yields

$$P_{\mathrm{b}} \geq \frac{P_{\mathrm{B}}}{b(m+1)^2} = P_{\mathrm{B}} 2^{-\log(b(m+1)^2)} = P_{\mathrm{B}} 2^{-o(1)mc} \tag{5.172}$$

Finally, we lower-bound P_{B} by (5.165) and obtain the following:

Theorem 5.15 For any rate $R = b/c$ convolutional code $\mathcal{C}$ encoded by a generator matrix of memory m and overall constraint length $\nu = bm$ that is used to communicate over the BSC with crossover probability ϵ, the bit error probability is lower-bounded by

$$P_{\mathrm{b}} > 2^{-\left(E_{\mathcal{C}}^{\mathrm{low}}(R) + o(1)\right)mc} \tag{5.173}$$

where $E_{\mathcal{C}}^{\mathrm{low}}(R)$ is the convolutional lower-bound exponent given by (5.166) and $0 \leq R < C$.

Asymptotically, for increasing memory, the lower bound on the bit error probability decreases exponentially with the same exponent as the lower bound on the burst error probability.

5.4 GENERAL BOUNDS FOR TIME-VARYING CONVOLUTIONAL CODES

In Sections 5.1 and 5.2, we studied the ensemble of periodically time-varying convolutional codes when used to communicate over the BSC. In this section we will consider a more general channel, viz., the binary-input, q-ary output discrete memoryless channel (DMC). We need a more general ensemble, viz., the ensemble of (nonperiodically) time-varying convolutional codes. For this ensemble we define

$$P_{\mathrm{B}} \stackrel{\mathrm{def}}{=} \lim_{T\to\infty} \frac{1}{T}\sum_{t=0}^{T-1} P\left(\bigcup_{j=0}^{T-1} \mathcal{L}_t(j)\right) \leq \lim_{T\to\infty} \frac{1}{T}\sum_{t=0}^{T-1}\sum_{j=0}^{T-1} P(\mathcal{L}_t(j)) \tag{5.174}$$

where $P(\mathcal{L}_t(j))$ is the probability that a burst starting at depth t has length $j+1$.

Consider the situation when we use a convolutional code together with maximum-likelihood (ML) decoding to communicate over a binary-input, q-ary output DMC with input $\boldsymbol{v}$ and output $\boldsymbol{r}$ and transition probabilities $P(\boldsymbol{r} \mid \boldsymbol{v})$.

First we will obtain an upper bound on the decoding error probability $P_2(\mathcal{E})$ when the code consists of only two codewords. This simple case is both nontrivial and interesting.

Let $P_2(\mathcal{E} \mid i)$ denote the conditional error probability given that the codeword $\boldsymbol{v}^{(i)}$, $i = 0, 1$, is transmitted, and let $\mathcal{D}_i$ denote the decoding region for the ith codeword, that is,

$$\mathcal{D}_i \stackrel{\text{def}}{=} \left\{ \boldsymbol{r} \;\middle|\; \boldsymbol{r} \text{ is decoded as } \boldsymbol{v}^{(i)} \right\} \tag{5.175}$$

Then it follows that

$$P_2(\mathcal{E} \mid i) = \sum_{\boldsymbol{r} \notin \mathcal{D}_i} P\left(\boldsymbol{r} \;\middle|\; \boldsymbol{v}^{(i)} \right) \tag{5.176}$$

Although (5.176) appears to be very simple, it will be most useful to have an upper bound on the error probability that is independent of i. Hence, we multiply each term of the sum of (5.176) by

$$\left(P\left(\boldsymbol{r} \;\middle|\; \boldsymbol{v}^{(\bar{i})} \right) / P\left(\boldsymbol{r} \;\middle|\; \boldsymbol{v}^{(i)} \right) \right)^{\lambda}$$

where $0 < \lambda \leq 1$ and $\bar{i}$ denotes the binary complement of i. Since we assume maximum-likelihood decoding, it follows that our estimate $\widehat{\boldsymbol{v}} = \boldsymbol{v}^{(i)}$, that is, $\boldsymbol{r} \in \mathcal{D}_i$, if $P\left(\boldsymbol{r} \mid \boldsymbol{v}^{(i)} \right) > P\left(\boldsymbol{r} \;\middle|\; \boldsymbol{v}^{(\bar{i})} \right)$ (ties are resolved arbitrarily) or, that $\widehat{\boldsymbol{v}} \neq \boldsymbol{v}^{(i)}$, that is, $\boldsymbol{r} \notin \mathcal{D}_i$, if $P\left(\boldsymbol{r} \mid \boldsymbol{v}^{(i)} \right) < P\left(\boldsymbol{r} \;\middle|\; \boldsymbol{v}^{(\bar{i})} \right)$. In other words, the ratio $\left(P\left(\boldsymbol{r} \;\middle|\; \boldsymbol{v}^{(\bar{i})} \right) / P\left(\boldsymbol{r} \mid \boldsymbol{v}^{(i)} \right) \right)^{\lambda}$ is at most 1 when $\boldsymbol{r} \in \mathcal{D}_i$ and at least 1 when $\boldsymbol{r} \notin \mathcal{D}_i$. Hence, we obtain the upper bound

$$\begin{aligned} P_2(\mathcal{E} \mid i) &\leq \sum_{\boldsymbol{r} \notin \mathcal{D}_i} P\left(\boldsymbol{r} \;\middle|\; \boldsymbol{v}^{(i)} \right) \left(P\left(\boldsymbol{r} \;\middle|\; \boldsymbol{v}^{(\bar{i})} \right) / P\left(\boldsymbol{r} \;\middle|\; \boldsymbol{v}^{(i)} \right) \right)^{\lambda} \\ &= \sum_{\boldsymbol{r} \notin \mathcal{D}_i} \left(P\left(\boldsymbol{r} \;\middle|\; \boldsymbol{v}^{(\bar{i})} \right) \right)^{\lambda} \left(P\left(\boldsymbol{r} \;\middle|\; \boldsymbol{v}^{(i)} \right) \right)^{1-\lambda} \\ &< \sum_{\boldsymbol{r}} \left(P\left(\boldsymbol{r} \;\middle|\; \boldsymbol{v}^{(\bar{i})} \right) \right)^{\lambda} \left(P\left(\boldsymbol{r} \;\middle|\; \boldsymbol{v}^{(i)} \right) \right)^{1-\lambda} \end{aligned} \tag{5.177}$$

Let $\lambda = 1/2$; then the last sum in the bound (5.177) is independent of the transmitted codeword. Hence, we have

$$P_2(\mathcal{E}) = P_2(\mathcal{E} \mid 0) = P_2(\mathcal{E} \mid 1) < \sum_{\boldsymbol{r}} \sqrt{P\left(\boldsymbol{r} \mid \boldsymbol{v}^{(0)} \right) P\left(\boldsymbol{r} \mid \boldsymbol{v}^{(1)} \right)} \tag{5.178}$$

Since the channel is memoryless we can simplify (5.178):

$$\begin{aligned} P_2(\mathcal{E}) &< \sum_{r_1}\sum_{r_2}\cdots\sqrt{\prod_j P(r_j \mid v_j^{(0)})P(r_j \mid v_j^{(1)})} \\ &= \prod_j \sum_r \sqrt{P(r \mid v_j^{(0)})P(r \mid v_j^{(1)})} \end{aligned} \tag{5.179}$$

where

$$\sum_r \sqrt{P(r \mid v_j^{(0)})P(r \mid v_j^{(1)})} = 1 \quad \text{for } v_j^{(0)} = v_j^{(1)} \tag{5.180}$$

Let d denote the Hamming distance between the two codewords, that is,

$$d = d_{\mathrm{H}}\left(\boldsymbol{v}^{(0)}, \boldsymbol{v}^{(1)}\right) \tag{5.181}$$

We have now proved the following:

Theorem 5.16 (Bhattacharyya bound) When using two codewords for communication over a binary-input DMC with transition probabilities $P(r \mid v)$ and maximum-likelihood decoding, the *decoding error probability* is upper-bounded by

$$p_d < \left(\sum_r \sqrt{P\left(r \mid 0\right) P\left(r \mid 1\right)}\right)^d \tag{5.182}$$

where d is the Hamming distance between the two codewords.

From Theorem 5.16 follows immediately the Bhattacharyya bound for the BSC (4.20).

■ **EXAMPLE 5.1**

Consider the binary-input, 8-ary output DMC given in Example 4.2. For this channel

$$\begin{aligned} &\sum_r \sqrt{P\left(r \mid 0\right) P\left(r \mid 1\right)} \\ &\quad = \sqrt{0.434 \cdot 0.002} + \cdots + \sqrt{0.002 \cdot 0.434} \\ &\quad = 2\left(\sqrt{0.434 \cdot 0.002} + \sqrt{0.197 \cdot 0.008} + \sqrt{0.167 \cdot 0.023} + \sqrt{0.111 \cdot 0.058}\right) \\ &\quad = 0.42 \end{aligned} \tag{5.183}$$

and

$$p_{10} < 0.42^{10} \approx 1.7 \cdot 10^{-4} \tag{5.184}$$

Consider a rate $R = b/c$, memory m time-invariant convolutional code with weight spectrum n_d (cf. (3.35)) used for communication over a binary-input DMC

and maximum-likelihood decoding. From (4.25) and (5.182) it follows that the burst error probability, P_{B}, is upper-bounded by

$$P_{\mathrm{B}} \leq \sum_{d=d_{\mathrm{free}}}^{\infty} n_d p_d \tag{5.185}$$

where

$$p_d < \left(\sum_r \sqrt{P(r \mid 0) P(r \mid 1)} \right)^d \stackrel{\mathrm{def}}{=} z^d \tag{5.186}$$

where z is the *Bhattacharyya parameter* for the binary-input DMC.

Next consider the ensemble $\mathcal{E}(b, c, m, \infty)$ of binary, rate $R = b/c$ time-varying convolutional codes encoded by polynomial, time-varying, generator matrices of memory m which we introduced in Section 3.6. The distance spectrum does in general depend on the time t, but its average $E[n_d]$ is independent of t.

The average of the burst error probability taken over this ensemble is upper-bounded by

$$E[P_{\mathrm{B}}] < \sum_{d=0}^{\infty} E[n_d] z^d \tag{5.187}$$

where $E[n_d]$ can be calculated by summing the probabilities for all incorrect paths of weight d. Since the number of incorrect paths of length $(j + m + 1)$ branches is upper-bounded by $2^{(j+1)b}$ and since the probability that each of these incorrect paths has weight d is equal to $\binom{(j+m+1)c}{d} 2^{-(j+m+1)c}$, we obtain

$$E[n_d] < \sum_{j=0}^{\infty} 2^{(j+1)b} \binom{(j+m+1)c}{d} 2^{-(j+m+1)c}, \quad d = 0, 1, 2, \ldots \tag{5.188}$$

Hence, by inserting (5.188) into (5.187) we have

$$\begin{aligned} E[P_{\mathrm{B}}] &< \sum_{d=0}^{\infty} \sum_{j=0}^{\infty} 2^{(j+1)b} \binom{(j+m+1)c}{d} 2^{-(j+m+1)c} z^d \\ &= \sum_{j=0}^{\infty} \sum_{d=0}^{(j+m+1)c} 2^{(j+1)b} \binom{(j+m+1)c}{d} 2^{-(j+m+1)c} z^d \\ &= \sum_{j=0}^{\infty} 2^{(j+1)b} \left(\frac{1+z}{2} \right)^{(j+m+1)c} \\ &= \left(\frac{1+z}{2} \right)^{mc} \sum_{j=0}^{\infty} \left(2^R \frac{1+z}{2} \right)^{(j+1)c} \end{aligned} \tag{5.189}$$

where the last sum converges if

$$2^R \frac{1+z}{2} < 1 \tag{5.190}$$

Before we proceed with upper-bounding the burst error probability, we generalize our definition of the computational cutoff rate for the BSC (5.6) to a general discrete memoryless channel:

Definition The *computational cutoff rate* R_0 for a general DMC with transition probabilities $P(r \mid v)$ and input distribution $Q(v)$ is given by

$$R_0 \stackrel{\text{def}}{=} -\log\left(\min_Q\left\{\sum_r\left(\sum_v \sqrt{P(r \mid v)}Q(v)\right)^2\right\}\right) \tag{5.191}$$

It is most interesting and, perhaps, somewhat surprising that for *binary*-input channels the probability distribution

$$Q(0) = Q(1) = \frac{1}{2} \tag{5.192}$$

is the minimizing one (Problem 5.2). Hence, expanding the square in (5.191) gives

$$\begin{aligned} R_0 &= -\log\left(\frac{1}{4}\sum_r\left(P(r \mid 0) + 2\sqrt{P(r \mid 0)\,P(r \mid 1)} + P(r \mid 1)\right)\right) \\ &= -\log\left(\frac{1}{2}\left(1 + \sum_r \sqrt{P(r \mid 0)\,P(r \mid 1)}\right)\right) \\ &= 1 - \log\left(1 + \sum_r \sqrt{P(r \mid 0)\,P(r \mid 1)}\right) \end{aligned} \tag{5.193}$$

and we have the following:

Lemma 5.17 The cutoff rate R_0 for a binary-input DMC with transition probabilities $P(r \mid v)$ is

$$R_0 = 1 - \log(1 + z) \tag{5.194}$$

where z is the Bhattacharyya parameter defined by (5.186).

■ **EXAMPLE 5.2**

For the BSC with crossover probability ϵ it follows from (5.194) that

$$R_0 = 1 - \log\left(1 + 2\sqrt{\epsilon(1-\epsilon)}\right) \tag{5.195}$$

which coincides with the definition for the BSC (5.6). As a specific instance, we find

$$R_0 = \frac{1}{2} \quad \text{when } \epsilon = 0.045 \tag{5.196}$$

■ **EXAMPLE 5.3**

Consider the *binary erasure channel* (BEC) with erasure probability δ shown in Fig. 1.22. Its cutoff rate is

$$\begin{aligned} R_0 &= 1 - \log\left(1 + \sqrt{(1-\delta)0} + \sqrt{\delta\delta} + \sqrt{0(1-\delta)}\right) \\ &= 1 - \log(1+\delta) \end{aligned} \tag{5.197}$$

As a specific instance, we find

$$R_0 = \frac{1}{2} \quad \text{when } \delta = \sqrt{2} - 1 = 0.414 \tag{5.198}$$

So 4.5% "channel errors" are as bad as 41.4% erasures!

Both the BSC and the BEC have

$$R_0 = 1 \quad \text{when } \epsilon = 0 \text{ and } \delta = 0, \text{ respectively} \tag{5.199}$$

and

$$R_0 = 0 \quad \text{when } \epsilon = \frac{1}{2} \text{ and } \delta = 1, \text{ respectively} \tag{5.200}$$

Now we return to our derivation and rewrite (5.194) as

$$2^{R_0}\frac{1+z}{2} = 1 \tag{5.201}$$

By comparing the inequality (5.190) with (5.201), we conclude that the last sum in (5.189) converges if $R < R_0$. Hence, we have

$$\begin{aligned} E[P_{\text{B}}] &< \left(\frac{1+z}{2}\right)^{mc} \frac{(2^R\frac{1+z}{2})^c}{1-(2^R\frac{1+z}{2})^c} = 2^{-R_0 mc}\frac{2^{(R-R_0)c}}{1-2^{(R-R_0)c}} \\ &= c(R)2^{-R_0 mc} = 2^{-(R_0+o(1))mc} \text{ for } R < R_0 \end{aligned} \tag{5.202}$$

where

$$c(R) = \frac{1}{2^{(R_0-R)c} - 1} \tag{5.203}$$

Since the average of the burst error probability taken over our ensemble is upper-bounded by (5.202), we have the following:

Theorem 5.18 There exists a binary, rate $R = b/c$, time-varying, convolutional code encoded by a polynomial, time-varying generator matrix of memory m such that its average burst error probability when used to communicate over a binary-input DMC with maximum-likelihood decoding is upper-bounded by

$$P_{\text{B}} < 2^{-(R_0+o(1))mc} \text{ for } R < R_0 \tag{5.204}$$

where R_0 is the computational cutoff rate.

Next we will obtain a corresponding upper bound on burst error probability for the sphere-packing region, that is, for rates $R_0 \leq R < C$. Then we need the Gallager function for the binary-input DMC with transition probabilities $P(r \mid v)$:

$$G(s) \stackrel{\text{def}}{=} -\log\left(\sum_r \left(\frac{1}{2}P(r \mid 0)^{\frac{1}{1+s}} + \frac{1}{2}P(r \mid 1)^{\frac{1}{1+s}}\right)^{1+s}\right) \tag{5.205}$$

where $0 < s < \infty$. For the BSC, definition (5.205) coincides with definition (5.98). We also notice that for $s = 1$ we have the important equality

$$G(1) = R_0 \tag{5.206}$$

For simplicity, in the sequel we will only consider binary-input and *output-symmetric* DMC, that is, we impose the restriction that

$$P(r \mid 0) = P(-r \mid 1), \quad \text{all } r \tag{5.207}$$

All channels we consider in this book are output-symmetric.

For the sphere-packing region we will again exploit the idea of separating the error event $\mathcal{E}$ into two disjoint events corresponding to "few" $\mathcal{F}$ and "many" $\mathcal{M}$ errors, respectively. Hence, we have (cf. (4.81))

$$E[P_{\text{B}}] = E[P(\mathcal{E})] \leq E[P(\mathcal{E}, \mathcal{F})] + P(\mathcal{M}) \tag{5.208}$$

Since we are considering output-symmetric channels we can without loss of generality assume that the allzero sequence is transmitted. Let $\boldsymbol{r} = \boldsymbol{r}_0\boldsymbol{r}_1\ldots$, where $\boldsymbol{r}_i = r_i^{(1)}r_i^{(2)}\ldots r_i^{(c)}$, denote the received sequence. Then we introduce the cumulative metric

$$S_t = \sum_{i=0}^{t} Z_i \tag{5.209}$$

where

$$Z_i = \sum_{\ell=1}^{c} \mu_{i\ell} \tag{5.210}$$

$$\mu_{i\ell} \stackrel{\text{def}}{=} \mu(r_i^{(\ell)}) = \log \frac{P\left(r_i^{(\ell)} \,\middle|\, 0\right)^{\frac{1}{1+s}}}{\frac{1}{2}P\left(r_i^{(\ell)} \,\middle|\, 0\right)^{\frac{1}{1+s}} + \frac{1}{2}P\left(r_i^{(\ell)} \,\middle|\, 1\right)^{\frac{1}{1+s}}} - R \tag{5.211}$$

and the value of the parameter s will be chosen later.

Remark: For the BSC, $\mu(0) = s\alpha$ and $\mu(1) = s\beta$, where α and β are defined by (4.86) and (4.87), respectively, the parameter a in (4.86) and (4.87) is given by $a = \epsilon^{\frac{1}{1+s}} / \left(\epsilon^{\frac{1}{1+s}} + (1-\epsilon)^{\frac{1}{1+s}}\right)$, and s satisfies $G(s) = sR$, where $G(s)$ is given by (5.98).

As in Sections 3.4 and 4.3 we have a random walk $0, S_0, S_1, S_2, \ldots$, and we say that those error patterns for which S_t hits or crosses (from above) a certain barrier u contain many errors.

From Wald's identity (Corollary B.6) follows

$$P(\mathcal{M}) = P(\min_t\{S_t\} \leq u) \leq 2^{-\lambda_0 u} \tag{5.212}$$

where $\lambda_0 < 0$ is a root of the equation

$$g(\lambda) \stackrel{\text{def}}{=} E[2^{\lambda\mu(r)}] = \sum_r 2^{\lambda\mu(r)} P(r \mid 0) = 1 \tag{5.213}$$

Combining (5.211) and (5.213) yields

$$2^{-\lambda_0 R}\left(\sum_r P\left(r \mid 0\right)^{\frac{\lambda_0}{1+s}+1}\left(\frac{1}{2}P\left(r \mid 0\right)^{\frac{1}{1+s}} + \frac{1}{2}P\left(r \mid 1\right)^{\frac{1}{1+s}}\right)^{-\lambda_0}\right) = 1 \tag{5.214}$$

Exploiting the output symmetry of the channel, we can rewrite (5.214) as

$$\begin{aligned} 2^{-\lambda_0 R}\Bigg(\sum_r \left(\frac{1}{2}P\left(r \mid 0\right)^{\frac{\lambda_0}{1+s}+1} + \frac{1}{2}P\left(r \mid 1\right)^{\frac{\lambda_0}{1+s}+1}\right) \\ \times \left(\frac{1}{2}P\left(r \mid 0\right)^{\frac{1}{1+s}} + \frac{1}{2}P\left(r \mid 1\right)^{\frac{1}{1+s}}\right)^{-\lambda_0}\Bigg) = 1 \end{aligned} \tag{5.215}$$

Now we choose $s = s_0$ to be a positive root of

$$G(s) = sR \tag{5.216}$$

where $G(s)$ is the Gallager function and $0 < s_0 \leq 1$ if $R \geq R_0$. We can easily verify that $\lambda_0 = -s_0$ satisfies (5.215). Hence, for $s \leq s_0$ we have

$$P(\mathcal{M}) \leq 2^{s_0 u} \leq 2^{su} \tag{5.217}$$

where s_0 is a positive root of (5.216).

To upper-bound the probability that a burst starts at the root and that we have an error pattern with few errors, $P(\mathcal{E}, \mathcal{F})$, we use, as before, the union bound and obtain

$$P(\mathcal{E}, \mathcal{F}) \leq \sum_k P(\mathcal{E}^{(k)}, \mathcal{F}) \tag{5.218}$$

where $\mathcal{E}^{(k)}$ is the event that a burst error starting at the root is caused by path k.

Let us assume that the allzero codeword is transmitted and that path k has weight w_k and remerges with the allzero path at depth $j + m + 1$, $j \geq 0$. Then,

$$P(\mathcal{E}^{(k)}, \mathcal{F}) = \sum_{\boldsymbol{r}_{[0,j+m]} \in \mathcal{D}_k, \mathcal{F}} P\left(\boldsymbol{r}_{[0,j+m]} \;\middle|\; \mathbf{0}_{[0,j+m]}\right) \tag{5.219}$$

where $\mathcal{D}_k$ is the decoding region for the kth path and $\mathcal{F}$ is the region corresponding to few errors. In the region $\mathcal{D}_k$ we have

$$\left(\frac{P\left(\boldsymbol{r}_{[0,j+m]} \;\middle|\; \boldsymbol{v}^{(k)}_{[0,j+m]}\right)}{P\left(\boldsymbol{r}_{[0,j+m]} \;\middle|\; \boldsymbol{0}_{[0,j+m]}\right)}\right)^{\lambda_1} \geq 1 \tag{5.220}$$

where $\boldsymbol{v}^{(k)}_{[0,j+m]}$ is the first $j+m+1$ c-tuples of the kth path and $\lambda_1 > 0$. Similarly, in the region $\mathcal{F}$ we have

$$2^{\lambda_2(S_{j+m}-u)} \geq 1 \tag{5.221}$$

where $\lambda_2 > 0$. Thus, combining (5.219), (5.220), and (5.221) yields

$$\begin{aligned} P(\mathcal{E}^{(k)}, \mathcal{F}) &\leq \sum_{\boldsymbol{r}_{[0,j+m]} \in \mathcal{D}_k, \mathcal{F}} \left(\frac{P\left(\boldsymbol{r}_{[0,j+m]} \;\middle|\; \boldsymbol{v}^{(k)}_{[0,j+m]}\right)}{P\left(\boldsymbol{r}_{[0,j+m]} \;\middle|\; \boldsymbol{0}_{[0,j+m]}\right)}\right)^{\lambda_1} \\ &\qquad \times 2^{\lambda_2(S_{j+m}-u)} P\left(\boldsymbol{r}_{[0,j+m]} \;\middle|\; \boldsymbol{0}_{[0,j+m]}\right) \\ &\leq \sum_{\text{all } \boldsymbol{r}_{[0,j+m]}} \left(\frac{P\left(\boldsymbol{r}_{[0,j+m]} \;\middle|\; \boldsymbol{v}^{(k)}_{[0,j+m]}\right)}{P\left(\boldsymbol{r}_{[0,j+m]} \;\middle|\; \boldsymbol{0}_{[0,j+m]}\right)}\right)^{\lambda_1} \\ &\qquad \times 2^{\lambda_2(S_{j+m}-u)} P\left(\boldsymbol{r}_{[0,j+m]} \;\middle|\; \boldsymbol{0}_{[0,j+m]}\right) \end{aligned} \tag{5.222}$$

Next we insert (5.209), (5.210), and (5.211) into (5.222) and exploit the memorylessness of the DMC. Then we obtain

$$\begin{aligned} P(\mathcal{E}^{(k)}, \mathcal{F}) &\leq 2^{-\lambda_2 u} \prod_{i=0}^{j+m} \prod_{\ell=1}^{c} 2^{-\lambda_2 R} \sum_{r_i^{(\ell)}} \left(\frac{P\left(r_i^{(\ell)} \;\middle|\; v_i^{(k)(\ell)}\right)^{\lambda_1}}{P\left(r_i^{(\ell)} \;\middle|\; 0\right)^{\lambda_1}} P\left(r_i^{(\ell)} \;\middle|\; 0\right)^{\frac{\lambda_2}{1+s}+1} \right. \\ &\qquad \left. \times \left(\frac{1}{2} P\left(r_i^{(\ell)} \;\middle|\; 0\right)^{\frac{1}{1+s}} + \frac{1}{2} P\left(r_i^{(\ell)} \;\middle|\; 1\right)^{\frac{1}{1+s}}\right)^{-\lambda_2}\right) \\ &= 2^{-\lambda_2 u} \left(2^{-\lambda_2 R} \sum_{r} P(r \mid 0)^{\frac{\lambda_2}{1+s}-\lambda_1+1} P(r \mid 1)^{\lambda_1} \right. \\ &\qquad \left. \times \left(\frac{1}{2} P(r \mid 0)^{\frac{1}{1+s}} + \frac{1}{2} P(r \mid 1)^{\frac{1}{1+s}}\right)^{-\lambda_2}\right)^{w_k} \\ &\qquad \times \left(2^{-\lambda_2 R} \sum_{r} P(r \mid 0)^{\frac{\lambda_2}{1+s}+1} \right. \\ &\qquad \left. \times \left(\frac{1}{2} P(r \mid 0)^{\frac{1}{1+s}} + \frac{1}{2} P(r \mid 1)^{\frac{1}{1+s}}\right)^{-\lambda_2}\right)^{(j+m+1)c-w_k} \end{aligned} \tag{5.223}$$

where $\boldsymbol{v}_i^{(k)} = v_i^{(k)(1)} v_i^{(k)(2)} \ldots v_i^{(k)(c)}$ is the ith c-tuple of the codeword corresponding to the kth path. Let

$$\lambda_1 = \frac{1}{1+s} \tag{5.224}$$

and

$$\lambda_2 = 1 - s \tag{5.225}$$

Then we have

$$P(\mathcal{E}^{(k)}, \mathcal{F}) \le 2^{-(1-s)u} Z_1^{w_k} Z_2^{c(j+m+1)-w_k} \tag{5.226}$$

where

$$\begin{aligned} Z_1 = 2^{-(1-s)R} \sum_r (P(r \mid 0)P(r \mid 1))^{\frac{1}{1+s}} \\ \times \left(\frac{1}{2} P(r \mid 0)^{\frac{1}{1+s}} + \frac{1}{2} P(r \mid 1)^{\frac{1}{1+s}} \right)^{-1+s} \end{aligned} \tag{5.227}$$

and

$$\begin{aligned} Z_2 &= 2^{-(1-s)R} \sum_r P(r \mid 0)^{\frac{2}{1+s}} \left(\frac{1}{2} P(r \mid 0)^{\frac{1}{1+s}} + \frac{1}{2} P(r \mid 1)^{\frac{1}{1+s}} \right)^{-1+s} \\ &= 2^{-(1-s)R} \sum_r \left(\frac{1}{2} P(r \mid 0)^{\frac{2}{1+s}} + \frac{1}{2} P(r \mid 1)^{\frac{2}{1+s}} \right) \\ &\quad \times \left(\frac{1}{2} P(r \mid 0)^{\frac{1}{1+s}} + \frac{1}{2} P(r \mid 1)^{\frac{1}{1+s}} \right)^{-1+s} \end{aligned} \tag{5.228}$$

where the last equality follows from the output symmetry of the channel.

By combining (5.218) and (5.226), we obtain

$$\begin{aligned} E[P(\mathcal{E}, \mathcal{F})] \le 2^{-(1-s)u} \sum_{j=0}^{\infty} \sum_{w=1}^{(j+m+1)c} E[n(w, j+m+1)] \\ \times Z_1^w Z_2^{(j+m+1)c-w} \end{aligned} \tag{5.229}$$

where $n(w, j+m+1)$ denotes the number of paths of weight w and length $j+m+1$ c-tuples.

For the ensemble $\mathcal{E}(b, c, m, \infty)$ we have

$$\begin{aligned} E[n(w, j+m+1)] &\le \left(2^{b(j+1)} - 1\right) \binom{(j+m+1)c}{w} \left(\frac{1}{2}\right)^{(j+m+1)c} \\ &< 2^{b(j+1)} \binom{(j+m+1)c}{w} 2^{-(j+m+1)c} \end{aligned} \tag{5.230}$$

Further upper-bounding of (5.229) by extending the sum over w to include $w = 0$ and inserting (5.230) into (5.229) yield

$$\begin{aligned}
E[P(\mathcal{E},\mathcal{F})] &\leq 2^{-(1-s)u}\sum_{j=0}^{\infty}2^{b(j+1)-(j+m+1)c} \\
&\quad\times\sum_{w=0}^{(j+m+1)c}\binom{(j+m+1)c}{w} \\
&\quad\times Z_1^w Z_2^{(j+m+1)c-w} \\
&= 2^{-(1-s)u}\sum_{j=0}^{\infty}2^{b(j+1)-(j+m+1)c}(Z_1+Z_2)^{(j+m+1)c} \\
&= 2^{-(1-s)u}\left(\frac{Z_1+Z_2}{2}\right)^{mc}\sum_{j=0}^{\infty}(2^{R-1}(Z_1+Z_2))^{(j+1)c} \\
&= 2^{-(1-s)u}\left(\frac{Z_1+Z_2}{2}\right)^{mc}\frac{(2^{R-1}(Z_1+Z_2))^c}{1-(2^{R-1}(Z_1+Z_2))^c} \qquad (5.231)
\end{aligned}$$

where the last equality requires that

$$R < \log\frac{2}{Z_1+Z_2} \qquad (5.232)$$

From (5.227) and (5.228) it follows that

$$\begin{aligned}
\frac{Z_1+Z_2}{2} &= 2^{-(1-s)R} \\
&\quad\times\sum_r\left(\frac{1}{2}P(r\mid 0)^{\frac{2}{1+s}}+P(r\mid 0)^{\frac{1}{1+s}}P(r\mid 1)^{\frac{1}{1+s}}+\frac{1}{2}P(r\mid 1)^{\frac{2}{1+s}}\right) \\
&\quad\times\left(\frac{1}{2}P(r\mid 0)^{\frac{1}{1+s}}+\frac{1}{2}P(r\mid 1)^{\frac{1}{1+s}}\right)^{-1+s} \\
&= 2^{-(1-s)R}\sum_r\left(\frac{1}{2}P(r\mid 0)^{\frac{1}{1+s}}+\frac{1}{2}P(r\mid 1)^{\frac{1}{1+s}}\right)^{2} \\
&\quad\times\left(\frac{1}{2}P(r\mid 0)^{\frac{1}{1+s}}+\frac{1}{2}P(r\mid 1)^{\frac{1}{1+s}}\right)^{-1+s} \\
&= 2^{-(1-s)R}\sum_r\left(\frac{1}{2}P(r\mid 0)^{\frac{1}{1+s}}+\frac{1}{2}P(r\mid 1)^{\frac{1}{1+s}}\right)^{1+s} \\
&= 2^{-(1-s)R-G(s)} \qquad (5.233)
\end{aligned}$$

Hence, inequality (5.232) is equivalent to

$$R < G(s)/s \qquad (5.234)$$

or, again equivalently,

$$s < s_0 \qquad (5.235)$$

where s_0 is defined by (5.216).

Combining the bounds (5.208), (5.217), and (5.231) with equality (5.233) yields

$$\begin{aligned} E[P_{\mathrm{B}}] &\le E[P(\mathcal{E},\mathcal{F})] + P(\mathcal{M}) \\ &\le 2^{-(1-s)u} 2^{-((1-s)R+G(s))mc} \frac{(2^{R-1}(Z_1+Z_2))^c}{1-(2^{R-1}(Z_1+Z_2))^c} + 2^{su} \end{aligned} \tag{5.236}$$

where $s < s_0$.

Let us choose

$$u = -((1-s)R + G(s))mc \tag{5.237}$$

Then we obtain

$$\begin{aligned} E[P_{\mathrm{B}}] &\le \left(1 + \frac{(2^{R-1}(Z_1+Z_2))^c}{1-(2^{R-1}(Z_1+Z_2))^c}\right) 2^{-s((1-s)R+G(s))mc} \\ &= \frac{1}{1-(2^{R-1}(Z_1+Z_2))^c} 2^{-s((1-s)R+G(s))mc} \end{aligned} \tag{5.238}$$

Now, if we choose s to be slightly less than s_0,

$$s = s_0 - 1/m \tag{5.239}$$

say, and use (5.216), then we obtain

$$\begin{aligned} E[P_{\mathrm{B}}] &\le \frac{2^{((1-s)R+G(s))c + (sG(s_0) - s_0 G(s))mc}}{1-(2^{R-1}(Z_1+Z_2))^c} 2^{-s_0 Rmc} \\ &= 2^{-(G(s_0)+o(1))mc} \end{aligned} \tag{5.240}$$

where $0 < s_0 \le 1$, or, equivalently, $R_0 \le R < C$.

Finally, since the average of the burst error probability can be upper-bounded by (5.240), we have proved the following:

Theorem 5.19 There exists a binary, rate $R = b/c$, time-varying convolutional code encoded by a polynomial, time-varying generator matrix of memory m such that its average burst error probability when used to communicate over a binary-input and output-symmetrical DMC with maximum-likelihood decoding is upper-bounded by

$$P_{\mathrm{B}} \le 2^{-(E_{\mathcal{C}}(R)+o(1))mc} \tag{5.241}$$

where

$$E_{\mathcal{C}}(R) = \begin{cases} R_0, & 0 \le R < R_0 \\ \begin{cases} G(s), \\ R = G(s)/s, \end{cases} & 0 < s \le 1, \;\; R_0 \le R < C \end{cases} \tag{5.242}$$

for the random coding and sphere-packing regions, respectively, and where $G(s)$ is the Gallager exponent for the DMC (5.205) and R_0 is the computational cutoff rate for the DMC (5.191).

Consider a rate $R = b/c$, memory m, minimal-basic convolutional encoding matrix whose overall constraint length is $\nu = bm$. Let $\mathcal{S}$ denote the set of encoder states of the controller canonical form realization of the encoding matrix. Thus, the complexity of the encoder is

$$|\mathcal{S}| = 2^{bm} \tag{5.243}$$

and, hence, the complexity of the encoder satisfying (5.241) is at most 2^{bm}. By combining (5.241), (5.242), and (5.243) we obtain the following:

Theorem 5.20 There exists at least one time-varying, rate $R < C$, convolutional code of memory m with burst error probability satisfying

$$P_{\mathrm{B}} < \begin{cases} (|\mathcal{S}|)^{-R_0/R+o(1)}, & 0 \le R < R_0 \\ (|\mathcal{S}|)^{-s_0+o(1)}, & R_0 \le R < C \end{cases} \tag{5.244}$$

where $o(1) \to 0$ when $|\mathcal{S}| \to \infty$ and s_0 is given by (5.216).

The burst error probability decreases algebraically with increasing complexity.

In particular, it follows from Theorem 5.20 that the single number R_0 not only determines a range of rates, $0 \le R < R_0$, over which reliable communication is possible but also determines the coding complexity necessary to obtain a given error probability.

5.5 ERROR BOUNDS FOR FINITE BACK-SEARCH LIMITS

In the Viterbi decoding described in Section 4.1, we postponed the decision of the decoded codeword until we had reached the end of the trellis, which was terminated using the zero-tail method. Here we will consider a suboptimal version of Viterbi decoding in which when the decoding has reached depth t, $t \ge \tau$, outputs a decision of the information b-tuple at depth $t - \tau$; τ is called the *back-search limit*. Our aim is to derive upper bounds on the burst error probability for the ensemble of rate $R = b/c$, time-invariant convolutional codes encoded by generator matrices of memory m for the back-search limit $\tau = m + 1$.

The decoding rule is simple: for each depth $t \ge \tau$ choose among the paths leading to the states at this depth the one or, in case of ties, arbitrarily one of those with maximum Viterbi metric and output as the decoder estimate the $(t-\tau)$th information b-tuple corresponding to the chosen path.

Since we consider the ensemble of randomly chosen time-invariant convolutional codes, without loss of generality we can study the burst error probability at the root.

To upper-bound the burst error probability we will use the block code $\mathcal{B}$ given by

$$\mathcal{B} = \left\{ \boldsymbol{v}_{[0,m]} \;\middle|\; \boldsymbol{v}_{[0,m]} = \mathbf{0} \text{ or arises from } \boldsymbol{u}_{[0,m]} \text{ with } \boldsymbol{u}_0 \neq 0 \right\} \tag{5.245}$$

The number of codewords is

$$M = (2^b - 1)2^{bm} + 1 \tag{5.246}$$

and the block length is

$$N = (m+1)c \tag{5.247}$$

Hence, the rate of the block code is slightly less than the rate of the convolutional code $R = b/c$.

Assume that the allzero sequence is transmitted over the BSC, and let $\mathcal{E}$ denote the error event for the Viterbi decoder with back-search limit $\tau = m + 1$. Then, a necessary and sufficient condition for $\mathcal{E}$ is that the block code $\mathcal{B}$ is erroneously decoded.

Lemma 5.21 (Random coding bound) There exists a binary, rate $R = b/c$, time-invariant convolutional code encoded by a generator matrix of memory m such that its burst error probability when used to communicate over the BSC with Viterbi decoding with back-search limit $\tau = m + 1$ is upper-bounded by the inequality

$$P_{\text{B}} < 2^{-(R_0 - R)(m+1)c}, \quad 0 \le R < R_{\text{crit}} \tag{5.248}$$

where R_0 and R_{crit} are the computational cutoff rate and the critical rate, respectively.

Proof: The required statistical properties follow from Lemma 3.13. Then from (4.20) it follows that over the ensemble $\mathcal{E}(b, c, m, 1)$ the probability that each codeword $\boldsymbol{v}_{[0,m]}$ that arises from an information sequence $\boldsymbol{u}_{[0,m]}$ with $\boldsymbol{u}_0 \neq \mathbf{0}$ causes an error is upper-bounded by

$$\sum_{i=0}^{(m+1)c} \binom{(m+1)c}{i} 2^{-(m+1)c} z^i = \left(\frac{1+z}{2}\right)^{(m+1)c} = 2^{-R_0(m+1)c} \tag{5.249}$$

where the last equality follows from (5.8) and z is the Bhattacharyya parameter (4.20). We combine (5.249) with the upper bound $2^{R(m+1)c}$ on the total number of codewords in $\mathcal{B}$ and the lemma follows. ∎

Lemma 5.22 (Expurgation bound) In the ensemble $\mathcal{E}(b, c, m, 1)$ of binary, rate $R = b/c$, time-invariant convolutional codes encoded by generator matrices of memory m, there exists a subset containing at least a 2^{-b}th fraction of the codes such that their average burst error probability when used to communicate over the BSC with Viterbi decoding with back-search limit $\tau = m + 1$ is upper-bounded by the inequality

$$E_{\text{exp}}[P_{\text{B}}] < (2^b - 1)2^{\rho\left(\log\left(2\sqrt{\epsilon(1-\epsilon)}\right)\right)(m+1)c}, \quad 0 \le R < R_{\text{exp}} \tag{5.250}$$

where ρ is the Gilbert-Varshamov parameter of the rate R, R_{exp} is the expurgation rate, and ϵ is the crossover probability for the BSC.

Proof: The probability that the minimum distance between the allzero codeword and any other codeword in the block code $\mathcal{B}$ with the number of codewords given in (5.246) does not exceed d_0 is upper-bounded by

$$(M-1)\sum_{i=0}^{d_0} \binom{(m+1)c}{i} 2^{-(m+1)c} \le (2^b - 1)2^{bm} 2^{\left(h\left(\frac{d_0}{(m+1)c}\right) - 1\right)(m+1)c} \tag{5.251}$$

Let

$$d_0 = \lfloor \rho(m+1)c \rfloor \tag{5.252}$$

where ρ is the Gilbert-Varshamov parameter. Then (5.251) can be further upper-bounded by

$$(2^b - 1)2^{bm}2^{(h(\rho)-1)(m+1)c} = (1 - 2^{-b})2^{b(m+1)}2^{-R(m+1)c} = 1 - 2^{-b} \tag{5.253}$$

Hence, at least the 2^{-b}th fraction of the block codes has such distances that are not less than $d_0 + 1$. These block codes form an expurgated ensemble. The average error probability for this subensemble is upper-bounded by (cf. (5.12))

$$E_{\text{exp}}[P(\mathcal{E})] \leq 2^b(M-1)\sum_{i=d_0+1}^{(m+1)c}\binom{(m+1)c}{i}2^{-(m+1)c}z^i \tag{5.254}$$

where z is the Bhattacharyya parameter (4.20). Then analogously to (5.13) we obtain

$$\begin{aligned} E_{\text{exp}}[P(\mathcal{E})] &\leq (2^b-1)2^{R(m+1)c}\sum_{i=d_0+1}^{(m+1)c}\binom{(m+1)c}{i}2^{\lambda(i-d_0-1)-(m+1)c}z^i \\ &= (2^b-1)2^{(R-1)(m+1)c}2^{-\lambda(d_0+1)}\sum_{i=d_0+1}^{(m+1)c}\binom{(m+1)c}{i}(2^\lambda z)^i \end{aligned} \tag{5.255}$$

where we have chosen λ to be

$$\lambda = \log\frac{\rho}{(1-\rho)z} > 0, \quad 0 \leq R < R_{\text{exp}} \tag{5.256}$$

(cf. (5.14)). Next we upper-bound (5.255) by extending the summation to start at $i = 0$. Then we obtain

$$\begin{aligned} E_{\text{exp}}[P(\mathcal{E})] &\leq (2^b-1)2^{(R-1)(m+1)c}2^{-\lambda(d_0+1)}\sum_{i=0}^{(m+1)c}\binom{(m+1)c}{i}(2^\lambda z)^i \\ &= (2^b-1)2^{(R-1)(m+1)c}2^{-\lambda(d_0+1)}(1+2^\lambda z)^{(m+1)c} \end{aligned} \tag{5.257}$$

Inserting (5.256) into (5.257) and using (5.252) yield

$$\begin{aligned} E_{\text{exp}}[P(\mathcal{E})] &\leq (2^b-1)2^{(R-1)(m+1)c}\left(\frac{(1-\rho)z}{\rho}\right)^{d_0+1}(1-\rho)^{-(m+1)c} \\ &< (2^b-1)2^{(R-1)(m+1)c}\left((1-\rho)^{-(1-\rho)}\rho^{-\rho}\right)^{(m+1)c}z^{\rho(m+1)c} \\ &= (2^b-1)2^{(R-1+h(\rho))(m+1)c}z^{\rho(m+1)c} \\ &= (2^b-1)z^{\rho(m+1)c} = (2^b-1)2^{\rho\left(\log\left(2\sqrt{\epsilon(1-\epsilon)}\right)\right)(m+1)c} \end{aligned} \tag{5.258}$$

and the proof is complete. ∎

For the sphere-packing region, $R_{\text{crit}} \leq R < C$, we have the following:

Lemma 5.23 (Sphere-packing bound) There exists a binary, rate $R = b/c$, time-invariant convolutional code encoded by a generator matrix of memory m such that its burst error probability when used to communicate over the BSC with crossover probability ϵ with Viterbi decoding with back-search limit $\tau = m + 1$ is upper-bounded by the inequality

$$P_{\text{B}} < 2^{-\left(\rho \log \frac{\rho}{\epsilon} + (1-\rho) \log \frac{1-\rho}{1-\epsilon}\right)(m+1)c}, \quad R_{\text{crit}} \leq R < C \tag{5.259}$$

where ρ is the Gilbert-Varshamov parameter of the rate R, that is,

$$\rho = h^{-1}(1 - R) \tag{5.260}$$

R_{crit} is the critical rate, and C is the channel capacity for the BSC.

Proof: As usual, the proof in the sphere-packing region uses the idea of splitting the error events in "few" ($\mathcal{F}$) and "many" ($\mathcal{M}$) errors introduced in Section 4.3. Let

$$\mathcal{M} = \left\{ \boldsymbol{r}_{[0,m+1]} \;\middle|\; w_{\text{H}}(\boldsymbol{r}_{[0,m+1]} \geq \rho(m+1)c) \right\} \tag{5.261}$$

and $\mathcal{F}$ is the complement of $\mathcal{M}$. The average burst error probability is now upper-bounded by (cf. (5.24))

$$E[P_{\text{B}}] = E[P(\mathcal{E})] \leq E[P(\mathcal{E}), \mathcal{F}] + P(\mathcal{M}) \tag{5.262}$$

where $P(\mathcal{M})$ is upper-bounded by (cf. (5.33))

$$P(\mathcal{M}) < 2^{-\lambda\rho(m+1)c}(1 - \epsilon + 2^{\lambda}\epsilon)^{(m+1)c} \tag{5.263}$$

where λ is given by (5.34).

Analogously, $E[P(\mathcal{E}), \mathcal{F}]$ is upper-bounded by (cf. (5.39))

$$\begin{aligned} E[P(\mathcal{E}), \mathcal{F}] \leq\; & 2^{\lambda'\rho(m+1)c} 2^{(R-1)(m+1)c} (1 + 2^{-\mu})^{(m+1)c} \\ & \times (1 - \epsilon + \epsilon 2^{\mu - \lambda'})^{(m+1)c} \end{aligned} \tag{5.264}$$

where μ and λ' are given by (5.40) and (5.41), respectively.

By inserting (5.263) and (5.264) into (5.262) we obtain (cf. (5.47))

$$E[P(\mathcal{E})] < 2 \cdot 2^{-\left(\rho \log \frac{\rho}{\epsilon} + (1-\rho) \log \frac{1-\rho}{1-\epsilon}\right)(m+1)c} \tag{5.265}$$

If we instead use more elegant bounds analogous to (5.32) and (5.38), we can avoid the factor 2 and obtain (5.259). ∎

We can now summarize our results in the following:

Theorem 5.24 There exists a binary, rate $R = b/c$, time-invariant convolutional code encoded by a generator matrix of memory m such that its burst error probability

when used to communicate over the BSC with crossover probability ϵ with Viterbi decoding with back-search limit $\tau = m + 1$ is upper-bounded by the inequality

$$P_{\mathrm{B}} < 2^{-E^{\mathrm{fbs}}(R)(m+1)c} \tag{5.266}$$

where $E^{\mathrm{fbs}}(R)$ is the *finite back-search limit exponent* given by

$$E^{\mathrm{fbs}}(R) = \begin{cases} -\rho \log\left(2\sqrt{\epsilon(1-\epsilon)}\right) + \frac{\log(2^b-1)}{(m+1)c}, & 0 \le R < R_{\mathrm{exp}} \\ E_{\mathcal{B}}(R), & R_{\mathrm{exp}} \le R < C \end{cases} \tag{5.267}$$

and where

$$\rho = h^{-1}(1 - R) \tag{5.268}$$

is the Gilbert-Varshamov parameter of the rate R, $E_{\mathcal{B}}(R)$ is the block coding exponent (5.50), R_{exp} is the expurgation rate, R_{crit} is the critical rate, and C is the channel capacity for the BSC.

We notice that when we use Viterbi decoding with back-search limit $\tau = m + 1$, the burst error probability is determined by the block coding exponent. When we use Viterbi decoding with the decoding postponed to the end of the trellis, the burst error probability is determined by the convolutional coding exponent.

Finally, we remark that for the ensemble $\mathcal{E}(b, c, m, \infty)$ of time-varying convolutional codes encoded by generator matrices of memory m, Viterbi decoding with back-search limit τ can also be analyzed for $\tau > m + 1$ [Zig72].

5.6 QUANTIZATION OF CHANNEL OUTPUTS

In Section 5.1 we introduced the computational cutoff rate R_0, which is an important design criterion. Massey [Mas74] showed that it can be effectively employed in modulation system design. Here we will use the R_0 criterion to determine the quantization thresholds on the AWGN channel when BPSK modulation is used.

It is easily seen from (5.194) that for the unquantized output channel we have the following:

Theorem 5.25 The computational cutoff rate R_0 for a binary-input, unquantized output channel is

$$R_0 = 1 - \log\left(1 + \int_{-\infty}^{\infty} \sqrt{p_{r|v}(\alpha \mid 0)\, p_{r|v}(\alpha \mid 1)}\, d\alpha\right) \tag{5.269}$$

where $p_{r|v}(\cdot \mid \cdot)$ denotes the transition density function.

Corollary 5.26 The computational cutoff rate R_0 for the AWGN channel with BPSK modulation at signal-to-noise ratio E_{s}/N_0 is

$$R_0 = 1 - \log\left(1 + e^{-E_{\mathrm{s}}/N_0}\right) \tag{5.270}$$

Proof: From (5.269) follows

$$\begin{aligned} R_0 &= 1 - \log\left(1 + \int_{-\infty}^{\infty} \sqrt{\frac{1}{\sqrt{\pi N_0}} e^{-\frac{(\alpha+\sqrt{E_s})^2}{N_0}} \frac{1}{\sqrt{\pi N_0}} e^{-\frac{(\alpha-\sqrt{E_s})^2}{N_0}}}\, d\alpha\right) \\ &= 1 - \log\left(1 + \int_{-\infty}^{\infty} \frac{1}{\sqrt{\pi N_0}} e^{-\frac{\alpha^2+E_s}{N_0}}\, d\alpha\right) \\ &= 1 - \log\left(1 + e^{-E_s/N_0}\right) \end{aligned} \tag{5.271}$$

■

■ **EXAMPLE 5.4**

The unquantized AWGN channel with BPSK modulation at signal-to-noise ratio $E_s/N_0 = 0$ dB is

$$R_0(\infty) = 1 - \log(1 + e^{-1}) = 0.548 \tag{5.272}$$

Using hard decisions, that is, only two output values, we obtain a BSC with crossover probability (see Problem 1.32):

$$\epsilon = Q\left(\sqrt{2}\right) = 0.079 \tag{5.273}$$

and, hence,

$$R_0(2) = 1 - \log\left(1 + 2\sqrt{0.079(1 - 0.079)}\right) = 0.379 \tag{5.274}$$

which is substantially smaller than $R_0(\infty)$.

Which signal-to-noise ratio is required for a BSC to achieve the same computational cutoff rate R_0 as for the unquantized AWGN channel at $E_s/N_0 = 0$ dB?

From (5.194) it follows that the crossover probability for the BSC can be expressed as

$$\epsilon = \frac{1}{2} - \sqrt{2^{-R_0}(1 - 2^{-R_0})} \tag{5.275}$$

By combining (5.272) and (5.275) we obtain

$$\epsilon = 0.035 \tag{5.276}$$

and by solving (1.12)

$$\epsilon = Q\left(\sqrt{2E_s/N_0}\right) \tag{5.277}$$

$$E_s/N_0 = 2.15 \text{ dB} \tag{5.278}$$

that is, we have a slightly more than *2-dB loss due to hard decisions*.

The binary-input, 8-ary output DMC in Example 4.2 is a quantization to $q = 8$ output levels of an AWGN channel with BPSK modulation at signal-to-noise ratio $E_s/N_0 = 0$ dB. This particular quantization was in actual use in the space program. The corresponding thresholds were set before we knew how to find the optimum ones [Mas74] (see also Problem 5.2).

Definition We will say that a given quantization scheme with q different outputs is *optimum* if it maximizes R_0 for all ways of quantizing the channel to q outputs.

For a given q, the AWGN channel with BPSK modulation reduces to a binary-input, q-ary output DMC. Let the q output symbols be $r_1, r_2, \ldots, r_q$. From (5.194) it follows that

$$R_0 = 1 - \log\left(1 + \sum_{j=1}^{q} \sqrt{P\left(r_j \mid 0\right) P\left(r_j \mid 1\right)}\right) \tag{5.279}$$

Maximizing R_0 is equivalent to minimizing the sum on the right-hand side of (5.279). Let T_i be the quantization threshold between the regions where r_i and r_{i+1} will be the output. A necessary condition for this minimization is that for each i the derivative of the sum with respect to T_i equals 0. Since only the terms for $j = i$ and $j = i + 1$ in the sum depend on T_i, we have

$$\begin{aligned}
&\frac{d}{dT_i}\left(\sqrt{P\left(r_i \mid 0\right) P\left(r_i \mid 1\right)} + \sqrt{P\left(r_{i+1} \mid 0\right) P\left(r_{i+1} \mid 1\right)}\right) \\
&\quad = \frac{1}{2\sqrt{P\left(r_i \mid 0\right) P\left(r_i \mid 1\right)}}\left(P\left(r_i \mid 0\right) p\left(T_i \mid 1\right) + p\left(T_i \mid 0\right) P\left(r_i \mid 1\right)\right) \\
&\qquad + \frac{1}{2\sqrt{P\left(r_{i+1} \mid 0\right) P\left(r_{i+1} \mid 1\right)}} \\
&\qquad \times \left(-P\left(r_{i+1} \mid 0\right) p\left(T_i \mid 1\right) - p\left(T_i \mid 0\right) P\left(r_{i+1} \mid 1\right)\right)
\end{aligned} \tag{5.280}$$

Equation (5.280) follows from

$$\frac{dP(r_j \mid v)}{dT_i} = \begin{cases} p\left(T_i \mid v\right), & j = i \\ -p\left(T_i \mid v\right), & j = i + 1 \\ 0, & j \neq i,\ \ j \neq i + 1 \end{cases} \tag{5.281}$$

where we have used the fact that

$$\frac{d}{dx}\int_a^x f(z)dz = f(x) \tag{5.282}$$

and

$$\frac{d}{dx}\int_x^b f(z)dz = -f(x) \tag{5.283}$$

The condition that (5.280) equals 0 can be written

$$\begin{aligned}
&\sqrt{\frac{P\left(r_i \mid 0\right)}{P\left(r_i \mid 1\right)}} p\left(T_i \mid 1\right) + \sqrt{\frac{P\left(r_i \mid 1\right)}{P\left(r_i \mid 0\right)}} p\left(T_i \mid 0\right) \\
&\quad = \sqrt{\frac{P\left(r_{i+1} \mid 0\right)}{P\left(r_{i+1} \mid 1\right)}} p\left(T_i \mid 1\right) + \sqrt{\frac{P\left(r_{i+1} \mid 1\right)}{P\left(r_{i+1} \mid 0\right)}} p\left(T_i \mid 0\right), \quad \text{all } i
\end{aligned} \tag{5.284}$$

which is equivalent to

$$\frac{p\left(T_i \mid 0\right)}{p\left(T_i \mid 1\right)} = \sqrt{\frac{P\left(r_i \mid 0\right)}{P\left(r_i \mid 1\right)}}\sqrt{\frac{P\left(r_{i+1} \mid 0\right)}{P\left(r_{i+1} \mid 1\right)}}, \quad \text{all } i \tag{5.285}$$

Given that $r = \alpha$, we define the *likelihood ratio on the unquantized channel* as

$$\lambda(\alpha) \stackrel{\text{def}}{=} \frac{p\left(\alpha \mid 0\right)}{p\left(\alpha \mid 1\right)} \tag{5.286}$$

For the DMC obtained by quantization of the continuous channel, we define the *likelihood ratio for the output letter* r_j as

$$\lambda_{r_j} \stackrel{\text{def}}{=} \frac{P\left(r_j \mid 0\right)}{P\left(r_j \mid 1\right)} \tag{5.287}$$

The necessary condition (5.285) can then be formulated as follows:

Theorem 5.27 A quantization scheme with thresholds T_i which converts a binary-input, continuous output channel into a DMC with q output levels is optimum in the sense of maximizing R_0 for all quantization schemes giving q output *only if*

$$\lambda(T_i) = \sqrt{\lambda_{r_i}\lambda_{r_{i+1}}}, \quad \text{all } i \tag{5.288}$$

that is, the likelihood ratio when $r = T_i$ on the unquantized channel must be the geometric mean of the likelihood ratios for the two output letters whose quantization regions the threshold T_i divides.

Remark: Lee [Lee76] has constructed an example showing that the optimality condition (5.288) is in general *not* sufficient.

■ **EXAMPLE 5.5**

For the AWGN with BPSK modulation at signal-to-noise ratio E_s/N_0 we have

$$\lambda(\alpha) = \frac{\frac{1}{\sqrt{\pi N_0}} e^{-\frac{\left(\alpha - \sqrt{E_\mathrm{s}}\right)^2}{N_0}}}{\frac{1}{\sqrt{\pi N_0}} e^{-\frac{\left(\alpha + \sqrt{E_\mathrm{s}}\right)^2}{N_0}}} = e^{\frac{4\alpha\sqrt{E_\mathrm{s}}}{N_0}} \tag{5.289}$$

which is a single-valued function of α.

Massey suggested the following algorithm to compute the set of thresholds [Mas74]:

Choose T_1 arbitrarily. This determines λ_{r_1} as well as $\lambda(T_1)$. Hence, we can then choose T_2 such that the resulting λ_{r_2} will satisfy (5.288). We can then choose T_3

such that the resulting λ_{r_3} will satisfy (5.288) and so on. If we can complete the procedure up to the choice of T_{q-1} and this choice gives as well

$$\lambda(T_{q-1}) = \sqrt{\lambda_{r_{q-1}}\lambda_{r_q}} \tag{5.290}$$

then we stop. If we are unable to choose some T_i along the way or if

$$\lambda(T_{q-1}) > \sqrt{\lambda_{r_{q-1}}\lambda_{r_q}} \tag{5.291}$$

then we have to decrease T_1 and repeat the procedure. On the other hand, if

$$\lambda(T_{q-1}) < \sqrt{\lambda_{r_{q-1}}\lambda_{r_q}} \tag{5.292}$$

then we know that our choice of T_1 was too small and we must adjust our guess of T_1 and repeat the procedure.

■ **EXAMPLE 5.6**

By applying the previous procedure, we obtain the following quantization thresholds for the AWGN channel with BPSK at signal-to-noise ratio $E_s/N_0 = 0$ dB and assuming $\sqrt{E_s} = 1$:

q	$R_0(q)$	T_1	T_2	T_3	T_4	T_5	T_6	T_7
2	0.378	0						
4	0.498	−0.731	0	−0.731				
8	0.534	−1.273	−0.761	−0.362	0	0.362	0.0.761	1.273
∞	0.548							

Symmetry considerations indicate that we have only one local maximum of R_0. This is also the global maximum; hence, the quantization thresholds are optimum.

■ **EXAMPLE 5.7**

To achieve a cutoff rate $R_0 = 0.548$ we need the following signal-to-noise ratios:

q	R_0	E_s/N_0 dB
2	0.548	2.148
4	0.548	0.572
8	0.548	0.157
∞	0.548	0

From the previous example it follows that compared to hard decisions there is a considerable gain in using four levels of quantization but not much of an advantage to go beyond eight levels.

5.7 COMMENTS

Error probability bounds for block codes when used to communicate over the BSC were already presented by Elias in 1955 [Eli55]. The modern versions of the upper bounds for block codes as presented here are inspired by Fano [Fan61] and Gallager [Gal65, Gal68]. The "few" and "many" idea which we have exploited several times in this volume can be found in Fano's textbook.

Lower bounds on the error probability for block codes were derived by Shannon, Gallager, and Berlekamp [SGB67].

For convolutional codes the upper bounds on the error probability were derived by Yudkin in 1964 [Yud64] and the lower bounds by Viterbi in 1967 [Vit67]. The tight upper bounds on the error probability were also derived in [Zig85].

The ideas of constructing the convolutional coding exponent from the block coding exponent and vice versa, as well as the concept of critical length, go back to Forney [For74].

For general, nonlinear trellis codes, Pinsker [Pin67] derived a lower bound on the error probability for decoding with finite back-search limit τ. His bound is similar to the sphere-packing bound for block codes of block length $N = \tau c$.

PROBLEMS

5.1 Show the following properties of the Gallager function (5.98).

- **a)** $G(s) > 0$ for $s > 0$
- **b)** $G'(s) > 0$ for $s \geq 0$
- **c)** $G''(s) < 0$ for $s > 0$
- **d)** $\lim_{s\to\infty} G'(s) = 0$
- **e)** $G'(0) = C$
- **f)** $G'(1) = R_{\text{crit}}$

5.2 Verify (5.162) for d odd.

5.3 Find a Bhattacharyya-type upper bound for the burst error probability when using a convolutional code with two codewords for communication over a binary-input, unquantized output DMC with transition density function $p_{r|v}$.

5.4 Show that for a binary-input DMC, R_0 is maximized when the inputs are used with equal probability.

5.5

- **a)** Find R_0 for the binary-input, 8-ary output DMC given in Example 4.2.
- **b)** Convert the channel in (a) to a *binary erasure channel* (BEC) by combining the soft-decision outputs $0_4, 0_3; 0_2, 0_1, 1_1, 1_2$; and $1_3, 1_4$ to hard-decision outputs $0, \Delta$, and 1, respectively. Find R_0 for this channel.
- **c)** Find R_0 for the BSC in Problem 4.5.

5.6

- **a)** Find R_0 for the AWGN channel with BPSK modulation at $E_s/N_0 = 1$ dB.
- **b)** Convert the channel in (a) to a 4-ary DMC with optimum R_0 and find the quantization thresholds.

c) Repeat (b) for an 8-ary DMC.

5.7 Repeat Problem 5.6 for $E_s/N_0 = 2$ dB.

CHAPTER 6

LIST DECODING

Viterbi decoding (Chapter 4) is an example of a nonbacktracking decoding method that at each time instant examines the total encoder state space. The error-correcting capability of the code is fully exploited. We first choose a suitable code and then design the decoder in order to "squeeze all juice" out of the chosen code.

Sequential decoding (Chapter 7) is a backtracking decoding method that (asymptotically) fully exploits the error-correcting capability of the code.

In *list decoding* we first limit the resources of the *decoder*; then we choose a generator matrix with a state space that is larger than the *decoder* state space. Thus, assuming the same decoder complexity, we use a more powerful code with list decoding than with Viterbi decoding. A list decoder is a powerful nonbacktracking decoding method that does not fully exploit the error-correcting capability of the code.

In this chapter we describe and analyze *list decoding*, which is an important and interesting decoding method based on the idea that we at each time instant create a list of the L most promising initial parts of the codewords. For a given decoder complexity, list decoding of convolutional codes encoded by systematic encoding matrices is in fact superior to Viterbi decoding of convolutional codes encoded by nonsystematic encoding matrices.

Fundamentals of Convolutional Coding, 2. Edition. By Rolf Johannesson, Kamil Sh. Zigangirov

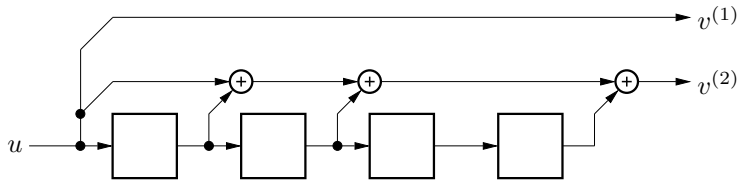


Figure 6.1 A rate $R = 1/2$ systematic convolutional encoder with encoding matrix $G(D) = (1 \quad 1 + D + D^2 + D^4)$.

6.1 LIST DECODING ALGORITHMS

List decoding is a nonbacktracking breadth-first search of the code tree. At each depth only the L most promising subpaths are extended, not all, as is the case with Viterbi decoding. These subpaths form a *list* of size L. (Needless to say, starting at the root, *all* subpaths are extended until we have obtained L or more subpaths.)

Since the search is breadth-first, all subpaths on the list are of the same length; finding the L best extensions reduces to choosing the L extensions with the largest values of the Viterbi metric (4.2).

Assuming a rate $R = b/c$ convolutional encoder of memory m, we append, as was the case with Viterbi decoding, a tail of bm dummy zeros to the information bits in order to terminate the convolutional code into a block code. The following algorithm is the simplest version of a list decoding algorithm:

Algorithm LD (List decoding)

LD1. Load the list with the root and metric zero; set $t = 0$.

LD2. Extend all stored subpaths to depth $t + 1$ and place the L best (largest Viterbi metric) of their extensions on the list.

LD3. If we have reached the end, then stop and choose as the decoded codeword a path to the terminating node with the largest Viterbi metric; otherwise increment t by 1 and go to **LD2**.

Assume that the allzero information sequence is encoded by the rate $R = 1/2$, memory $m = 4$, systematic encoder with encoding matrix $G(D) = (1 \quad 1 + D + D^2 + D^4)$ given in Fig. 6.1 and that the sequence $\boldsymbol{r} = 00\,11\,01\,00\,00\,00\,00\,00\ldots$ is received over a BSC.

In Fig. 6.2 we show the code tree that is partially explored by the list decoding algorithm. We have used a list of size $L = 3$, which should be compared with the 16 states that would have been examined at each depth by the Viterbi algorithm. At the root and at depth 1, only one and two states, respectively, are extended. The upper (lower) branch stemming from an extended state represents information bit 0 (1). We notice that the correct path is lost at depth 4 and that the correct state is recovered at depth 6. A *correct path loss* is a serious kind of error event that is typical for list decoding. It will be discussed in depth later in this chapter.

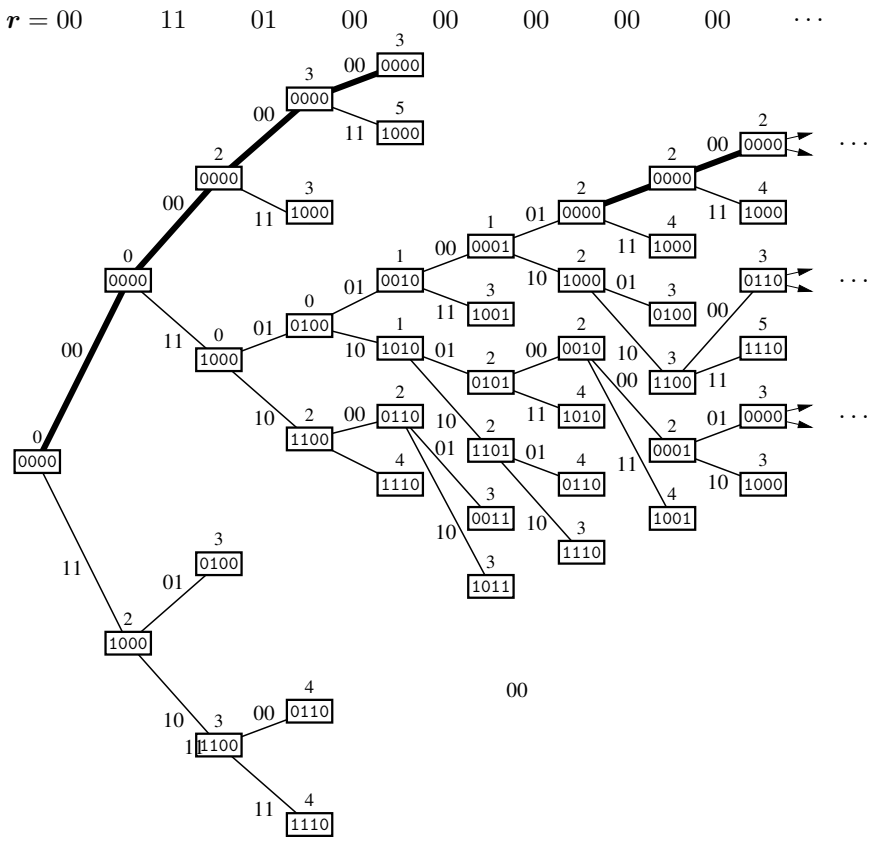


Figure 6.2 List decoding ($L = 3$). Partially explored tree demonstrating a correct path loss at depth 4 and a spontaneous recovery of the correct state at depth 6.

The extension process that takes place in Fig. 6.2 is illustrated in Fig. 6.3 in a way that resembles an implementation of the list decoding algorithm. The upper branches enter states at even positions counted from the top (positions $0, 2, \ldots, 2L-2$), and the lower branches enter states at odd positions (positions $1, 3, \ldots, 2L-1$).

An efficient representation of the extension process is given in Fig. 6.4. It is understood that we have the states in the same positions as in Fig. 6.3. The extended states are denoted by "$\cdot$". The extended paths can be traced backward through the array in the following way. Suppose that the $*$ at breadth k and depth $j+1$ represents the best path traced backward so far. The best position at depth j is then represented by the $(\lfloor k/2 \rfloor + 1)$th $*$ (counted from the top). Furthermore, the decided information bit is $k \pmod 2$.

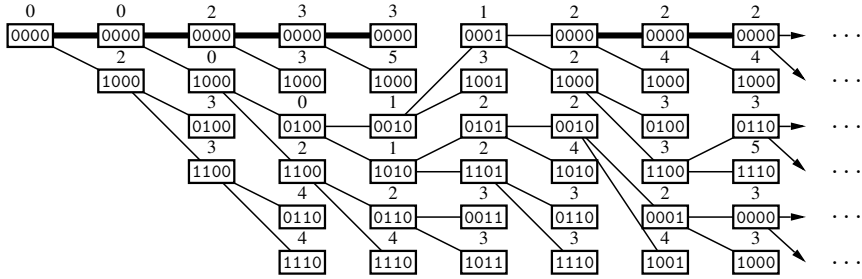


Figure 6.3 An illustration of the extension process for list decoding ($L = 3$).

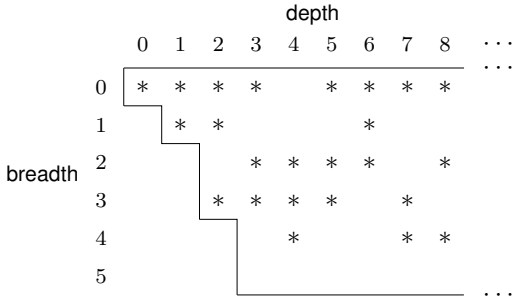


Figure 6.4 An example of an array representing the extension process for list decoding ($L = 3$).

■ EXAMPLE 6.1

Consider the array in Fig. 6.4 and suppose that the $*$ at breadth 0 and depth 5 represents the best path traced backward so far. Then we obtain $(\lfloor 0/2 \rfloor + 1) = 1$, that is, the first $*$ at depth 4, and $\widehat{u}_4 = 0 \pmod 2 = 0$.

- The first $*$ at depth 4 is at breadth 2. Thus, continuing backward we obtain $(\lfloor 2/2 \rfloor + 1) = 2$, that is, the second $*$ at depth 3, and $\widehat{u}_3 = 2 \pmod 2 = 0$.
- The second $*$ at depth 3 is at breadth 2. Hence, we have $(\lfloor 2/2 \rfloor + 1) = 2$, that is, the second $*$ at depth 2, and $\widehat{u}_2 = 2 \pmod 2 = 0$.
- The second $*$ at depth 2 is at breadth 1. Thus, $(\lfloor 1/2 \rfloor + 1) = 1$, that is, the first $*$ at depth 1, and $\widehat{u}_1 = 1 \pmod 2 = 1$.
- Finally, the first $*$ at depth 1 is at breadth 0 and, hence, $\widehat{u}_0 = 0$.

In conclusion, the decided information sequence is $\widehat{\boldsymbol{u}} = (01000\ldots)$.

Let us return to the partially explored tree in Fig. 6.2. We notice that at depth 8 we have a duplicate of the state (0000). The two states (0000) have accumulated Hamming distances 2 and 3, respectively. The two subtrees stemming from these two nodes are, of course, identical. Hence, a path passing the first of these two states will always be superior to a corresponding path passing the latter one. We will obtain

an improved version of the list decoding algorithm if at each depth we delete inferior duplicates before we select the L best extensions to be put on the list. Searching only for duplicates of only one state (in our case the tentatively best) can easily be done linearly in L. We do not worry about duplicates of other states since subpaths stemming from nonoptimal states are deleted from the list of the L best subpaths very fast. Furthermore, obtaining the Lth poorest path in the list is of order L [Knu73], and comparing the remaining paths to this one is also of order L. Hence, the list decoding algorithm is actually linear in L [MoA84, AnM91].

6.2 LIST DECODING—PERFORMANCE

Consider a rate $R = b/c$, memory m encoder for a convolutional code $\mathcal{C}$, the BSC, and the received binary sequence

$$\boldsymbol{r}_{[0,t]} = \boldsymbol{r}_0 \boldsymbol{r}_1 \ldots \boldsymbol{r}_t \tag{6.1}$$

where $\boldsymbol{r}_i \in \mathbb{F}_2^c$, $0 \leq i \leq t$. We look at the sphere $\mathcal{S}_\delta(\boldsymbol{r}_{[0,t]})$ with radius δ around the received sequence $\boldsymbol{r}_{[0,t]}$, that is,

$$\mathcal{S}_\delta(\boldsymbol{r}_{[0,t]}) = \left\{ \boldsymbol{y}_{[0,t]} \mid d_{\mathrm{H}}(\boldsymbol{r}_{[0,t]}, \boldsymbol{y}_{[0,t]}) \leq \delta \right\} \tag{6.2}$$

where $\boldsymbol{y}_{[0,t]} \in \mathbb{F}_2^{(1+t)c}$. The number of (initial parts of length $(1+t)c$ of the) codewords $\boldsymbol{v}$ in this sphere is

$$N_t(\delta, \boldsymbol{r}_{[0,t]}) = \left| \left\{ \boldsymbol{v}_{[0,t]} \in \mathcal{S}_\delta(\boldsymbol{r}_{[0,t]}) \mid \boldsymbol{v} \in \mathcal{C} \right\} \right| \tag{6.3}$$

Let $N(e, \boldsymbol{r})$ be the maximal number of codewords which can occur in a sphere with center $\boldsymbol{r}$ and radius e over all t, that is,

$$N(e, \boldsymbol{r}) = \max_t \{ N_t(e, \boldsymbol{r}_{[0,t]}) \} \tag{6.4}$$

Maximizing over all possible received sequences, we get the sphere of radius e with the maximal number of codewords for the code $\mathcal{C}$:

$$N_{\max}(e) \stackrel{\text{def}}{=} \max_{\boldsymbol{r}} \{ N(e, \boldsymbol{r}) \} \tag{6.5}$$

A list decoder that has to correct all e-error combinations, no matter where they start, must keep at least $L = N_{\max}(e)$ paths in the decoder memory. Otherwise the decoder might lose the correct path, and there exists at least one e-error sequence that may not be corrected.

How large should we choose L? If

$$L \geq N_{\max}(e) \tag{6.6}$$

then the following statements hold:

(i) If at most e errors occur, then a list decoder of list size L will not lose the correct path.

(ii) If the used code has free distance $d_{\text{free}} \geq 2e+1$, then all e-error combinations will be decoded correctly.

The parameter $N_{\max}(e)$ can be illustrated by assuming that all codewords at some trellis level are points in a plane and by using a coin [And89]. Move a coin with radius e around until it covers the largest number of codewords. The largest such number at any trellis level is $N_{\max}(e)$, the least L which is necessary to avoid losing the correct path.

■ **EXAMPLE 6.2**

The following table shows $N_{\max}(e)$ for the systematic rate $R = 1/2$, memory $m = 11$, convolutional encoding matrix $G(D) = (1 \quad 1+D+D^2+D^5+D^6+D^8+D^{10}+D^{11})$ with free distance $d_{\text{free}} = 9$.

e	2	3	4
$N_{\max}(e)$	4	9	19

A list decoder for this encoding matrix decodes all 4-error combinations correctly if L is large enough. The sphere around any received sequence with radius 4 contains at most $N_{\max}(4) = 19$ codewords. So we have to use at least $L = 19$ paths in order to fully reach the 4-error correction potential of the given code.

In Chapter 2 we showed in Theorem 2.74 that every convolutional generator matrix is equivalent to a systematic rational encoding matrix. Consider a rate $R = b/c$, memory m, nonsystematic polynomial convolutional encoding matrix and its systematic rational equivalent. Expand the rational functions of the systematic encoding matrix into power series in D and truncate after D^m. Then we obtain a *systematic polynomial* encoding matrix that is equivalent to the *nonsystematic* one *over the first memory length*, that is, their code trees are *identical* over the first $m+1$ branches. Hence, these two encoding matrices have exactly the same distance profile.

Suppose that both a nonsystematic polynomial encoding matrix of memory m and a systematic polynomial encoding matrix, in general also of memory m, that are equivalent over the first memory length are used together with a list decoder.

For a range of interesting values of L, the list decoder will operate mostly in the identical parts of the code trees encoded by the two encoders. The burst error probability *measured at the root* will be almost the same for both encoders.

Consider the memory $m = 31$ ODP nonsystematic convolutional encoding matrix (octal notation, see Chapter 10) $G_{\text{nonsys}} = (74041567512 \quad 54041567512)$ with $d_{\text{free}} = 25$. By long division of the generator polynomials and truncation, we obtain the following memory $m = 31$ ODP systematic convolutional encoding matrix $G_{\text{sys}} = (40000000000 \quad 67115143222)$ with $d_{\text{free}} = 16$. These two encoding matrices are equivalent over the first memory length. In Fig. 6.5 we compare for various L the burst error probability P_{B} measured at the root of the code tree when the received

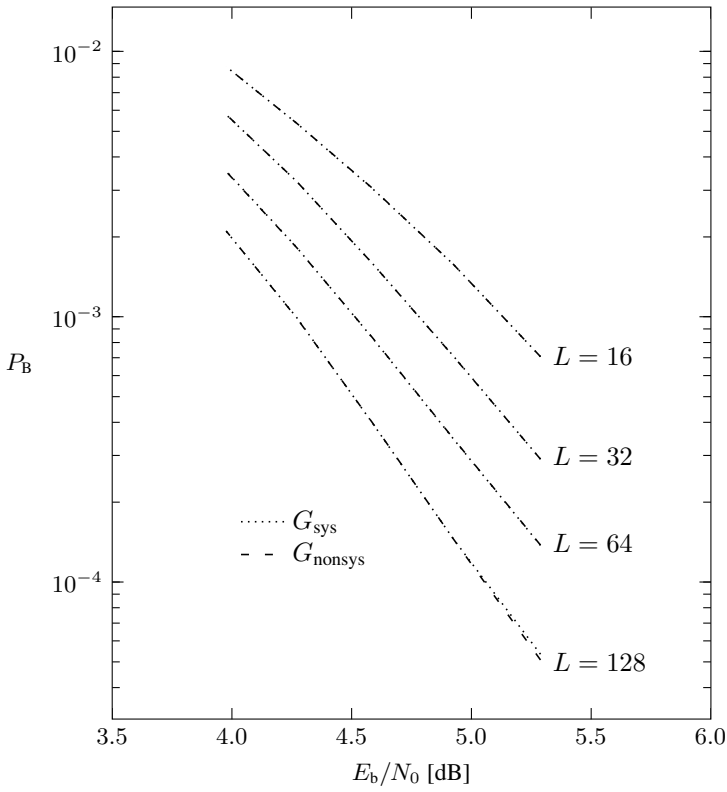


Figure 6.5 Burst error probability, P_{B}, measured at the root for list decoding—systematic vs. nonsystematic convolutional encoding matrices.

sequences are decoded by a list decoder. The simulations give striking support of the conclusions given above.

A very different result occurs when encoding matrices with widely varying distance profiles are tested. In Fig. 6.6 we compare the burst error probabilities at $L = 32$ for five encoding matrices, all with $d_{\mathrm{free}} = 10$, each of whose distance profile successively underbounds the others. The result is a sequence of widely varying P_{B} curves arranged in the same order. The encoding matrices are

$$
\begin{aligned}
G_1 &= (400000 \quad 714474), \quad m = 15, \quad d_{\mathrm{free}} = 10 \\
&\qquad \boldsymbol{d} = (2, 3, 3, 4, 4, 5, 5, 6, 6, 6, 7, 7, 8, 8, 8, 8) \\
G_2 &= (400000 \quad 552234), \quad m = 15, \quad d_{\mathrm{free}} = 10 \\
&\qquad \boldsymbol{d} = (2, 2, 3, 4, 4, 4, 57, 5, 5, 6, 6, 6, 7, 7, 8, 8)
\end{aligned}
$$

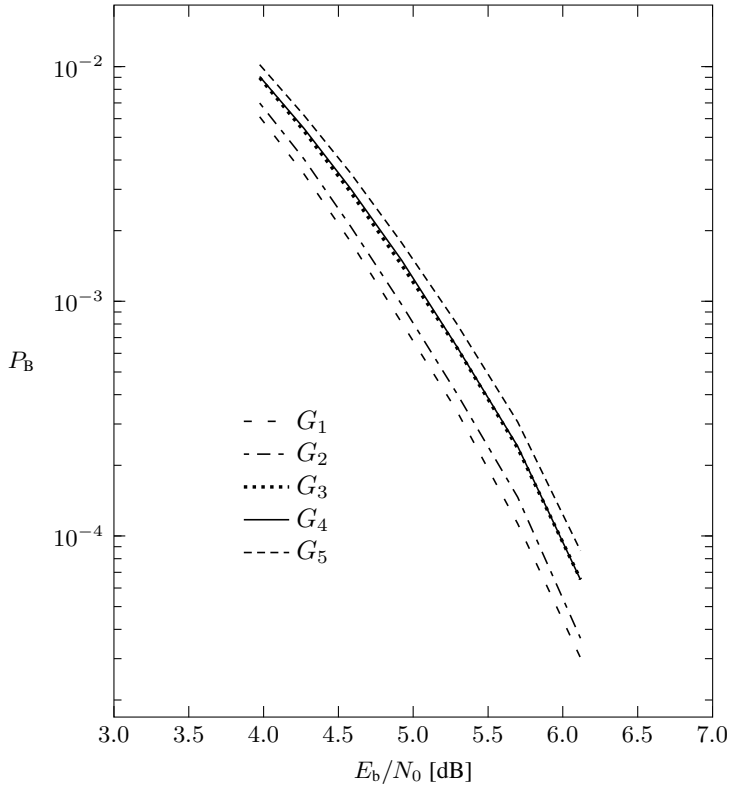


Figure 6.6 A nested set of distance profiles leads to a nested set of P_B curves.

$$
\begin{aligned}
G_3 &= (400000 \quad 447254), \quad m = 15, \quad d_{\text{free}} = 10 \\
&\qquad \boldsymbol{d} = (2, 2, 2, 3, 3, 3, 3, 4, 5, 6, 6, 6, 6, 7, 7, 7) \\
G_4 &= (400000 \quad 427654), \quad m = 15, \quad d_{\text{free}} = 10 \\
&\qquad \boldsymbol{d} = (2, 2, 2, 2, 3, 3, 4, 5, 5, 5, 5, 5, 6, 6, 6) \\
G_5 &= (400000 \quad 417354), \quad m = 15, \quad d_{\text{free}} = 10 \\
&\qquad \boldsymbol{d} = (2, 2, 2, 2, 2, 3, 4, 5, 5, 6, 6, 6, 6, 6, 7)
\end{aligned}
$$

It appears that for a given list size L almost all the variations in the burst error probability for different encoding matrices can be traced to variations in the distance profiles. The burst error probability performance depends almost entirely on the distance profile and not on either the free distance or the type of the encoding matrix (systematic or nonsystematic).

Now we will turn to perhaps the most important comparison: list versus Viterbi decoding. In Fig. 6.7 we show the results when nonsystematic encoding matrices with optimum free distances are used. The list decoder decodes a memory $m = 9$ encoding matrix at $L = 16, 32, 64, 128, 256$, while the Viterbi algorithm decodes

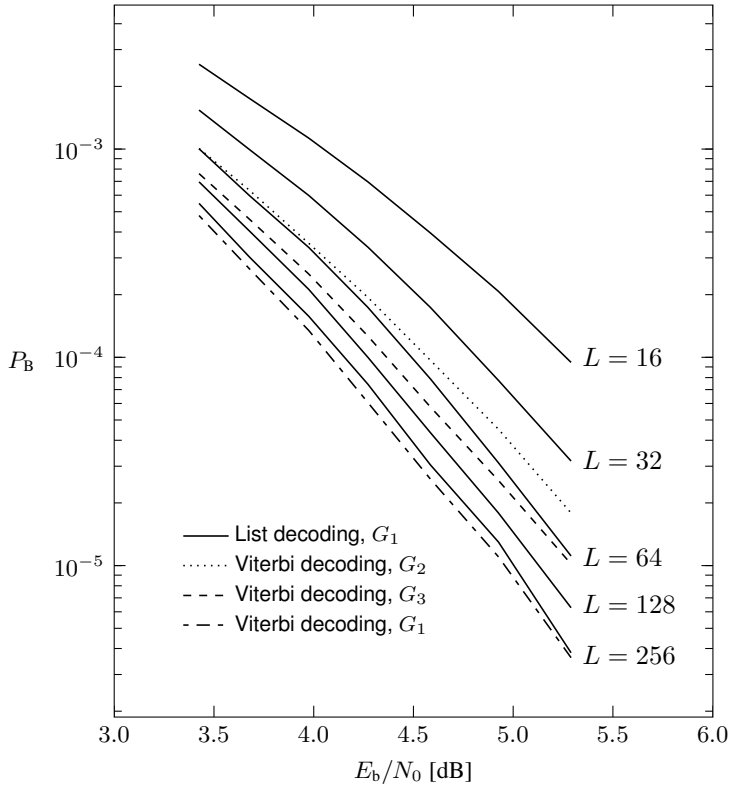


Figure 6.7 Burst error probability, P_B. List vs. Viterbi decoding.

three encoding matrices of memory $7, 8$, and 9 whose survivor numbers are $128, 256$, and 512, respectively. From the simulations we conclude that for the same burst error probability the Viterbi algorithm requires somewhat more than twice the survivors of the list decoder. The encoding matrices are

$$G_1 = (4734 \quad 6624), \quad m = 9, \quad d_{\text{free}} = 12 \tag{6.7}$$

$$G_2 = (712 \quad 476), \quad m = 7, \quad d_{\text{free}} = 10 \tag{6.8}$$

$$G_3 = (561 \quad 753), \quad m = 8, \quad d_{\text{free}} = 12 \tag{6.9}$$

So far we have considered only burst error probability. The situation is quite different when we deal with bit error probability. A correct path loss is a severe event that causes many bit errors. If the decoder cannot recover a lost correct path it is, of course, a "catastrophe," that is, a situation similar to the catastrophic error propagation that can occur when a catastrophic encoding matrix is used to encode the information sequence. The list decoder's ability to recover a lost correct path depends heavily on the type of *encoding matrix* that is used.

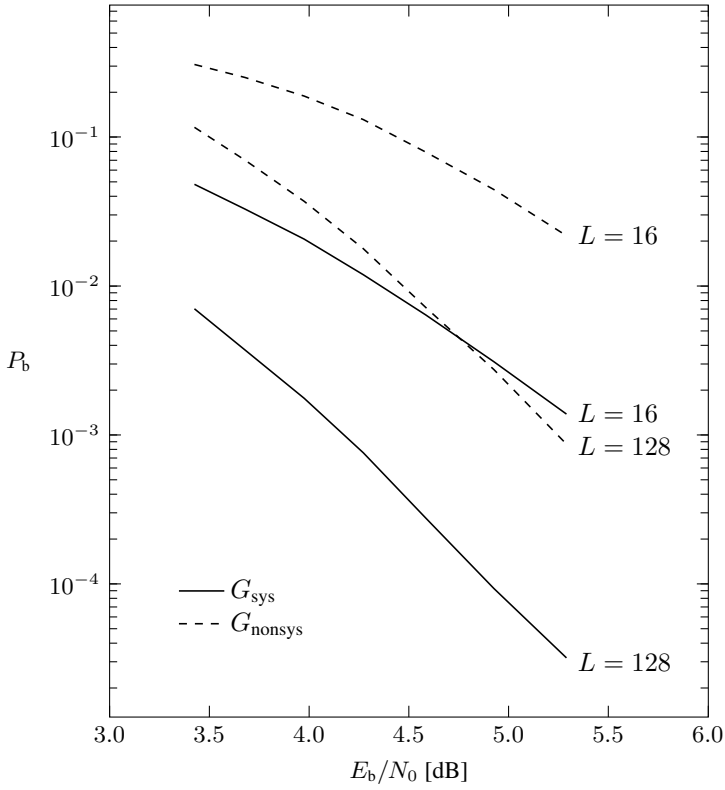


Figure 6.8 Bit error probability, P_b, for list decoding—systematic vs. nonsystematic, polynomial convolutional encoding matrices.

A systematic, polynomial encoding matrix supports a spontaneous recovery. This is illustrated in Fig. 6.8 where we compare the bit error probability for list decoders with various list sizes when they are used to decode sequences received over a BSC and encoded with both systematic and nonsystematic, polynomial encoding matrices with the same distance profile. The free distance of the systematic, polynomial encoding matrix is by far the least, yet its bit error probability is more than 10 times better. The encoding matrices are

$$G_{sys} = (4000000 \quad 7144655), \quad m = 20, \quad d_{free} = 13 \tag{6.10}$$

$$G_{nonsys} = (6055051 \quad 4547537), \quad m = 20, \quad d_{free} = 22 \tag{6.11}$$

The only advantage of the nonsystematic encoding matrix is its larger free distance. Yet this extra distance has almost no effect on either the burst or the bit error probability. Nor does it change the list size L needed to correct e errors as long as e falls within the powers of the systematic, polynomial encoding matrix.

A suggestion of why systematic, polynomial encoding matrices offer rapid recovery of a lost correct path may be found by considering the trellises of rate $R = 1/2$ random systematic, polynomial, and nonsystematic encoding matrices. Suppose the correct path is the allzero one and no errors occur for a time, and consider an arbitrary trellis node. The 0-branch (the one that inserts a zero into the encoder shift register) is the one leading back to the correct path. For a systematic, polynomial encoding matrix, the distance increment of this "correct" branch is 0.5 on the average, while the incorrect branch has increment 1.5. For a nonsystematic encoding matrix, these average increments are both 1 and give no particular direction of the search back to the correct path.

In conclusion, using systematic, polynomial convolutional encoding matrices essentially solves the correct path loss problem with list decoders. Since both systematic and nonsystematic encoding matrices have the same error rate in the absence of correct path loss, systematic, polynomial encoding matrices are clearly superior to nonsystematic ones.

In Fig. 6.9 we compare list and Viterbi decoding with respect to the bit error probability. We have chosen Viterbi decoders with complexities that are twice the list sizes of the list decoders. The list decoders outperform the corresponding Viterbi decoders. The encoding matrices are

$$G_1 = (4000000 \quad 7144655), \quad m = 20, \quad d_{\text{free}} = 13 \tag{6.12}$$

$$G_2 = (712 \quad 476), \quad m = 7, \quad d_{\text{free}} = 10 \tag{6.13}$$

$$G_3 = (561 \quad 753), \quad m = 8, \quad d_{\text{free}} = 12 \tag{6.14}$$

$$G_4 = (4734 \quad 6624), \quad m = 9, \quad d_{\text{free}} = 12 \tag{6.15}$$

6.3 THE LIST MINIMUM WEIGHT

In this section we will introduce the *list minimum weight* for convolutional codes. It is an important parameter when we analyze the error performance of list decoding. It is related to the number of errors that can be guaranteed to be corrected by a list decoder.

In the previous section we used a sphere of fixed radius e. By counting the number of codewords within the sphere when we moved the center of the sphere to all possible received sequences with at most e errors, we obtained guidelines on how to choose the list size L for a given guaranteed error-correcting capability e. Now we will turn the problem around. Consider a list decoder with a fixed list size L. For every $t = 0, 1, \ldots$ and every received sequence $\boldsymbol{r}_{[0,t]} \in \mathbb{F}_2^{(1+t)c}$, let $\delta_L(\boldsymbol{r}_{[0,t]})$ denote the largest radius of a sphere $\mathcal{S}_{\delta_L(\boldsymbol{r}_{[0,t]})}(\boldsymbol{r}_{[0,t]})$ with center $\boldsymbol{r}_{[0,t]}$ such that the number of codewords in the sphere is

$$N_t(\delta_L(\boldsymbol{r}_{[0,t]}), \boldsymbol{r}_{[0,t]}) \leq L \tag{6.16}$$

The smallest such radius is of particular significance, and we have the following:

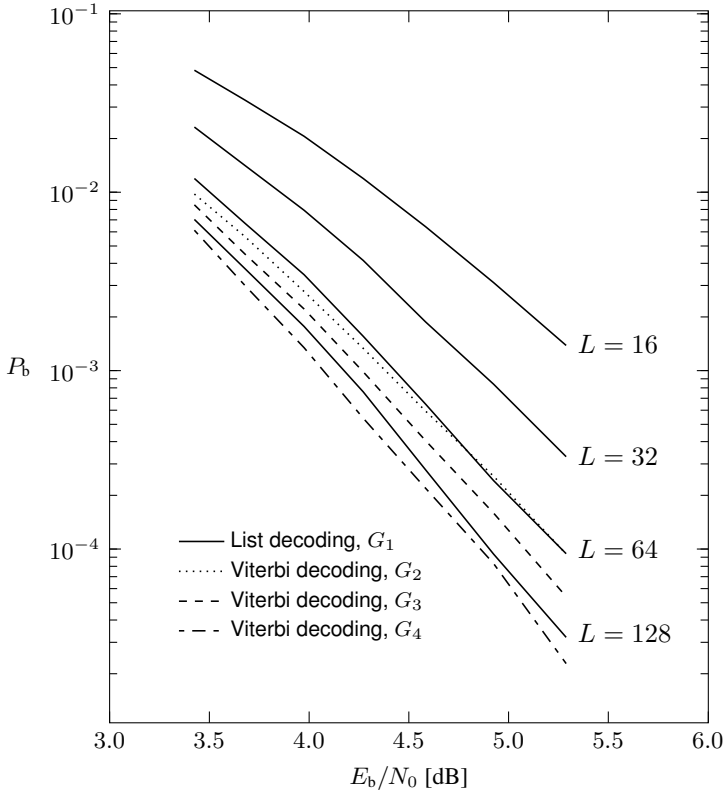


Figure 6.9 Bit error probability, P_{b}. List vs. Viterbi decoding.

Definition For a list decoder with a given list size L, the *list minimum weight* $w_{\min}$ is

$$w_{\min} = \min_{t} \min_{\boldsymbol{r}_{[0,t]}} \left\{ \delta_L(\boldsymbol{r}_{[0,t]}) \right\} \tag{6.17}$$

where $\boldsymbol{r}_{[0,t]}$ is the initial part of the received sequence $\boldsymbol{r}$.

We immediately have the next theorem.

Theorem 6.1 Given a list decoder of list size L and a received sequence with at most $w_{\min}$ errors. Then the correct path will not be forced outside the list of L survivors.

Unfortunately, $w_{\min}$ is hard to estimate. This leads us to restrict the minimization to those received sequences that are codewords. Thus, we obtain the following:

Definition For a given list size L the *list weight* w_{list} of the convolutional code $\mathcal{C}$ is

$$w_{\text{list}} = \min_{t} \min_{\boldsymbol{v}_{[0,t]}} \{ \delta_L(\boldsymbol{v}_{[0,t]}) \} \tag{6.18}$$

where $\boldsymbol{v}_{[0,t]}$ is the initial part of the codeword $\boldsymbol{v} \in \mathcal{C}$.

The importance of the *list minimum weight* can be inferred from the following:

Theorem 6.2 The list minimum weight $w_{\min}$ is upper- and lower-bounded by w_{list} according to

$$\left\lfloor \frac{1}{2} w_{\text{list}} \right\rfloor \leq w_{\min} \leq w_{\text{list}} \tag{6.19}$$

Proof: The right inequality of (6.19) follows immediately from (6.17) and (6.18).

To prove the left inequality of (6.19), we consider the spheres $\mathcal{S}_{\lfloor \frac{1}{2} w_{\text{list}} \rfloor}(\boldsymbol{r}_{[0,t]})$ of radius $\lfloor \frac{1}{2} w_{\text{list}} \rfloor$ with centers at all received sequences $\boldsymbol{r}_{[0,t]}$. If all such spheres contain less than L codewords, we are done. If not, consider any one of the nonempty spheres and take another sphere $\mathcal{S}_{w_{\text{list}}}(\boldsymbol{v}_{[0,t]})$ of radius w_{list} with center at any codeword $\boldsymbol{v}^{(i)}$ in the smaller sphere (Fig. 6.10). Clearly, we have

$$\mathcal{S}_{\lfloor \frac{1}{2} w_{\text{list}} \rfloor}(\boldsymbol{r}_{[0,t]}) \subseteq \mathcal{S}_{w_{\text{list}}}(\boldsymbol{v}^{(i)}_{[0,t]}) \tag{6.20}$$

By the definition of w_{list}, the larger sphere contains at most L codewords. Thus, from (6.20) it follows that the smaller sphere contains at most L codewords. Furthermore, the correct path cannot be forced outside the list if the received sequence $\boldsymbol{r}_{[0,t]}$ contains at most $\lfloor \frac{1}{2} w_{\text{list}} \rfloor$ errors. ■

From Theorems 6.1 and 6.2 we immediately have the following:

Corollary 6.3 Given a list decoder of list size L and a received sequence with at most $\lfloor \frac{1}{2} w_{\text{list}} \rfloor$ errors. Then the correct path will not be forced outside the list of L survivors.

If the number of errors exceeds $\lfloor \frac{1}{2} w_{\text{list}} \rfloor$, then it depends on the code $\mathcal{C}$ and on the received sequence $\boldsymbol{r}$ whether the correct path is not forced outside the list.

■ **EXAMPLE 6.3**

Both the list minimum weight $w_{\min}$ and the list weight w_{list} for the convolutional code $\mathcal{C}$ encoded by the rate $R = 1/2$, memory $m = 4$, systematic convolutional encoding matrix $G(D) = (1 \quad 1 + D + D^2 + D^4)$ are shown in Fig. 6.11.

To prove a random coding lower bound on w_{list} we consider the ensemble of $\mathcal{E}(b, c, \infty, 1)$, that is, the ensemble of infinite-memory, time-invariant convolutional codes with generator matrix

$$\boldsymbol{G} = \begin{pmatrix} G_0 & G_1 & \dots & \\ & G_0 & G_1 & \dots \\ & & \ddots & \ddots \end{pmatrix} \tag{6.21}$$

in which each digit in each of the matrices G_i, $i = 0, 1, \ldots$, is chosen independently with probability $1/2$. Hence, over the ensemble $\mathcal{E}(b, c, \infty, 1)$ all code symbols on a

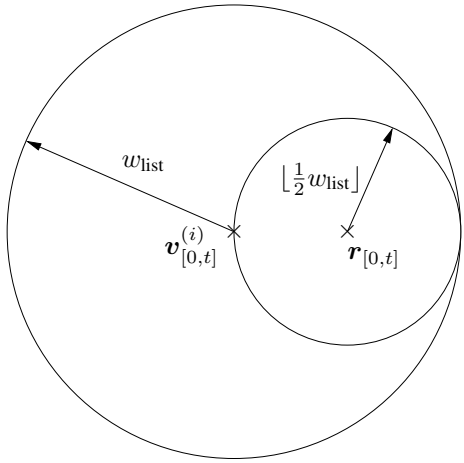


Figure 6.10 Illustration for the proof of Theorem 6.2.

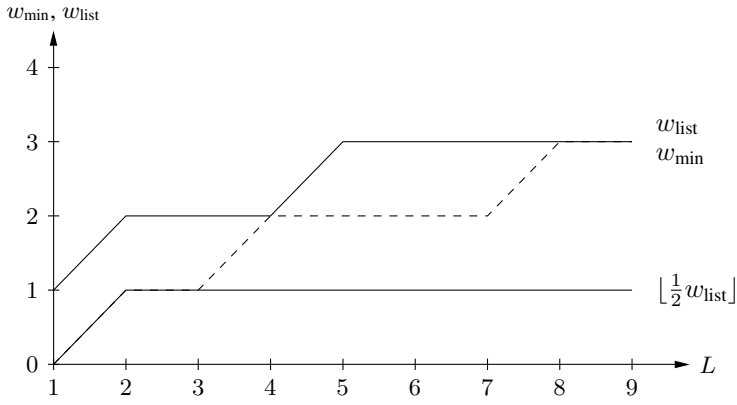


Figure 6.11 The list minimum weight $w_{\min}$ and the list weight w_{list} for the convolutional code $\mathcal{C}$ given in Example 6.3.

subpath diverging from the allzero path are mutually independent (cf. Theorem 3.25). Furthermore, these code symbols are also equally probable binary digits.

The next lemma and theorem establish lower bounds on the list weight w_{list} that are similar to Costello's lower bound on the free distance of a convolutional code (Theorem 3.28):

Lemma 6.4 The fraction of binary, rate $R = b/c$, infinite-memory, time-invariant convolutional codes with polynomial encoding matrices used with list decoding of

list size L, having list weight w_{list}, and satisfying the inequality

$$w_{\text{list}} > \frac{\log L}{-\log(2^{1-R}-1)} + \frac{\log((2^R-1)(1-f))}{-\log(2^{1-R}-1)} - 1 \tag{6.22}$$

exceeds f, where $0 \le f < 1$.

Proof: Suppose that $\mathcal{C}$ belongs to the ensemble of infinite-memory, time-invariant convolutional codes. Let $\mathcal{C}_{[0,t)}$ denote the truncation at depth t of the code $\mathcal{C}$. One of the paths in the code tree representing $\mathcal{C}_{[0,t)}$ corresponds to the allzero path. In this code tree, $2^b - 1$ subpaths exist differing from the allzero path at depth $t-1$, and, in general, there exist $(2^b-1)2^{\ell b}$ subpaths differing from the allzero path at depth $t-\ell-1$, $\ell = 0, 1, \ldots, t-1$. Since all code symbols on the subpaths differing from the allzero path are independent and equiprobable binary digits (Lemma 3.13), it follows that the probability that a subpath differing from the allzero path at depth $t-\ell-1$ has weight i is equal to

$$\binom{(\ell+1)c}{i} \left(\frac{1}{2}\right)^{(\ell+1)c}$$

The probability that the weight of this subpath is less than w is equal to

$$\sum_{i=0}^{w-1} \binom{(\ell+1)c}{i} \left(\frac{1}{2}\right)^{(\ell+1)c}$$

Let $\boldsymbol{v}_{k[0,t)}$ be a code sequence arising from a nonzero information sequence, and let us introduce the indicator function

$$\varphi_w(\boldsymbol{v}_{k[0,t)}) = \begin{cases} 1 & \text{if } w_{\text{H}}(\boldsymbol{v}_{k[0,t)}) < w \\ 0 & \text{else} \end{cases} \tag{6.23}$$

Let $N_w(t)$ denote the number of subpaths in the truncated code tree for $\mathcal{C}_{[0,t)}$ having weight less than w. Clearly,

$$N_w(t) = \sum_k \varphi_w(\boldsymbol{v}_{k[0,t)}) \tag{6.24}$$

Hence, we have

$$\begin{aligned} E[N_w(t)] &= \sum_k E[\varphi_w(\boldsymbol{v}_{k[0,t)})] \\ &= \sum_k P(w_{\text{H}}(\boldsymbol{v}_{k[0,t)}) < w) \\ &= 1 + \sum_{\ell=0}^{t-1} (2^b-1)2^{\ell b} \sum_{i=0}^{w-1} \binom{(\ell+1)c}{i} \left(\frac{1}{2}\right)^{(\ell+1)c} \end{aligned} \tag{6.25}$$

By extending the sum over ℓ in (6.25) to all nonnegative integers, we obtain an upper bound that is independent of t. Omitting the argument t in the random variable $N_w(t)$ and upper-bounding $(2^b - 1)2^{\ell b}$ by $2^{(\ell+1)b}$, we obtain

$$E[N_w] < 1 + \sum_{\ell=0}^{\infty} \sum_{i=0}^{w-1} 2^{(\ell+1)b} \binom{(\ell+1)c}{i} 2^{-(\ell+1)c} \tag{6.26}$$

We use the substitution

$$k = (\ell + 1)c \tag{6.27}$$

and upper-bound the right side of (6.26) by summing over $k = 0, 1, 2, \ldots$,

$$E[N_w] < \sum_{k=0}^{\infty} \sum_{i=0}^{w-1} \binom{k}{i} 2^{k(R-1)} \tag{6.28}$$

Substituting

$$x = 2^{R-1} \tag{6.29}$$

into (6.28) and using the formulas

$$\sum_{k=0}^{\infty} \binom{k}{i} x^k = \frac{x^i}{(1-x)^{i+1}} \tag{6.30}$$

and

$$\sum_{i=0}^{w-1} \frac{x^i}{(1-x)^{i+1}} = \frac{1}{1-x} \frac{\left(\frac{x}{1-x}\right)^w - 1}{\left(\frac{x}{1-x}\right) - 1} \tag{6.31}$$

for $0 < x < 1$, we obtain

$$E[N_w] < \frac{1}{(2^R - 1)(2^{1-R} - 1)^w} \tag{6.32}$$

For a given list size L and value f, we choose a $\widehat{w}$ such that

$$\frac{1}{(2^R - 1)(2^{1-R} - 1)^{\widehat{w}}} \leq L(1 - f) < \frac{1}{(2^R - 1)(2^{1-R} - 1)^{\widehat{w}+1}} \tag{6.33}$$

Then, from (6.32) and the left inequality of (6.33) it follows that

$$E[N_{\widehat{w}}] < L(1 - f) \tag{6.34}$$

and, thus, for more than a fraction f of the codes in the ensemble, we must have $N_{\widehat{w}} \leq L$, which implies a list weight w_{list} that is not less than $\widehat{w}$. By rewriting the right inequality of (6.33) as

$$\widehat{w} > \frac{\log L}{-\log(2^{1-R} - 1)} + \frac{\log((2^R - 1)(1 - f))}{-\log(2^{1-R} - 1)} - 1 \tag{6.35}$$

we have completed the proof. ∎

Next we shall show that our lower bound on the list weight w_{list} for general convolutional codes also holds for convolutional codes encoded by *systematic, polynomial* encoding matrices.

Consider the ensemble $\mathcal{E}(b, c, \infty, 1)$ of binary, rate $R = b/c$, infinite-memory, time-invariant, convolutional codes with systematic, polynomial encoding matrices

$$\boldsymbol{G} = \begin{pmatrix} G_0 & G_1 & \dots & & \\ & G_0 & G_1 & \dots & \\ & & \ddots & \ddots & \end{pmatrix} \tag{6.36}$$

in which each $(b \times c)$ submatrix G_i, $i = 0, 1, \dots$, is systematic, that is,

$$G_i = \begin{pmatrix} 1 & 0 & \dots & 0 & \vdots & g_{1,b+1} & g_{g_1,b+2} & \cdots & g_{1,c} \\ 0 & 1 & \dots & 0 & \vdots & g_{2,b+1} & g_{g_2,b+2} & \cdots & g_{2,c} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \dots & 1 & \vdots & g_{b,b+1} & g_{g_b,b+2} & \cdots & g_{b,c} \end{pmatrix} \tag{6.37}$$

and each digit g_{ij}, $i = 1, 2, \dots, b$, $j = b+1, b+2, \dots, c$, is chosen independently with probability $1/2$ to be 0 or 1.

Lemma 6.5 The fraction of binary, rate $R = b/c$, infinite-memory, time-invariant convolutional codes with systematic, polynomial encoding matrices used with list decoding of list size L, having a list weight w_{list}, and satisfying inequality (6.22), exceeds f, where $0 \leq f < 1$.

Proof: Suppose that $\mathcal{C}$ belongs to the ensemble of infinite-memory, time-invariant convolutional codes with systematic, polynomial encoding matrices. Let $\mathcal{C}_{[0,t)}$ denote the truncation at depth t of the code $\mathcal{C}$. In the code tree there exist $\binom{b}{i}$ subpaths differing from the allzero path at depth $t-1$ with a weight of information symbols equal to i, $i = 1, 2, \dots, b$. Correspondingly, there exist no more than $\binom{(\ell+1)b}{i}$ subpaths differing from the allzero path at depth $t - \ell - 1$, with a weight of its information symbols equal to i, $i = 1, 2, \dots, b(\ell+1)$. The probability that the weight of the parity check symbols of the subpath differing from the allzero path at depth $t - \ell - 1$ has weight $j - i$, $j \geq i$, is equal to

$$\binom{(\ell+1)(c-b)}{j-i} \left(\frac{1}{2}\right)^{(\ell+1)(c-b)}$$

Thus (cf. (6.26)), the average of the number of subpaths of $\mathcal{C}_{[0,t)}$ having weights less than w is upper-bounded by the inequality

$$E[N_w] < 1 + \sum_{\ell=0}^{\infty} \sum_{j=0}^{w-1} \sum_{i=0}^{j} \binom{(\ell+1)b}{i} \binom{(\ell+1)(c-b)}{j-i} \left(\frac{1}{2}\right)^{(\ell+1)(c-b)}$$

$$= 1 + \sum_{\ell=0}^{\infty} \sum_{j=0}^{w-1} \binom{(\ell+1)c}{j} 2^{-(\ell+1)(c-b)} \tag{6.38}$$

where the equality follows from the identity

$$\sum_{i=0}^{j} \binom{n}{j-i} \binom{k}{i} = \binom{n+k}{j} \tag{6.39}$$

Since (6.38) is analogous to (6.26), the remaining part of the proof is similar to the proof of Lemma 6.4. ∎

By letting $f = 0$ and combining Lemmas 6.4 and 6.5, we obtain the following:

Theorem 6.6 There exists a binary, rate $R = b/c$, infinite-memory, time-invariant convolutional code with nonsystematic and systematic, polynomial generator matrices used with list decoding of list size L, having a list weight w_{list}, and satisfying the inequality

$$w_{\text{list}} > \frac{\log L}{-\log(2^{1-R} - 1)} + O(1) \tag{6.40}$$

where

$$O(1) = \frac{\log((2^R - 1)(2^{1-R} - 1))}{-\log(2^{1-R} - 1)} \tag{6.41}$$

Since a list decoder has L states and a Viterbi decoder for a convolutional code of overall constraint length ν has 2^ν states, our lower bound on w_{list} (6.40) is similar to Costello's lower bound on d_{free} (3.164).

It follows from Theorem 6.6 that convolutional codes encoded by both nonsystematic generator matrices and systematic, polynomial encoding matrices have principal determiners of the correct path loss probability that are lower-bounded by the same bound. For the free distance, which is the principal determiner of the error probability with Viterbi decoding, Costello's lower bounds on the free distance differ by the factor $1 - R$, depending on whether the convolutional code is encoded by a nonsystematic generator matrix or by a systematic, polynomial encoding matrix (cf. Theorem 3.28 and Problem 3.17). This different behavior reflects a fundamental and important difference between list and maximum-likelihood decoding.

Next we will derive an upper bound on the list weight that resembles our lower bound. In the derivation we will use a Hamming-type upper bound on the list weight for convolutional codes, but first we need the corresponding bound for block codes (cf. the Hamming bound for binary block codes given in Problem 1.17).

The list weight w_{list} for a block code $\mathcal{B}$ of block length N is obtained if we let $t = 0$ and use $c = N$ code symbols on each branch in the definition of the list weight w_{list} for a convolutional code $\mathcal{C}$ (6.18).

Lemma 6.7 The list weight w_{list} for a binary block code $\mathcal{B}$ of block length N and rate R when used with list decoding of list size L satisfies the inequality

$$\sum_{i=0}^{\lfloor \frac{1}{2} w_{\text{list}} \rfloor} \binom{N}{i} \leq L 2^{N(1-R)} \tag{6.42}$$

Proof: Assume that the inequality does not hold. Then, counting all points in the spheres of radius $\lfloor \frac{1}{2} w_{\text{list}} \rfloor$ around the 2^{NR} codewords we will fill the whole space (2^N points) more than L-fold. Hence, at least one of these points in the spheres must have been counted at least $L + 1$ times; that is, it must belong to at least $L + 1$ spheres. The codewords at the centers of these at least $L + 1$ spheres with one point in common are at most $2\lfloor \frac{1}{2} w_{\text{list}} \rfloor$ apart. Let any of these codewords be centered in a sphere of radius $2\lfloor \frac{1}{2} w_{\text{list}} \rfloor$. That sphere contains at least $L + 1$ codewords, which contradicts the definition of w_{list}. Thus, the inequality (6.42) must hold. ∎

Corollary 6.8 The list weight w_{list} for a binary block code $\mathcal{B}$ of block length N and rate R when used with list decoding of list size L satisfies the inequality

$$\left(h\left(\frac{1}{N} \left\lfloor \frac{1}{2} w_{\text{list}} \right\rfloor \right) - 1 + R \right) N < \frac{1}{2} \log N + \log L + \frac{1}{2} \tag{6.43}$$

where $h(\ )$ is the binary entropy function (1.22).

Proof: We have the following series of inequalities:

$$\begin{aligned} \sum_{i=0}^{\lfloor \frac{1}{2} w_{\text{list}} \rfloor} \binom{N}{i} &> \binom{N}{\lfloor \frac{1}{2} w_{\text{list}} \rfloor} > \sqrt{\frac{N}{8\lfloor \frac{1}{2} w_{\text{list}} \rfloor (N - \lfloor \frac{1}{2} w_{\text{list}} \rfloor)}} 2^{Nh\left(\frac{1}{N}\lfloor \frac{1}{2} w_{\text{list}} \rfloor\right)} \\ &\geq \frac{1}{\sqrt{2N}} 2^{Nh\left(\frac{1}{N}\lfloor \frac{1}{2} w_{\text{list}} \rfloor\right)} \end{aligned} \tag{6.44}$$

where the first inequality is obvious and the second inequality follows from Lemma 5.1. By combining Lemma 6.7 and (6.44), we obtain

$$\frac{1}{\sqrt{2N}} 2^{Nh\left(\frac{1}{N}\lfloor \frac{1}{2} w_{\text{list}} \rfloor\right)} < L 2^{N(1-R)} \tag{6.45}$$

Taking the logarithm completes the proof. ∎

For convolutional codes we have the following counterpart to Lemma 6.7:

Lemma 6.9 The list weight w_{list} for a convolutional code of rate $R = b/c$ when used with list decoding of list size L satisfies the inequality

$$\sum_{i=0}^{\lfloor \frac{1}{2} w_{\text{list}} \rfloor} \binom{(t+1)c}{i} \leq L 2^{(t+1)c(1-R)} \tag{6.46}$$

for $t = 0, 1, \ldots$.

Proof: Follows immediately from Lemma 6.7 applied to $\mathcal{C}_{[0,t)}$ for each $t = 0, 1, \ldots$. ∎

Theorem 6.10 The list weight w_{list} for a binary, rate $R = b/c$ convolutional code when used with list decoding of list size L satisfies the inequality

$$w_{\text{list}} < \frac{2 \log L}{-\log(2^{1-R} - 1)} + o(\log L) \tag{6.47}$$

Proof: For convolutional codes we obtain from (6.46) the following counterpart to (6.43):

$$\begin{aligned} &\left(h\left(\frac{1}{(t+1)c} \left\lfloor \frac{1}{2} w_{\text{list}} \right\rfloor \right) - 1 + R \right) (t+1)c \\ &\quad < \frac{1}{2} \log((t+1)c) + \log L + \frac{1}{2} \end{aligned} \tag{6.48}$$

for $t = 0, 1, \ldots$. Let

$$t_0 + 1 = \left\lfloor \frac{1}{c(1 - 2^{R-1})} \left\lfloor \frac{1}{2} w_{\text{list}} \right\rfloor \right\rfloor \tag{6.49}$$

Then, for any given rate $R > 0$ there exists a sufficiently large $\lfloor \frac{1}{2} w_{\text{list}} \rfloor$ such that the inequalities

$$\frac{2}{c} \left\lfloor \frac{1}{2} w_{\text{list}} \right\rfloor < t_0 + 1 \leq \frac{1}{c(1 - 2^{R-1})} \left\lfloor \frac{1}{2} w_{\text{list}} \right\rfloor \tag{6.50}$$

hold.

Let $t = t_0$ in (6.48). From the left inequality in (6.50) it follows that the argument for the binary entropy function in (6.48) is less than $1/2$. From the right inequality in (6.50) it follows that this argument is at least $(1 - 2^{R-1})$. Since the binary entropy function is increasing in the interval $[0, \frac{1}{2}]$, we obtain

$$(h(1 - 2^{R-1}) - 1 + R)(t_0 + 1)c < \frac{1}{2} \log((t_0 + 1)c) + \log L + \frac{1}{2} \tag{6.51}$$

By simple manipulations, we can show that

$$h(1 - 2^{R-1}) - 1 + R = -(1 - 2^{R-1}) \log(2^{1-R} - 1) \tag{6.52}$$

Combining (6.51) and (6.52) gives the important inequality

$$-(t_0 + 1)c(1 - 2^{R-1}) \log(2^{1-R} - 1) < \frac{1}{2} \log((t_0 + 1)c) + \log L + \frac{1}{2} \tag{6.53}$$

By using the well-known inequality

$$\log x \leq x - 1, \quad x > 0 \tag{6.54}$$

it follows from (6.53) that

$$(t_0+1)Ac - \sqrt{(t_0+1)c} - \log L < 0 \tag{6.55}$$

where

$$A = -(1-2^{R-1})\log(2^{1-R}-1) \tag{6.56}$$

By combining (6.50) and (6.55) we obtain

$$\begin{aligned} w_{\text{list}} &< \frac{2\log L}{-\log(2^{1-R}-1)} + (1-2^{R-1})\frac{\sqrt{1+4A\log L}}{A^2} + \frac{1-2^{R-1}}{A^2} \\ &\quad + (1-2^{R-1})c + 2 = \frac{2\log L}{-\log(2^{1-R}-1)} + O\left(\sqrt{\log L}\right) \end{aligned} \tag{6.57}$$

Thus, the proof is complete. ■

It is interesting to notice that the main term in the upper bound for w_{list} is exactly twice the main term in the corresponding lower bound.

Finally, from Theorems 6.6 and 6.10 we obtain the following:

Theorem 6.11 Given any received sequence with at most e errors. Then there exists a rate $R = b/c$ convolutional code such that the correct path is not lost when it is used with list decoding of list size L satisfying (asymptotically) the inequalities

$$\left(\frac{1}{2^{1-R}-1}\right)^{\frac{1}{2}e} \lesssim L \lesssim \left(\frac{1}{2^{1-R}-1}\right)^{e} \tag{6.58}$$

In particular, we have the next corollary.

Corollary 6.12 Given any received sequence with at most e errors. Then there exists a rate $R = 1/2$ convolutional code such that the correct path is not lost when it is used with list decoding of list size L satisfying (asymptotically) the inequalities

$$\left(1+\sqrt{2}\right)^{\frac{1}{2}e} \lesssim L \lesssim \left(1+\sqrt{2}\right)^{e} \tag{6.59}$$

As expected, the required list size grows exponentially with the number of errors to be corrected.

6.4 UPPER BOUNDS ON THE PROBABILITY OF CORRECT PATH LOSS

The correct path loss on the tth step of a list decoding algorithm is a random event $\mathcal{E}_t^{\text{cpl}}$ which consists of deleting at the tth step the correct codeword from the list of the L most likely codewords. In this section, we derive both expurgation and sphere-packing upper bounds on the probability of this correct path loss for the ensemble $\mathcal{E}(b, c, \infty, 1)$ of infinite-memory, time-invariant convolutional codes.

Our expurgation bound is valid for transmission rates R less than the computational cutoff rate R_0:

Lemma 6.13 (Expurgation bound for list decoding) For a list decoder of list size L and the BSC with crossover probability ϵ there exist infinite-memory, time-invariant, binary convolutional codes of rate $R = b/c$ with systematic and nonsystematic, polynomial generator matrices such that the probability of correct path loss is upper-bounded by the inequality

$$P(\mathcal{E}_t^{\text{cpl}}) \leq L^{-s} O(1) \tag{6.60}$$

for rates $0 \leq R < R_0$, where s, $1 \leq s < \infty$, satisfies

$$R = G_{\text{exp}}(s)/s \tag{6.61}$$

$G_{\text{exp}}(s)$ is the expurgation function (5.75), and $O(1)$ is independent of L.

Remark: When the rate $R \to R_0$ the exponent of the L-dependent factor of the upper bound in Lemma 6.13 approaches -1, while the second factor approaches $+\infty$.

In the proof we need the following:

Lemma 6.14 (Markov inequality) If a nonnegative random variable X has an average $E[X]$, then the probability that the outcome exceeds any given positive number a satisfies

$$P(X \geq a) \leq \frac{E[X]}{a} \tag{6.62}$$

Proof: The lemma follows immediately from

$$\begin{aligned} E[X] &= \sum_x x P(X = x) \geq \sum_{x \geq a} x P(X = x) \\ &\geq a \sum_{x \geq a} P(X = x) = a P(X \geq a) \end{aligned} \tag{6.63}$$

■

Proof (Lemma 6.13): Consider the ensembles $\mathcal{E}(b, c, \infty, 1)$ of binary, rate $R = b/c$, infinite-memory, time-invariant convolutional codes with systematic or nonsystematic, polynomial encoding matrices, respectively. Let $w_{\lfloor L/2 \rfloor\text{-list}}$ denote the list weight when the list size is $\lfloor L/2 \rfloor$. Then we let the $\lfloor L/2 \rfloor$ codewords of weight $w_{\lfloor L/2 \rfloor\text{-list}}$ or less be on the list of size L. The remaining $\lceil L/2 \rceil$ places are filled with codewords having weights more than $w_{\lfloor L/2 \rfloor\text{-list}}$. This is done as follows. Expurgate from the code ensemble all codes for which $w_{\lfloor L/2 \rfloor\text{-list}}$ does not satisfy the inequality

$$w_{\lfloor L/2 \rfloor\text{-list}} > \left\lceil \frac{\log \lfloor L/2 \rfloor}{-\log(2^{1-R} - 1)} + \frac{\log\left((2^R - 1)(1 - f)\right)}{-\log(2^{1-R} - 1)} - 1 \right\rceil \stackrel{\text{def}}{=} w_0 \tag{6.64}$$

and consider from now on only the subensembles for which the inequality (6.64) is satisfied.

According to Lemmas 6.4 and 6.5, each of these subensembles contains more than a fraction f of the codes of the ensemble. Assuming list decoding of list size L of the codes of each subensemble, a necessary condition to lose the correct path at the tth step of the decoding is that the number of subpaths in the tree at the depth t, having weight *more* than $w_{\lfloor L/2 \rfloor\text{-list}}$ and surpassing the correct path, exceed $\lceil L/2 \rceil$ since then the total number of paths exceeds L. The probability that a particular subpath of weight j would surpass the correct (allzero) path is upper-bounded by the Bhattacharyya bound (4.20), viz., by $\left(2\sqrt{\epsilon(1-\epsilon)}\right)^j$. The fraction of subpaths diverging from the allzero path at depth $t-\ell-1$ having weight j is upper-bounded by

$$\binom{(\ell+1)c}{j} 2^{-(\ell+1)c}$$

The average of the number of subpaths having weight not less than w_0 and surpassing the correct path evaluated over the ensemble is upper-bounded by

$$\sum_{\ell=0}^{\infty} \sum_{j=w_0}^{\infty} 2^{b(\ell+1)} \binom{(\ell+1)c}{j} 2^{-(\ell+1)c} \left(2\sqrt{\epsilon(1-\epsilon)}\right)^j$$

If we evaluate the average over the subensemble containing only the fraction f of the codes, then the upper bound is weakened by a factor of $1/f$. Hence, for $f > 0$ in the subensemble, the mathematical expectation of the number N of subpaths, which surpass the correct path, is upper-bounded by

$$\begin{aligned} E[N] &< \frac{1}{f} \sum_{\ell=0}^{\infty} \sum_{j=w_0}^{\infty} \binom{(\ell+1)c}{j} 2^{-(\ell+1)(c-b)} \left(2\sqrt{\epsilon(1-\epsilon)}\right)^j \\ &< \frac{1}{f} \sum_{k=0}^{\infty} \sum_{j=w_0}^{\infty} \binom{k}{j} 2^{k(R-1)} \left(2\sqrt{\epsilon(1-\epsilon)}\right)^j \\ &= \frac{1}{(1-2^{R-1})f} \sum_{j=w_0}^{\infty} \left(\frac{2\sqrt{\epsilon(1-\epsilon)}}{(2^{1-R}-1)}\right)^j \end{aligned} \tag{6.65}$$

The sum on the right-hand side of (6.65) converges if

$$\frac{2\sqrt{\epsilon(1-\epsilon)}}{2^{1-R}-1} = \frac{2^{1-R_0}-1}{2^{1-R}-1} < 1 \tag{6.66}$$

that is, if $R < R_0$. From (6.66) we have for $f > 0$ that

$$\begin{aligned} E[N] &< \frac{1}{(1-2^{R-1})f}\left(\frac{2\sqrt{\epsilon(1-\epsilon)}}{2^{1-R}-1}\right)^{w_0}\frac{1}{1-\frac{2\sqrt{\epsilon(1-\epsilon)}}{2^{1-R}-1}} \\ &= \frac{2^{1-R}}{f(2^{1-R}-2^{1-R_0})}\left(\frac{2\sqrt{\epsilon(1-\epsilon)}}{2^{1-R}-1}\right)^{w_0} = \frac{1}{f(1-2^{R-R_0})}\left(\frac{2\sqrt{\epsilon(1-\epsilon)}}{2^{1-R}-1}\right)^{w_0} \\ &\le (\lfloor L/2\rfloor)^{-\frac{\log\left(2\sqrt{\epsilon(1-\epsilon)}\right)}{\log(2^{1-R}-1)}}\ (\lfloor L/2\rfloor)\ \frac{1}{f}\ \frac{1}{1-2^{R-R_0}} \\ &\times \left((2^R-1)(1-f)(2^{1-R}-1)\right)^{-\frac{\log\left(2\sqrt{\epsilon(1-\epsilon)}\right)}{\log(2^{1-R}-1)}+1} \end{aligned} \tag{6.67}$$

where the last inequality follows from (6.64).

Let s satisfy (6.61). Then combining (6.61) and (5.57) yields

$$s\left(1-\log\left(1+\left(2\sqrt{\epsilon(1-\epsilon)}\right)^{1/s}\right)\right) = sR \tag{6.68}$$

Assuming $s \neq 0$, we obtain

$$s = \frac{\log\left(2\sqrt{\epsilon(1-\epsilon)}\right)}{\log(2^{1-R}-1)} \tag{6.69}$$

Thus, inequality (6.67) can be rewritten as

$$E[N] < (\lfloor L/2\rfloor)^{1-s}\,\frac{1}{f}\frac{1}{1-2^{R-R_0}}\left((2^R-1)(1-f)(2^{1-R}-1)\right)^{1-s} \tag{6.70}$$

From Lemma 6.14 it follows that the probability of correct path loss at the tth step of list decoding for rates $R < R_0$ is upper-bounded by the inequalities

$$P(\mathcal{E}_t^{\text{cpl}}) < \frac{E[N]}{\lceil L/2\rceil} < L^{-s}O_f(1) \tag{6.71}$$

where

$$O_f(1) = \frac{1}{1-2^{R-R_0}}2^s\frac{1}{f}\left((2^R-1)(1-f)(2^{1-R}-1)\right)^{1-s} \tag{6.72}$$

If we choose $f = 1/2$ say, we get

$$\begin{aligned} O_{1/2}(1) &= \frac{1}{1-2^{R-R_0}}\left((2^R-1)(2^{1-R}-1)\right)^{1-s}(4\epsilon(1-\epsilon))^{\frac{1}{\log(2^{1-R}-1)}} \\ &\stackrel{\text{def}}{=} O(1) \end{aligned} \tag{6.73}$$

and the proof is complete. ∎

For the sphere-packing region, $R_0 \leq R < C$, we have the following upper bound on the probability of correct path loss:

Lemma 6.15 (Sphere-packing bound for list decoding) For a list decoder of list size L and the BSC with crossover probability ϵ, there exist infinite-memory, time-invariant, binary convolutional codes of rate $R = b/c$ with systematic and nonsystematic, polynomial generator matrices such that the probability of correct path loss is upper-bounded by the inequality

$$P(\mathcal{E}_t^{\text{cpl}}) < L^{-s} O(\log L) \tag{6.74}$$

for rates $R_0 \leq R < C$, where s, $0 < s \leq 1$, satisfies

$$R = G(s)/s \tag{6.75}$$

$G(s)$ is the Gallager function (5.98), and $\frac{O(\log L)}{\log L} \to$ constant when $L \to \infty$.

Proof: For the sphere-packing region we shall as before exploit the idea of separating the error event $\mathcal{E}_t^{\text{cpl}}$ into two disjoint events corresponding to "few" $\mathcal{F}$ and "many" $\mathcal{M}$ errors, respectively. Hence, we have (cf. (4.81))

$$E[P(\mathcal{E}_t^{\text{cpl}})] \leq E[P(\mathcal{E}_t^{\text{cpl}}, \mathcal{F})] + P(\mathcal{M}) \tag{6.76}$$

Without loss of generality, we assume that the allzero sequence is transmitted. Let $\boldsymbol{r}_{[0,t)} = \boldsymbol{r}_0 \boldsymbol{r}_1 \ldots \boldsymbol{r}_{t-1}$, where $\boldsymbol{r}_i = (r_{i1} r_{i2} \ldots r_{ic})$, denote the received sequence of length t c-tuples.

We introduce a *backward* random walk $0, S_0, S_1, \ldots, S_{t-1}$, where

$$S_\ell = \sum_{i=0}^{\ell} Z_i, \quad 0 \leq \ell < t \tag{6.77}$$

starting with 0 at a node at depth t, S_0 at depth $t-1$, and so on, until we have S_{t-1} at the root. The branch metric Z_i is given by

$$Z_i = \sum_{j=1}^{c} Y_{ij} \tag{6.78}$$

where

$$Y_{ij} = \begin{cases} \alpha & \text{if } r_{(t-1-i)j} = 0 \\ \beta & \text{otherwise} \end{cases} \tag{6.79}$$

and (cf. (5.211))

$$\begin{cases} \alpha &= \log \dfrac{(1-\epsilon)^{\frac{1}{1+s}}}{\frac{1}{2}(1-\epsilon)^{\frac{1}{1+s}} + \frac{1}{2}\epsilon^{\frac{1}{1+s}}} - R \\ \beta &= \log \dfrac{\epsilon^{\frac{1}{1+s}}}{\frac{1}{2}(1-\epsilon)^{\frac{1}{1+s}} + \frac{1}{2}\epsilon^{\frac{1}{1+s}}} - R \end{cases} \tag{6.80}$$

The parameter s will be chosen later. We say that those error patterns for which S_t hits or crosses (from above) a certain barrier u contain "many" errors.

Following the same steps as in the derivation given by (5.212) to (5.217) we obtain (see also Corollary B.6)

$$P(\mathcal{M}) = P(\min_{\ell}\{S_\ell\} \leq u) \leq 2^{su} \tag{6.81}$$

where $0 < s \leq s_0$ and s_0 is a positive root of the equation

$$G(s_0) = s_0 R \tag{6.82}$$

Next we upper-bound the probability that the correct path is lost at depth t and that we have an error pattern with "few" errors. Let N_t denote the number of paths at depth t which surpass the correct path. Then, since the correct path is lost at depth t if $N_t \geq L$, it follows from the Markov inequality (Lemma 6.14) that

$$P(\mathcal{E}_t^{\text{cpl}}, \mathcal{F}) \leq \frac{E[N_t \mid \mathcal{F}]P(\mathcal{F})}{L} \tag{6.83}$$

Consider an arbitrary subpath which diverges from the correct path at depth $t-\ell-1$, $0 \leq \ell < t$. A necessary condition that this path surpasses the correct one is that the Hamming distance between this path and the received sequence is not more than the number of channel errors i in the last $\ell+1$ subblocks. In case of the event $\mathcal{F}$, that is, that we have "few" errors, the random walk must satisfy $S_\ell > u$. Thus, assuming nonsystematic generator matrices we obtain

$$E[N_t \mid \mathcal{F}]P(\mathcal{F}) \leq \sum_{\ell=0}^{t-1} \sum_{i|S_\ell > u} \sum_{w=0}^{i} 2^{b(\ell+1)} \binom{(\ell+1)c}{i} \epsilon^i (1-\epsilon)^{(\ell+1)c-i} \times \binom{(\ell+1)c}{w} 2^{-(\ell+1)c} \tag{6.84}$$

For $S_\ell > u$ we have

$$2^{\lambda_2(S_\ell - u)} > 1 \tag{6.85}$$

where $\lambda_2 > 0$. Combining (6.84) and (6.85) yields

$$\begin{aligned} E[N_t \mid \mathcal{F}]P(\mathcal{F}) &\le \sum_{\ell=0}^{t-1} \sum_{i|S_\ell > u} \sum_{w=0}^{i} 2^{b(\ell+1)} \binom{(\ell+1)c}{i} \epsilon^i (1-\epsilon)^{(\ell+1)c-i} \\ &\quad \times \binom{(\ell+1)c}{w} 2^{-(\ell+1)c} 2^{\lambda_2(S_t-u)} \\ &\le 2^{-\lambda_2 u} \sum_{\ell=0}^{t-1} \sum_{i=0}^{(\ell+1)c} \sum_{w=0}^{i} 2^{b(\ell+1)} \binom{(\ell+1)c}{i} \epsilon^i (1-\epsilon)^{(\ell+1)c-i} \\ &\quad \times \binom{(\ell+1)c}{w} 2^{-(\ell+1)c} 2^{\lambda_2(\alpha((\ell+1)c-i)+\beta i)} \\ &\le 2^{-\lambda_2 u} \sum_{\ell=0}^{t-1} \sum_{i=0}^{(\ell+1)c} \sum_{w=0}^{(\ell+1)c} 2^{b(\ell+1)} \binom{(\ell+1)c}{i} \epsilon^i (1-\epsilon)^{(\ell+1)c-i} \\ &\quad \times \binom{(\ell+1)c}{w} 2^{-(\ell+1)c} 2^{\lambda_1(i-w)} 2^{\lambda_2(\alpha((\ell+1)c-i)+\beta i)} \end{aligned} \tag{6.86}$$

where $\lambda_1 > 0$ and $\lambda_2 > 0$. From (6.86) it follows that

$$\begin{aligned} E[N_t \mid \mathcal{F}]P(\mathcal{F}) < 2^{-\lambda_2 u} \sum_{\ell=0}^{\infty} 2^{b(\ell+1)} &\left(\frac{1 + 2^{-\lambda_1}}{2} \right)^{(\ell+1)c} \\ &\times \left(\epsilon 2^{\lambda_1 + \lambda_2 \beta} + (1-\epsilon) 2^{\lambda_2 \alpha} \right)^{(\ell+1)c} \end{aligned} \tag{6.87}$$

Let

$$\lambda_1 = \frac{1}{1+s} \log \frac{1-\epsilon}{\epsilon} \tag{6.88}$$

$$\lambda_2 = 1 - s \tag{6.89}$$

and

$$u = -\log L \tag{6.90}$$

Then we obtain

$$\begin{aligned}
E[N_t \mid \mathcal{F}]P(\mathcal{F}) &< 2^{(1-s)\log L} \sum_{\ell=0}^{\infty} 2^{b(\ell+1)} \left(\frac{\epsilon^{\frac{1}{1+s}} + (1-\epsilon)^{\frac{1}{1+s}}}{2(1-\epsilon)^{\frac{1}{1+s}}} \right)^{(\ell+1)c} \\
&\quad \times \left(\frac{\left(\frac{1-\epsilon}{\epsilon}\right)^{\frac{1}{1+s}} \epsilon^{1+\frac{1-s}{1+s}} 2^{-(1-s)R} + (1-\epsilon)^{1+\frac{1-s}{1+s}} 2^{-(1-s)R}}{\left(\frac{1}{2}\epsilon^{\frac{1}{1+s}} + \frac{1}{2}(1-\epsilon)^{\frac{1}{1+s}}\right)^{1-s}} \right)^{(\ell+1)c} \\
&= L^{1-s} \sum_{\ell=0}^{\infty} 2^{b(\ell+1)} \left(\epsilon^{\frac{1}{1+s}} + (1-\epsilon)^{\frac{1}{1+s}} \right)^{(1+s)(\ell+1)c} 2^{-s(\ell+1)c} 2^{(s-1)R(\ell+1)c} \\
&= L^{1-s} \sum_{\ell=0}^{\infty} 2^{R(\ell+1)} 2^{-G(s)(\ell+1)c} 2^{(s-1)R(\ell+1)c} \\
&= L^{1-s} \sum_{\ell=0}^{\infty} 2^{-(G(s)-sR)(\ell+1)c} \\
&= L^{1-s} \sum_{\ell=0}^{\infty} 2^{-(G(s)-sG(s_0)/s_0)(\ell+1)c} \\
&= L^{1-s} \frac{1}{2^{(G(s)-sG(s_0)/s_0)c} - 1} \qquad (6.91)
\end{aligned}$$

where $G(s)$ is the Gallager function (5.98) and the last sum converges for all s such that $0 < s < s_0$ and s_0 satisfies (6.75). Combining (6.76), (6.83), (6.90), and (6.91) yields

$$\begin{aligned}
E[P(\mathcal{E}_t^{\text{cpl}})] &\le L^{-s} \frac{1}{2^{(G(s)-sG(s_0)/s_0)c} - 1} + 2^{-s\log L} \\
&= L^{-s} \frac{1}{1 - 2^{-(G(s)-sG(s_0)/s_0)c}} \qquad (6.92)
\end{aligned}$$

Let us choose

$$s = s_0 - \frac{1}{\log L} \qquad (6.93)$$

Then we obtain

$$E[P(\mathcal{E}_t^{\text{cpl}})] \le L^{-s_0} O(\log L) \qquad (6.94)$$

where $\frac{O(\log L)}{\log L} \to$ constant when $L \to \infty$, and the proof for nonsystematic generator matrices is complete.

For systematic, polynomial generator matrices, we replace (6.84) by (cf. (6.38))

$$\begin{aligned}
E[N_t \mid \mathcal{F}]P(\mathcal{F}) \le \sum_{\ell=0}^{t-1} \sum_{i|S_\ell > u} \sum_{w=0}^{i} \sum_{k=0}^{w} \binom{(\ell+1)c}{i} \epsilon^i (1-\epsilon)^{(\ell+1)c-i} \\
\times \binom{(\ell+1)b}{k} \binom{(\ell+1)(c-b)}{w-k} 2^{-(\ell+1)(c-b)} \qquad (6.95)
\end{aligned}$$

Using the identity (6.39), we can rewrite (6.95) as (6.84) and then repeat the steps (6.86)–(6.97). ■

Since

$$\frac{O(\log L)}{\log L} = 2^{O(\log\log L)} = L^{\frac{O(\log\log L)}{\log L}} = L^{o(1)} \tag{6.96}$$

where $o(1) \to 0$ when $L \to \infty$, it follows that the expurgated bound (6.60) can be written

$$P(\mathcal{E}_t^{\text{cpl}}) \leq L^{-s+o(1)} \tag{6.97}$$

for rates $0 \leq R < R_0$, where s satisfies (6.61).

We can now summarize Lemmas 6.13 and 6.15 in the following:

Theorem 6.16 For a list decoder of list size L and the BSC with crossover probability ϵ, there exist infinite-memory, time-invariant, binary convolutional codes of rate $R = b/c$ with systematic and nonsystematic, polynomial generator matrices such that the probability of correct path loss is upper-bounded by the inequality

$$P(\mathcal{E}_t^{\text{cpl}}) \leq L^{-s+o(1)} \tag{6.98}$$

where s satisfies

$$\begin{cases} R = G_{\text{exp}}(s)/s, & 0 \leq R < R_0 \\ R = G(s)/s, & R_0 \leq R < C \end{cases} \tag{6.99}$$

$G_{\text{exp}}(s)$ is the expurgation function (5.75) and $G(s)$ is the Gallager function (5.98), and $o(1) \to 0$ when $L \to \infty$.

The upper bound in Theorem 6.16 can be rewritten as

$$P(\mathcal{E}_t^{\text{cpl}}) \leq L^{-s+o(1)} = 2^{-(s+o(1))\log L} = 2^{-(E_{\mathcal{C}}(R)+o(1))(\log L)/R} \tag{6.100}$$

where $E_{\mathcal{C}}(R)$ is the convolutional coding exponent (5.105). Hence, if we choose the list size L equal to the number of encoder states, assuming that $\nu = bm$, that is,

$$L = 2^{bm} \tag{6.101}$$

then our upper bound on the probability of correct path loss (6.100) coincides with the upper bound on the burst error probability (5.104).

For the ensemble of general, nonlinear trellis codes it can be shown that for list decoding the exponent of (6.98) in the sphere-packing region, viz., $s = G(s)/R$, is somewhat surprisingly correct for all rates, $0 \leq R < C$ [ZiK80]! We conjecture that list decoding of convolutional codes encoded by systematic, polynomial generator matrices is superior to Viterbi decoding of convolutional codes encoded by nonsystematic generator matrices. This conjecture is in fact given strong support by the experiments reported in Section 6.2.

6.5 LOWER BOUND ON THE PROBABILITY OF CORRECT PATH LOSS

As a counterpart to the upper bound on the probability of correct path loss for list decoding used when we communicate over the BSC, we will derive a lower bound. Following the path of reasoning in Section 6.3 we begin by proving the corresponding lower bound for block codes.

Theorem 6.17 Suppose that a block code $\mathcal{B}$ of rate r and block length N with list decoding of list size L is used to communicate over the BSC with crossover probability ϵ. If

$$\tilde{r} = r - \frac{\log L}{N} \tag{6.102}$$

and

$$N = \frac{E_{\mathcal{C}}^{\text{sph}}(r) \log L}{E_{\mathcal{B}}^{\text{sph}}(\tilde{r}) r} \tag{6.103}$$

where $E_{\mathcal{C}}^{\text{sph}}(r)$ and $E_{\mathcal{B}}^{\text{sph}}(\tilde{r})$ are the convolutional sphere-packing exponent (5.148) and block sphere-packing exponent (5.146), respectively. Then the probability that the correct codeword will *not* be on the list of L codewords chosen by the list decoder is lower-bounded by

$$P_L(\mathcal{E}^{\text{cpl}}) > L^{-s+o(1)} \tag{6.104}$$

where s, $0 < s < \infty$, satisfies

$$r = G(s)/s \tag{6.105}$$

$G(s)$ is the Gallager function (5.98), and $o(1) \to 0$ when $L \to \infty$.

Proof: Let $P(\overline{\mathcal{E}^{\text{cpl}}} \mid i)$ denote the probability that the transmitted codeword $\boldsymbol{v}^{(i)}$, $i = 0, 1, \ldots, 2^{rN} - 1$, will appear on the list of the L most probable codewords that is produced by the list decoder. For each received sequence $\boldsymbol{r}$, a list of L codewords is produced. Let $\mathcal{D}_i$ denote the set of received sequences $\boldsymbol{r}$ such that the codeword $\boldsymbol{v}^{(i)}$ is on the list corresponding to $\boldsymbol{r}$. Then we have

$$P(\overline{\mathcal{E}^{\text{cpl}}} \mid i) = \sum_{\boldsymbol{r} \in \mathcal{D}_i} P(\boldsymbol{r} \mid \boldsymbol{v}^{(i)}) \tag{6.106}$$

Clearly,

$$\min_i \{ \mid \mathcal{D}_i \mid \} \leq L \frac{2^N}{2^{rN}} = 2^{(1-\tilde{r})N} \tag{6.107}$$

where $\tilde{r}$ is given by (6.102).

Let

$$P(\mathcal{E}^{\text{cpl}}) \stackrel{\text{def}}{=} \max_i \{ P(\mathcal{E}^{\text{cpl}} \mid i) \} \tag{6.108}$$

Then, by repeating the steps (5.123)–(5.130) we obtain the parametric lower bound

$$\begin{array}{rcl} 2^{(1-\tilde{r})N} & > & \binom{N}{k_0 - 1} \\ P(\mathcal{E}^{\text{cpl}}) & > & \binom{N}{k_0 + 1} \epsilon^{k_0+1} (1-\epsilon)^{N-k_0-1} \end{array} \tag{6.109}$$

where k_0 is the largest integer such that (cf. (5.124))

$$\sum_{k=0}^{k_0-1} \binom{N}{k} \leq \min\{|\ \mathcal{D}_i\ |\} \tag{6.110}$$

Following the steps in the proof of Theorem 5.11 yields

$$P(\mathcal{E}^{\text{cpl}}) > 2^{-\left(E_{\mathcal{B}}^{\text{sph}}(\tilde{r})+o(1)\right)N} \tag{6.111}$$

where $E_{\mathcal{B}}^{\text{sph}}(\tilde{r})$ is given by (5.146) and $o(1) \to 0$ when $N \to \infty$.

Inserting (6.103) into (6.111) yields

$$P(\mathcal{E}^{\text{cpl}}) > 2^{-\left(E_{\mathcal{C}}^{\text{sph}}(r)/r+o(1)\right)\log L} = L^{-s+o(1)} \tag{6.112}$$

where s, $0 < s < \infty$, satisfies (6.105).

We notice that $\tilde{r}$ as given by (6.102) is related to r by the equation

$$E_{\mathcal{C}}^{\text{sph}}(r) = \frac{r}{r-\tilde{r}} E_{\mathcal{B}}^{\text{sph}}(\tilde{r}) \tag{6.113}$$

which shows the existence (at least for large N) of $\tilde{r}$ and N satisfying (6.102) and (6.103), respectively. ∎

We will now prove the corresponding lower bound on the probability of correct path loss for convolutional codes:

Theorem 6.18 Suppose that a rate $R = b/c$ convolutional code with list decoding of list size L is used to communicate over the BSC with crossover probability ϵ. Then the probability of correct path loss is lower-bounded by the inequality

$$P(\mathcal{E}^{\text{cpl}}) > L^{-s+o(1)} \tag{6.114}$$

where s, $0 < s < \infty$, satisfies

$$R = G(s)/s \tag{6.115}$$

and $o(1) \to 0$ when $L \to \infty$.

Proof: The proof is similar to the proof of Lemma 5.12. The theorem states that for any $\varepsilon > 0$ there exists a list size L_ε such that for any $L > L_\varepsilon$ we have

$$P(\mathcal{E}^{\text{cpl}}) > L^{-(s+\varepsilon)} \tag{6.116}$$

Now suppose that inequality (6.116) does not hold. Then, as a consequence, a convolutional code and a certain $\varepsilon > 0$ exist such that for any large L_ε there exists a list decoder with list size $L > L_\varepsilon$ such that

$$P(\mathcal{E}^{\text{cpl}}) < L^{-(s+2\varepsilon)} \tag{6.117}$$

Construct a block code by truncating this convolutional code so that its rate is equal to the rate R of the convolutional code (no zero termination) and its block length N is given by

$$N = \frac{E_{\mathcal{C}}^{\text{sph}}(R) \log L}{E_{\mathcal{B}}^{\text{sph}}(\tilde{r}) R} \tag{6.118}$$

where

$$\tilde{r} = R - \frac{\log L}{N} \tag{6.119}$$

For this block code the probability that the correct codeword will not be on the list is upper-bounded by the probability of correct path loss for the convolutional code at depths $1, 2, \ldots, N/c$. Using the union bound we obtain from (6.117) and (6.108) that

$$P_L(\mathcal{E}^{\text{cpl}}) < \frac{N}{c} L^{-(s+2\varepsilon)} = \frac{E_{\mathcal{C}}^{\text{sph}}(R) \log L}{E_{\mathcal{B}}^{\text{sph}}(\tilde{r}) Rc} L^{-(s+2\varepsilon)} \tag{6.120}$$

Let us choose L_ε such that

$$\frac{E_{\mathcal{C}}^{\text{sph}}(R) \log L_\varepsilon}{E_{\mathcal{B}}^{\text{sph}}(\tilde{r}) Rc} < (L_\varepsilon)^{\varepsilon} \tag{6.121}$$

Then, for any $L > L_\varepsilon$ we have

$$P_L(\mathcal{E}^{\text{cpl}}) < L^{-(s+\varepsilon)} \tag{6.122}$$

in contradiction to Theorem 6.17. Hence, we conclude that inequality (6.116) must hold and the proof is complete. ∎

Let us rewrite the lower bound in Theorem 6.18 as

$$P(\mathcal{E}^{\text{cpl}}) > L^{-s+o(1)} = 2^{-(s+o(1)) \log L} = 2^{-\left(E_{\mathcal{C}}^{\text{sph}}(R)+o(1)\right)(\log L)/R} \tag{6.123}$$

where $E_{\mathcal{C}}^{\text{sph}}(R)$ is the convolutional sphere-packing exponent (5.148). Thus, if we choose

$$L = 2^{bm} \tag{6.124}$$

then our lower bound on the probability of correct path loss coincides with the lower bound on the burst error probability for high rates given in Lemma 5.12. For maximum-likelihood decoding we derived a tighter lower bound for low rates (Lemma 5.13). Such a bound does not exist for list decoding.

6.6 CORRECT PATH LOSS FOR TIME-INVARIANT CONVOLUTIONAL CODES

In this section, we use a path weight enumerator and derive upper bounds on the probability of correct path loss for time-invariant convolutional codes which are similar to Viterbi's bounds in Section 4.2. Consider the trellis for a rate $R = b/c$,

time-invariant convolutional code encoded by a generator matrix of memory m. Assuming a realization in controller canonical form, the signal flowchart introduced in Section 3.10 consists of $2^{bm}+1$ states, since the zero state is split into two—a source and a sink state. Let $\xi_j(W)$, $j = 1, 2, \ldots, 2^{bm}-1$, be dummy variables representing the generating functions of the weights of all paths leading from the left zero state to the $2^{bm}-1$ intermediate states, respectively. Assuming $W < 1$, order these dummy variables in decreasing order, that is,

$$\xi_1(W) \geq \xi_2(W) \geq \cdots \geq \xi_{2^{bm}-1}(W) \tag{6.125}$$

Notice that since $W < 1$, a large value of $\xi(W)$ corresponds to low weight on the paths that are represented by $\xi(W)$.

Let us introduce the *ℓ-list path weight enumerator*

$$T_\ell(W) \stackrel{\text{def}}{=} \sum_{j=\ell+1}^{2^{bm}-1} \xi_j(W) \tag{6.126}$$

As before $\mathcal{E}_t^{\text{cpl}}$ denotes the event that the correct path is deleted at the tth step from the list of the L most likely codewords. Then we have the following:

Theorem 6.19 For convolutional codes encoded by a generator matrix with ℓ-list path weight enumerator $T_\ell(W)$ and used to communicate over the BSC with crossover probability ϵ, the probability of correct path loss for a list decoder of list size L is upper-bounded by

$$P(\mathcal{E}_t^{\text{cpl}}) \leq \min_{0 \leq \ell < L} \left\{ \frac{T_\ell(W) \mid_{W=2\sqrt{\epsilon(1-\epsilon)}}}{L-\ell} \right\} \tag{6.127}$$

Proof: Since a convolutional code is linear, we can without loss of generality assume that the allzero codeword has been transmitted. The states at depth t are ordered according to increasing weights of the best paths leading to these states. If the allzero state (that is, the state corresponding to the transmitted path) is not among the L best states, a correct path loss has occurred. The probability that a certain path of weight w will be ranked by the list decoder higher than the allzero path is upper-bounded by the Bhattacharyya bound (4.20), viz., $(2\sqrt{\epsilon(1-\epsilon)})^w$. Then, it follows that the average of the number of states ranked higher than the allzero state is upper-bounded by

$$T_0(W) = \sum_{j=1}^{2^{bm}-1} \xi_j(W) \tag{6.128}$$

where

$$W = 2\sqrt{\epsilon(1-\epsilon)} \tag{6.129}$$

From (6.128) and Lemma 6.14 we obtain the upper bound

$$P(\mathcal{E}_t^{\text{cpl}}) = P(N \geq L) \leq \frac{T_0(W) \mid_{W=2\sqrt{\epsilon(1-\epsilon)}}}{L} \tag{6.130}$$

where N denotes the number of states ranked higher than the allzero state.

The bound (6.130) can be improved if we assume that the ℓ, $0 \leq \ell < L$, states represented by the ℓ largest values of the dummy variables, viz., ξ_j, $1 \leq j \leq \ell$, always remain on list. This assumption holds when both the crossover probability ϵ and the list size L are relatively small (that is, in the situations that are of practical interest). Assuming that these ℓ states will always stay on the list, we expurgate these states and consider only list size $L - \ell$. Then the average of the number of remaining states ranked higher than the allzero state is upper-bounded by

$$T_\ell(W) = \sum_{j=\ell+1}^{2^{bm}-1} \xi_j(W) \tag{6.131}$$

where W is given by (6.129).

Again, we apply Lemma 6.14 and obtain

$$P(\mathcal{E}_t^{\text{cpl}}) = P(n \geq L - \ell) \leq \frac{T_\ell(W)\,|_{W=2\sqrt{\epsilon(1-\epsilon)}}}{L - \ell} \tag{6.132}$$

Optimizing over ℓ completes the proof. ■

■ **EXAMPLE 6.4**

The signal flowchart for the rate $R = 1/2$, memory $m = 3$, systematic encoding matrix $G(D) = (1 \quad 1 + D + D^3)$, realized in controller canonical form, is given in Fig. 6.12. We obtain the following linear system of equations:

$$\begin{array}{rcl}
\xi_1(W) &=& \xi_2(W) + W\xi_3(W) \\
\xi_2(W) &=& W\xi_4(W) + \xi_5(W) \\
\xi_3(W) &=& W\xi_6(W) + \xi_7(W) \\
\xi_4(W) &=& W^2 + W\xi_1(W) \\
\xi_5(W) &=& W^2\xi_2(W) + W\xi_3(W) \\
\xi_6(W) &=& W\xi_4(W) + W^2\xi_5(W) \\
\xi_7(W) &=& W\xi_6(W) + W^2\xi_7(W)
\end{array} \tag{6.133}$$

Solving (6.133) gives

$$\begin{array}{rcl}
\xi_1(W) &=& W^3 \frac{1+3W^2-6W^4+4W^6-W^8}{1-3W^2-4W^4+7W^6-4W^8+W^{10}} \stackrel{\text{def}}{=} \xi_2^o(W) \\
\xi_2(W) &=& W^4 \frac{1-W^2+W^4}{1-3W^2-4W^4+7W^6-4W^8+W^{10}} \stackrel{\text{def}}{=} \xi_5^o(W) \\
\xi_3(W) &=& W^4 \frac{2-3W^2+3W^4-W^6}{1-3W^2-4W^4+7W^6-4W^8+W^{10}} \stackrel{\text{def}}{=} \xi_4^o(W) \\
\xi_4(W) &=& W^2 \frac{1-2W^2-W^4+W^6}{1-3W^2-4W^4+7W^6-4W^8+W^{10}} \stackrel{\text{def}}{=} \xi_1^o(W) \\
\xi_5(W) &=& W^5 \frac{3-2W^2}{1-3W^2-4W^4+7W^6-4W^8+W^{10}} \stackrel{\text{def}}{=} \xi_7^o(W) \\
\xi_6(W) &=& W^3 \frac{1-2W^2+2W^4-W^6}{1-3W^2+4W^4+7W^6-4W^8+W^{10}} \stackrel{\text{def}}{=} \xi_3^o(W) \\
\xi_7(W) &=& W^4 \frac{1-W^2+W^4}{1-3W^2-4W^4+7W^6-4W^8+W^{10}} \stackrel{\text{def}}{=} \xi_6^o(W)
\end{array} \tag{6.134}$$

where $\xi_i^o(W)$, $1 \leq i \leq 7$, denotes a reordering in decreasing order for small values of W.

For small values of W we obtain from Theorem 6.19 the following upper bounds on the probability of the correct path loss:

$$L = 1 : P(\mathcal{E}_t^{\text{cpl}}) \leq \sum_{j=1}^{7} \xi_j^o(W) \approx W^2 \tag{6.135}$$

$$L = 2 : P(\mathcal{E}_t^{\text{cpl}}) \leq \min \left\{ \frac{1}{2} \sum_{j=1}^{7} \xi_j^o(W), \sum_{j=2}^{7} \xi_j^o(W) \right\} \approx 2W^3 \tag{6.136}$$

$$L = 3 : P(\mathcal{E}_t^{\text{cpl}}) \leq \min \left\{ \frac{1}{3} \sum_{j=1}^{7} \xi_j^o(W), \frac{1}{2} \sum_{j=2}^{7} \xi_j^o(W), \sum_{j=3}^{7} \xi_j^o(W) \right\} \approx W^3 \tag{6.137}$$

$$L = 4 : P(\mathcal{E}_t^{\text{cpl}}) \leq \min \left\{ \frac{1}{4} \sum_{j=1}^{7} \xi_j^o(W), \frac{1}{3} \sum_{j=2}^{7} \xi_j^o(W), \frac{1}{2} \sum_{j=3}^{7} \xi_j^o(W), \sum_{j=4}^{7} \xi_j^o(W) \right\} \approx 3W^4 \tag{6.138}$$

$$L = 5 : P(\mathcal{E}_t^{\text{cpl}}) \leq \min \left\{ \frac{1}{5} \sum_{j=1}^{7} \xi_j^o(W), \frac{1}{4} \sum_{j=2}^{7} \xi_j^o(W), \frac{1}{3} \sum_{j=3}^{7} \xi_j^o(W), \frac{1}{2} \sum_{j=4}^{7} \xi_j^o(W), \sum_{j=5}^{7} \xi_j^o(W) \right\} \approx 2W^4 \tag{6.139}$$

A probability of correct path loss of the order W^3 requires $L = 2$, while a probability of correct path loss of the order W^4 requires $L = 4$.

For the Gaussian channel we have the corresponding bound:

Theorem 6.20 For convolutional codes of rate R encoded by generator matrices with ℓ-list path weight enumerator $T_\ell(W)$ and used to communicate over the AWGN channel with signal-to-noise ratio E_{b}/N_0 the probability of correct path loss for a list decoder is upper-bounded by

$$P(\mathcal{E}_t^{\text{cpl}}) \leq \min_{0 \leq \ell < L} \left\{ \frac{T_\ell(W) \mid_{W = e^{-RE_{\text{b}}/N_0}}}{L - \ell} \right\} \tag{6.140}$$

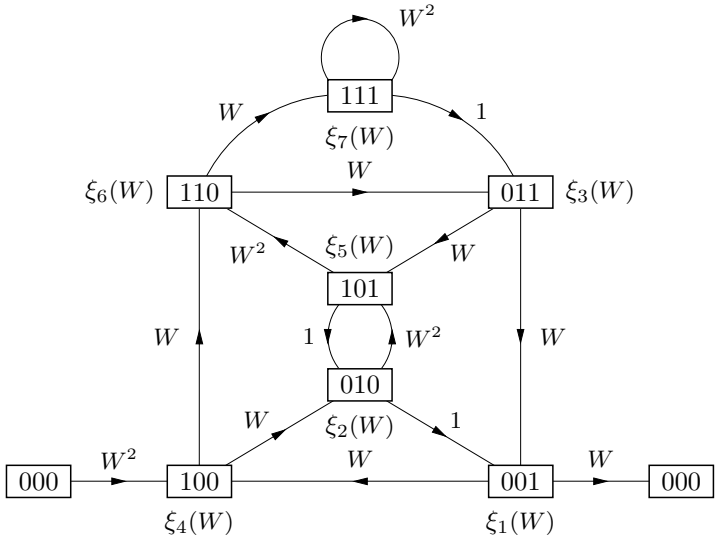


Figure 6.12 Signal flowchart for the encoding matrix in Example 6.4.

Proof: This theorem follows immediately from the proof of Theorem 6.19 if we replace $\left(2\sqrt{\epsilon(1-\epsilon)}\right)^w$ by the corresponding Bhattacharyya bound for the Gaussian channel, viz., $(e^{-RE_b/N_0})^w$ (cf. Theorem 4.7). ∎

6.7 COMMENTS

Elias introduced list decoding for block codes in 1955 in the very same paper where convolutional codes were introduced for the first time [Eli55]. Anderson introduced a list algorithm for source coding in his M.Sc. thesis 1969 [And69]. He called it the "M-algorithm," where M denotes the number of extended paths. Actually, it was conceived in August 1968 during a camping trip in Quebec, Canada. "It was very cold and I tried to forget how cold it was!"

Early theoretical work on list decoding was done by Zigangirov and Kolesnik [ZiK80]. Further investigations of the list decoding algorithm (the M-algorithm) were done by Anderson and his students; see, for example, [Lin86, And89, And92]. The bounds on the list weight were derived by Johannesson and Zigangirov, see [JoZ96], which together with [ZiO93] contains various bounds of the same type as those presented here.

Use of systematic convolutional encoders to solve the correct path loss problem was first reported in [Ost93]. This phenomenon is discussed in depth in [OAJ98].

PROBLEMS

6.1 Consider a binary symmetric channel BSC with the error sequence $e = 01\,01\,10\,00\,00\ldots$. Decode the received sequence for the following combinations of encoders and decoders:

- **a)** Systematic convolutional encoding matrix $G(D) = (1 \quad 1+D+D^2+D^4)$, $d_{\text{free}} = 5$. List decoder with $L = 3$.
- **b)** Systematic convolutional encoding matrix $G(D) = (1 \quad 1 + D + D^3 + D^4 + D^5 + D^8)$, $d_{\text{free}} = 7$. List decoder with $L = 3$.
- **c)** Nonsystematic convolutional encoding matrix $G(D) = (1+D+D^2 \quad 1+D^2)$, $d_{\text{free}} = 5$. Viterbi algorithm.
- **d)** Nonsystematic convolutional encoding matrix $G(D) = (1+D+D^4 \quad 1+D^2+D^3+D^4)$, $d_{\text{free}} = 7$. Viterbi algorithm.

6.2 Consider a binary symmetric channel BSC with the error sequence $e = 01\,01\,10\,01\,00\,00\ldots$. Decode the received sequence for the following combinations of encoders and decoders:

- **a)** Systematic convolutional encoding matrix $G(D) = (1 \quad 1+D+D^2+D^4)$, $d_{\text{free}} = 5$. List decoder with $L = 3$.
- **b)** Systematic convolutional encoding matrix $G(D) = (1 \quad 1 + D + D^2 + D^5 + D^6 + D^8 + D^{10} + D^{11})$, $d_{\text{free}} = 9$. List decoder with $L = 3$.
- **c)** Nonsystematic convolutional encoding matrix $G(D) = (1+D+D^2 \quad 1+D^2)$, $d_{\text{free}} = 5$. Viterbi algorithm.

6.3 Consider the rate $R = 1/2$ convolutional encoding matrix $G(D) = (1+D^3+D^4 \quad 1+D^4)$. Suppose that the encoder is used to communicate over a BSC with crossover probability ϵ. Use the list decoder with $L = 4$ to decode the received sequence $\boldsymbol{r} = 11\,01\,10\,10\,11\,00\,10\,11\;\;10\,01\,01\,11$. Eight information bits followed by four dummy zeros have been encoded.

6.4 Consider the binary-input, 8-ary output DMC shown in Fig. 4.6 with transition probabilities $P(r \mid v)$ given by the following table:

		r							
		0_4	0_3	0_2	0_1	1_1	1_2	1_3	1_4
v	0	0.2196	0.2556	0.2144	0.1521	0.0926	0.0463	0.0167	0.0027
	1	0.0027	0.0167	0.0463	0.0926	0.1521	0.2144	0.2556	0.2196

An information sequence is encoded by the encoding matrix $G(D) = (1 + D + D^2 \quad 1 + D^2)$. Use a list decoder of list size L to decode

$$\boldsymbol{r} = 0_2 1_3 \quad 1_2 0_2 \quad 0_1 1_3 \quad 1_1 1_1 \quad 1_2 1_4 \quad 1_1 1_1$$

Find the minimum value of L for which the output from the list decoder is a maximum-likelihood estimate of the transmitted information sequence. (Use appropriate scaling and rounding of the metrics.)

6.5 Consider the systematic convolutional encoding matrix $G(D) = (1 \quad 1 + D)$ with $d_{\text{free}} = 3$. Find the following parameters:

a) $N_{\max}(e)$ for $e = 1, 2$
b) $w_{\min}$ for $L = 1, 2, 3$
c) w_{list} for $L = 1, 2, 3$

6.6 Repeat Problem 6.5 for the systematic convolutional encoding matrix $G(D) = (1 \quad 1 + D + D^2)$ with $d_{\text{free}} = 4$.

6.7 Consider the nonsystematic convolutional encoding matrix $G(D) = (1 + D + D^2 \quad 1 + D^2)$ with $d_{\text{free}} = 5$. Find the following parameters:
a) $N_{\max}(e)$ for $e = 1, 2$
b) $w_{\min}$ for $L = 1, 2, 3$
c) w_{list} for $L = 1, 2, 3$
d) Find two systematic convolutional encoding matrices that are equivalent over the first memory length.

6.8 Consider the rate $R = 1/2$, memory $m = 3$, nonsystematic encoding matrix $G(D) = (1 + D + D^2 + D^3 \quad 1 + D^2 + D^3)$ and assume that it is used together with list decoding with list size $L = 3$ to communicate over the BSC.

Show that $P(\mathcal{E}_t^{\text{cpl}}) \lesssim 1.5W^3$, where $W = 2\sqrt{\epsilon(1-\epsilon)}$.

CHAPTER 7

SEQUENTIAL DECODING

In Chapter 4, we saw that Viterbi decoding is maximum-likelihood and that the complexity of Viterbi's algorithm grows exponentially with the overall constraint length ν of the encoder. If a certain application requires extremely low error probability, we may have to use an encoder with such a large overall constraint length, $\nu = 25$ say, that the Viterbi decoder would be hopelessly complex.

In this chapter, we describe a class of algorithms, called *sequential decoding algorithms*, whose complexity is essentially independent of the memory of the encoder. In sequential decoding, only one encoder state is examined at each time instant. By allowing backtracking, sequential decoding approaches asymptotically a maximum-likelihood decision for the transmitted sequence (with increasing encoder memory). It is an example of a decoding method that (asymptotically) fully exploits the error-correcting capability of the code. We choose a code whose encoder memory is long enough to warrant essentially error-free decoding. Sequential decoding algorithms are almost maximum-likelihood, and they can be easily implemented.

When we consider convolutional encoders with overall constraint lengths so large that Viterbi decoding is impractical, we usually view the encoding process as a walk through a tree instead of through a trellis. The fundamental idea behind sequential decoding is that in the decoding process we should explore only the most promising paths. If a path to a node looks "bad" we can discard all paths stemming from this

Fundamentals of Convolutional Coding, 2. Edition. By Rolf Johannesson, Kamil Sh. Zigangirov

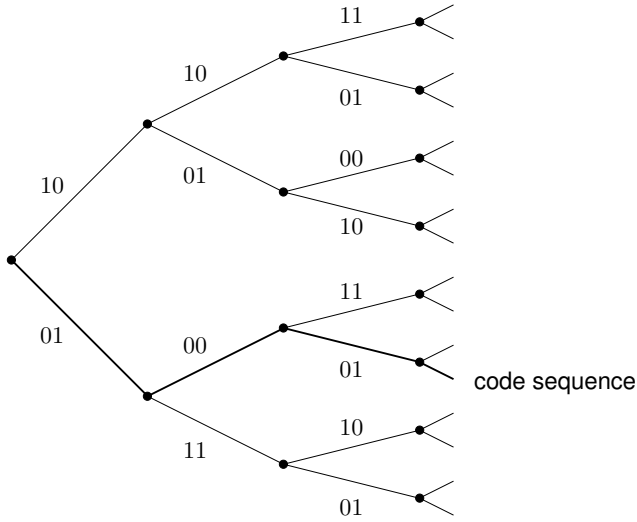


Figure 7.1 A rate $R = 1/2$ tree code.

node without any essential loss in performance from that of a maximum-likelihood decoder.

We begin by introducing a quality measure that could guide a decoder in its search for the most promising path to explore. This will lead us naturally to the *stack algorithm*, which is the simplest one to describe and analyze. After discussing the stack algorithm, we will be well prepared for the more intricate *Fano algorithm*. Then we discuss *Creeper*, which combines the best properties of the other two algorithms. For all three algorithms we analyze their computational performances and obtain upper bounds on the decoding error probabilities.

7.1 THE FANO METRIC

Consider the tree diagram for a rate $R = 1/2$, binary *tree code* shown in Fig. 7.1. Starting at the root at time 0, we traverse the tree from left to right and choose the upper branch when the corresponding information symbol is 0 and the lower branch when it is 1. Thus the information sequence $101\ldots$ is encoded as the code sequence $01\,00\,01\ldots$.

First we will consider "hard decisions" and then we will extend our results to "soft decisions."

Suppose that we receive the sequence $\boldsymbol{r} = 00\,10\,01\ldots$ over a BSC and that in the decoding process we have obtained in some unexplained way the partially explored tree shown in Fig. 7.2.

Regardless of which path the decoder eventually will choose, it must pass through exactly one of the leaves in this tree. It is tempting to ask: "Which of these four paths is the most promising to extend?" In the Viterbi algorithm we always compare

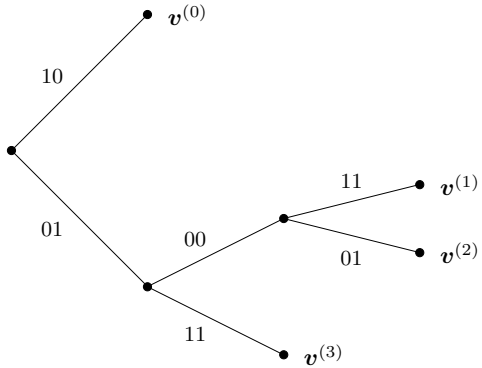


Figure 7.2 Partially explored tree.

paths of the same length and prefer the one closest to the received sequence—in, for example, Hamming distance. Here the situation is more subtle. Should we, for instance, prefer $\boldsymbol{v}^{(0)}$ with one erroneous symbol of two instead of $\boldsymbol{v}^{(2)}$ with two erroneous symbols of six? It depends on the channel—if the crossover probability is very small, then $\boldsymbol{v}^{(0)}$ is more promising than $\boldsymbol{v}^{(2)}$, but if it is very large, then we should extend $\boldsymbol{v}^{(2)}$! We will now analyze the situation in more detail.

Let us assume that the tree code shown in Fig. 7.1 belongs to the ensemble of binary tree codes in which each digit on each branch is chosen independently with probability $1/2$.

All information bits are also chosen independently with equal probability. Let X denote a particular configuration of all symbols in the partially explored tree, and let H_i denote the hypothesis that the transmitted path corresponds to the information sequence $\boldsymbol{u}^{(i)}$, $i = 0, 1, \ldots$.

The noise symbols are independent and 1 and 0 occur with probability ϵ and $1 - \epsilon$, respectively, where ϵ is the crossover probability of the BSC.

Let n_i denote the length of the path $\boldsymbol{v}^{(i)}$ in c-tuples, and let $d_i = d_{\mathrm{H}}(\boldsymbol{r}_{[0,n_i)}, \boldsymbol{v}^{(i)})$ denote the Hamming distance between the path $\boldsymbol{v}^{(i)}$ and the corresponding part of the received sequence, $\boldsymbol{r}_{[0,n_i)}$, $i = 0, 1, \ldots$.

Suppose that H_j maximizes $P(H_i \mid X, \boldsymbol{r})$, $i = 0, 1, \ldots$. Then the natural choice for the most promising path to extend is $\boldsymbol{v}^{(j)}$! Using Bayes' rule, we have

$$P(H_i \mid X, \boldsymbol{r}) = \frac{P(X \mid H_i, \boldsymbol{r})\, P(H_i \mid \boldsymbol{r})}{P(X \mid \boldsymbol{r})} \tag{7.1}$$

where

$$P(X \mid H_i, \boldsymbol{r}) = \frac{P(\boldsymbol{r} \mid X, H_i)\, P(X \mid H_i)}{P(\boldsymbol{r} \mid H_i)} \tag{7.2}$$

and, since the hypothesis H_i and the received sequence $\boldsymbol{r}$ are independent if we do not know the configuration X,

$$P(H_i \mid \boldsymbol{r}) = P(H_i) = 2^{-n_i R c} \tag{7.3}$$

Without additional knowledge, neither the received sequence $\boldsymbol{r}$ nor the configuration X is dependent on the hypotheses. Hence, we have

$$P\left(\boldsymbol{r} \mid H_i\right) = P(\boldsymbol{r}) \tag{7.4}$$

and

$$P\left(X \mid H_i\right) = P(X) \tag{7.5}$$

Inserting (7.4) and (7.5) into (7.2) and then inserting (7.2) and (7.3) into (7.1) and multiplying by the factor $P(\boldsymbol{r})P\left(X \mid \boldsymbol{r}\right)/P(X)$, we see that we can equivalently maximize

$$\begin{aligned} P\left(\boldsymbol{r} \mid X, H_i\right) P(H_i) &= (1-\epsilon)^{n_i c - d_i} \epsilon^{d_i} 2^{-(n-n_i)c} 2^{-n_i R c} \\ &= 2^{-nc} \left((1-\epsilon)2^{1-R}\right)^{n_i c - d_i} \left(\epsilon 2^{1-R}\right)^{d_i} \end{aligned} \tag{7.6}$$

where $n \geq \max_j\{n_j\}$ is the length of the received sequence in c-tuples.

Again, equivalently, in the case of hard decisions we can maximize

$$\mu_{\mathrm{F}}(\boldsymbol{r}_{[0,n_i)}, \boldsymbol{v}^{(i)}_{[0,n_i)}) \stackrel{\text{def}}{=} (n_i c - d_i)(\log(1-\epsilon) + 1 - R) + d_i(\log\epsilon + 1 - R) \tag{7.7}$$

$i = 0, 1, \ldots$, which quantity is usually called the hard-decision *Fano metric* (although it is not a metric in the mathematical sense).

The most promising path to extend is the one whose Fano metric $\mu_{\mathrm{F}}(\boldsymbol{r}_{[0,n_i)}, \boldsymbol{v}^{(i)}_{[0,n_i)})$ is the largest!

In general, the Fano metric for the path $\boldsymbol{v}_{[0,t)}$ is

$$\mu_{\mathrm{F}}(\boldsymbol{r}_{[0,t)}, \boldsymbol{v}_{[0,t)}) = \sum_{k=0}^{t-1} \mu_{\mathrm{F}}(\boldsymbol{r}_k, \boldsymbol{v}_k) \tag{7.8}$$

where $\mu_{\mathrm{F}}(\boldsymbol{r}_k, \boldsymbol{v}_k)$ is the *Fano branch metric*.

For notational convenience we sometimes use

$$\begin{array}{rcl} \alpha & \stackrel{\text{def}}{=} & \log(1-\epsilon) + 1 - R \\ \beta & \stackrel{\text{def}}{=} & \log\epsilon \quad\quad + 1 - R \end{array} \tag{7.9}$$

Then the Fano branch metric can be expressed as

$$\mu_{\mathrm{F}}(\boldsymbol{r}_k, \boldsymbol{v}_k) = j\alpha + (c-j)\beta \tag{7.10}$$

where $c - j = d_{\mathrm{H}}(\boldsymbol{r}_k, \boldsymbol{v}_k)$. It has the distribution

$$P(\mu_{\mathrm{F}}(\boldsymbol{r}_k, \boldsymbol{v}_k) = j\alpha + (c-j)\beta) = \binom{c}{j}(1-\epsilon)^j \epsilon^{c-j} \tag{7.11}$$

if the branch is on the *correct (transmitted)* path and for the ensemble of random tree codes, the distribution

$$P(\mu_{\mathrm{F}}(\boldsymbol{r}_k, \boldsymbol{v}_k) = j\alpha + (c-j)\beta) = \binom{c}{j} 2^{-c} \tag{7.12}$$

if the branch is on an *incorrect (erroneous)* path.

From (7.11) it follows that along the correct path we have

$$\begin{aligned} E[\mu_{\mathrm{F}}(\boldsymbol{r}_k, \boldsymbol{v}_k)] &= ((1-\epsilon)\alpha + \epsilon\beta)c \\ &= ((1-\epsilon)(\log(1-\epsilon) + 1 - R) + \epsilon(\log\epsilon + 1 - R))c \\ &= (1 - h(\epsilon) - R)c = (C - R)c \end{aligned} \tag{7.13}$$

where $h(\epsilon)$ is the binary entropy function (1.22) and $C = 1 - h(\epsilon)$ is the channel capacity for the BSC.

If $R < C$, then on the average we have an *increase* in the Fano metric along the *correct* path.

Along an incorrect path we have

$$\begin{aligned} E[\mu_{\mathrm{F}}(\boldsymbol{r}_k, \boldsymbol{v}_k)] &= \left(\frac{1}{2}\alpha + \frac{1}{2}\beta\right)c \\ &= \left(\frac{1}{2}(\log(1-\epsilon) + 1 - R) + \frac{1}{2}(\log\epsilon + 1 - R)\right)c \\ &= \left(1 - R + \log\sqrt{\epsilon(1-\epsilon)}\right)c \leq -Rc \end{aligned} \tag{7.14}$$

with equality if and only if $\epsilon = 1/2$. On the average, the Fano metric will always *decrease* along an *incorrect* path!

The *Fano symbol metric* is

$$\mu_{\mathrm{F}} \stackrel{\text{def}}{=} \mu_{\mathrm{F}}(r_k^{(\ell)}, v_k^{(\ell)}) \stackrel{\text{def}}{=} \begin{cases} \alpha & \text{if } r_k^{(\ell)} = v_k^{(\ell)} \\ \beta & \text{if } r_k^{(\ell)} \neq v_k^{(\ell)} \end{cases} \tag{7.15}$$

where α and β are given by (7.9).

■ **EXAMPLE 7.1**

Consider a rate $R = 1/2$ convolutional code used to communicate over a BSC with crossover probability $\epsilon = 0.045$. Then we have the following Fano symbol metric:

$$\mu_{\mathrm{F}}(r_k^{(\ell)}, v_k^{(\ell)}) = \begin{cases} \alpha = \log 0.955 + 1 - 1/2 = 0.434 & \text{if } r_k^{(\ell)} = v_k^{(\ell)} \\ \beta = \log 0.045 + 1 - 1/2 = -3.974 & \text{if } r_k^{(\ell)} \neq v_k^{(\ell)} \end{cases} \tag{7.16}$$

For simplicity, we scale and round the metrics and obtain

$$\mu'_{\mathrm{F}}(r_k^{(\ell)}, v_k^{(\ell)}) = \begin{cases} +0.5, & \text{no error} \\ -4.5, & \text{error} \end{cases} \tag{7.17}$$

for the BSC with $\epsilon = 0.045$.

Remark: We can scale the Fano metrics with any positive factor γ. If we choose

$$\gamma = (1 + s)/s \tag{7.18}$$

where s satisfies

$$R = G(s)/s \tag{7.19}$$

and $G(s)$ is the Gallager function (5.98), then we obtain the metric (5.211) used to prove the error bound for time-varying convolutional codes for the sphere-packing region (Theorem 5.19).

We will now extend the Fano metrics to the situation when soft demodulator outputs are available. The derivation given above is valid up to equation (7.6), which should be replaced by

$$\begin{aligned} P(\boldsymbol{r} \mid X, H_i)\, P(H_i) &= \prod_{j=0}^{n_i-1} P\left(\boldsymbol{r}_j \;\middle|\; \boldsymbol{v}_j^{(i)}\right) \prod_{k=n_i}^{n} P(\boldsymbol{r}_k) 2^{-n_i R c} \\ &= \prod_{k=0}^{n} P(\boldsymbol{r}_k) \prod_{j=0}^{n_i-1} \frac{P\left(\boldsymbol{r}_j \;\middle|\; \boldsymbol{v}_j^{(i)}\right)}{P(\boldsymbol{r}_j)} 2^{-n_i R c} \end{aligned} \tag{7.20}$$

where n_i is the length of the codeword $\boldsymbol{v}^{(i)} = \boldsymbol{v}_0^{(i)} \boldsymbol{v}_1^{(i)} \ldots \boldsymbol{v}_{n_i-1}^{(i)}$ and $n \geq \max_j\{n_j\}$ is the length of the received sequence. Again we take the logarithm and, equivalently, maximize

$$\begin{aligned} \mu_{\mathrm{F}}(\boldsymbol{r}_{[0,n_i)}^{(i)}, \boldsymbol{v}_{[0,n_i)}) &\stackrel{\text{def}}{=} \sum_{j=0}^{n_i-1} \left(\log \frac{P\left(\boldsymbol{r}_j \;\middle|\; \boldsymbol{v}_j^{(i)}\right)}{P(\boldsymbol{r}_j)} - Rc \right) \\ &= \sum_{j=0}^{n_i-1} \sum_{\ell=1}^{c} \left(\log \frac{P\left(r_j^{(\ell)} \;\middle|\; v_j^{(i)(\ell)}\right)}{P(r_j^{(\ell)})} - R \right) \end{aligned} \tag{7.21}$$

where $\boldsymbol{r}_j = r_j^{(1)} r_j^{(2)} \ldots r_j^{(c)}$ and $\boldsymbol{v}_j^{(i)} = v_j^{(i)(1)} v_j^{(i)(2)} \ldots v_j^{(i)(c)}$, $i = 0, 1, \ldots$. The terms in the first sum are the Fano branch metrics, and the terms in the double sum are the Fano symbol metrics.

■ **EXAMPLE 7.2**

Consider a rate $R = 1/2$ convolutional code used to communicate over a binary-input, 8-ary output DMC with equiprobable inputs and transition probabilities $P(r \mid v)$ given by the following table:

		r							
		0_4	0_3	0_2	0_1	1_1	1_2	1_3	1_4
v	0	0.434	0.197	0.167	0.111	0.058	0.023	0.008	0.002
	1	0.002	0.008	0.023	0.058	0.111	0.167	0.197	0.434

Then we have the following Fano symbol metrics:

		r							
		0_4	0_3	0_2	0_1	1_1	1_2	1_3	1_4
v	0	0.49	0.44	0.31	−0.11	−1.04	−2.55	−4.18	−7.27
	1	−7.27	−4.18	−2.55	−1.04	−0.11	0.31	0.44	0.49

For simplicity we scale and round to integers and obtain:

		r							
		0_4	0_3	0_2	0_1	1_1	1_2	1_3	1_4
v	0	9	8	6	-2	-18	-46	-75	-131
	1	-131	-75	-46	-18	-2	6	8	9

7.2 THE STACK ALGORITHM

The discussion in the previous section leads us naturally to the obvious *stack algorithm*, in which we store the Fano metrics for all paths leading to a terminal node and then extend to next depth the path with the greatest Fano metric. The adjective "stack" is used to indicate that the paths in the partially explored tree are ordered by the Fano metrics. The best path is placed at the top of the stack; then we have the second best path and so on.

Remark: Although it is an abuse of notations, we have chosen to follow a deeply rooted tradition in using the word "stack" instead of the proper word "list."

When we use a rate $R = b/c$ convolutional encoder of memory m to communicate a finite number of information bits, ℓb say, we append a *tail* of mb dummy zeros to the information bits in order to terminate the convolutional code into a block code. If the string of dummy zeros is shorter than mb we do not fully exploit the error-correcting capability of the code to protect the last information symbols.

Algorithm S (Stack)

S1. Load the stack with the root and the metric zero.

S2. Remove the top node and place its successors in the stack according to their metrics.

S3. If the top path leads to the end of the tree, then stop and choose the top path to be the decoded codeword; otherwise go to **S2**.

■ **EXAMPLE 7.3**

Consider the rate $R = 1/2$ binary convolutional encoding matrix $G(D) = (1 + D + D^2 \quad 1 + D^2)$ with memory $m = 2$. We encode $\ell = 3$ information symbols and $m = 2$ dummy zeros. Assuming that the code is used together with a BSC with $\epsilon = 0.045$, we use the Fano symbol metrics given in Example 7.4, viz.,

$$\mu_{\mathrm{F}} = \begin{cases} +0.5, & \text{no error} \\ -4.5, & \text{error} \end{cases} \tag{7.22}$$

The received sequence $\boldsymbol{r} = 01\,10\,01\,10\,11$ is decoded by the stack algorithm as follows (the stack entries are the paths $\boldsymbol{u}^{(i)}$ and the corresponding Fano metrics):

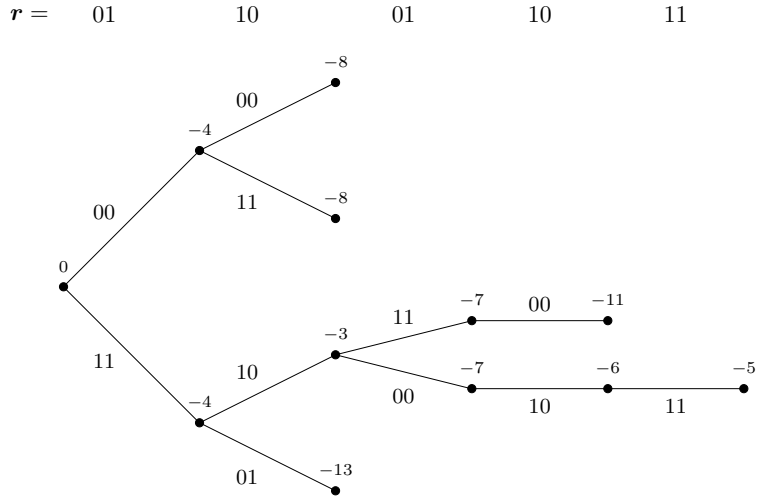


Figure 7.3 Partially explored binary tree—hard decisions.

Step	0	1	2	3	4
stack contents	$-, 0$	$0, -4$	$1, -4$	$10, -3$	$100, -7$
		$1, -4$	$00, -8$	$00, -8$	$101, -7$
			$01, -8$	$01, -8$	$00, -8$
				$11, -13$	$01, -8$
					$11, -13$

5	6	7
$101, -7$	$1010, -6$	$10100, -5$
$00, -8$	$00, -8$	$00, -8$
$01, -8$	$01, -8$	$01, -8$
$1000, -11$	$1000, -11$	$1000, -11$
$11, -13$	$11, -13$	$11, -13$

The partially explored tree is shown in Fig. 7.3 together with the Fano metrics. The stack algorithm's decision is the path 10100 or the information sequence 101.

■ EXAMPLE 7.4

The same code as that used in Example 7.2 is used to communicate over a binary-input, 8-ary output DMC with equiprobable inputs and transition probabilities $P(r \mid v)$ given in Example 7.1 where the following Fano metrics per symbol were calculated:

		r							
		0_4	0_3	0_2	0_1	1_1	1_2	1_3	1_4
v	0	9	8	6	-2	-18	-46	-75	-131
	1	-131	-75	-46	-18	-2	6	8	9

The soft demodulator outputs $\boldsymbol{r} = 0_1 1_2\ 1_3 0_2\ 0_4 1_3\ 1_1 0_3\ 1_2 1_1$ are decoded by the stack algorithm as follows:

Step	0	1	2	3
stack contents	$-, 0$	$0, -12$	$10, +2$	$0, -48$
		$1, -48$	$0, -48$	$101, -64$
			$11, -133$	$100, -121$
				$11, -133$

4	5	6
$101, -64$	$1010, -58$	$10100, -54$
$01, -86$	$01, -86$	$01, -86$
$00, -117$	$00, -117$	$00, -117$
$100, -121$	$100, -121$	$100, -121$
$11, -133$	$11, -133$	$11, -133$

The partially explored tree is shown in Fig. 7.4 together with the Fano metrics per symbol. The stack algorithm's decision is the path 10100 or the information sequence 101.

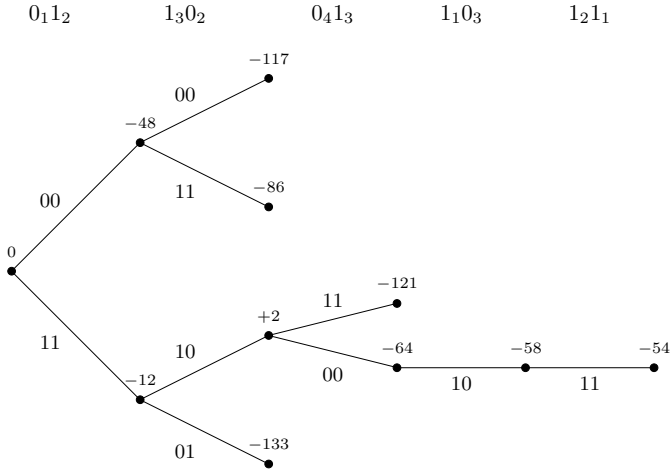


Figure 7.4 Partially explored binary tree—soft decisions.

7.3 THE FANO ALGORITHM

The *Fano algorithm* is an extremely clever algorithm for sequential decoding. It generally visits more nodes than the stack algorithm, but since it requires essentially no memory it is more suitable for hardware implementations. In practice, this more than compensates for the additional computations needed to decode a given sequence.

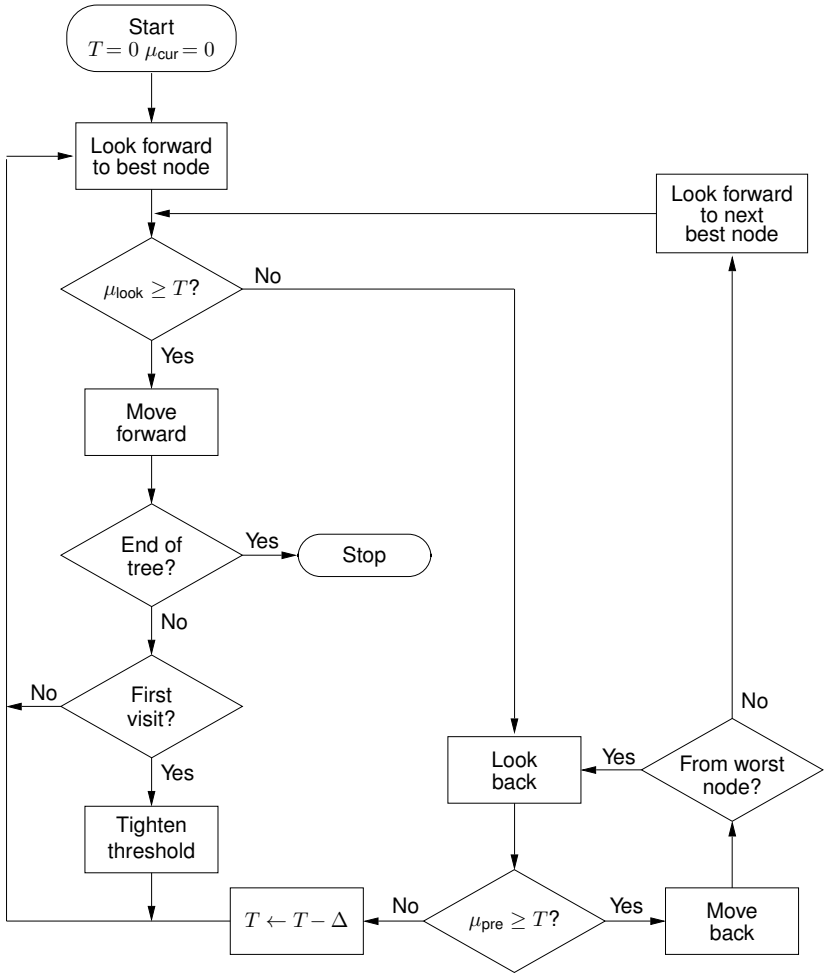


Figure 7.5 Flowchart of the Fano algorithm.

The Fano decoder moves from a certain (current) node either to its predecessor or to one of its immediate successors—it never jumps. The flowchart of the Fano algorithm is shown in Fig. 7.5, where μ_{look} and μ_{pre} are the Fano metrics of the successor node which we look at and the predecessor node, respectively. If the decoder looks backward from the root, we assume that the "predecessor" of the root has a metric value of $-\infty$.

The decoder can visit a node only if its Fano metric μ_{F} is larger than or equal to the current value of a certain *threshold* T which takes on only discrete values $\ldots, -2\Delta, -\Delta, 0, \Delta, 2\Delta, \ldots$, where Δ is the *stepsize*.

> If the decoder visits a *new* node then the threshold should be increased by the largest multiple of Δ such that the new threshold T does not exceed the current

metric, that is, such that $T \leq \mu_{\text{cur}} < T + \Delta$ holds. If it explores the immediate predecessor of the current node and the predecessors metric is lower than the threshold then the threshold should be lowered by Δ.

If the successor's metric falls below the threshold, then the Fano decoder first moves backward and then tries to move forward to its next best successor. If this fails, another move backward is necessary, and so on. Eventually, all paths with metrics above the threshold T have been systematically visited.

```
Init;
Look forward to best node;
while μ_look < T do                                          ▷ see note 1.
   T ← T − Δ
end while
Move forward;
while not end of tree do                                     ▷ see note 2.
   if First visit then
      Tighten threshold;
   end if
   Look forward to best node;
   while μ_look<T do                                         ▷ see note 3.
      if μ_pre ≥ T then
         repeat                                              ▷ see note 5.
            Move back;
         until not ( (From worst node) & (μ_pre ≥ T) )
         if not (From worst node) then                       ▷ see note 7.
            Look forward to next best node;
         else
            T ← T − Δ                                        ▷ see note 6.
            Look forward to nest node;
         end if
      else
         T ← T − Δ
         Look forward to best node;
      end if
   end while
   Move Forward;
end while
```

When the Fano decoder reaches the node where the threshold T was increased the previous time, the Fano decoder lowers T by Δ and tries to move forward again, now with a lower threshold. This means that when a node is revisited, $T + \Delta \leq \mu_{\text{cur}}$ holds. Eventually, the Fano decoder will reach the end of the tree and complete the decoding procedure.

"First visit?" can be detected using μ_{cur} and μ_{pre} as follows:

If $\mu_{\text{pre}} < T + \Delta$ holds, the *previous* node n_{pre} cannot have been visited with a threshold higher than T. Hence, the previous visit of n_{pre} was a "first visit" and, since n_{cur} is a successor of n_{pre}, n_{cur} is also visited for the first time.

If $\mu_{\text{pre}} \geq T + \Delta$ and $\mu_{\text{cur}} < T + \Delta$ hold, then it follows that n_{cur} cannot have been visited before, but n_{pre} has. Since $\mu_{\text{cur}} < T + \Delta$, the threshold is already tight.

If $\mu_{\text{pre}} \geq T + \Delta$ and $\mu_{\text{cur}} \geq T + \Delta$ hold, both n_{pre} and n_{cur} have been visited before and the threshold should *not* be increased.

Hence, the condition for the threshold to be increased can be written

$$\begin{array}{rcl} \mu_{\text{pre}} & < & T + \Delta \\ \mu_{\text{cur}} & \geq & T + \Delta \end{array} \tag{7.23}$$

where the first inequality in (7.23) indicates a first visit and the second inequality in (7.23) indicates that the threshold is not already tight.

Notes

1. If the threshold is too high, then it must be lowered.
2. This is the main loop. As long as we do not encounter any channel errors we will execute this loop.
3. The metric is decreased on the chosen path.
4. We have moved forward and tightened the threshold, so we cannot move backward. Since we cannot proceed forward either, we have to lower the threshold in order to continue.
5. After having examined both successor nodes, we have returned to the present node. We will continue backward as long as we are coming from the worst node and are staying above the threshold.
6. As in 5 but here the threshold is too high. We have to decrease the threshold before we can proceed forward.
7. We have moved backward and are looking forward to the next best node that we have not examined.

7.4 THE CREEPER ALGORITHM*

The stack algorithm is simple to describe and analyze. It is very attractive from a pedagogical point of view. However, any practical implementation includes accesses to the large stack (external) memory. These accesses will limit the clock frequency. In the Fano algorithm we can eliminate these external memory accesses and thus execute the algorithm at a higher clock frequency. On the other hand, the Fano algorithm will visit more nodes; this drawback does not override the strong implementational advantages.

In this section we will describe an algorithm that is a compromise between the stack and Fano algorithms. It visits more nodes than the stack algorithm but fewer than the Fano algorithm, while it can be implemented without the need of an external memory that slows down the clock frequency. We call this sequential decoding algorithm *Creeper* since it explores the code tree in a way that resembles the behavior of a creeping plant.

Consider the partially explored tree in Fig. 7.6. Let n_{cur} denote the *current node* with metric μ_{cur}, and let its two *successors* be n_x and n_y with metrics μ_x and μ_y, respectively. Also, assume without loss of generality that $\mu_x \geq \mu_y$ holds. We call the path leading to the current node the *stem* of the partially explored tree. The *subtrees* stemming from the stem are denoted $\mathcal{T}_1, \mathcal{T}_2, \ldots$. When we have computed (for the first time) the metrics of the successor nodes of a given node, we say that these successor nodes are *examined* but not visited. Let $\mu_1, \mu_2, \ldots, \mu_7$ be the metrics of the best, that is, largest, metric examined but not visited nodes in the subtrees $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_7$, respectively. In Fig. 7.7 we let only the root of subtree $\mathcal{T}_i$, viz., n_{τ_i}, together with the largest metric of an examined node in that subtree, μ_i, represent subtree $\mathcal{T}_i$.

Assume that the metric of the current node n_{cur} is the largest among the metrics of all examined but not visited nodes in the subtrees, that is,

$$\mu_{\text{cur}} \geq \max\{\mu_1, \mu_2, \ldots, \mu_7\} \tag{7.24}$$

Furthermore, assume that the metric μ_3 is the second largest.

The stack algorithm removes the current top node of the stack, n_{cur}, and calculates the metrics of its two successor nodes, n_x and n_y, and compares these metrics with μ_3, the metric of n_3, the new top node of the stack. If $\mu_x \geq \mu_3 \geq \mu_y$, then the stack algorithm proceeds to node n_x, which will be the next top node; n_3 will again be the node with the second largest metric on the stack. If $\mu_3 > \mu_x \geq \mu_y$, then node n_3 will remain as the top node of the stack. In the next step, the stack algorithm removes the top node (n_3) from the stack and so on.

Creeper has only stored (some of) the nodes shown in Fig. 7.7. Hence, it cannot move directly to n_3. Instead, since μ_3 is the largest metric, Creeper will move to n_{τ_3} and with n_{τ_3} as the new current node work its way to node n_3 in the subtree $\mathcal{T}_3$. When n_{τ_3} is made the current node, the stem is pruned at node n_3^s, which is stored together with the metric value μ_3' representing the metrics of all examined nodes stemming from n_3^s, viz.,

$$\mu_3' = \max\{\mu_4, \mu_5, \mu_6, \mu_7, \mu_x\} \tag{7.25}$$

and the decoding process is continued. (In general, Creeper stores only a subset of the nodes along the stem. Moreover, only a subset of these are considered important enough to warrant the storage of the associated μ-value.)

If we, as in the stack algorithm, always visit the nodes with the largest metrics, we will jump to n_{τ_3} also when μ_3 is only slightly larger than μ_x. This causes a substantial increase in the number of computations. By introducing a *threshold* T, we can restrict the algorithm to return to only those nodes that are *sufficiently promising*, that is, with metrics above the threshold T. These nodes are called *buds* and are stored on a *node stack*.

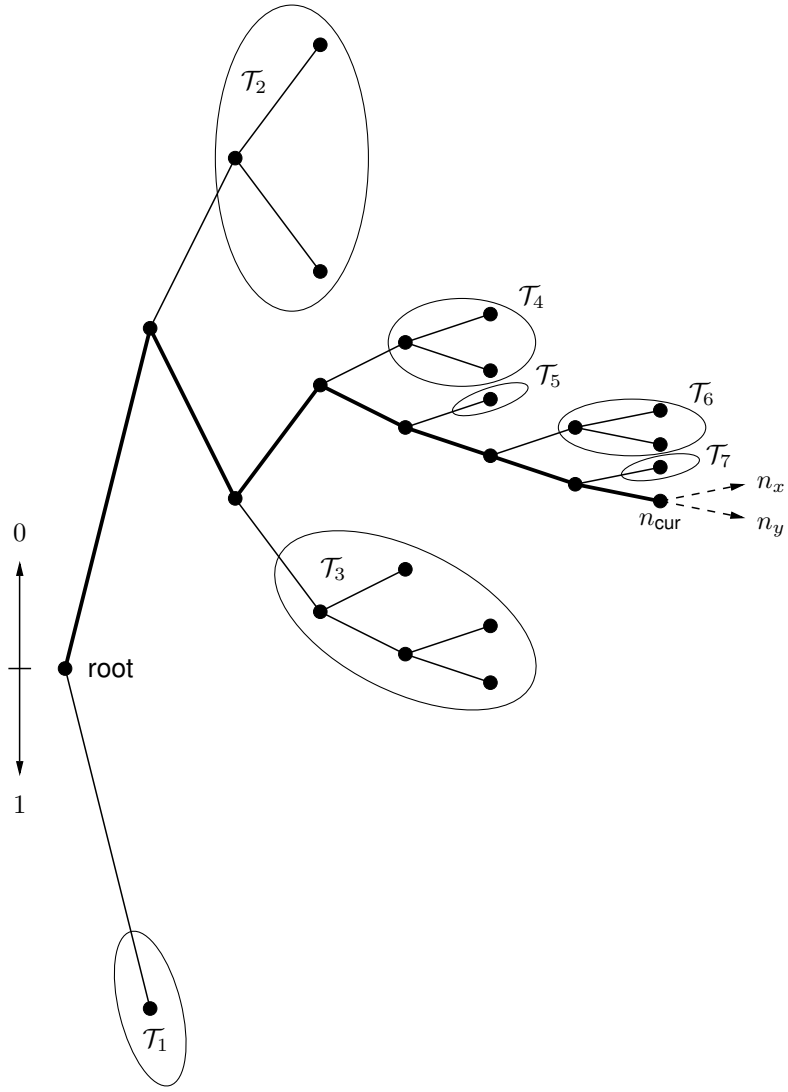


Figure 7.6 A partially explored tree.

The nodes on the node stack are referred to by the node stack pointer NP. The top node is denoted $n(NP)$, the second $n(NP-1)$, and so on. The node $n(NP-1)$ corresponds to the closest bud.

Some of the nodes on the node stack are considered so good that their metrics should affect the threshold, viz., increase it. These nodes are called *T-nodes*, and to each T-node there is a metric value stored on the *threshold stack*. This metric value, denoted μ_τ, is *the metric of the best examined but not visited node in the subtree*

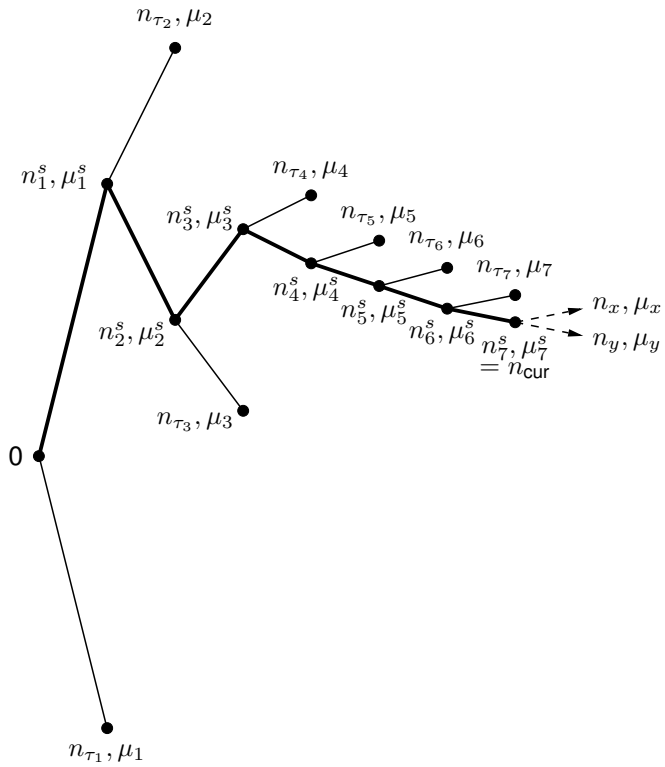


Figure 7.7 The stem corresponding to the partially explored tree in Fig. 7.6.

stemming from that node the previous time that subtree was explored. This value is a measure of what we expect to find if we later should (re-)visit that subtree. For each node on the node stack there is also a flag, F, denoted $F(NP)$, that indicates whether or not the corresponding node is a T-node.

The elements on the threshold stack are referred to by the threshold stack pointer TP, and the elements on the stack are denoted $\mu_\tau(TP)$, $\mu_\tau(TP-1)$, and so on. We also denote the subtree stemming from the T-node corresponding to the value $\mu_\tau(TP)$ by $\tau(TP)$, and so on. The current threshold T is computed as $Q(\mu_\tau(TP-1))$, where

$$Q(x) = \left\lfloor \frac{x}{\Delta} \right\rfloor \Delta \tag{7.26}$$

is a quantization function that converts x into the largest multiple of a positive parameter Δ not larger than x. By convention, we let $Q(-\infty) = -\infty$. The threshold for the current subtree is constructed with the help of the metric of the best examined but not visited node in $\tau(TP-1)$.

Creeper must remember the maximum metric, $\mu_{\max}$, of all nodes that have been examined so far. If it finds a node (n_x) with a metric larger than $\mu_{\max}$ and also n_y has a metric above T, then these nodes are stored as T-nodes and the new threshold will be computed as $T = Q(\mu_y)$. Thus, the threshold is raised only when it is likely that

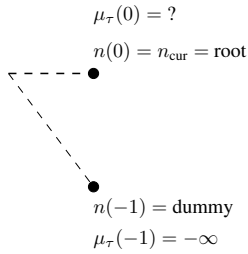


Figure 7.8 The stack looks like this when Creeper starts. The purpose of the dummy node, or rather its associated threshold $\mu_\tau(-1) = -\infty$, is to guarantee that Rule One is the very first rule to be applied. Thus, in the sequel none of the root's successors will be removed.

the decoder is following the correct path, that is, when a new $\mu_{\max}$ value is found. It is less likely that a threshold raise occurs while examining an incorrect path. All *non-T-nodes* in $\tau(TP)$ that are stacked have metrics μ satisfying $T \leq \mu \leq \mu_{\max}$, but these non-$T$-node buds are not necessarily stacked according to increasing metrics.

The threshold T will determine the further actions of the algorithm. Creeper will continue to explore the tree forward to n_x if $\mu_x \geq T$. If $\mu_y \geq T$, n_y should also be stored on the node stack. If $\mu_x < T$, the decoder cannot move forward and will leave the current subtree for the subtree stemming from $n(NP-1)$. Depending on whether or not that node is a T-node, different actions must be taken. If it is a T-node, the decoder must decide whether or not to keep the T-node on the threshold stack. If it is a non-T-node, a backtracking move is made in order to systematically examine all paths above the threshold.

We are now well prepared to give a formal description of Creeper. For simplicity we consider only rate $R = 1/c$ convolutional codes. The Creeper algorithm for rate $R = b/c$ convolutional codes is given at the end of this section.

Algorithm C (Creeper rate $R = 1/c$)

C1. Store the root node, $n(0)$, and a dummy sibling, $n(-1)$, on the node stack as T-nodes, that is, $F(0) = F(-1) \leftarrow 1$.
Set $\mu_\tau(-1) \leftarrow -\infty$ on the threshold stack.
Set $\mu_{\max} \leftarrow -\infty$, $TP = NP \leftarrow 0$, and $n_{\text{cur}} \leftarrow root$. See Fig. 7.8.

C2. Compute the successor metrics μ_x, μ_y ($\mu_x \geq \mu_y$) and the threshold $T = Q(\mu_\tau(TP-1))$. Ties are resolved by choosing n_x among the successors in an arbitrary way.

C3. Perform one of six actions depending on the corresponding conditions. The actions are specified in Fig. 7.9.

C4. If we have reached the end of the tree, STOP, and output the current path as the estimated information sequence; otherwise go to **C2**.

The $exchange(\quad,\quad)$ operation above simply changes places of the two elements on top of the appropriate stack. Stacking at most two (2^b in general) elements at each

Rule	Condition	Decoder actions
One	$T \le \mu_y$ and $\mu_x > \mu_{\max}$	$n_{\text{cur}} \leftarrow n_x$ $n(NP+1) \leftarrow n_y$ $n(NP+2) \leftarrow n_x$ $\mu_\tau(TP+1) \leftarrow \mu_y$ $\mu_\tau(TP+2) \leftarrow -\infty$ $F(NP+1) \leftarrow 1$ $F(NP+2) \leftarrow 1$ $NP \leftarrow NP+2$ $TP \leftarrow TP+2$ $\mu_{\max} \leftarrow \mu_x$
Two	$T \le \mu_y$ and $\mu_x \le \mu_{\max}$	$n_{\text{cur}} \leftarrow n_x$ $n(NP+1) \leftarrow n_y$ $n(NP+2) \leftarrow n_x$ $F(NP+1) \leftarrow 0$ $F(NP+2) \leftarrow 0$ $NP \leftarrow NP+2$
Three	$\mu_y < T \le \mu_x$	$n_{\text{cur}} \leftarrow n_x$ $\mu_\tau(TP) \leftarrow \max\{\mu_y, \mu_\tau(TP)\}$ $\mu_{\max} \leftarrow \max\{\mu_x, \mu_{\max}\}$
Four	$\mu_x < T$ and $F(NP-1)=1$ and $\max(\mu_x, \mu_\tau(TP)) \ge Q(\mu_\tau(TP-3))$	$n_{\text{cur}} \leftarrow n(NP-1)$ $\mu_\tau(TP) \leftarrow \max\{\mu_x, \mu_\tau(TP)\}$ exchange $(\mu_\tau(TP), \mu_\tau(TP-1))$ exchange $(n(NP), n(NP-1))$ exchange $(F(NP), F(NP-1))$ $\mu_\tau(TP) \leftarrow -\infty$
Five	$\mu_x < T$ and $F(NP-1)=1$ and $\max\{\mu_x, \mu_\tau(TP)) < Q(\mu_\tau(TP-3)\}$	$n_{\text{cur}} \leftarrow n(NP-1)$ $\mu_\tau(TP-2) \leftarrow \max\{\mu_x,$ $\mu_\tau(TP), \mu_\tau(TP-2)\}$ $NP \leftarrow NP-2$ $TP \leftarrow TP-2$
Six	$\mu_x < T$ and $F(NP-1)=0$	$n_{\text{cur}} \leftarrow n(NP-1)$ $\mu_\tau(TP) \leftarrow \max\{\mu_x, \mu_\tau(TP)\}$ $NP \leftarrow NP-2$

Figure 7.9 Six rules that specify the behavior of Creeper for rate $R = 1/c$.

depth makes the hardware memory requirements roughly proportional to the code tree depth. Note that the initial conditions guarantee that at the very first iteration Rule One always will apply. It is easy to see that exactly one of the six conditions above will always apply. If the received sequence is error-free, condition one will always hold, thus, making all nodes in the decoder stack as T-nodes. In Figs. 7.10–7.17 we illustrate how Creeper explores the tree and how the stacks are affected by the different rules. Note that in these figures TP and NP have not been updated in order to make it easier to follow the actions.

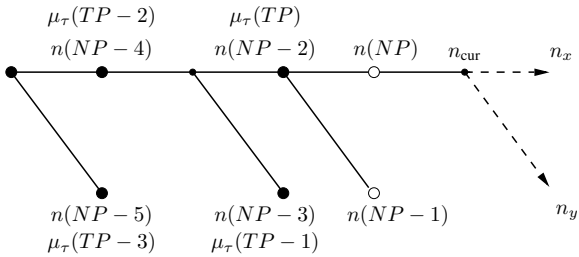


Figure 7.10 The situation could look like this before one of the rules One, Two, Three, or Six is applied. The nodes $n(NP-2)$, $n(NP-3)$, $n(NP-4)$, and $n(NP-5)$ are T-nodes and marked $\bullet$, the nodes $n(NP-1)$ and $n(NP)$ are non-T-nodes and marked $\circ$, the nodes marked $\bullet$ are not stored on the stack.

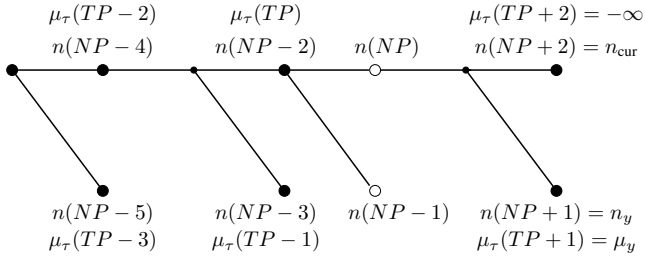


Figure 7.11 Rule One: Move forward and stack both successors as T-nodes. Two elements, corresponding to the two new T-nodes, are stacked on the threshold stack.

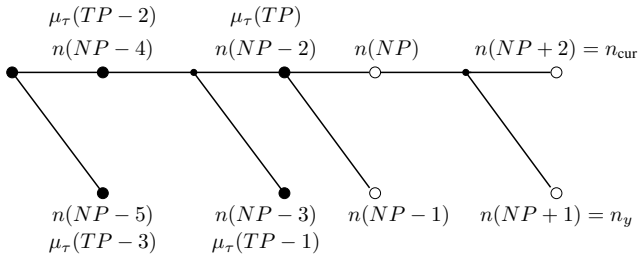


Figure 7.12 Rule Two: Move forward and stack both successors as non-T-nodes.

Consider the rate $R = 1/2$ binary convolutional encoding matrix $G(D) = (1 + D + D^2 \quad 1 + D^2)$ with memory $m = 2$. Assuming the encoder is used together with a BSC with crossover probability $\epsilon = 0.045$, we use the Fano symbol metrics

$$\mu_{\text{F}} = \begin{cases} \alpha = +0.5, & \text{no error} \\ \beta = -4.5, & \text{error} \end{cases} \tag{7.27}$$

Since we have two symbols on each branch, the Fano symbol metrics in (7.27) correspond to the Fano branch metrics $+1, -4$, and -9 when we have 0, 1, and 2 assumed errors per branch, respectively. Let $\boldsymbol{r} = (01\,01\,00\,10\,00\ldots)$ be the received

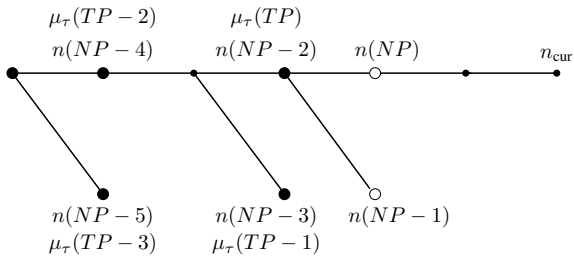


Figure 7.13 Rule Three: Move forward and update $\mu_\tau(TP)$ with $\max\{\mu_y, \mu_\tau(TP)\}$. Since n_y has been examined but will not be visited this time, $\tau(TP)$ is visited and consequently $\mu_\tau(T)$ will be updated using μ_y.

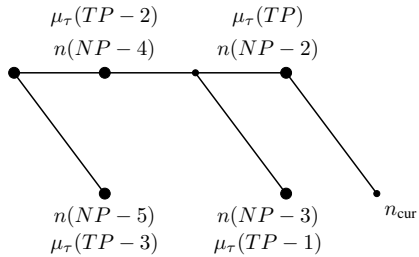


Figure 7.14 Rule Six: Move sideways-backward to the closest (non-T-)node on the node stack and delete the two top elements on the node stack. Update $\mu_\tau(TP)$ with $\max\{\mu_x, \mu_\tau(TP)\}$.

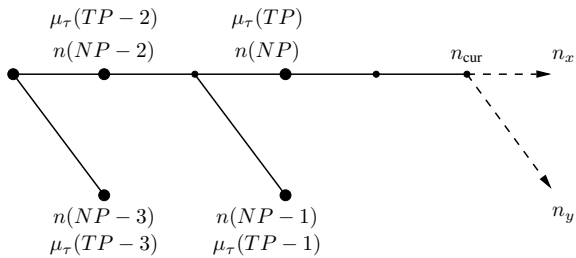


Figure 7.15 The situation could look like this before one of the rules Four or Five is applied. The metric value $\max\{\mu_x, \mu_\tau(TP)\}$ of the best examined but not visited node in $\tau(TP)$ is compared to $Q(\mu_\tau(TP-3)$.

sequence. The code tree and its node metrics are given in Fig. 7.18. Finally, we assume the threshold spacing $\Delta = 3$.

In Fig. 7.19(a–h) we show the first eight steps of Creeper's travel through the code tree and how the variables involved change. In case of a tie between the successor metrics, we assume that the zero path is always chosen first. Big black circles are T-nodes, big white circles are non-T-nodes, and the small black circles correspond to nodes that have been visited but that are not stored on the node stack.

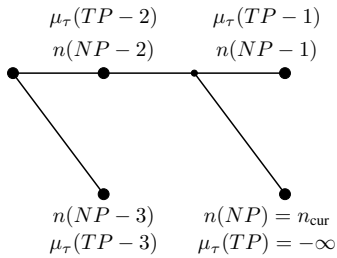


Figure 7.16 Rule Four: Move to closest (T-)node on the stack and exchange the two top elements of the node and threshold stacks.

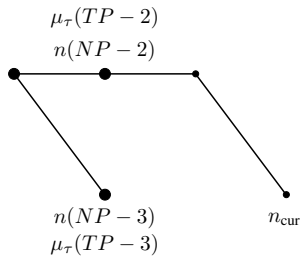


Figure 7.17 Rule Five: Move to the closest (T-)node on the stack and delete the two top elements on the node and threshold stacks.

The node currently being visited is the node from which the two dotted arrows originate. These arrows indicate that the nodes pointed to are currently being examined. The dotted lines correspond to the subtrees in $\tau(TP)$ that have been examined or visited and that have been discarded. The metrics at the end of these lines are the metrics of the nodes at the end of these paths, that is, nodes that have been examined but not visited. Evidently, these metrics lie below the current threshold. For each graphic in Fig. 7.19(a–h), we describe the situation and state which condition will hold for that situation.

(a) The initial condition guarantees that Condition One holds, leading to stacking of the successors as T-nodes and updating $\mu_{\max}$ to -4.

(b) The threshold T is given by $Q(-4) = -6$, so both successors lie below T. Thus, $\mu_\tau(2)$ is updated to -8. The closest node on the stack is a T-node, and $\max\{-8, -\infty\}$ is not less than $Q(\mu_\tau(-1)) = -\infty$, so Condition Four holds.

(c) Only one successor node lies above the threshold, so Condition Three holds, making the other node update $\mu_\tau(2)$. Also, $\mu_{\max}$ is updated to -3.

(d) Both successors lie above T, but none exceeds $\mu_{\max}$, so Condition Two holds.

(e) Both successors lie below T and the closest node on the stack is a non-T-node. Thus, Condition Six holds, so $\mu_\tau(2)$ is updated to -11.

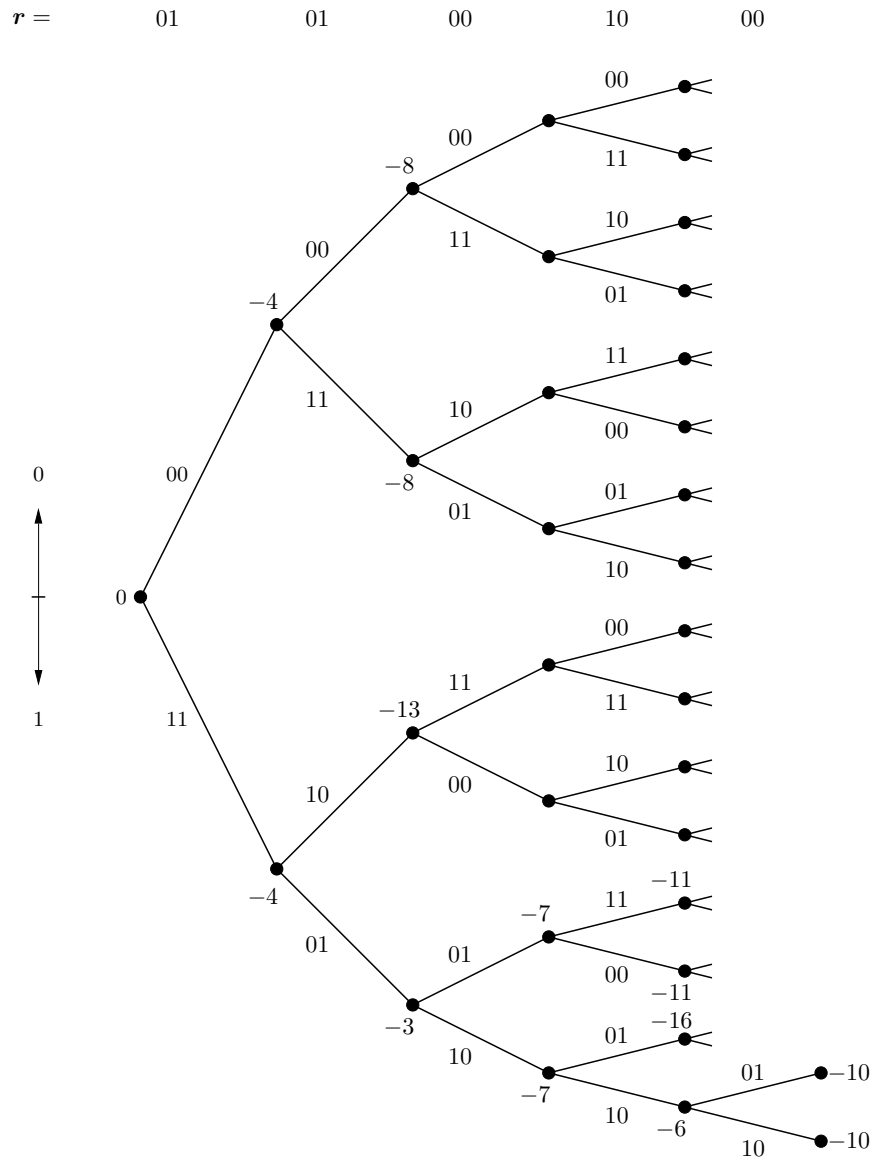


Figure 7.18 The part of the code tree with its node metrics that is used in the example.

(f) Only one successor node lies above the threshold, so Condition Three holds, making the other node update $\mu_\tau(2)$, which will not change.

(g) Both successors lie below T and the closest node is a T-node, and $\max\{-10, -11\}$ is not less than $Q(\mu(-1)) = -\infty$, so Condition Four holds.

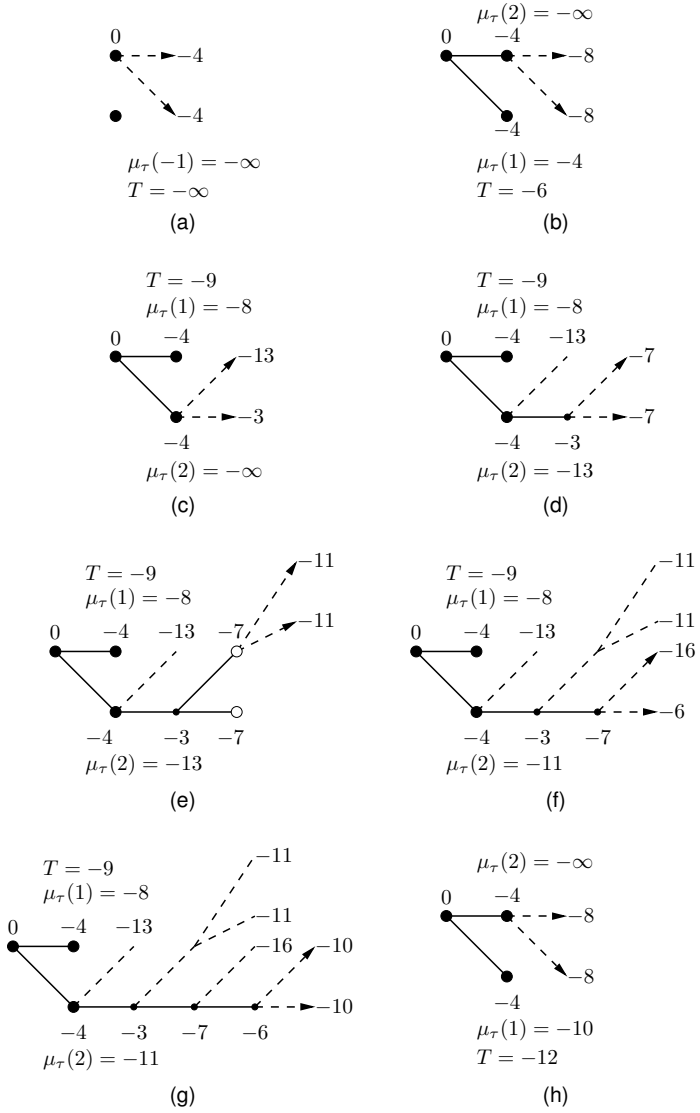


Figure 7.19 The first eight steps of Creeper when $\Delta = 3$ and the received sequence is $(01\ 01\ 00\ 10\ 00\ \ldots)$. The node metrics are shown next to the corresponding nodes.

(h) We have the same situation as in (b) except that we have a threshold that is lowered by an amount of 2Δ compared to (b), and a new $\mu_{\max}$ value.

From Fig. 7.19(g) it is evident that all nodes in $\tau(2)$ with metrics above $T = -9$ have been visited (metrics $-4, -3, -7, -7, -6$) and that no node with metric below

T has been visited. Furthermore, the best metric of all nodes that have been examined but not visited (metrics $-13, -11, -11, -16, -10, -10$) is remembered ($\mu_\tau(1)$ in Fig. 7.19(h) after the exchange of stack elements).

The example shows that Creeper, contrary to the Fano algorithm, does not always increase (tighten) the threshold when a node is visited for the first time. Instead, Creeper continues in the current subtree as long as the best known alternative is not better; the threshold is constructed using the metric of the best known alternative as $T = Q(\mu_\tau(TP - 1))$. When a new largest (so far) metric, $\mu_{\max}$, has been found, a new T-node is stacked (Rule One) and a threshold increase occurs. Usually, the threshold increases occur along the correct path; along the incorrect paths the threshold is usually kept constant, allowing Creeper to exhaust most of the incorrect subpaths to be explored in one forward attempt. In the incorrect subtree the Fano algorithm moves forward, tightens the threshold, backs up, lowers the threshold, continues forward, and so on, visiting the same incorrect nodes several times. After the threshold T has been decreased (as a consequence of Rule Four or Five), Creeper will always visit at least one node that it had not visited before.

We conclude this section by briefly discussing an extension of Creeper to rate $R = b/c$ convolutional codes. In this case, any number of the 2^b successor nodes to the current node could be suitable for further investigation, that is, have a metric above the current threshold. At each depth a variable number of nodes could be stacked, not just zero or two as in the original Creeper for $R = 1/c$.

Stems from earlier levels are not allowed to be visited until all possible paths stemming from nodes at larger depths are exhausted. Therefore, we have to store the number of promising nodes at each depth along with these nodes, in order to keep track of remaining candidate stems.

This number is decreased as paths leading into "bad" parts of the tree are discarded using a certain threshold but could be increased when the parent node is revisited with a lower threshold, which may allow more successor nodes to be promising than at the previous visits.

Since we have to control a varying number of stacked nodes and corresponding μ_τ values at a certain depth, we introduce two stacks similar to those in the rate $R = 1/c$ version, viz., a *node object stack* and a *threshold object stack*. An object (of any of the two kinds) now holds several elements, all associated with sibling nodes at the same depth. The node object stack pointer, NP, is used as reference to the objects on the node object stack and the threshold object stack pointer, TP, as reference to objects on the threshold stack.

A node object contains: N, denoting the number of promising sibling nodes at that depth; the N sibling nodes denoted $n_1, \ldots, n_N$; and a flag F, denoting whether the N nodes are T-nodes ($F = 1$) or not ($F = 0$). The nodes are initially sorted such that n_i has a metric not less than that of n_{i+1}.

A node object is denoted $N(NP)$, $N(NP - 1)$, and so on, depending on the relative position of the object from the top of the node object stack.

A threshold object contains N metric values, $\mu_\tau^1, \mu_\tau^2, \ldots, \mu_\tau^N$, where N is the number of corresponding T-nodes. These metrics are kept sorted so that the relations $\mu_\tau^i \geq \mu_\tau^{i+1}$, $i = 1, \ldots, N - 1$, hold.

Each such value is the metric of a node that has been examined but not yet visited in the subtree stemming from the corresponding node in an appropriate node object on the node object stack.

A threshold object is denoted $\mu(TP)$, $\mu(TP-1)$, and so forth, depending on the relative position of the object from the top of the threshold object stack.

A specific value or node within a particular object is denoted $N(NP).N$ (the number of promising siblings in the topmost node object). The flag F and the N nodes are referred to in a similar way, and so are the μ_τ^i values in the threshold objects in the threshold object stack.

The currently visited node is denoted n_{cur}.

The 2^b successors to n_{cur} are denoted $n_1, \ldots, n_{2^b}$ and their corresponding metrics are $\mu_1 \geq \cdots \geq \mu_{2^b}$, possibly after sorting.

Now we are ready to give a formal description of the rate $R = b/c$ version of Creeper.

Algorithm C (Creeper rate $R = b/c$)

C1. Let $TP = 0$. Let $NP = 0$. Store a node object, $N(NP)$, on the node object stack with $N(NP).N = 2$, $N(NP).F = 1$, $N(NP).n1 =$ root, and $N(NP).n2 =$ a dummy node. Let $\mu(TP)$ be a corresponding threshold object with

$$\mu(TP).\mu_\tau^1 = \mu(TP).\mu_\tau^2 = -\infty$$

Finally, let $T = -\infty$, $n_{\text{cur}} =$ root, and $\mu_{\max} = -\infty$.

C2. Compute the threshold $T = Q(\mu(TP).\mu_\tau^2)$. Compute and sort the metrics of all successors to n_{cur}. Ties are resolved in an arbitrary way. Let N be the number of successors having metrics not smaller than T.

C3. Perform one of six actions specified in Fig. 7.20.

C4. If we have reached the end of the tree, STOP, and output the current path as the estimated information sequence; otherwise go to **C2**.

The "resort" operation in Rule Four is simple. All thresholds in the threshold object are already sorted except for threshold one, which only has to be inserted in the proper place. Node one is inserted in the same place in the node object so that the node and its corresponding threshold always have the same order number.

The "renumber" operation in Rules Five and Six is even simpler since after threshold one has been deleted the remaining thresholds remain sorted as before.

The "resorting" operation corresponds to an exchange of two elements when we have a rate $R = 1/c$ code.

7.5 SIMULATIONS

In the table below we compare the stack and Fano algorithms with Creeper. We have simulated the transmission of a large number of frames over a BSC with crossover

Rule	Condition	Decoder actions
One	$N \geq 2$ and $\mu_1 > \mu_{\max}$	$n_{\text{cur}} = n_1$ $N(NP+1).n_i = n_i,\ i = 1, \ldots, N$ $N(NP+1).N = N$ $N(NP+1).F = 1$ $NP = NP + 1$ $\mu(TP+1).\mu_\tau^1 = -\infty$ $\mu(TP+1).\mu_\tau^i = \mu_i, i = 2, \ldots, N$ if $N < 2^b$, then $\mu(TP).\mu_\tau^1 = \max\{\mu_{N+1}, \mu(TP).\mu_\tau^1\}$ $TP = TP + 1$ $\mu_{\max} = \max\{\mu_{\max}, \mu_1\}$
Two	$N \geq 2$ and $\mu_1 \leq \mu_{\max}$	$n_{\text{cur}} = n_1$ $N(NP+1).n_i = n_i,\ i = 1, \ldots, N$ $N(NP+1).N = N$ $N(NP+1).F = 0$ $NP = NP + 1$ if $N < 2^b$, then $\mu(TP).\mu_\tau^1 = \max\{\mu_{N+1}, \mu(TP).\mu_\tau^1\}$
Three	$N = 1$	$n_{\text{cur}} = n_1$ $\mu_{\max} = \max\{\mu_{\max}, \mu_1\}$ $\mu(TP).\mu_\tau^1 = \max\{\mu_2, \mu(TP).\mu_\tau^1\}$
Four	$N = 0$ and $N(NP).F = 1$ and $\max\{\mu_1, \mu(TP).\mu_\tau^1\} \geq Q(\mu(TP-1).\mu_\tau^2)$	$n_{\text{cur}} = N(NP).n_2$ $\mu(TP).\mu_\tau^1 = \max\{\mu_1, \mu(TP).\mu_\tau^1\}$ Resort the $N(NP).N$ thresholds in $\mu(TP)$ Resort the $N(NP).N$ nodes in $N(NP)$ *accordingly* $\mu(TP).\mu_\tau^1 = -\infty$
Five	$N = 0$ and $N(NP).F = 1$ and $\max\{\mu_1, \mu(TP).\mu_\tau^1\} < Q(\mu(TP-1).\mu_\tau^2)$	$n_{\text{cur}} = N(NP).n_2$ $\mu(TP-1).\mu_\tau^1 = \max\{\mu_1, \mu(TP).\mu_\tau^1, \mu(TP-1).\mu_\tau^1\}$ Delete $N(NP).n_1$ Delete $\mu(TP).\mu_\tau^1$ Renumber the remaining $N(NP).N - 1$ thresholds in $\mu(TP)$ Renumber the remaining $N(NP).N - 1$ nodes in $N(NP)$ $N(NP).N = N(NP).N - 1$ if $N(NP).N = 1$, then $NP = NP - 1$, and $TP = TP - 1$
Six	$N = 0$ and $N(NP).F = 0$	$n_{\text{cur}} = N(NP).n_2$ $\mu(TP).\mu_\tau^1 = \max\{\mu_1, \mu(TP).\mu_\tau^1\}$ Delete $N(NP).n_1$ Renumber the remaining $N(NP).N - 1$ nodes in $N(NP)$ $N(NP).N = N(NP).N - 1$ if $N(NP).N = 1$, then $NP = NP - 1$

Figure 7.20 Six rules that specify the behavior of Creeper for rate $R = b/c$.

probability $\epsilon = 0.034$ which corresponds to a rate $R = 0.9R_0$, that is, 10% below the computational cutoff rate R_0 when $R = 1/2$. We used the rate $R = 1/2$, memory 23, ODP convolutional encoding matrix $G(D) = (75744143 \quad 55346125)$ with free distance $d_{\text{free}} = 25$. (The polynomials are given in octal notation.) Information

sequences of 500 bits were encoded as 1046 bit code sequences (frames) before the transmission; that is, $m = 23$ dummy information zeros were used to terminate the convolutional code (ZT). We have simulated the transmission of 1,000,000 frames (200,000 for Fano) when the average number of computations needed to decode the *first* branch $\widehat{C}_1$ was estimated. When the average number of computations needed to decode *a* branch $\widehat{C}$ was estimated, 200,000 frames were transmitted. The frames were aborted when C and C_1 exceeded 100. Due to the large encoder memory, no frames whose decoding terminated normally were erroneously decoded. (The Δ values in the table are optimized empirically and shown before scaling and quantizing to integers.)

Algorithm	$\widehat{C}_1$ sim.	$\widehat{C}$ sim.
Stack	1.20	1.29
Creeper, $\Delta = 2.6$	1.23	1.54
Fano, $\Delta = 4.8$	1.71	2.26

The estimations of $\widehat{C}$ suggest that Creeper is significantly more efficient than the Fano algorithm and only slightly worse than the stack algorithm.

Finally, in Figs. 7.21 and 7.22, we show simulations of the computational distributions for the number of computations $P(C_1 \geq x)$ and $P(C \geq x)$, respectively, for the three algorithms.

7.6 COMPUTATIONAL ANALYSIS OF THE STACK ALGORITHM

The curse of sequential decoding is that the number of computations needed to decode a received sequence varies with the channel noise in a particularly nasty way.

Consider the ensemble $\mathcal{E}(b, c, \infty, 1)$ of infinite-memory, time-invariant, rate $R = b/c$ convolutional codes and the partially explored binary tree shown in Fig. 7.23. For simplicity we assume that the decoder operates continuously in the infinite code tree, and since the codes have infinite-memory we assume that the decoder *never* makes any decoding errors. Let $\mathcal{T}_i$, $i = 1, 2, \ldots$, denote the set of *extended* nodes in the ith incorrect subtree ($\emptyset$ denotes the empty set).

If we count as a computation every extension of a node, then the number of computations needed to decode the ith *correct* node, C_i, can be written as

$$C_i = 1 + |\mathcal{T}_i| \tag{7.28}$$

$i = 1, 2, \ldots$, where $|\mathcal{T}_i|$ denotes the cardinality of the set $\mathcal{T}_i$. Since for the ensemble of random infinite tree codes the statistical properties are the same for all subtrees, the random variables C_i, $i = 1, 2, \ldots$, all have the same distribution, but they are certainly not independent. Thus, for the *average number of computations per branch*, C_{av}, we have

$$C_{\text{av}} = E[C_i] \tag{7.29}$$

all i. Without loss of generality, we will only consider the first incorrect subtree, that is, the incorrect subtree stemming from the root.

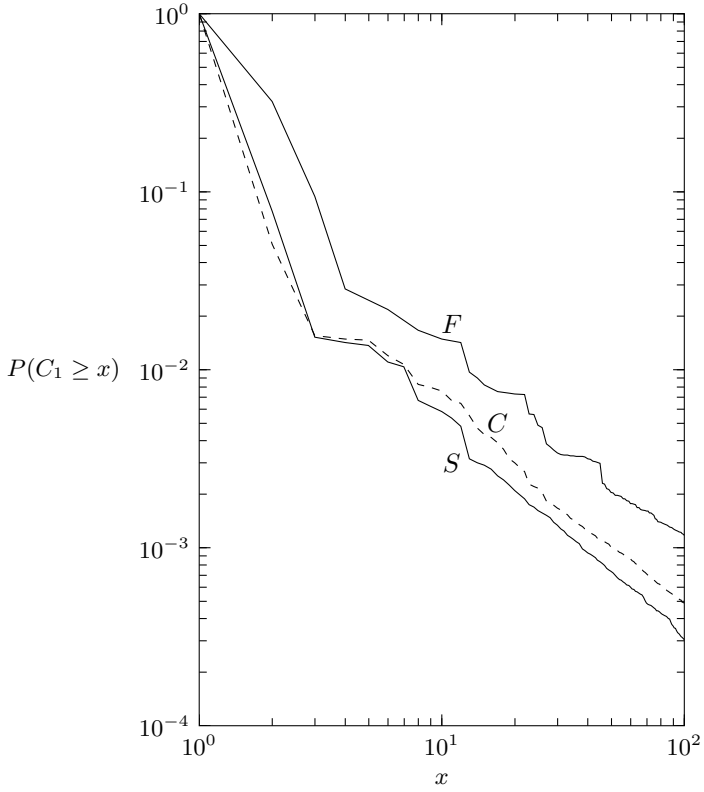


Figure 7.21 Simulations of the computational distribution functions $P(C_1 \geq x)$ for the Fano (F), Creeper (C), and stack (S) algorithms for the BSC with crossover probability $\epsilon = 0.034$.

It is obvious that C_{av} will depend on the distribution of the minimum of the metrics along the correct path.

In Section 7.1 we introduced the Fano metric. Now we will consider a more general symbol metric for the BSC, viz.,

$$\mu = \begin{cases} \alpha = \log(1-\epsilon) + 1 - B, & \text{no error} \\ \beta = \log \epsilon + 1 - B, & \text{error} \end{cases} \tag{7.30}$$

where the parameter B is called the *bias*. When $B = R$, we have the Fano symbol metric.

The following lemma is important when we want to characterize the behavior of the metric along the correct path. It follows from Wald's identity (Corollary B.6), and it gives a most useful upper bound on the minimum of the metric along the correct path.

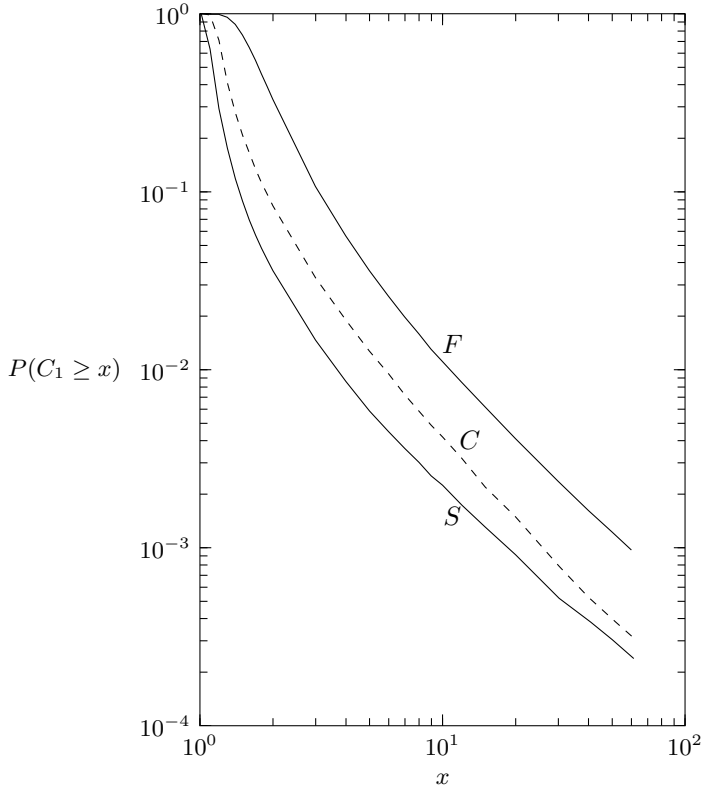


Figure 7.22 Simulations of the computational distribution functions $P(C \geq x)$ for the Fano (F), Creeper (C), and stack (S) algorithms for the BSC with crossover probability $\epsilon = 0.034$.

Lemma 7.1 Consider the BSC and let $\mu_{\min}$ denote the minimum of the metric $\mu(\boldsymbol{r}_{[0,t)}, \boldsymbol{v}_{[0,t)})$ along the correct path, that is,

$$\mu_{\min} = \min_t \{\mu(\boldsymbol{r}_{[0,t)}, \boldsymbol{v}_{[0,t)})\} \tag{7.31}$$

Then

$$P(\mu_{\min} \leq x) \leq 2^{-\lambda_0 x} \tag{7.32}$$

where λ_0 is the smallest root of the equation

$$g(\lambda) \stackrel{\text{def}}{=} E[2^{\lambda_0 \mu(\boldsymbol{r}_k, \boldsymbol{v}_k)}] = \left((1-\epsilon)2^{\lambda_0 \alpha} + \epsilon 2^{\lambda_0 \beta}\right)^c = 1 \tag{7.33}$$

where α and β are given by (7.30).

Equation (7.33) always has the root $\lambda_0 = 1$. To obtain a nontrivial bound from (7.32), equation (7.33) should have a root $\lambda_0 < 0$. This problem is addressed in the following (see Example B.1):

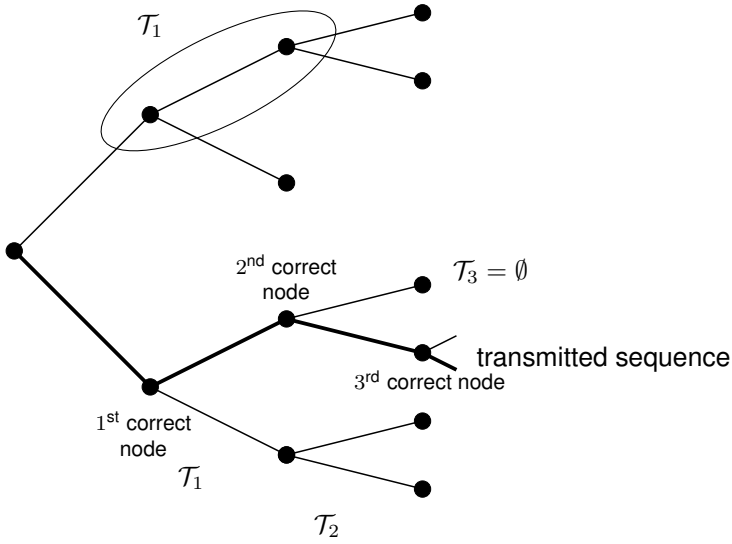


Figure 7.23 Partially explored binary tree.

Corollary 7.2 Consider a convolutional code of rate $R < C$, where C is the channel capacity of the BSC. Then, for the Fano metric there exists a negative root of (7.33), viz.,

$$\lambda_0 = -\frac{s}{1+s} \tag{7.34}$$

where s satisfies the equality

$$G(s) = sR \tag{7.35}$$

and $G(s)$ is the Gallager function (5.98).

Let the information sequence $\boldsymbol{u}'_{[0,j)}$ also denote the corresponding node at depth j in the first incorrect subtree, and let us introduce the function

$$\varphi(\boldsymbol{u}'_{[0,j)}) = \begin{cases} 1 & \text{if node } \boldsymbol{u}'_{[0,j)} \text{ is extended} \\ 0 & \text{else} \end{cases} \tag{7.36}$$

Then, from (7.28) it follows that

$$C_{\text{av}} = E[C_1] = 1 + \sum_{i=1}^{\infty} \sum_{\boldsymbol{u}'_{[0,j)} \in \mathcal{T}_1} E[\varphi(\boldsymbol{u}'_{[0,j)})] \tag{7.37}$$

Consider the random walk $S'_0 = 0, S'_1, \ldots, S'_j$ associated with the node $\boldsymbol{u}'_{[0,j)}$, where

$$S'_t = \sum_{i=0}^{t-1} Z'_i \tag{7.38}$$

and $Z'_i = \mu(\boldsymbol{v}'_i, \boldsymbol{r}_i)$ is the ith branch metric along the incorrect path $\boldsymbol{u}'_{[0,j)}$.

In the ensemble $\mathcal{E}(b, c, \infty, 1)$ the branch metrics along an incorrect path, Z'_i, are independent, identically distributed, random variables with the distribution (see Example B.2)

$$P(Z'_i = j\alpha + (c-j)\beta) = \binom{c}{j}\left(\frac{1}{2}\right)^c \tag{7.39}$$

$j = 0, 1, \ldots, c$. Hence, we have

$$E[Z'_i] = \left(\frac{1}{2}\alpha + \frac{1}{2}\beta\right)c = \left(1 - R + \sqrt{\epsilon(1-\epsilon)}\right)c < 0 \tag{7.40}$$

and

$$g_{Z'_i}(\lambda) = E[2^{\lambda Z_i}] = \left(\frac{1}{2}2^{\lambda\alpha} + \frac{1}{2}2^{\lambda\beta}\right)^c \tag{7.41}$$

For a node $\boldsymbol{u}'_{[0,j)}$ in the incorrect subtree to be extended, it is necessary that the random walk $S'_0, S'_1, \ldots, S'_j$ does *not* drop below a barrier at $\mu_{\min}$. What will happen when $\min_t\{S'_t\} = \mu_{\min}$? It depends on how the stack algorithm is implemented. A common and efficient method of resolving the ties when two nodes have exactly the same cumulative metrics is to use the "last in/first out" principle. In our analysis, we will adopt the more pessimistic view: in case of a tie between a node on the correct path and an incorrect node, always extend the incorrect node.

Let

$$f_j(y, z) \stackrel{\text{def}}{=} P(S'_t \geq y, \quad 0 \leq t < j, \quad S'_j = z) \tag{7.42}$$

Then the probability that a node $\boldsymbol{u}'_{[0,j)}$ in the first incorrect subtree will be extended, given that $\mu_{\min} = y$, can be upper-bounded by

$$E[\varphi(\boldsymbol{u}'_{[0,j)}) \mid \mu_{\min} = y] = \sum_{z \geq y} f_j(y, z) \tag{7.43}$$

Thus, we have

$$\begin{aligned} E[C_1 \mid \mu_{\min} = y] &= 1 + \sum_{j=1}^{\infty} \sum_{\boldsymbol{u}'_{[0,j)} \in \mathcal{T}_1} E\left[\varphi(\boldsymbol{u}'_{[0,j)}) \;\middle|\; \mu_{\min} = y\right] \\ &= 1 + \sum_{j=1}^{\infty} (2^b - 1)2^{b(j-1)} \sum_{z \geq y} f_j(y, z) \end{aligned} \tag{7.44}$$

From (7.40) and Lemma B.1 it follows that the infinite random walk $S'_0, S'_1, \ldots, S'_j, \ldots$ will cross from above any finite barrier at $y < 0$, that is,

$$\sum_{j=1}^{\infty} \sum_{z < y} f_j(y, z) = 1 \tag{7.45}$$

Furthermore, since the random walk S'_t will eventually cross the barrier y, it follows that

$$\sum_{z \geq y} f_j(y,z) = \sum_{t=j+1}^{\infty} \sum_{z<y} f_t(y,z) \tag{7.46}$$

Hence, inserting (7.46) into (7.44) yields

$$E\left[C_1 \mid \mu_{\min} = y\right] = 1 + \sum_{j=1}^{\infty} (2^b - 1) 2^{b(j-1)} \sum_{t=j+1}^{\infty} \sum_{z<y} f_t(y,z) \tag{7.47}$$

Next we replace the "1" in (7.47) by the double sum in (7.45) and obtain

$$\begin{aligned} E\left[C_1 \mid \mu_{\min} = y\right] &= \sum_{t=1}^{\infty} \sum_{z<y} f_t(y,z) + \sum_{j=1}^{\infty} (2^b - 1) 2^{b(j-1)} \sum_{t=j+1}^{\infty} \sum_{z<y} f_t(y,z) \\ &= \sum_{t=1}^{\infty} \sum_{z<y} f_t(y,z) + \sum_{j=1}^{\infty} 2^{bj} \sum_{t=j+1}^{\infty} \sum_{z<y} f_t(y,z) \qquad (7.48) \\ &\quad - \sum_{j=1}^{\infty} 2^{b(j-1)} \sum_{t=j+1}^{\infty} \sum_{z<y} f_t(y,z) \\ &= \sum_{j=0}^{\infty} 2^{bj} \sum_{t=j+1}^{\infty} \sum_{z<y} f_t(y,z) - \sum_{j=0}^{\infty} 2^{bj} \sum_{t=j+2}^{\infty} \sum_{z<y} f_t(y,z) \\ &= \sum_{j=0}^{\infty} 2^{bj} \sum_{z<y} f_{j+1}(y,z) = \sum_{t=1}^{\infty} 2^{b(t-1)} \sum_{z<y} f_t(y,z) \qquad (7.49) \end{aligned}$$

From Wald's identity (Corollary B.4) it follows that

$$\sum_{t=1}^{\infty} 2^{bt} \sum_{z<y} f_t(y,z) \leq 2^{-\lambda_1 (y + c\beta)} \tag{7.50}$$

where $\lambda_1 > 0$ is a positive root of

$$g_{Z'_t}(\lambda_1) = 2^{-b} \tag{7.51}$$

(That $g'_{Z'_i}(\lambda_1) < 0$ and the existence of such a λ_1 for the Fano metric are shown in Example B.2.) Then, by combining (7.48) and (7.50), we obtain the upper bound

$$E\left[C_1 \mid \mu_{\min} = y\right] \leq 2^{-b} 2^{-\lambda_1 (y + c\beta)} \tag{7.52}$$

Thus, we have proved the following:

Lemma 7.3 The average number of computations needed to decode the first correct node given that the minimum metric along the correct path is $\mu_{\min} = y$ is upper-bounded by

$$E\left[C_1 \mid \mu_{\min} = y\right] \leq 2^{-\lambda_1 (y + c\beta) - b} \tag{7.53}$$

where λ_1 is a positive root of equation (7.51) and β is given by (7.30).

For the Fano metric we show in Example B.2 that $\lambda_1 = 1/(1+s)$ is a positive root of (7.51), where s satisfies (7.35). Hence, we have the following:

Corollary 7.4 For the Fano metric we have the following:

$$E\left[C_1 \mid \mu_{\min} = y\right] \leq 2^{-\frac{1}{1+s}(y+c\beta)-b} \tag{7.54}$$

where s is the solution of equation (7.35).

Without any essential loss of accuracy we assume that the metric values can be written

$$\alpha = \alpha_0 \delta \tag{7.55}$$

and

$$\beta = \beta_0 \delta \tag{7.56}$$

where α_0 and β_0 are integers and $\delta > 0$. Then $\{\mu_{\min}\} = \{0, -\delta, -2\delta, -3\delta, \ldots\}$ and we have

$$\begin{aligned}
C_{\text{av}} = E[C_1] &= \sum_{i=0}^{\infty} E\left[C_1 \mid \mu_{\min} = -i\delta\right] P(\mu_{\min} = -i\delta) \\
&\leq \sum_{i=0}^{\infty} 2^{\lambda_1(i\delta - c\beta)-b} P(\mu_{\min} = -i\delta) \\
&= 2^{-\lambda_1 c\beta - b}\left(\sum_{i=0}^{\infty} 2^{i\lambda_1\delta}(P(\mu_{\min} \leq -i\delta) - P(\mu_{\min} \leq -(i+1)\delta))\right) \\
&= 2^{-\lambda_1 c\beta - b}\left(\sum_{i=0}^{\infty} 2^{i\lambda_1\delta} P(\mu_{\min} \leq -i\delta) - \sum_{i=1}^{\infty} 2^{(i-1)\lambda_1\delta} P(\mu_{\min} \leq -i\delta)\right) \\
&= 2^{-\lambda_1 c\beta - b}\left(\left(1 - 2^{-\lambda_1\delta}\right)\left(\sum_{i=0}^{\infty} 2^{i\lambda_1\delta} P(\mu_{\min} \leq -i\delta)\right) + 2^{-\lambda_1\delta}\right)
\end{aligned} \tag{7.57}$$

Now we use Lemma 7.1 and obtain

$$C_{\text{av}} \leq 2^{-\lambda_1 c\beta - b}\left(\left(1 - 2^{-\lambda_1\delta}\right)\left(\sum_{i=0}^{\infty} 2^{i\lambda_1\delta} 2^{i\lambda_0\delta}\right) + 2^{-\lambda_1\delta}\right) \tag{7.58}$$

The sum converges if $\lambda_0 + \lambda_1 < 0$ and, finally, we have

$$\begin{aligned}
C_{\text{av}} &\leq 2^{-\lambda_1 c\beta - b}\left(\frac{1 - 2^{-\lambda_1\delta}}{1 - 2^{(\lambda_0+\lambda_1)\delta}} + 2^{-\lambda_1\delta}\right) \\
&= 2^{-\lambda_1 c\beta - b}\frac{1 - 2^{\lambda_0\delta}}{1 - 2^{(\lambda_0+\lambda_1)\delta}} \leq 2^{-\lambda_1 c\beta - b}\frac{\lambda_0}{\lambda_0 + \lambda_1}
\end{aligned} \tag{7.59}$$

where the last inequality follows from the fact that $(1 - e^{-x})/x$ is decreasing for $x > 0$. Thus we have proved the next theorem.

Theorem 7.5 The average number of computations per branch for the stack algorithm when used to communicate over the BSC is upper-bounded by

$$C_{\text{av}} \leq \frac{\lambda_0}{\lambda_0 + \lambda_1} 2^{-\lambda_1 c\beta - b} \tag{7.60}$$

where λ_0 and λ_1 are the negative and positive roots of equations (7.33) and (7.51), respectively, and $\lambda_0 + \lambda_1 < 0$.

The factor $\lambda_0/(\lambda_0+\lambda_1)$ in the upper bound (7.60) regarded as a function of the bias B achieves its minimum for $B = R$, that is, for the Fano metric (see Problem 7.10). Hence, the following corollary is of particular interest.

Corollary 7.6 If we use the Fano metric, then the average number of computations per branch for the stack algorithm when used to communicate over the BSC is upper-bounded by

$$C_{\text{av}} \leq \frac{s}{s-1} 2^{-\frac{1}{1+s} c\beta - b} \quad \text{for } R < R_0 \tag{7.61}$$

where s is the solution of equation (7.35) and R_0 is the computational cutoff rate.

Proof: For the Fano metric the inequality

$$\lambda_0 + \lambda_1 = \frac{1-s}{1+s} < 0 \tag{7.62}$$

is satisfied if and only if $s > 1$, that is, if and only if $R < R_0$. ∎

By choosing the bias $B = R_0$ we obtain the *Gallager (symbol) metric* [Gal68] for the BSC,

$$\mu_{\text{G}} = \begin{cases} \alpha = \log(1-\epsilon) + 1 - R_0 \\ \beta = \log \epsilon + 1 - R_0 \end{cases} \tag{7.63}$$

Remark: Although the Gallager metric gives a weaker bound on the average number of computations, it is, as we will see in the sequel, important when we analyze the error probability for sequential decoding.

Since the Gallager metric is the same as the Fano metric when $R = R_0$, it follows from (7.34) that $\lambda_0 = -1/2$ for the Gallager metric (where we also have used that $s = 1$ when $R = R_0$, cf. (5.206)). For $R < R_0$ and bias $B \leq R_0$ the root $\lambda_1 < 1/2$ (see Problem 7.11).

Corollary 7.7 If we use the Gallager metric, then the average number of computations per branch for the stack algorithm when used to communicate over the BSC is upper-bounded by

$$C_{\text{av}} \leq \frac{\lambda_0}{\lambda_0 + \lambda_1} 2^{-\lambda_1 c\beta - b} \quad \text{for } R < R_0 \tag{7.64}$$

where $\lambda_0 = -1/2$, λ_1 is the positive root of the equation

$$\left(\frac{1}{2}2^{\lambda_1\alpha} + \frac{1}{2}2^{\lambda_1\beta}\right)^c = 2^{-b} \tag{7.65}$$

α and β are the Gallager symbol metrics given in (7.63), and $\lambda_0 + \lambda_1 < 0$.

Since C_{av} is finite for $R < R_0$, we obtain the following from the Markov inequality (Lemma 6.14):

Theorem 7.8 When the stack algorithm is used to communicate over the BSC, then the computational distribution for the number of computations needed to decode any correct node, C_i, $i = 1, 2, \ldots$, is upper-bounded by

$$P(C_i \geq x) \leq C_{\text{av}} x^{-1} \quad \text{for } R < R_0 \tag{7.66}$$

where C_{av} is the average number of computations per branch and R_0 is the computational cutoff rate.

Remark: For the ensemble $\mathcal{E}(b, c, \infty, 1)$ of rate $R = b/c$, infinite-memory, time-invariant convolutional codes, Zigangirov [Zig66] has shown that for the stack algorithm used with the Fano metric the sth moment of C_i, $E[C_i^s]$, is finite for $0 < s \leq 1$ if $R < G(s)/s$ and, hence, that

$$P(C_i \geq x) \leq E[C_i^s] x^{-s} \tag{7.67}$$

Furthermore, for the ensemble of general, nonlinear, infinite-memory trellis codes, Zigangirov strengthened (7.67) to

$$P(C_i \geq x) \leq O_R(1) x^{-s} \tag{7.68}$$

for

$$R < R^{(s)} = G(s)/s \tag{7.69}$$

where $0 < s < \infty$ and $O_R(1)$ depends on the rate R but not on x.

As a counterpart to the upper bound on the computational distribution (7.66) we have the following lower bound:

Theorem 7.9 When the stack algorithm is used to communicate over the BSC, then the computational distribution for the number of computations needed to decode the information b-tuple $\boldsymbol{u}_{i-1}$, C_i, $i = 1, 2, \ldots$, is lower-bounded by

$$P(C_i \geq x) \geq x^{-s+o(1)}, \quad 0 \leq R < C \tag{7.70}$$

for at least one information b-tuple $\boldsymbol{u}_{i-1}$, $i = 1, 2, \ldots$, where s is the solution of (7.35), that is, $s = E_{\mathcal{C}}^{\text{sph}}(R)/R$, and $o(1) \to 0$ when $x \to \infty$.

Proof: The proof is similar to the proof of Lemma 5.12 and Theorem 6.18. The theorem states that for at least one i, $i = 1, 2, \ldots$, and any $\varepsilon > 0$ there exists a value x_ε such that for any $x > x_\varepsilon$ we have

$$P(C_i \geq x) > x^{-(s+\varepsilon)} \tag{7.71}$$

Suppose that (7.71) does not hold. Then as a consequence there exist a rate $R = b/c$ convolutional code decoded with the stack algorithm and a certain $\varepsilon > 0$ such that for all i, $i = 1, 2, \ldots$, and any large x_ε there exists an $x > x_\varepsilon$ such that

$$P(C_i \geq x) < x^{-(s+2\varepsilon)}, \quad i = 1, 2, \ldots \tag{7.72}$$

We terminate this convolutional code (with very good computational distribution according to (7.72)) into a block code $\mathcal{B}$ of rate R (no zero tail) and block length N.

In order to decode this code, we use the stack algorithm as a (block) list decoder with list size $L = x$. Assume that the decoder operates only up to depth N/c, where the block length is

$$N = \frac{E_{\mathcal{C}}^{\text{sph}}(R) \log x}{E_{\mathcal{B}}^{\text{sph}}(\tilde{r}) R} \tag{7.73}$$

and

$$\tilde{r} = R - \frac{\log x}{N} \tag{7.74}$$

Each time the decoder reaches the depth N/c, it stores the corresponding node on the list and chooses the next node from the stack and operates according to the rules of the stack algorithm. Assume that the stack algorithm stops after making $x(2^b - 1)^{-1}$ computations. Since each computation of the stack algorithm increases the number of nodes on the stack by $(2^b - 1)$, the number of stored paths is equal to x.

Extend in an arbitrary way all paths that are shorter than N/c c-tuples. The list decoder will not make an error if the number of computations for none of the N/c information b-tuples exceeds $x/(N/c)$. Thus, using the union bound and (7.72), the probability of error for the (block) list decoder is upper-bounded by

$$P_L(\mathcal{E}) < \frac{N}{c} \left(\frac{xc}{N} \right)^{-(s+2\varepsilon)} \left(\frac{N}{c} \right)^{1+s+2\varepsilon} x^{-(s+2\varepsilon)} \tag{7.75}$$

Choose x_ε such that

$$\left(\frac{E_{\mathcal{C}}^{\text{sph}}(R) \log x_\varepsilon}{E_{\mathcal{B}}^{\text{sph}}(r) Rc} \right)^{1+s+2\varepsilon} < (x_\varepsilon)^{\varepsilon} \tag{7.76}$$

Then, for any $x > x_\varepsilon$ we have

$$P_L(\mathcal{E}) < x^{-(s+\varepsilon)} \tag{7.77}$$

in contradiction to Theorem 6.17. Hence, we conclude that inequality (7.71) must hold and the proof is complete. ∎

7.7 ERROR PROBABILITY ANALYSIS OF THE STACK ALGORITHM

In this section, we will upper-bound the probability of decoding error for the stack algorithm. Hence, we must assume that the codes have finite memory since otherwise we have zero probability of decoding errors. We consider the ensemble $\mathcal{E}(b, c, m, \infty)$ of binary, rate $R = b/c$, time-varying convolutional codes of memory m. Then we have

$$\boldsymbol{v} = \boldsymbol{u}\boldsymbol{G}_t \tag{7.78}$$

where

$$\boldsymbol{G}_t = \begin{pmatrix} G_0(t) & G_1(t+1) & \cdots & G_m(t+m) & \\ & G_0(t+1) & G_1(t+2) & \cdots & G_m(t+1+m) \\ & & \ddots & \ddots & & \ddots \end{pmatrix} \tag{7.79}$$

in which each digit in each of the matrices $G_i(t)$ for $0 \leq i \leq m$ and $t = 0, 1, \ldots$ is chosen independently and is equally likely to be 0 and 1.

Assume without loss of generality that the allzero sequence is transmitted over the BSC and that the stack algorithm with the metric μ (7.30) with a general bias $B > 0$ is used to decode the received sequence $\boldsymbol{r}$.

Since the analysis of the probability of decoding error for the stack algorithm is the same for all correct nodes, we consider only the root and the decoding of the first information b-tuple $\boldsymbol{u}_0$.

In the first incorrect subtree stemming from the root, there are $(2^b - 1)$ nodes at depth $m+1$ which have the same state as the $(m+1)$th correct node. In general, in the first incorrect subtree there are $(2^b - 1)2^{bj}$ nodes at depth $(j + m + 1)$, $j = 0, 1, \ldots$, which have the same state as the $(j + m + 1)$th correct node. We call these nodes *mergeable*.

First, we assume that the rate is less than the computational cutoff rate, that is, $0 < R < R_0$.

Consider an arbitrary mergeable node $\boldsymbol{u}'_{[0,j+m]}$, $j = 0, 1, \ldots$, in the first incorrect subtree. A necessary condition that an error burst will start at the root is that at least one of the mergeable nodes in the first incorrect subtree will be extended. Introduce the function

$$\varphi(\boldsymbol{u}'_{[0,j+m]}) = \begin{cases} 1 & \text{if the mergeable node } \boldsymbol{u}'_{[j+m]} \text{ is extended} \\ 0 & \text{else} \end{cases} \tag{7.80}$$

Then the probability that the mergeable node $\boldsymbol{u}'_{[j+m]}$ is extended is simply $E[\varphi(\boldsymbol{u}'_{[j+m]})]$. Hence, the probability that an error burst will start at the root is upper-bounded by the average number of the extended, mergeable nodes in the first incorrect subtree, that is,

$$P(\mathcal{E}_1) < \sum_{j=0}^{\infty} \sum_{\boldsymbol{u}'_{[0,j+m]} \in \mathcal{T}_1^m} E[\varphi(\boldsymbol{u}'_{[0,j+m]})] \tag{7.81}$$

where $\mathcal{T}_1^m$ is the set of mergeable nodes in the first incorrect subtree.

Consider the random walk $S'_0 = 0, S'_1, \ldots, S'_{j+m+1}$ associated with the node $\boldsymbol{u}'_{[0,j+m]}$, where

$$S'_t = \sum_{i=0}^{t-1} Z'_i \tag{7.82}$$

and $Z'_i = \mu(\boldsymbol{r}_i, \boldsymbol{v}_i)$ is the ith branch metric along the path $\boldsymbol{u}'_{[0,j+m]}$. A necessary condition for a mergeable node $\boldsymbol{u}'_{[0,j+m]}$ in the first incorrect subtree to be extended is that the random walk $S'_0, S'_1, \ldots, S'_{j+m+1}$ does not cross (from above) a barrier at $\mu_{\min}$, where $\mu_{\min}$ is the minimum metric along the correct path. Assume that $\mu_{\min} = y$ and introduce (see (7.42))

$$f_{j+m+1}(y, z) \stackrel{\text{def}}{=} P(S'_t \geq y, \quad 0 \leq t \leq j+m, \quad S'_{j+m+1} = z) \tag{7.83}$$

Then

$$E[\varphi(\boldsymbol{u}'_{[0,j+m]}) \mid \mu_{\min} = y] = \sum_{z \geq y} f_{j+m+1}(y, z) \tag{7.84}$$

Analogously to (7.81) we have

$$P(\mathcal{E}_1 \mid \mu_{\min} = y) < \sum_{j=0}^{\infty} \sum_{\boldsymbol{u}'_{[0,j+m]} \in \mathcal{T}_1^m} E\left[\varphi(\boldsymbol{u}'_{[0,j+m]}) \;\middle|\; \mu_{\min} = y\right] \tag{7.85}$$

Inserting (7.84) into (7.85) yields

$$\begin{aligned} P(\mathcal{E}_1 \mid \mu_{\min} = y) &< \sum_{j=0}^{\infty} \sum_{\boldsymbol{u}'_{[0,j+m]} \in \mathcal{T}_1^m} \sum_{z \geq y} f_{j+m+1}(y, z) \\ &= \sum_{j=0}^{\infty} (2^b - 1) 2^{bj} \sum_{z \geq y} f_{j+m+1}(y, z) \end{aligned} \tag{7.86}$$

Since in the incorrect subtree we will eventually cross the barrier we have, analogously to (7.45),

$$\sum_{z \geq y} f_{j+m+1}(y, z) = \sum_{t=j+m+2}^{\infty} \sum_{z < y} f_t(y, z) \tag{7.87}$$

Then we insert (7.87) into (7.86) and obtain

$$\begin{aligned} P(\mathcal{E}_1 \mid \mu_{\min} = y) &< \sum_{j=0}^{\infty} (2^b - 1) 2^{bj} \sum_{t=j+m+2}^{\infty} \sum_{z < y} f_t(y, z) \\ &= 2^{-bm} (2^b - 1) \sum_{j=0}^{\infty} 2^{b(j+m)} \sum_{t=j+m+2}^{\infty} \sum_{z < y} f_t(y, z) \end{aligned} \tag{7.88}$$

The right side of (7.88) can be further upper-bounded by

$$\begin{aligned} P\left(\mathcal{E}_1 \mid \mu_{\min} = y\right) &< 2^{-bm}(2^b - 1)\sum_{j=0}^{\infty} 2^{bj} \sum_{t=j+1}^{\infty} \sum_{z<y} f_t(y,z) \\ &= 2^{-bm}(2^b - 1)\sum_{t=1}^{\infty} \left(\sum_{j=0}^{t-1} 2^{bj}\right) \sum_{z<y} f_t(y,z) \\ &= 2^{-bm}(2^b - 1)\sum_{t=1}^{\infty} \frac{2^{bt}-1}{2^b-1} \sum_{z<y} f_t(y,z) \\ &< 2^{-bm} \sum_{t=1}^{\infty} 2^{bt} \sum_{z<y} f_t(y,z) \end{aligned} \tag{7.89}$$

(cf. (7.47)). Analogously to (7.53), we obtain

$$P\left(\mathcal{E}_1 \mid \mu_{\min} = y\right) < 2^{-bm} 2^{-\lambda_1(y+c\beta)} \tag{7.90}$$

where λ_1 satisfies (7.50) and β is given by (7.30) and, then, analogously to (7.57) we can show that

$$\begin{aligned} P(\mathcal{E}_1) &= \sum_{i=0}^{\infty} P\left(\mathcal{E}_1 \mid \mu_{\min} = -i\delta\right) P(\mu_{\min} = -i\delta) \\ &< 2^{-bm} 2^{-\lambda_1 c\beta} \left((1 - 2^{-\lambda_1\delta}) \left(\sum_{i=0}^{\infty} 2^{i(\lambda_0+\lambda_1)\delta} \right) + 2^{-\lambda_1\delta} \right) \end{aligned} \tag{7.91}$$

The sum in inequality (7.91) converges if $\lambda_0 + \lambda_1 < 0$, and we obtain analogously to (7.59) that

$$P(\mathcal{E}_1) < 2^{-\lambda_1 c\beta} \frac{\lambda_0}{\lambda_0 + \lambda_1} 2^{-Rmc} \tag{7.92}$$

where $\lambda_0 + \lambda_1 < 0$.

If we choose the Fano metric, then

$$P(\mathcal{E}_1) < O(1) 2^{-Rmc} \text{ for } R < R_0 \tag{7.93}$$

where

$$O(1) = 2^{-\frac{1}{1+s} c\beta} \frac{s}{s-1} \tag{7.94}$$

We notice that for the Fano metric we have obtained an upper bound on the error probability for the stack algorithm whose exponent R *decreases* linearly with the rate R! The reason is that although the Fano metric should be chosen in order to obtain a good computational performance, it is not optimal from an error probability point of view. In fact, for rates R close to zero, it is far from optimal.

Next we consider the sphere-packing region $R_0 < R < C$. Then, as always when we consider this region, we separate the event that an error burst starts at the root

node into two disjoint events corresponding to "few" ($\mathcal{F}$) and "many" ($\mathcal{M}$) errors, respectively. Then we have

$$\begin{aligned} P(\mathcal{E}_1) &= P(\mathcal{E}_1 \mid \mathcal{F})\, P(\mathcal{F}) + P(\mathcal{E}_1 \mid \mathcal{M})\, P(\mathcal{M}) \\ &\leq P(\mathcal{E}_1 \mid \mathcal{F})\, P(\mathcal{F}) + P(\mathcal{M}) \end{aligned} \tag{7.95}$$

where we have upper-bounded $P(\mathcal{E}_1 \mid \mathcal{M})$ by 1.

To state more precisely what we mean by "few" and "many" errors, we consider the minimum metric $\mu_{\min}$ along the correct path. Let those error patterns for which the metric stays above the barrier $u < 0$, that is, for which

$$\mu_{\min} > u \tag{7.96}$$

contain "few" errors and those error patterns for which the metric hits or crosses the barrier contain "many" errors. From Lemma 7.1 it follows that

$$P(\mathcal{M}) = P(\mu_{\min} \leq u) \leq 2^{-\lambda_0 u} \tag{7.97}$$

where λ_0 is the smallest root of (7.33).

The probability

$$P(\mathcal{E}_1, \mathcal{F}) = P(\mathcal{E}_1 \mid \mathcal{F}) P(\mathcal{F}) \tag{7.98}$$

is equal to the joint probability that $\mu_{\min} > u$ and that at least one of the mergeable nodes in the first incorrect subtree is extended. Then, for $\mu_{\min} = y > u$ we have (cf. (7.90))

$$P(\mathcal{E}_1 \mid \mu_{\min} = y) < 2^{-bm} 2^{-\lambda_1 (y + c\beta)} \tag{7.99}$$

where λ_1 satisfies (7.50) and β is given by (7.30). We let

$$u = -(i_0 + 1)\delta \tag{7.100}$$

$$y = -i\delta \tag{7.101}$$

and, then, analogously to (7.57), we get

$$
\begin{aligned}
P\left(\mathcal{E}_1 \mid \mathcal{F}\right) P(\mathcal{F}) &< \sum_{i=0}^{i_0} 2^{-bm} 2^{-\lambda_1(-i\delta+c\beta)} P(\mu_{\min} = -i\delta) \\
&= 2^{-bm-\lambda_1 c\beta} \left(\sum_{i=0}^{i_0} 2^{\lambda_1 i\delta} (P(\mu_{\min} \leq -i\delta) - P(\mu_{\min} \leq -(i+1)\delta)) \right) \\
&= 2^{-bm-\lambda_1 c\beta} \left(\sum_{i=0}^{i_0} 2^{\lambda_1 i\delta} P(\mu_{\min} \leq -i\delta) - \sum_{i=1}^{i_0+1} 2^{\lambda_1 (i-1)\delta} P(\mu_{\min} \leq -i\delta) \right) \\
&= 2^{-bm-\lambda_1 c\beta} \left((1 - 2^{-\lambda_1\delta}) \left(\sum_{i=0}^{i_0} 2^{\lambda_1 i\delta} P(\mu_{\min} \leq -i\delta) \right) \right. \\
&\qquad \left. + 2^{-\lambda_1\delta} P(\mu_{\min} \leq 0) - 2^{\lambda_1 i_0 \delta} P(\mu_{\min} \leq -(i_0+1)\delta) \right) \\
&< 2^{-bm-\lambda_1 c\beta} \left((1 - 2^{-\lambda_1\delta}) \left(\sum_{i=0}^{i_0} 2^{\lambda_1 i\delta} 2^{\lambda_0 i\delta} \right) + 2^{-\lambda_1\delta} \right) \qquad (7.102)
\end{aligned}
$$

where the last inequality follows from Lemma 7.1.

Assuming that $\lambda_0 + \lambda_1 \neq 0$, we have

$$
\begin{aligned}
P\left(\mathcal{E}_1 \mid \mathcal{F}\right) P(\mathcal{F}) &< 2^{-bm-\lambda_1 c\beta} \left(\frac{(1 - 2^{-\lambda_1\delta})(2^{(\lambda_0+\lambda_1)(i_0+1)\delta} - 1)}{2^{(\lambda_0+\lambda_1)\delta} - 1} + 2^{-\lambda_1\delta} \right) \\
&= 2^{-bm-\lambda_1 c\beta} \frac{(1 - 2^{-\lambda_1\delta}) 2^{-(\lambda_0+\lambda_1)u} + 2^{\lambda_0\delta} - 1}{2^{(\lambda_0+\lambda_1)\delta} - 1} \qquad (7.103)
\end{aligned}
$$

By inserting (7.103) and (7.97) into (7.95), we obtain

$$
P(\mathcal{E}_1) < 2^{-bm-\lambda_1 c\beta} \frac{(1 - 2^{-\lambda_1\delta}) 2^{-(\lambda_0+\lambda_1)u} + 2^{\lambda_0\delta} - 1}{2^{(\lambda_0+\lambda_1)\delta} - 1} + 2^{-\lambda_0 u} \qquad (7.104)
$$

For the Fano metric we conclude from (7.34) and Example B.2 that

$$
\lambda_0 = -\frac{s}{1+s} \qquad (7.105)
$$

and

$$
\lambda_1 = \frac{1}{1+s} \qquad (7.106)
$$

where s is the solution of

$$
G(s) = sR \qquad (7.107)
$$

and $G(s)$ is the Gallager function (5.98). Hence,

$$
\lambda_0 + \lambda_1 = \frac{1-s}{1+s} > 0 \qquad (7.108)
$$

if and only if $s < 1$, that is, if and only if $R > R_0$.

Since $\lambda_0 < 0$ and $\lambda_0 + \lambda_1 > 0$, it follows that

$$\frac{2^{\lambda_0 \delta} - 1}{2^{(\lambda_0+\lambda_1)\delta} - 1} < 0 \tag{7.109}$$

Furthermore,

$$\begin{aligned} \frac{(1 - 2^{-\lambda_1 \delta})}{2^{(\lambda_0+\lambda_1)\delta} - 1} &< \left(\frac{1 - 2^{-\lambda_1 \delta}}{\lambda_1 \delta}\right) \left(\frac{(\lambda_0 + \lambda_1)\delta}{1 - 2^{-(\lambda_0+\lambda_1)\delta}}\right) \left(\frac{\lambda_1}{\lambda_0 + \lambda_1}\right) \\ &< \frac{\lambda_1}{\lambda_0 + \lambda_1} \end{aligned} \tag{7.110}$$

where in order to obtain the last inequality we have used that $(1-e^{-x})/x$ is decreasing with increasing x. Thus, we can further upper-bound (7.104):

$$P(\mathcal{E}_1) < \frac{\lambda_1}{\lambda_0 + \lambda_1} 2^{-bm - \lambda_1 c\beta} 2^{-(\lambda_0+\lambda_1)u} + 2^{-\lambda_0 u} \tag{7.111}$$

In order to transform our bound (7.111) into a more illuminative form, when $R > R_0$ we use the optimal value of the barrier, viz.,

$$u = -\frac{1 + s}{s} G(s)mc \tag{7.112}$$

Next we insert (7.105), (7.106), (7.108), and (7.112) into (7.111) and obtain

$$\begin{aligned} P(\mathcal{E}_1) &< \frac{1}{1 - s} 2^{-\frac{1}{1+s} c\beta} 2^{\frac{1-s}{s} G(s)mc - bm} + 2^{-G(s)mc} \\ &= \left(\frac{1}{1 - s} 2^{-\frac{1}{1+s} c\beta} 2^{\frac{G(s)}{s} mc - Rmc} + 1\right) 2^{-G(s)mc} \\ &= \left(\frac{1}{1 - s} 2^{-\frac{1}{1+s} c\beta} + 1\right) 2^{-G(s)mc} \end{aligned} \tag{7.113}$$

where the last inequality follows from (7.107). Finally, we obtain

$$P(\mathcal{E}_1) < O(1) 2^{-G(s)mc} \quad \text{for } R > R_0 \tag{7.114}$$

where

$$O(1) = \frac{1}{1 - s} 2^{-\frac{1}{1+s} c\beta} + 1 > 0 \tag{7.115}$$

Remark: It is somewhat surprising that the constant factor in the upper bound (7.114), that is, $O(1)$, is better than the corresponding factor for maximum-likelihood decoding (cf. Theorem 5.19). The explanation is that here we have used a more advanced bounding procedure.

Next we consider the special case $R = R_0$, which implies that $s = 1$. Hence, assuming the Fano metric, we have

$$\lambda_0 = -\lambda_1 = -1/2 \tag{7.116}$$

That is, we have $\lambda_0 + \lambda_1 = 0$. We return to the last inequality of (7.102) and obtain

$$\begin{aligned} P(\mathcal{E}_1 \mid \mathcal{F})\, P(\mathcal{F}) &< 2^{-bm-\lambda_1 c\beta} \left((1 - 2^{-\delta/2}) \sum_{i=0}^{i_0} 1 + 2^{-\delta/2} \right) \\ &= 2^{-bm-c\beta/2}((1 - 2^{-\delta/2})(i_0 + 1) + 2^{-\delta/2}) \\ &< 2^{-bm-c\beta/2}(1 + (i_0 + 1)\delta/2) \end{aligned} \tag{7.117}$$

By inserting (7.117) and (7.97) into (7.95), we obtain

$$P(\mathcal{E}_1) < 2^{-bm-c\beta/2}(1 - u/2) + 2^{u/2} \tag{7.118}$$

Again we use the optimal value of the barrier, viz.,

$$u = -2R_0 mc \tag{7.119}$$

and obtain

$$\begin{aligned} P(\mathcal{E}_1) &< 2^{-R_0 mc - c\beta/2}(1 + R_0 mc) + 2^{-R_0 mc} \\ &< \left(2^{-c\beta/2}(1 + R_0 mc) + 1 \right) 2^{-R_0 mc} \end{aligned} \tag{7.120}$$

Finally, we obtain

$$P(\mathcal{E}_1) < O(m) 2^{-R_0 mc} \quad \text{for } R = R_0 \tag{7.121}$$

where

$$O(m) = 2^{-c\beta/2}(1 + R_0 mc) + 1 \tag{7.122}$$

We summarize our results in the following:

Theorem 7.10 If we use the Fano metric, then for the ensemble of rate $R = b/c$, time-varying convolutional codes of memory m the burst error probability for the stack algorithm when used to communicate over the BSC is upper-bounded by

$$P(\mathcal{E}_1) < \begin{cases} O(1)2^{-Rmc}, & 0 \le R < R_0 \\ O(m)2^{-R_0 mc}, & R = R_0 \\ O(1)2^{-G(s)mc}, & R_0 < R < C \end{cases} \tag{7.123}$$

where s is the solution of (7.107), $G(s)$ is the Gallager function given by (5.98), and R_0 is the computational cutoff rate.

For rates $R_0 \le R < C$ we have the same exponents in our upper bounds on the error probability for sequential decoding as we have for maximum-likelihood (ML) decoding, but in the region $0 \le R < R_0$ the bound for sequential decoding is much worse than that for ML decoding. This is because from the *error probability* point of view the Fano metric is not the best choice for rates $R < R_0$. By replacing the Fano metric by the Gallager metric for rates $R < R_0$ we can obtain a stronger bound

on the error probability at the expense of an increased number of computations as follows:

By replacing $b\ (= Rc)$ by

$$b_0 = R_0 c \tag{7.124}$$

in (7.86), we can obtain a strengthening for rates $R < R_0$. As before, λ_0 is the smallest root of (7.33) but now λ_1 is the positive root of

$$\frac{1}{2}2^{\lambda_1 \alpha} + \frac{1}{2}2^{\lambda_1 \beta} = 2^{-R_0} \tag{7.125}$$

For the Gallager metric it is easily verified that

$$\lambda_0 = -\frac{1}{2} \tag{7.126}$$

and

$$\lambda_1 = \frac{1}{2} \tag{7.127}$$

Since $\lambda_0 + \lambda_1 = 0$ we can use the Gallager metric and repeat the derivation of (7.121) and obtain the following:

Theorem 7.11 If we use the Gallager metric, then for the ensemble of rate $R = b/c$, time-varying convolutional codes of memory m the burst error probability for the stack algorithm when used to communicate over the BSC is upper-bounded by

$$P(\mathcal{E}_1) < \begin{cases} O(m)2^{-R_0 mc}, & 0 \le R \le R_0 \\ O(1)2^{-G(s)mc}, & R_0 < R < C \end{cases} \tag{7.128}$$

where s is the solution of (7.107), $G(s)$ is the Gallager function given in (5.98), and R_0 is the computational cutoff rate.

Hitherto we have assumed that the sequential decoder had no *back-search limit*, that is, the decoder could, if required, back up as far as to the root of the code tree. Now we assume (cf. Section 5.5) that we have a finite back-search limit τ that is an excursion further back than τ branches from the foremost node is prohibited. For simplicity we will restrict our analysis to the case $\tau = m + 1$, where m is the encoder memory.

Consider the ensemble $\mathcal{E}(b, c, m, 1)$ of binary, rate $R = b/c$, time-invariant convolutional codes of memory m. In the first incorrect subtree there are $(2^b - 1)2^{b(\tau-1)}$ nodes at depth τ. A necessary condition that the subpath corresponding to an error burst will start at the root is that it will pass one of these nodes at depth τ, that is, that the metric of such a node is not less than the minimum metric along the correct path. In order to minimize the probability that an error burst starts at the root when we have the finite back-search limit $\tau = m + 1$, $P(\mathcal{E}_1^{\text{fbs}})$, we need a metric different from the Fano and Gallager metrics, viz., the *Zigangirov metric* for the BSC:

$$\mu_{\text{Z}} = \begin{cases} \alpha = \log(1 - \epsilon) + 1 - G(s_z)/s_z \\ \beta = \log \epsilon + 1 - G(s_z)/s_z \end{cases} \tag{7.129}$$

where s_z, $0 < s_z \leq 1$, is the solution of

$$G'(s) = R \tag{7.130}$$

if this solution is upper-bounded by 1 and where $s_z = 1$ if the solution of (7.130) is greater than 1. As before, $G(s)$ is the Gallager function (5.98). (The value of s_z maximizes the reliability function

$$E_{\mathcal{B}}(R) = G(s) - sR \quad \text{for } R_{\text{crit}} \leq R < C \tag{7.131}$$

and the pair $(E_{\mathcal{B}}(R), G'(s_z))$ defines parametrically the *sphere-packing* bound for block codes in this region.) For the BSC it is easily verified that

$$s_z = \frac{\log \frac{1-\epsilon}{\epsilon}}{\log \frac{1-\rho}{\rho}} - 1 \quad \text{for } R_{\text{crit}} \leq R < C \tag{7.132}$$

where ρ is the Gilbert-Varshamov parameter with respect to the rate R, that is,

$$\rho = h^{-1}(1 - R) \tag{7.133}$$

The rate corresponding to $s_z = 1$ is the critical rate R_{crit} (5.27).

For rates $R \leq R_{\text{crit}}$ the Zigangirov metric coincides with the Gallager metric.

It is easily verified that for the Zigangirov metric the negative root λ_0 of (7.33) is

$$\lambda_0 = -\frac{s_z}{1 + s_z} \tag{7.134}$$

Thus, $P(\mathcal{M})$ satisfies (7.97) when λ_0 is given by (7.134).

Remark: For rates $R_{\text{crit}} \leq R < C$, the Zigangirov metric (7.129) can be written as

$$\begin{cases} \alpha = \frac{1+s_z}{s_z} \log \frac{(1-\epsilon)}{1-\rho} \\ \beta = \frac{1+s_z}{s_z} \log \frac{\epsilon}{\rho} \end{cases} \tag{7.135}$$

which can be scaled to

$$\begin{cases} \alpha' = \log \frac{1-\epsilon}{1-\rho} \\ \beta' = \log \frac{\epsilon}{\rho} \end{cases} \tag{7.136}$$

Analogously, for rates $0 \leq R < R_{\text{crit}}$ we obtain

$$\begin{cases} \alpha' = \log \frac{1-\epsilon}{1-\rho_{\text{crit}}} \\ \beta' = \log \frac{\epsilon}{\rho_{\text{crit}}} \end{cases} \tag{7.137}$$

where ρ_{crit} is the Gilbert-Varshamov parameter with respect to the rate R_{crit}, viz.,

$$\rho_{\text{crit}} = h^{-1}(1 - R_{\text{crit}}) \tag{7.138}$$

We have $\lambda_1 = 1/2$ and

$$\lambda_1 = \frac{1}{1+s_z} < 1/2 \tag{7.139}$$

for rates $0 < R \le R_{\text{crit}}$ and $R_{\text{crit}} < R < C$, respectively, and it is easily verified that

$$\frac{1}{2}2^{\lambda_1\alpha} + \frac{1}{2}2^{\lambda_1\beta} = 2^{-G(s_z)/s_z} \tag{7.140}$$

Consider the rates $R_{\text{crit}} < R < C$. The conditional probability that at least one of the nodes at depth $\tau = m+1$ in the first incorrect subtree is visited given that we have few errors, that is, $P(\mathcal{E}_1^{\text{fbs}} \mid \mathcal{F})$, is upper-bounded by the conditional expectation of the number of visited nodes at depth $m+1$ in the first incorrect subtree given the event $\mathcal{F}$. Then we obtain (for $\tau = m+1$)

$$\begin{aligned}
P\left(\mathcal{E}_1^{\text{fbs}} \mid \mu_{\min} = y\right) &\le \sum_{z \ge y} (2^b - 1)2^{bm} f_{m+1}(y,z) \\
&\le 2^{b(m+1)} \sum_{z \ge y} P(\mu_{m+1} = z) 2^{\lambda_1(z-y)} \\
&< 2^{b(m+1)} \left(\sum_{\text{all } z} P(\mu_{m+1} = z) 2^{\lambda_1 z} \right) 2^{-\lambda_1 y} \\
&= 2^{b(m+1)} \left(\frac{1}{2}2^{\lambda_1\alpha} + \frac{1}{2}2^{\lambda_1\beta} \right)^{(m+1)c} 2^{-\lambda_1 y}
\end{aligned} \tag{7.141}$$

where $\lambda_1 > 0$. Next, we let $u = -(i_0+1)\delta$, $y = -i\delta$, and assume that $\lambda_0 + \lambda_1 > 0$, that is, $R > R_{\text{crit}}$. Then analogously to (7.102) and (7.103) we can show that

$$\begin{aligned}
P\left(\mathcal{E}_1^{\text{fbs}} \mid \mathcal{F}\right) P(\mathcal{F}) &= \sum_{i=0}^{i_0} P\left(\mathcal{E}_1^{\text{fbs}} \mid \mu_{\min} = -i\delta\right) P(\mu_{\min} = -i\delta) \\
&< 2^{b(m+1)} \left(\frac{1}{2}2^{\lambda_1\alpha} + \frac{1}{2}2^{\lambda_1\beta} \right)^{(m+1)c} \\
&\quad \times \left(\frac{(1 - 2^{-\lambda_1\delta})2^{(\lambda_0+\lambda_1)(i_0+1)\delta} + 2^{\lambda_0\delta} - 1}{2^{(\lambda_0+\lambda_1)\delta} - 1} \right) \\
&= 2^{b(m+1)} \left(\frac{1}{2}2^{\lambda_1\alpha} + \frac{1}{2}2^{\lambda_1\beta} \right)^{(m+1)c} \\
&\quad \times \frac{(1 - 2^{-\lambda_1\delta})2^{-(\lambda_0+\lambda_1)u} + 2^{\lambda_0\delta} - 1}{2^{(\lambda_0+\lambda_1)\delta} - 1}
\end{aligned} \tag{7.142}$$

Analogously to (7.104) we obtain

$$\begin{aligned}
P(\mathcal{E}_1^{\text{fbs}}) &< 2^{b(m+1)} \left(\frac{1}{2}2^{\lambda_1\alpha} + 2^{\lambda_1\beta} \right)^{(m+1)c} \\
&\quad \times \left(\frac{(1 - 2^{-\lambda_1\delta})2^{-(\lambda_0+\lambda_1)u} + 2^{\lambda_0\delta} - 1}{2^{(\lambda_0+\lambda_1)\delta}} \right) + 2^{-\lambda_0 u}
\end{aligned} \tag{7.143}$$

Using (7.109), (7.110), and (7.140), it follows from (7.143) that

$$P(\mathcal{E}_1^{\text{fbs}}) < \frac{\lambda_1}{\lambda_0+\lambda_1} 2^{b(m+1)} 2^{-\frac{G(s_z)}{s_z}(m+1)c} 2^{-(\lambda_0+\lambda_1)u} + 2^{-\lambda_0 u} \tag{7.144}$$

We choose the barrier to be

$$u = -\frac{1+s_z}{s_z}(G(s_z) - s_z R)(m+1)c \tag{7.145}$$

insert (7.134), (7.139), and (7.145) into (7.144), and obtain

$$\begin{aligned} P(\mathcal{E}_1^{\text{fbs}}) &< \frac{\lambda_1}{\lambda_0+\lambda_1} 2^{b(m+1)} 2^{-\frac{G(s_z)}{s_z}(m+1)c} 2^{\frac{1-s_z}{s_z}(G(s_z)-s_z R)(m+1)c} \\ &\quad + 2^{-(G(s_z)-s_z R)(m+1)c} = \frac{\lambda_0+2\lambda_1}{\lambda_0+\lambda_1} 2^{-(G(s_z)-s_z R)(m+1)c} \end{aligned} \tag{7.146}$$

Finally, we have

$$P(\mathcal{E}_1^{\text{fbs}}) < O(1) 2^{-(G(s_z)-s_z R)(m+1)c} \quad \text{for } R > R_{\text{crit}} \tag{7.147}$$

and

$$O(1) = \frac{\lambda_0+2\lambda_1}{\lambda_0+\lambda_1} = \frac{2-s_z}{1-s_z} \tag{7.148}$$

Next, we consider the special case when $R = R_{\text{crit}}$, which implies that $s_z = 1$. Hence, assuming the Zigangirov metric we have

$$\lambda_0 = -\lambda_1 = -1/2 \tag{7.149}$$

Then, analogously to (7.117) and (7.118), we can show that for the barrier

$$u = -2R_0(m+1)c \tag{7.150}$$

we have

$$P(\mathcal{E}_1^{\text{fbs}}) < O(m) 2^{-(R_0-R)(m+1)c} \quad \text{for } R = R_{\text{crit}} \tag{7.151}$$

where

$$O(m) = (R_0(m+1)c + 1) + 1 \tag{7.152}$$

Finally, to make the analysis complete, we consider rates $R < R_{\text{crit}}$. Then the Zigangirov metric coincides with the Gallager metric, and analogously to (7.128) we obtain

$$P(\mathcal{E}_1^{\text{fbs}}) < O(m) 2^{-(R_0-R)(m+1)c}, \quad 0 \le R < R_{\text{crit}} \tag{7.153}$$

where $O(m)$ is given by (7.152).

We summarize our results in the following (cf. Theorem 5.24):

Theorem 7.12 If we use the Zigangirov metric, then for the ensemble of rate $R = b/c$, time-invariant convolutional codes of memory m the burst error probability for

the stack algorithm with back-search limit $m+1$ when used to communicate over the BSC is upper-bounded by

$$P(\mathcal{E}_1^{\text{fbs}}) < \begin{cases} O(m)2^{-(R_0-R)(m+1)c}, & 0 \le R \le R_{\text{crit}} \\ O(1)2^{-(G(s_z)-s_zR)(m+1)c}, & R_{\text{crit}} < R < C \end{cases} \tag{7.154}$$

where s_z is given by (7.132), $G(s_z)$ is the Gallager function given by (5.98), R_0 is the computational cutoff rate, and R_{crit} is the critical rate.

Remark: If we use a tail that is shorter than bm the protection of the last information symbols will be weakened [Joh77b].

Remark: The sequential decoder will perform about as well with a rate $R = b/c$ systematic encoder of memory $mc/(c-b)$ as with a rate $R = b/c$ nonsystematic encoder of memory m. Due to the systematicity, the first b symbols on each branch in the tail, which is used to terminate the encoded sequence, are zeros. The zeros are known beforehand and can be omitted before transmission. Hence, the two encoders require the same allotted space for transmission of their corresponding tails. The factor $c/(c-b)$ by which the memories differ is the same factor by which Heller's upper bounds on the free distance differ when systematic and nonsystematic encoders are used.

7.8 ANALYSIS OF THE FANO ALGORITHM

Our analysis of the Fano algorithm is based on the results we obtained for the stack algorithm. First we analyze the computational behavior and consider, as for the stack algorithm, the ensemble $\mathcal{E}(b, c, \infty, 1)$ of infinite-memory, time-invariant, rate $R = b/c$ convolutional codes. We need the following properties of the Fano algorithm. They follow immediately from the description of the algorithm.

Property F1: A node $\boldsymbol{u}_{[0,t)}$ is extended by the Fano algorithm only if its metric $\mu(\boldsymbol{r}_{[0,t)}, \boldsymbol{v}_{[0,t)})$ is not less than the threshold T.

Property F2: Let $\mu_{\min}$ denote the minimum metric along the correct path. The threshold T is always greater than $\mu_{\min} - \Delta$, where Δ is the stepsize, that is,

$$T_{\min} \stackrel{\text{def}}{=} \min\{T\} > \mu_{\min} - \Delta \tag{7.155}$$

Property F3: For any value of the threshold T, each node can be visited at most once.

From Properties F1 and F3 it follows that given $T_{\min} = y$, the number of visits to any node with metric $\mu = z$, where $z \ge y$, is upper-bounded by

$$\left\lceil \frac{z-y}{\Delta} \right\rceil < \frac{z-y}{\Delta} + 1 \tag{7.156}$$

Since the root has metric 0, it can be visited at most $-y/\Delta + 1$ times. Analogously to (7.44), we can show that the conditional average of the number of computations needed to decode the first correct node, given $T_{\min} = y$, is upper-bounded by

$$\begin{aligned} E\left[C_1 \mid T_{\min} = y\right] &< (-y/\Delta + 1) + \sum_{j=1}^{\infty} (2^b - 1) 2^{b(j-1)} \sum_{z \geq y} \left(\frac{z-y}{\Delta} + 1 \right) f_j(y,z) \\ &= -y/\Delta + \left(1 + \sum_{j=1}^{\infty} (2^b - 1) 2^{b(j-1)} \sum_{z \geq y} f_j(y,z) \right) \\ &\quad - \Delta^{-1} \sum_{j=1}^{\infty} (2^b - 1) 2^{b(j-1)} \sum_{z \geq y} (y-z) f_j(y,z) \end{aligned} \tag{7.157}$$

In order to upper-bound the right side of (7.157), we use the following inequality from Appendix B (see Corollary B.5):

$$\sum_{t=1}^{\infty} t \sum_{x<y} f_t(y,x) \geq \frac{y \ln 2}{g'_\lambda(0)} \tag{7.158}$$

where

$$g'_\lambda(0) = \left(\frac{1}{2}\alpha + \frac{1}{2}\beta \right) c \ln 2 = E[Z'_i] \ln 2 \tag{7.159}$$

and Z'_i is the ith branch metric $\mu(\boldsymbol{r}_i, \boldsymbol{v}'_i)$ along the incorrect path $\boldsymbol{u}'_{[0,j)}$.

Combining (7.157) and (7.158) yields

$$\begin{aligned} E\left[C_1 \mid T_{\min} = y\right] < -y/\Delta &+ \left(1 + \sum_{j=1}^{\infty} (2^b - 1) 2^{b(j-1)} \sum_{z \geq y} f_j(y,z) \right) \\ &- E[Z'_i] \Delta^{-1} \sum_{j=1}^{\infty} (2^b - 1) 2^{b(j-1)} \\ &\times \sum_{z \geq y} \sum_{t=1}^{\infty} t \sum_{x<y-z} f_t(y-z,x) f_j(y,z) \end{aligned} \tag{7.160}$$

From (7.44) and (7.52) we conclude that

$$1 + \sum_{j=1}^{\infty} (2^b - 1) 2^{b(j-1)} \sum_{z \geq y} f_j(y,z) \leq 2^{-b - \lambda_1(y + c\beta)} \tag{7.161}$$

The sum

$$\sum_{x<y-z} f_t(y-z,x)$$

is the probability that the random walk, given that it started from level z, where $z \geq y$, crosses the barrier y for the first time at the depth t. Then,

$$\sum_{z \geq y} f_j(y,z) \sum_{x<y-z} f_t(y-z,x)$$

is the probability that the random walk, given that it started from level 0, crosses the barrier y for the first time at the depth $j+t$. This latter probability can also be expressed as

$$\sum_{z<y} f_{j+t}(y,z)$$

Hence, we have

$$\sum_{z\geq y} f_j(y,z) \sum_{x<y-z} f_t(y-z,x) = \sum_{z<y} f_{j+t}(y,z) \tag{7.162}$$

Using (7.161) and (7.162) we can upper-bound (7.160):

$$\begin{aligned}
E[C_1 \mid T_{\min}=y] &< -y/\Delta + 2^{-b-\lambda_1(y+c\beta)} \\
&\quad -E[Z_i']\Delta^{-1}\sum_{j=1}^{\infty}(2^b-1)2^{b(j-1)}\sum_{t=1}^{\infty} t\sum_{z<y} f_{j+t}(y,z) \\
&= -y/\Delta + 2^{-b-\lambda_1(y+c\beta)} \\
&\quad -E[Z_i']\Delta^{-1}(1-2^{-b})\sum_{k=1}^{\infty}\sum_{j=1}^{k}(k-j)2^{bj}\sum_{z<y} f_k(y,z) \\
&= -y/\Delta + 2^{-b-\lambda_1(y+c\beta)} - E[Z_i']\Delta^{-1}(1-2^{-b}) \\
&\quad \times \sum_{k=1}^{\infty}\left(k2^b\frac{2^{bk}-1}{2^b-1} - \frac{k2^{b(k+2)} - 2^{b(k+1)}(k+1)+2^b}{(2^b-1)^2}\right)\sum_{z<y} f_k(y,z) \\
&= -y/\Delta + 2^{-b-\lambda_1(y+c\beta)} - E[Z_i']\Delta^{-1}(1-2^{-b}) \\
&\quad \times \sum_{k=1}^{\infty}\left(-\frac{k2^b}{2^b-1} + \frac{2^{b(k+1)}}{(2^b-1)^2} - \frac{2^b}{(2^b-1)^2}\right)\sum_{z<y} f_k(y,z)
\end{aligned} \tag{7.163}$$

In order to further upper-bound (7.163) we use the following two simplifications: From (7.158) it follows that

$$\sum_{k=1}^{\infty} k\sum_{z<y} f_k(y,z) \geq \frac{y}{E[Z_i']} \tag{7.164}$$

and from (7.50) it follows that

$$\sum_{k=1}^{\infty} 2^{bk}\sum_{z<y} f_k(y,z) \leq 2^{-\lambda_1(y+c\beta)} \tag{7.165}$$

Furthermore, from (7.45) we have

$$\sum_{k=1}^{\infty}\sum_{z<y} f_k(y,z) = 1 \tag{7.166}$$

Inserting (7.164), (7.165), and (7.166) into (7.163) yields

$$\begin{aligned}
E\left[C_1 \mid T_{\min} = y\right] &< -y/\Delta + 2^{-b-\lambda_1(y+c\beta)} \\
&\quad -E[Z_i']\Delta^{-1}(1-2^{-b})\left(-\frac{y2^b}{(2^b-1)E[Z_i']} + \frac{2^b 2^{-\lambda_1(y+c\beta)}}{(2^b-1)^2} - \frac{2^b}{(2^b-1)^2}\right) \\
&= -y/\Delta + 2^{-b-\lambda_1(y+c\beta)} + y/\Delta - \frac{E[Z_i']2^{-\lambda_1(y+c\beta)}}{\Delta(2^b-1)} + \frac{E[Z_i']}{\Delta(2^b-1)} \\
&< 2^{-b-\lambda_1(y+c\beta)} - \frac{E[Z_i']2^{-\lambda_1(y+c\beta)}}{\Delta(2^b-1)} \\
&= \left(2^{-b} - \frac{E[Z_i']}{\Delta(2^b-1)}\right) 2^{-\lambda_1 c\beta} 2^{-\lambda_1 y} \qquad (7.167)
\end{aligned}$$

By combining (7.155) and (7.167), we obtain

$$\begin{aligned}
E\left[C_1 \mid \mu_{\min} = y\right] &< \left(2^{-b} - \frac{E[Z_i']}{(2^b-1)\Delta}\right) 2^{-\lambda_1 c\beta + \lambda_1 \Delta - \lambda_1 y} \\
&= \left(1 - \frac{E[Z_i']}{(1-2^{-b})\Delta}\right) 2^{\lambda_1 \Delta} 2^{-b-\lambda_1 c\beta} 2^{-\lambda_1 y} \qquad (7.168)
\end{aligned}$$

The function $2^{\lambda_1 \Delta}/\Delta$ attains its minimum when

$$\Delta = \frac{1}{\lambda_1 \ln 2} \qquad (7.169)$$

Since in a practical situation the second term in the parentheses on the right-hand side of (7.168) is essentially greater than 1, we use (7.169) as a good approximation of the optimal value of the stepsize. Inserting (7.169) into (7.168) gives

$$E\left[C_1 \mid \mu_{\min} = y\right] < C_{\mathrm{F}} 2^{-\lambda_1 y} \qquad (7.170)$$

where

$$C_{\mathrm{F}} = \left(1 - \frac{(\alpha+\beta)\lambda_1 \ln 2}{2(1-2^{-b})}\right) e 2^{-b-\lambda_1 c\beta} \qquad (7.171)$$

Without any essential loss of accuracy we assume that the metric values can be written

$$\alpha = \alpha_0 \delta \qquad (7.172)$$

and

$$\beta = \beta_0 \delta \qquad (7.173)$$

where α_0 and β_0 are integers and $\delta > 0$. Then $\mu_{\min} \in \{0, -\delta, -2\delta, -3\delta, \ldots\}$ and we have

$$\begin{aligned} C_{\text{av}} = E[C_1] &= \sum_{i=0}^{\infty} E\left[C_1 \mid \mu_{\min} = -i\delta\right] P(\mu_{\min} = -i\delta) \\ &< \sum_{i=0}^{\infty} C_{\text{F}} 2^{i\lambda_1\delta} (P(\mu_{\min} \le -i\delta) - P(\mu_{\min} \le -(i+1)\delta)) \\ &= C_{\text{F}} \left(\sum_{i=0}^{\infty} 2^{i\lambda_1\delta} P(\mu_{\min} \le -i\delta) - \sum_{i=1}^{\infty} 2^{(i-1)\lambda_1\delta} P(\mu_{\min} \le -i\delta) \right) \\ &= C_{\text{F}} \left((1 - 2^{-\lambda_1\delta}) \left(\sum_{i=0}^{\infty} 2^{i\lambda_1\delta} P(\mu_{\min} \le -i\delta) \right) + 2^{-\lambda_1\delta} \right) \end{aligned} \tag{7.174}$$

Now we use Lemma 7.1 and obtain

$$C_{\text{av}} < C_{\text{F}} \left((1 - 2^{-\lambda_1\delta}) \left(\sum_{i=0}^{\infty} 2^{i\lambda_1\delta} 2^{i\lambda_0\delta} \right) + 2^{-\lambda_1\delta} \right) \tag{7.175}$$

The sum converges if $\lambda_0 + \lambda_1 < 0$ and, finally, we have

$$\begin{aligned} C_{\text{av}} &< C_{\text{F}} \left(\frac{1 - 2^{-\lambda_1\delta}}{1 - 2^{(\lambda_0+\lambda_1)\delta}} + 2^{-\lambda_1\delta} \right) \\ &= C_{\text{F}} \frac{1 - 2^{\lambda_0\delta}}{1 - 2^{(\lambda_0+\lambda_1)\delta}} \le C_{\text{F}} \frac{\lambda_0}{\lambda_0 + \lambda_1} \end{aligned} \tag{7.176}$$

where the last inequality follows from the fact that $(1 - e^{-x})/x$ is decreasing for all x. Thus, we have proved the following

Theorem 7.13 The average number of computations per branch for the Fano algorithm when used to communicate over the BSC is upper-bounded by

$$C_{\text{av}} < C_{\text{F}} \frac{\lambda_0}{\lambda_0 + \lambda_1} \tag{7.177}$$

where λ_0 and λ_1 are the negative and positive roots of equations (7.33) and (7.51), respectively, such that $\lambda_0 + \lambda_1 < 0$ and C_{F} is given in (7.171).

From (7.105) and (7.108), we have the next corollary.

Corollary 7.14 If we use the Fano metric, the average number of computations per branch for the Fano algorithm when used to communicate over the BSC is upper-bounded by

$$C_{\text{av}} < C_{\text{F}} \frac{s}{s-1} \quad \text{for } R < R_0 \tag{7.178}$$

where s is the solution of (7.35) and R_0 is the computational cutoff rate.

Also for the Fano algorithm we can use the Markov inequality (Lemma 6.14) and obtain the following:

Theorem 7.15 When the Fano algorithm is used to communicate over the BSC then the computational distribution for the number of computations needed to decode any correct node, C_i, $i = 1, 2, \ldots$, is upper-bounded by

$$P(C_i \geq x) \leq C_{\text{av}} x^{-1} \quad \text{for } R < R_0 \tag{7.179}$$

where C_{av} is the average number of computations per branch and R_0 is the computational cutoff rate.

Remark: For the ensemble $\mathcal{E}(b, c, \infty, 1)$ of rate $R = b/c$, infinite-memory, time-invariant convolutional codes Falconer [Fal66] has shown that for the Fano algorithm the sth moment of C_i, $E[C_i^s]$, is finite for $0 < s \leq 1$ if $R < G(s)/s$ and, hence, that

$$P(C_i \geq x) \leq E[C_i^s] x^{-s} \tag{7.180}$$

For the ensemble of general, nonlinear, infinite-memory trellis codes Savage [Sav66] strengthened (7.180) to

$$P(C_i \geq x) \leq O_R(1) x^{-s} \tag{7.181}$$

for

$$R < R^{(s)} = G(s)/s \tag{7.182}$$

where s is a strictly positive integer and $O_R(1)$ depends on the rate R but not on x.

The lower bound on the computational distribution that we derived in Section 7.6 (Theorem 7.9) is also valid for the Fano algorithm.

The error probability analysis of the Fano algorithm is quite similar to the corresponding analysis of the stack algorithm. The only difference is in the necessary condition for a mergeable node $\boldsymbol{u}'_{[0,j+m]}$ in the incorrect subtree to be extended. For the Fano algorithm, the condition is that the random walk $S'_0, S'_1, \ldots, S'_{j+m+1}$ does not cross a barrier at $T_{\min}$, while for the stack algorithm the barrier is at $\mu_{\min}$. Thus, we use inequality (7.155) and instead of (7.92) we obtain

$$P(\mathcal{E}_1) < 2^{-\lambda_1 c\beta + \lambda_1 \Delta} \frac{\lambda_0}{\lambda_0 + \lambda_1} 2^{-Rmc} \tag{7.183}$$

where $\lambda_0 + \lambda_1 < 0$ and $R < R_0$.

Analogously to Theorem 7.10 we can prove the next theorem.

Theorem 7.16 If we use the Fano metric, then for the ensemble of rate $R = b/c$, time-varying convolutional codes of memory m the burst error probability for the Fano algorithm when used to communicate over the BSC is upper-bounded by

$$P(\mathcal{E}_1) < \begin{cases} O(1)2^{-Rmc} & 0 \leq R < R_0 \\ O(m)2^{-R_0 mc} & R = R_0 \\ O(1)2^{-G(s)mc} & R_0 < R < C \end{cases} \tag{7.184}$$

where s is the solution of (7.107), $G(s)$ is the Gallager function given by (5.98), and R_0 is the computational cutoff rate.

For the Gallager metric we can show the following counterpart to Theorem 7.11:

Theorem 7.17 If we use the Gallager metric, then for the ensemble of rate $R = b/c$, time-varying convolutional codes of memory m the burst error probability for the Fano algorithm when used to communicate over the BSC is upper-bounded by

$$P(\mathcal{E}_1) < \begin{cases} O(m)2^{-R_0 mc} & 0 \le R \le R_0 \\ O(1)2^{-G(s)mc} & R_0 < R < C \end{cases} \tag{7.185}$$

where s is the solution of (7.107), $G(s)$ is the Gallager function given by (5.98), and R_0 is the computational cutoff rate.

Finally, when we use the Fano algorithm with a finite back-search limit, we can show the following counterpart to Theorem 7.12:

Theorem 7.18 If we use the Zigangirov metric, then for the ensemble of rate $R = b/c$, time-invariant convolutional codes of memory m the burst error probability for the Fano algorithm with back-search limit $m + 1$ when used to communicate over the BSC is upper-bounded by

$$P(\mathcal{E}_1^{\text{fbs}}) < \begin{cases} O(m)2^{-(R_0 - R)(m+1)c} & 0 \le R \le R_{\text{crit}} \\ O(1)2^{-(G(s_z) - s_z R)(m+1)c} & R_{\text{crit}} < R < C \end{cases} \tag{7.186}$$

where s_z is given by (7.132), $G(\ )$ is the Gallager function given by (5.98), R_0 is the computational cutoff rate, and R_{crit} is the critical rate.

7.9 ANALYSIS OF CREEPER*

For the analysis of computational behavior we consider as before the ensemble $\mathcal{E}(b, c, \infty, 1)$ of binary, rate $R = b/c$, time-invariant convolutional codes with infinite-memory.

The following properties are quite similar to those of the Fano algorithm:

Property C1: A node $\boldsymbol{u}_{[0,t)}$ is extended by Creeper only if its metric $\mu(\boldsymbol{r}_{[0,t)}, \boldsymbol{v}_{[0,t)})$ is not less than the threshold T.

From the description of Creeper it follows that the value of the threshold is a multiple of the stepsize Δ and that it is decreased at least twice between two successive visits to a certain node. Hence, we have the following:

Property C2: The threshold decrement between two successive visits to a certain node is at least 2Δ.

Property C2 implies that Creeper will not get stuck in a loop.

Property C3: Let $\mu_{\min}$ denote the minimum metric along the correct path. The minimum value of the threshold for Creeper is lower-bounded by

$$T_{\min} > \mu_{\min} - 2\Delta + c\beta \tag{7.187}$$

Property C4: For each value of the threshold T Creeper can visit a node at most once.

From Properties C1 and C4 it follows that for given $T_{\min} = y$ the number of visits to any node with metric $\mu = z$, where $z \geq y$, is upper-bounded by

$$\left\lceil \frac{z-y}{2\Delta} \right\rceil < \frac{z-y}{2\Delta} + 1 \tag{7.188}$$

Thus, since the root has metric 0 it can be visited at most $-y/2\Delta + 1$ times.

Analogously to (7.157) we can show that the conditional expectation of the number of computations needed to decode the first correct node given that $T_{\min} = y$ is upper-bounded by the inequality

$$E\left[C_1 \mid T_{\min} = y\right] < -\frac{y}{2\Delta} + 1 + \sum_{j=1}^{\infty} (2^b - 1) 2^{b(j-1)} \sum_{z \geq y} \left(\frac{z-y}{2\Delta} + 1 \right) f_j(y, z) \tag{7.189}$$

The upper bound (7.189) for Creeper differs from its counterpart for the Fano algorithm only by the factor of 2 in front of the stepsize Δ. Therefore, repeating the steps (7.158)–(7.167) yields

$$E\left[C_1 \mid T_{\min} = y\right] < \left(1 - \frac{E[Z_i']}{2(1-2^{-b})\Delta}\right) 2^{2\lambda_1\Delta - b - \lambda_1 c\beta} 2^{-\lambda_1 y} \tag{7.190}$$

Combining Property C3 and (7.190) yields (cf. (7.168))

$$E\left[C_1 \mid \mu_{\min} = y\right] < \left(1 - \frac{E[Z_i']}{2(1-2^{-b})\Delta}\right) 2^{4\lambda_1\Delta - b - 2\lambda_1 c\beta} 2^{-\lambda_1 y} \tag{7.191}$$

The function $2^{4\lambda_1\Delta}/\Delta$ attains its minimum when

$$\Delta = \frac{1}{4\lambda_1 \ln 2} \tag{7.192}$$

Since in a practical situation the second term in the parentheses on the right-hand side of (7.191) is essentially greater than 1, we use (7.192) as a good approximation of the optimal value of the stepsize. Inserting (7.192) into (7.191) gives

$$E\left[C_1 \mid \mu_{\min} = y\right] < C_{\mathrm{C}} 2^{-\lambda_1 y} \tag{7.193}$$

where

$$C_{\mathrm{C}} = \left(1 - \frac{2(\alpha+\beta)\lambda_1 \ln 2}{1-2^{-b}}\right) e 2^{-b - 2\lambda_1 c\beta} \tag{7.194}$$

and analogously to (7.176) we obtain

$$C_{\text{av}} = E[C_1] < C_{\text{C}} \frac{\lambda_0}{\lambda_0 + \lambda_1} \tag{7.195}$$

Thus, we have proved the following:

Theorem 7.19 The average number of computations per branch for Creeper when used to communicate over the BSC is upper-bounded by

$$C_{\text{av}} < C_{\text{C}} \frac{\lambda_0}{\lambda_0 + \lambda_1} \tag{7.196}$$

where λ_0 and λ_1 are the negative and positive roots of equations (7.33) and (7.51), respectively, such that $\lambda_0 + \lambda_1 < 0$ and C_{C} is given by (7.194).

From (7.105) and (7.108), we have the next corollary.

Corollary 7.20 If we use the Fano metric, the average number of computations per branch for Creeper when used to communicate over the BSC is upper-bounded by

$$C_{\text{av}} < C_{\text{C}} \frac{s}{s-1} \quad \text{for } R < R_0 \tag{7.197}$$

where s is the solution of (7.35) and R_0 is the computational cutoff rate.

Again we can use the Markov inequality (Lemma 6.14) and obtain the following:

Theorem 7.21 When Creeper is used to communicate over the BSC then the computational distribution for the number of computations needed to decode any correct node, C_i, $i = 1, 2, \ldots$, is upper-bounded by

$$P(C_i \geq x) \leq C_{\text{av}} x^{-1} \quad \text{for } R < R_0, \tag{7.198}$$

where C_{av} is the average number of computations per branch and R_0 is the computational cutoff rate.

The lower bound on the computational distribution given in Theorem 7.9 is also valid for Creeper.

By comparing C_{C} and C_{F}, one might expect Creeper to perform worse than the Fano algorithm. In our proof technique, we (have to) use a lower bound on the threshold, viz., $T_{\min} > \mu_{\min} - 2\Delta + c\beta$, which does not show that the average behavior of Creeper is much better. For the Fano and stack algorithms, we can use the more realistic bounds, $T_{\min} > \mu_{\min} - \Delta$ and $T_{\min} = \mu_{\min}$, respectively. The simulations reported in Section 7.5 show the superior computational performance of Creeper compared to the Fano algorithm.

For the error probability, we can analogously to Theorem 7.16 prove the next theorem.

Theorem 7.22 If we use the Fano metric, then for the ensemble of rate $R = b/c$, time-varying convolutional codes of memory m the burst error probability for Creeper when used to communicate over the BSC is upper-bounded by

$$P(\mathcal{E}_1) < \begin{cases} O(1)2^{-Rmc} & 0 \le R < R_0 \\ O(m)2^{-R_0 mc} & R = R_0 \\ O(1)2^{-G(s)mc} & R_0 < R < C \end{cases} \tag{7.199}$$

where s is the solution of (7.107), $G(s)$ is the Gallager function given by (5.98), and R_0 is the computational cutoff rate.

For the Gallager metric, we can show the following counterpart to Theorem 7.17:

Theorem 7.23 If we use the Gallager metric, then for the ensemble of rate $R = b/c$, time-varying convolutional codes of memory m the burst error probability for Creeper when used to communicate over the BSC is upper-bounded by

$$P(\mathcal{E}_1) < \begin{cases} O(m)2^{-R_0 mc} & 0 \le R \le R_0 \\ O(1)2^{-G(s)mc} & R_0 < R < C \end{cases} \tag{7.200}$$

where s is the solution of (7.107), $G(s)$ is the Gallager function given by (5.98), and R_0 is the computational cutoff rate.

Finally, when we use Creeper with a finite back-search limit, we can show the following counterpart to Theorem 7.18:

Theorem 7.24 If we use the Zigangirov metric, then for the ensemble of rate $R = b/c$, time-invariant convolutional codes of memory m the burst error probability for Creeper with back-search limit $m + 1$ when used to communicate over the BSC is upper-bounded by

$$P(\mathcal{E}_1^{\text{fbs}}) < \begin{cases} O(m)2^{-(R_0 - R)(m+1)c} & 0 \le R \le R_{\text{crit}} \\ O(1)2^{-(G(s_z) - s_z R)(m+1)c} & R_{\text{crit}} < R < C \end{cases} \tag{7.201}$$

where s_z is given by (7.132), $G(\ )$ is the Gallager function given by (5.98), R_0 is the computational cutoff rate, and R_{crit} is the critical rate.

7.10 COMMENTS

Wozencraft presented the first sequential decoding algorithm [Woz57]. It inspired all the subsequent discoveries but is much more complicated as well as less effective than the other sequential decoding algorithms.

In 1963, Fano published a most ingenious algorithm for sequential decoding [Fan63]. It is still considered to be the most practical sequential decoding algorithm. The Fano algorithm was intensively studied by graduate students at MIT during the

1960s; among them were Falconer [Fal66], Haccoun [Hac66], Heller [Hel67], Savage [Sav66], Stiglitz [Sti63], and Yudkin [Yud64]. It was used for space and military applications in the late 1960s.

Zigangirov published the stack algorithm in 1966 [Zig66] and Jelinek in 1969 [Jel69]. Zigangirov used recursive equations to analyze sequential decoding algorithms [Zig74].

An embryo of Creeper appears in [Zig75]. It was further developed in [CJZ84b], and the final version and its analysis are given in [Nys93, NJZ97].

Among all of those analysts who addressed the problem of bounding the distribution function of the number of computation, we would like to mention Jacobs and Berlekamp [JaB67].

Massey [Mas75]: "It has taken a long time to reach the point where one understands why the early sequential decoding algorithms 'worked' and what they were really doing."

PROBLEMS

7.1 Consider the rate $R = 1/2$, memory $m = 2$ convolutional encoding matrix $G(D) = (1 + D + D^2 \quad 1 + D^2)$.

- **a)** Draw the complete code tree for length $\ell = 3$ and tail $m = 2$.
- **b)** Find the Fano metrics for a BSC with crossover probability $\epsilon = 0.045$, that is, $R = R_0 = 1/2$.
- **c)** Use the stack algorithm to decode the received sequence $\boldsymbol{r} = 00\,01\,10\,00\,00$. How many computations are needed?
- **d)** Repeat (c) with the Fano algorithm with threshold increment $\Delta = 4$.
- **e)** Are the decisions in (c) and (d) maximum-likelihood?

7.2 Repeat Problem 7.1 (c–e) for $\boldsymbol{r} = 01\,01\,00\,00\,11$.

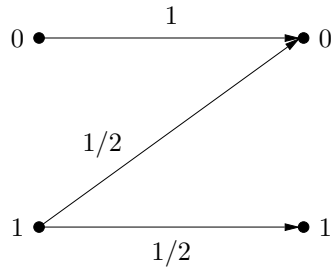
7.3 Consider a binary-input, 8-ary output DMC with transition probabilities $P(r \mid v)$ given in Example 7.1. Suppose that four information symbols (and a tail of three dummy zeros) are encoded by $G(D) = (1 + D + D^2 + D^3 \quad 1 + D^2 + D^3)$. Use the stack algorithm to decode the received sequence $\boldsymbol{r} = 0_2 0_2\ 0_1 0_4\ 0_3 0_4\ 0_2 1_3\ 1_4 1_1\ 1_3 1_2\ 0_4 0_3$.

7.4 Repeat Problem 7.3 for $G(D) = (1 + D + D^2 \quad 1 + D^2)$ and $\boldsymbol{r} = 1_1 0_4\ 0_1 1_2\ 1_1 0_1\ 0_1 1_1\ 1_3 0_1\ 0_4 0_3$. Is the decision maximum-likelihood?

7.5 Consider the BEC with erasure probability $\delta = 1/2$. Suppose that three information symbols (and a tail of two dummy zeros) are encoded by the encoding matrix $G(D) = (1 + D + D^2 \quad 1 + D^2)$.

- **a)** Find the Fano metric.
- **b)** Use the stack algorithm to decode the received sequence $\boldsymbol{r} = 1\Delta\ \Delta\Delta\ 0\Delta\ \Delta 1\ \ \Delta\Delta$.

7.6 Consider the following Z channel:



Suppose that three information symbols (and a tail of two dummy zeros) are encoded by the encoding matrix $G(D) = (1 + D + D^2 \quad 1 + D^2)$.

a) Find the Fano metric.

b) Use the stack algorithm to decode the received sequence $\boldsymbol{r} = 01\ 00\ 00\ 10\ 10$.

c) Is the decision in (b) maximum-likelihood?

7.7 Consider a rate $R = 1/2$ convolutional code with encoding matrix $G(D) = (1+D+D^2 \quad 1+D^2)$ that is used to communicate over a binary-input, 8-ary output DMC with transition probabilities $P(r \mid v)$ given by the following table:

		r							
		0_4	0_3	0_2	0_1	1_1	1_2	1_3	1_4
v	0	0.2196	0.2556	0.2144	0.1521	0.0926	0.0463	0.0167	0.0027
	1	0.0027	0.0167	0.0463	0.0926	0.1521	0.2144	0.2556	0.2196

Suppose that four information symbols (and a tail of two dummy zeros) are encoded.

a) Use an appropriate choice of integer metrics and the stack algorithm to decode the received sequence $\boldsymbol{r} = 0_2 1_3\ 1_2 0_2\ 0_1 1_3\ 1_1 1_1\ 1_2 1_4\ 1_1 1_1$.

b) Is the decision in (a) maximum-likelihood?

7.8 Consider a rate $R = 1/3$ convolutional code with encoding matrix $G(D) = (1+D+D^3 \quad 1+D^2+D^3 \quad 1+D+D^2+D^3)$ that is used to communicate over the BSC with crossover probability $\epsilon = 0.095$.

a) Find R_0 for this channel.

b) Suppose that a tail of three information symbols is used. Make an appropriate choice of integer metrics and use the stack algorithm to decode the received sequence $\boldsymbol{r} = 010\ 001\ 101\ 100\ 000\ 011\ 110$.

7.9 Consider the rate $R = 1/2$, convolutional code with the systematic encoding matrix $G(D) = (1 \quad 1+D+D^2+D^4)$ that is used to communicate over the binary-input, 8-ary output DMC with transition probabilities $P(r \mid v)$ given by the following table:

		r							
		0_4	0_3	0_2	0_1	1_1	1_2	1_3	1_4
v	0	0.1402	0.3203	0.2864	0.1660	0.0671	0.0177	0.0024	0.0001
	1	0.0001	0.0024	0.0177	0.0671	0.1660	0.2864	0.3203	0.1402

Suppose that after the information symbols and a tail of four dummy zeros have been encoded, the four "known" (systematic encoder!) zeros in the encoded tail are deleted before transmission. The sequence $\boldsymbol{r} = 1_2 0_1\ 0_2 1_2\ 1_3 1_1\ 1_2 1_1\ 0_3 1_2$ is received.

a) How many information symbols were encoded?

b) After an appropriate choice of integer metrics, use the stack algorithm to decode $\boldsymbol{r}$.

7.10 Show that the factor $\lambda_0/(\lambda_0 + \lambda_1)$ in the upper bound on the average number of computations per branch for the stack algorithm (Theorem 7.5) regarded as a function of the bias B achieves its minimum for $B = R$, that is, for the Fano metric.

7.11 Show that for rates $R < R_0$ and bias $B \leq R_0$ the positive root λ_1 of the equation

$$\left(\frac{1}{2}2^{\lambda_1 \alpha} + \frac{1}{2}2^{\lambda_1 \beta}\right) = 2^{-R}$$

is strictly less than $1/2$ (cf. Corollary 7.7).

CHAPTER 8

LOW-DENSITY PARITY-CHECK CODES

The previous chapters are devoted to classical convolutional codes. In this chapter, we consider a special class of convolutional codes, namely *low-density parity-check* (LDPC) *convolutional codes*. These codes are defined using *sparse syndrome formers*, that is, syndrome formers for which entries are mostly zeros. As we shall show, these codes are suitable for low-complexity *iterative decoding*. In parallel to describing and analyzing the LDPC convolutional codes we describe their block code counterpart, the *low-density parity-check block codes*

In Section 8.1, we give the formal definition of the LDPC block codes introduced in Chapter 1. Then, in Section 8.2, we introduce LDPC convolutional codes. Section 8.3 is devoted to the description of *permutors*. Analyses of the minimum/free distances of LDPC block/convolutional codes are given in Section 8.4. In Section 8.5, we describe Gallager's iterative decoding algorithm, also known as the *belief propagation* algorithm, both for LDPC block and LDPC convolutional codes. Iterative limits and thresholds of LDPC codes are investigated in Section 8.6. In Section 8.7, we introduce *braided block codes*, a variant of LDPC convolutional codes.

8.1 LDPC BLOCK CODES

In Section 1.3, we briefly introduced LDPC block codes. In this section, we shall give a formal definition of LDPC block codes. When constructing and analyzing LDPC convolutional codes it is convenient to operate with the transposed parity-check matrix H^{T} (that is, the syndrome former). Thus, we shall use the transposed parity-check matrix also in our description of the LDPC block codes.

Definition A binary block code of block length N, defined by the $N \times L$, $L < N$, transposed parity-check matrix H^{T},

$$H^{\mathrm{T}} = \begin{pmatrix} \boldsymbol{h}_0 \\ \boldsymbol{h}_1 \\ \vdots \\ \boldsymbol{h}_{N-1} \end{pmatrix} \tag{8.1}$$

is called a *low-density parity-check* (LDPC) block code if all rows $\boldsymbol{h}_n = (h_{n0}\, h_{n1}\, \ldots\, h_{n(L-1)})$ of H^{T} are sparse, that is, if

$$w_{\mathrm{H}}(\boldsymbol{h}_n) \ll L, \quad n = 0, \ldots, N-1 \tag{8.2}$$

A code defined by a transposed parity-check matrix H^{T} of rank r, $r \leq L$, has the code rate $R = 1 - r/N$. This rate is lower-bounded by the *design rate* of the code $R_d \stackrel{\mathrm{def}}{=} 1 - L/N$. The code rate and the design rate coincide when the parity-check matrix has full rank.

Definition A *regular* binary (N, J, K) LDPC block code is a code defined by an $N \times L$, $L < N$, transposed parity-check matrix H^{T}, having exactly J 1s in each row ($J \ll L$), K 1s in each column ($K \ll N$), and 0s elsewhere.

It follows from this definition that the number of columns in such a transposed parity-check matrix equals $L = NJ/K$ and, hence, that the design rate of the code is $R_d = 1 - J/K$. The fraction of 1s in the transposed parity-check matrices of regular (N, J, K) LDPC block codes is decreasing with increasing block length N.

In contrast to regular LDPC block codes, the number of 1s in the rows/columns of a transposed parity-check matrix in an *irregular* LDPC block code varies. We consider only regular LDPC codes.

As we know from Section 1.3, the connections between parity-check equations and code symbols of an LDPC block code can be illustrated not only by a parity-check matrix but also by a Tanner graph. To each row in the transposed parity-check matrix H^{T} corresponds a symbol (variable) node in the Tanner graph, and to each column corresponds a constraint (check) node. A symbol node is connected with a constraint node by an edge if the element at the intersection of the corresponding row and column in H^{T} is equal to unity. A sparse parity-check matrix will therefore result in a weakly connected graph with a relatively small amount of edges.

Consider the two sets of nodes

$$\mathcal{N}(l) = \{n \mid h_{nl} = 1\} \tag{8.3}$$

and

$$\mathcal{L}(n) = \{l \mid h_{nl} = 1\} \tag{8.4}$$

where h_{nl} is the element in row n and column l of the matrix H^{T}. In the Tanner graph, $\mathcal{N}(l)$ enumerates the symbol nodes connected to the constraint node l and $\mathcal{L}(n)$ enumerates the constraint nodes connected to the symbol node n.

The number of edges which leave a symbol/constraint node is called the *degree* of this node. Each symbol node of a regular (N, J, K) LDPC block code has degree J and each constraint node has degree K. For *irregular* LDPC block codes, the degrees of the symbol and constraint nodes vary. Usually one considers random ensembles of irregular LDPC codes and the ensembles are defined by the degree distributions of symbol and constraint nodes.

We shall focus on two ensembles of regular LDPC block codes, introduced by Gallager [Gal62, Gal63]. First we consider the ensemble $\mathcal{B}_1(N, J, K)$ of regular (N, J, K) LDPC codes with parity-check matrices (1.77). Then we shall study the ensemble $\mathcal{B}_2(N, J, K)$ of regular (N, J, K) LDPC block codes with parity-check matrices that are composed from permutation matrices.[14]

The transposed parity-check matrices of codes in the ensemble $\mathcal{B}_1(N, J, K)$ are (cf. (1.77))

$$H^{\mathrm{T}} = \left(H_1^{\mathrm{T}}\, H_2^{\mathrm{T}}\, \ldots\, H_J^{\mathrm{T}}\right) \tag{8.5}$$

where each submatrix H_j^{T}, $j = 1, 2, \ldots, J$, is an $N \times M$ matrix, where $M = N/K = L/J$. All submatrices have one 1 in each row and K 1s in each column and all the other entries are 0s.

[14] A permutation matrix is a binary square matrix having exactly one 1 in each column and in each row.

■ **EXAMPLE 8.1**

The following transposed parity-check matrix defines a regular $(12, 2, 3)$ LDPC block code[15] from the ensemble $\mathcal{B}_1(12, 2, 3)$:

$$H^{\mathrm{T}} = \left(\begin{array}{cccc:cccc} 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \end{array}\right) \tag{8.6}$$

Its design rate is $R_d = 1 - J/K = 1 - 2/3 = 1/3$.

The constraint nodes of the Tanner graph of this code (Fig. 8.1), $c_0, c_1, \ldots, c_7$, are divided into $J = 2$ groups; the constraint nodes c_l, $l \in \{0, 1, 2, 3\}$, are in the first group and the constraint nodes c_l, $l \in \{4, 5, 6, 7\}$, are in the second group. Each of the symbol nodes, $v_0, v_1, \ldots, v_{11}$, is connected with exactly one constraint node from each group. For example, the symbol node corresponding to code symbol v_5 is connected with the constraint nodes c_l, $l \in \mathcal{L}(5) = \{2, 6\}$ (thick lines). These constraint nodes are connected with the symbol nodes corresponding to the code symbols v_n, $n \in \mathcal{N}(2) = \{5, 6, 11\}$ and $n \in \mathcal{N}(6) = \{1, 5, 10\}$, respectively (thick and dashed lines).

Lemma 8.1 Each $N \times M$ submatrix H_j^{T}, $j = 1, 2, \ldots, J$, of the transposed parity-check matrix (8.5) can be chosen in $(KM)!/(K!)^M$ different ways, where $M = N/K$.

Proof: There are $\binom{KM}{K}$ ways to form the first column of H_j^{T}. If the first column is fixed, then the second column can be chosen in $\binom{K(M-1)}{K}$ different ways, and so on. We conclude that there exists

$$\binom{KM}{K}\binom{K(M-1)}{K}\binom{K(M-2)}{K}\cdots\binom{K}{K} = \frac{(KM)!}{(K!)^M} \tag{8.7}$$

different submatrices H_j^{T}. ■

[15]Gallager's condition "containing mostly 0s and relatively few 1s" for the transposed parity-check matrix H^{T} in this example is not fulfilled because the code is short, but we still call it an LDPC code.

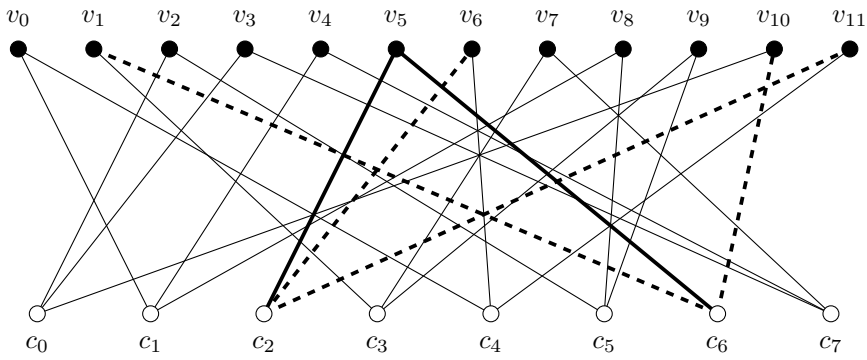


Figure 8.1 The Tanner graph for the code in Example 8.1.

From Lemma 8.1 it follows that the number of different codes in the ensemble $\mathcal{B}_1(N, J, K)$ equals $\left((KM)!/(K!)^M\right)^J$. Using Stirling's formula [Fel68]

$$n! = n^n e^{-n+o(n)} \tag{8.8}$$

where $o(n)/n \to 0$ when $n \to \infty$, we obtain that the number of different codes in $\mathcal{B}_1(N, J, K)$ equals

$$e^{JN \ln N - \frac{JN}{K} \ln(K!) - JN + o(N)} \tag{8.9}$$

Consider now the ensemble $\mathcal{B}_2(N, J, K)$ of regular LDPC block codes with parity-check matrices composed from permutation matrices. The convolutional counterpart of block codes from the ensemble $\mathcal{B}_2(N, J, K)$ is convenient for analyzing the free distance and the thresholds of LDPC convolutional codes (see Sections 8.4 and 8.6).

The transposed parity-check matrices H^{T} of the codes in the ensemble $\mathcal{B}_2(N, J, K)$ are defined as

$$H^{\mathrm{T}} = \begin{pmatrix} P^{(11)} & P^{(12)} & \dots & P^{(1J)} \\ P^{(21)} & P^{(22)} & \dots & P^{(2J)} \\ \vdots & & & \\ P^{(K1)} & P^{(K2)} & \dots & P^{(KJ)} \end{pmatrix} \tag{8.10}$$

where $P^{(kj)}$, $k = 1, 2, \dots, K$, $j = 1, 2, \dots, J$, are $M \times M$ permutation matrices and $M = N/K$. If these matrices are chosen randomly and independently of each other such that all $M!$ values are equiprobable, then we get an ensemble $\mathcal{B}_2(N, J, K)$ of regular (N, J, K) LDPC block codes with transposed parity-check matrices composed of permutation matrices. The total number of different codes in the ensemble $\mathcal{B}_2(N, J, K)$ equals $(M!)^{JK}$. Using Stirling's formula (8.8) we obtain that the number of different codes in $\mathcal{B}_2(N, J, K)$ equals

$$e^{JN \ln N - JN \ln K - JN + o(N)} \tag{8.11}$$

From (8.9) and (8.11) it follows that the number of codes in the ensemble $\mathcal{B}_1(N, J, K)$ is about $e^{JN/K \ln(K^K/K!)}$ times larger than that of the codes in the ensemble $\mathcal{B}_2(N, J, K)$.

■ **EXAMPLE 8.2**

The following transposed parity-check matrix composed of 4×4 permutation matrices defines a regular $(12, 2, 3)$ LDPC block code in the ensemble $\mathcal{B}_2(12, 2, 3)$:

$$H^{\mathrm{T}} = \left(\begin{array}{cccc:cccc} 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ \hdashline 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ \hdashline 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \end{array}\right) \tag{8.12}$$

As in the Tanner graph of the code considered in Example 8.1, the constraint nodes of the Tanner graph of the code defined by the transposed parity-check matrix (8.12) are divided into $J = 2$ groups and each symbol node is connected with exactly one node from each of these two groups. Moreover, the symbol nodes of the graph are divided into $K = 3$ groups and each constraint node is connected with exactly one symbol node from each group (Problem 8.2).

When we study iterative decoding of LDPC block codes, it is convenient to use a *computational tree* stemming from a symbol node representing an arbitrary code symbol v_n. In Fig. 8.2 we illustrate such a computational tree for a regular (N, J, K) LDPC block code.

Each node at an even level $k = 2\ell$, $\ell = 0, 1, \ldots$, of the tree represents one of the N symbols of the codeword and each node at an odd level $k = 2\ell + 1$, $\ell = 0, 1, \ldots$, represents one of the J parity-check equations. We call the set of symbols corresponding to the nodes at even levels of the tree a v_n-*clan*. The set of symbols in the 2ℓth level, where ℓ is a nonnegative integer, is called the ℓth *generation* of the clan. The symbol v_n, represented by the root at level zero, is called the *clan head* of the v_n-clan.

The symbols in the clan, excluding v_n, are called the *descendants* of the clan head v_n. The J edges leaving the root, together with the nodes they lead to, correspond to the J parity-check equations that include the clan head. The nodes at this first level are called the *families* of the clan head. Each of these nodes, in turn, is connected to $K - 1$ edges leading to $K - 1$ nodes at the second level. These $K - 1$ second-level nodes correspond to the $K - 1$ symbols that (together with the clan head) are included in the parity-check equation represented by a family of the clan head. They are called *children* or *direct descendants* of the clan head. Hence, the clan head has J families and in each family there are $K - 1$ children.

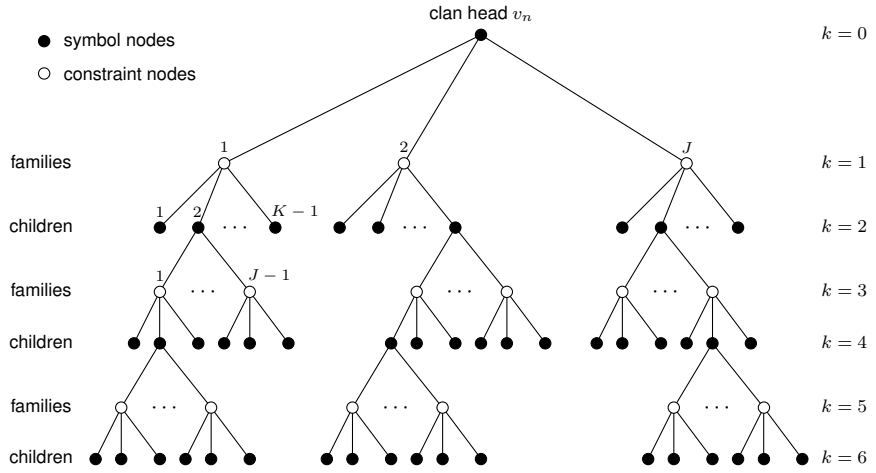


Figure 8.2 Computational tree of an (N, J, K) LDPC code.

Each child of the clan head has $J - 1$ families, that is, $J - 1$ edges leave each second-level node leading to $J - 1$ nodes in the third level of the tree. These $J - 1$ third-level nodes correspond to the $J - 1$ parity-check equations that include a child of the clan head but not the clan head itself. Each of the third-level families includes $K - 1$ children, that is, the $K - 1$ edges leaving each third-level node lead to $K - 1$ nodes at the fourth level, and so on. This expansion of levels can be continued indefinitely, but eventually, since the total number of symbols in a codeword and the number of constraints are finite, different nodes in the tree on level k will represent the same symbol or constraint. In this case we say that the v_n-clan graph has a *cycle*, and starting from level k this code cannot be described by a *cyclefree* graph, that is, by a tree.

Definition The v_n-clan graph is called $2\ell_0$-*cyclefree* if all symbol nodes of the clan graph up to the ℓ_0th generation correspond to different symbols, but in the set of symbol nodes of the clan graph in the $(\ell_0 + 1)$th generation there are at least two nodes that represent the same symbol.

Definition An LDPC block code is called $2\ell_0$-*cyclefree* if all v_n-clan graphs, $n = 0, \ldots, N-1$, are 2ℓ-cyclefree, where $\ell \geq \ell_0$, and there exists at least one $2\ell_0$-*cyclefree* clan graph.

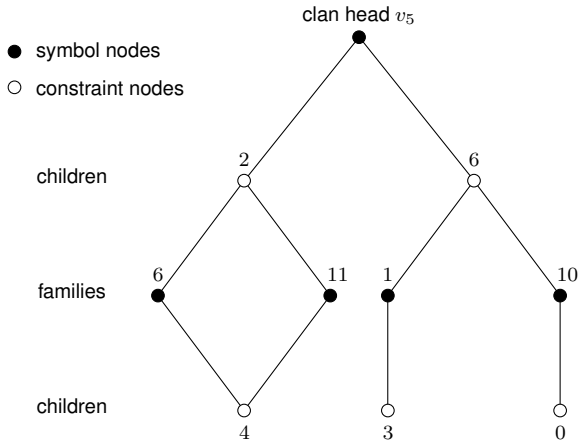


Figure 8.3 The v_5-clan graph for the LDPC block code given by the Tanner graph in Fig. 8.1.

■ **EXAMPLE 8.1** ***(Cont'd)***

In Fig. 8.3 we show the v_5-clan graph for the LDPC block code given by the Tanner graph in Fig. 8.1. It is 2-cyclefree ($\ell_0 = 1$).

The parameter ℓ_0 characterizes the *number of independent iterations* during belief propagation decoding of $2\ell_0$-cyclefree LDPC block codes (see Section 8.5). When analyzing the thresholds of LDPC block codes we assume that the iterative decoder operates on cyclefree graphs with independent decision statistics.

Theorem 8.2 (Gallager [Gal63]) If a regular (N, J, K) LDPC block code is $2\ell_0$-cyclefree, then

$$\ell_0 < \frac{\log N}{\log\left((J-1)(K-1)\right)} \tag{8.13}$$

Proof: Consider an arbitrary v_n-clan of a regular (N, J, K) LDPC block code. To the root (the 0th generation) corresponds one symbol, namely v_n, in the 1st generation (level 2) there are $J(K-1)$ symbols, ..., and in the ℓth generation there are

$$J(K-1)\left((J-1)(K-1)\right)^{(\ell-1)} > \left((J-1)(K-1)\right)^{\ell} \tag{8.14}$$

symbols. Since the number of different symbols in the ℓ_0th generation of a clan does not exceed N, we have

$$\left((J-1)(K-1)\right)^{\ell_0} < N \tag{8.15}$$

From (8.15) we conclude that inequality (8.13) holds. ■

The proof of a nontrivial lower bound for ℓ_0 is more difficult. Gallager constructively proved [Gal62, Gal63] that for $J \geq 2$ the ensemble $\mathcal{B}_2(N, J, K)$ contains a

regular (N, J, K) $2\ell_0$-cyclefree LDPC block code such that

$$\ell_0 > \frac{\log N}{2\log\left((J-1)(K-1)\right)} - \gamma \tag{8.16}$$

where γ is a constant depending on J and K but not on N. This proof is valid only for regular LDPC block codes from the ensemble $\mathcal{B}_2(N, J, K)$. Similar bounds for more general LDPC block codes can be obtained if we "expurgate" symbols which cause short cycles [LTZ05]. We preassign to these symbols known values, for example, 0s, and will not transmit them over the channel. If the fraction of these expurgated symbols goes to zero when $N \to \infty$, then the rate loss also goes to zero when $N \to \infty$.

In Section 8.4, we shall prove that if $J \geq 3$ then there exists a regular (N, J, K) LDPC block code whose minimum distance asymptotically grows linearly with N. The following theorem is due to Gallager [Gal63] and shows that if $J = 2$ then the minimum distance of regular (N, J, K) LDPC block codes grows not faster than $O(\log N)$.

Theorem 8.3 The minimum distance of a regular $(N, 2, K)$ LDPC block code with design rate $R_d = 1 - 2/K$ is upper-bounded by

$$d_{\min} < 2 + \frac{2\log N/2}{\log(K-1)} \tag{8.17}$$

Proof: Consider a v_n-clan graph of a regular $(N, 2, K)$ LDPC block code. The number of symbol nodes in the ℓth generation of the clan is $2(K-1)^\ell < N$. If the clan graph is $2\ell_0$-cyclefree, then

$$\ell_0 < \frac{\log N/2}{\log(K-1)} \tag{8.18}$$

and there exists a cycle of length (in edges) $4\ell_0 + 4$ which includes $2\ell_0 + 2$ symbol nodes. Assume that all symbols of a codeword corresponding to nodes along this cycle are 1s and all the other symbols of this codeword are 0s. From (8.16) it follows that the number of symbols along this cycle is less than $2 + \frac{2\log N/2}{\log(K-1)}$. Since the allzero sequence is a codeword, the minimum distance $d_{\min}$ is upper-bounded by (8.17). ∎

The following theorem establishes an upper bound for the minimum distance of LDPC block codes from the ensemble $\mathcal{B}_2(N, J, K)$.

Theorem 8.4 The minimum distance of a regular (N, J, K) LDPC block code with a parity-check matrix composed of $M \times M$ permutation matrices from the ensemble $\mathcal{B}_2(N, J, K)$ can be upper-bounded by the inequality

$$d_{\min} \leq 2M = \frac{2N}{K} \tag{8.19}$$

Proof: Consider N-tuple $\boldsymbol{v}$, $N = KM$, consisting of K M-tuples, that is, $\boldsymbol{v} = \boldsymbol{v}_1\, \boldsymbol{v}_2\, \ldots \boldsymbol{v}_K$, where $\boldsymbol{v}_k = \boldsymbol{v}_{k0}\, \boldsymbol{v}_{k1}\, \ldots \boldsymbol{v}_{k(M-1)}$, $k = 1, 2, \ldots, K$. Suppose that two of the M-tuples $\boldsymbol{v}_k$, $k = 1, 2, \ldots, K$, for example, $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$, are allone tuples and all the other M-tuples of $\boldsymbol{v}$ are allzero vectors. Then $\boldsymbol{v}$ is a codeword having weight $2M = 2N/K$ and the minimum distance of the code cannot exceed $2N/K$. ∎

The bound (8.19) is nontrivial for large K; it is tight for small design rates $R_d = 1 - J/K$ and loose for large R_d.

Next we introduce *generalized low-density parity-check* (GLDPC) block codes. In Section 1.3 we considered a regular (N, J, K) LDPC block code as a code having single-error-detecting codes of length K as constituent codes. If we allow other constituent block codes, then we obtain GLDPC block codes. For example, a length $(\mu - \kappa)N$ GLDPC block code, having the (μ, κ) Hamming codes as constituent codes, can be constructed from a regular (N, J, K) LDPC block code, where $K = \mu$, as follows.

Let $H^{\mathrm{T}}_{\mathrm{LDPC}}$ be an $N \times L$ transposed parity-check matrix of a regular (N, J, K) LDPC block code, where $K = \mu = 2^{\lambda} - 1$, $\lambda = 3, 4, \ldots$, and let $H^{\mathrm{T}}_{\mathrm{Ham}}$ be a $\mu \times \lambda$ transposed parity-check matrix of the (μ, κ) Hamming code, where $\kappa = \mu - \lambda$. The L binary column vectors $\boldsymbol{h}_l^{\mathrm{T}}$ of $H^{\mathrm{T}}_{\mathrm{LDPC}}$ have length N and Hamming weight $w_{\mathrm{H}}(\boldsymbol{h}_l) = \mu = K$.

We obtain the *Hamming expansion* of the vector $\boldsymbol{h}_l^{\mathrm{T}}$ by replacing K 1s in $\boldsymbol{h}_l^{\mathrm{T}}$ by the rows of the $\mu \times (\mu - \kappa)$ matrix $H^{\mathrm{T}}_{\mathrm{Ham}}$, taken in any order, and by replacing the $N - K$ 0s in $\boldsymbol{h}_l^{\mathrm{T}}$ by the allzero row vector $(0\,0\,\ldots\,0)$ of length $\lambda = \mu - \kappa$. The Hamming expansion of the vector $\boldsymbol{h}_l^{\mathrm{T}}$ is an $N \times (\mu - \kappa)$ matrix which we denote $\widetilde{H}^{\mathrm{T}}(\boldsymbol{h}_l^{\mathrm{T}})$.

A transposed parity-check matrix $H^{\mathrm{T}}_{\mathrm{GLDPC}}$ of the GLDPC code having Hamming codes as constituent codes can be constructed by replacing the L column vectors $\boldsymbol{h}_l^{\mathrm{T}}$, $l = 0, 1, \ldots, L - 1$, of the matrix $H^{\mathrm{T}}_{\mathrm{LDPC}}$ by the Hamming expansions of these vectors.

■ **EXAMPLE 8.3**

Consider the construction of a length $N = 21$ GLDPC block code with $(7, 4)$ Hamming codes as constituent codes. In the transposed parity-check matrix

$$H_{\text{LDPC}}^{\text{T}} = \left(\begin{array}{ccc:ccc} 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \end{array}\right) \tag{8.20}$$

defining an LDPC $(21, 2, 7)$ block code, we replace the $K = 7$ 1s in the column vectors $\boldsymbol{h}_l^{\text{T}}$, $l = 0, 1, \ldots, 5$, by the $K = \mu = 7$ rows of the following transposed parity-check matrices for the equivalent Hamming codes, taken in order,

$$H_{\text{Ham}}^{\text{T(0)}} = \begin{pmatrix} 1\,0\,0 \\ 0\,1\,1 \\ 1\,0\,1 \\ 0\,1\,0 \\ 0\,0\,1 \\ 1\,1\,1 \\ 1\,1\,0 \end{pmatrix} \quad H_{\text{Ham}}^{\text{T(1)}} = \begin{pmatrix} 0\,1\,1 \\ 0\,1\,0 \\ 0\,0\,1 \\ 1\,0\,1 \\ 1\,0\,0 \\ 1\,1\,1 \\ 1\,1\,0 \end{pmatrix} \quad H_{\text{Ham}}^{\text{T(2)}} = \begin{pmatrix} 1\,0\,1 \\ 0\,1\,0 \\ 1\,1\,0 \\ 1\,0\,0 \\ 1\,1\,1 \\ 0\,0\,1 \\ 0\,1\,1 \end{pmatrix}$$

$$H_{\text{Ham}}^{\text{T(3)}} = \begin{pmatrix} 0\,1\,0 \\ 0\,1\,1 \\ 1\,1\,0 \\ 1\,1\,1 \\ 1\,0\,0 \\ 1\,0\,1 \\ 0\,0\,1 \end{pmatrix} \quad H_{\text{Ham}}^{\text{T(4)}} = \begin{pmatrix} 0\,1\,1 \\ 1\,0\,0 \\ 0\,1\,0 \\ 1\,1\,0 \\ 1\,1\,1 \\ 0\,0\,1 \\ 1\,0\,1 \end{pmatrix} \quad H_{\text{Ham}}^{\text{T(5)}} = \begin{pmatrix} 1\,0\,1 \\ 0\,1\,1 \\ 1\,1\,1 \\ 1\,1\,0 \\ 0\,0\,1 \\ 0\,1\,0 \\ 1\,0\,0 \end{pmatrix} \tag{8.21}$$

and the 0s in the columns by allzero row vectors $(0\,0\,0)$. The resulting transposed parity-check matrix $H^{\mathrm{T}}_{\mathrm{GLDPC}}$ consists of $J = 2$ submatrices

$$H^{\mathrm{T}}_{\mathrm{GLDPC}} = \left(\begin{array}{ccc:ccc}
000 & 011 & 000 & 000 & 000 & 101 \\
100 & 000 & 000 & 000 & 011 & 000 \\
000 & 000 & 101 & 000 & 000 & 011 \\
000 & 000 & 010 & 010 & 000 & 000 \\
000 & 000 & 110 & 000 & 100 & 000 \\
011 & 000 & 000 & 000 & 010 & 000 \\
000 & 000 & 100 & 000 & 000 & 111 \\
000 & 010 & 000 & 011 & 000 & 000 \\
000 & 001 & 000 & 000 & 110 & 000 \\
000 & 101 & 000 & 000 & 111 & 000 \\
101 & 000 & 000 & 000 & 001 & 000 \\
000 & 000 & 111 & 000 & 000 & 110 \\
000 & 100 & 000 & 110 & 000 & 000 \\
000 & 000 & 001 & 000 & 000 & 001 \\
010 & 000 & 000 & 111 & 000 & 000 \\
000 & 111 & 000 & 000 & 000 & 010 \\
000 & 000 & 011 & 100 & 000 & 000 \\
001 & 000 & 000 & 101 & 000 & 000 \\
000 & 110 & 000 & 000 & 000 & 100 \\
111 & 000 & 000 & 000 & 101 & 000 \\
110 & 000 & 000 & 001 & 000 & 000
\end{array}\right) \tag{8.22}$$

Each submatrix has a single nonzero 3-tuple in all of its $N = 21$ rows. The transposed parity-check matrix $H^{\mathrm{T}}_{\mathrm{GLDPC}}$, having irregular structure, defines a GLDPC block code. The design rate of this code is (Problem 8.6)

$$R_d = 1 - \frac{J(\mu - \kappa)}{\mu} = \frac{1}{7} \tag{8.23}$$

In a similar way we can construct a GLDPC block code from a regular (N, J, K) LDPC block code using as constituent codes arbitrary (μ, κ) block codes with $\mu = K$. The parity-check matrix of such a GLDPC block code has, in general, an irregular structure.

More detailed descriptions and analyses of GLDPC block codes are given in [LeZ98, LeZ99].

8.2 LDPC CONVOLUTIONAL CODES

Low-density parity-check convolutional codes are time-varying convolutional codes. In Section 3.6 we introduced time-varying convolutional codes via their time-varying generator matrices. When we study LDPC convolutional codes it is more convenient to use transposed parity-check matrices (syndrome formers); thus, we let the time-varying convolutional code be determined by the set of sequences $\boldsymbol{v}$ satisfying the equation

$$\boldsymbol{v}\boldsymbol{H}^{\mathrm{T}} = \boldsymbol{0} \tag{8.24}$$

where $\boldsymbol{v} = \ldots \boldsymbol{v}_{-1}\, \boldsymbol{v}_0\, \boldsymbol{v}_1 \ldots \boldsymbol{v}_t \ldots$ and where $\boldsymbol{v}_t = v_t^{(1)}\, v_t^{(2)} \ldots v_t^{(c)}$, $v_t^{(i)} \in \mathbb{F}_2$, $i = 1, 2, \ldots, c$, and $\boldsymbol{H}^{\mathrm{T}}$ is a bi-infinite time-varying syndrome former. We have the following straightforward generalization of (2.412) to time-varying syndrome formers:

$$\boldsymbol{v}_t H_0^{\mathrm{T}}(t) + \boldsymbol{v}_{t-1} H_1^{\mathrm{T}}(t) + \cdots + \boldsymbol{v}_{t-m_{\mathrm{s}}} H_{m_{\mathrm{s}}}^{\mathrm{T}}(t) = \mathbf{0}, \quad t \in \mathbb{Z} \tag{8.25}$$

where m_{s} is the memory of the syndrome former and the submatrices

$$H_i^{\mathrm{T}}(t) = \begin{pmatrix} h_i^{(1,1)}(t) & \cdots & h_i^{(1,c-b)}(t) \\ \vdots & & \vdots \\ h_i^{(c,1)}(t) & \cdots & h_i^{(c,c-b)}(t) \end{pmatrix} \tag{8.26}$$

with $0 \leq i \leq m_{\mathrm{s}}$, are the $c \times (c-b)$ entries of the *bi-infinite* syndrome former

$$\boldsymbol{H}^{\mathrm{T}} = \begin{pmatrix} \ddots & \ddots & & \ddots & & & & \\ & H_0^{\mathrm{T}}(t) & H_1^{\mathrm{T}}(t+1) & \ldots & H_{m_{\mathrm{s}}}^{\mathrm{T}}(t+m_{\mathrm{s}}) & & & \\ & & H_0^{\mathrm{T}}(t+1) & H_1^{\mathrm{T}}(t+2) & \ldots & H_{m_{\mathrm{s}}}^{\mathrm{T}}(t+m_{\mathrm{s}}+1) & & \\ & & & \ddots & \ddots & & \ddots & \end{pmatrix} \tag{8.27}$$

To simplify the description of LDPC convolutional codes we use periodically time-varying syndrome formers.

Definition Suppose that there exists a positive integer T such that the submatrices (8.26) of the syndrome former (8.27) satisfy the equalities

$$H_i^{\mathrm{T}}(t) = H_i^{\mathrm{T}}(t+T), \quad i = 0, 1, \ldots, m_s \tag{8.28}$$

for all $t \in \mathbb{Z}$. The convolutional code determined by the section $\boldsymbol{H}_{[0,T)}^{\mathrm{T}}$ of the bi-infinite syndrome former $\boldsymbol{H}_{[-\infty,\infty]}^{\mathrm{T}}$ is called *periodic*. The minimal T for which (8.28) is fulfilled is the *period*.

We assume that $H_0^{\mathrm{T}}(t)$, $t \in \mathbb{Z}$, have full rank and, without loss of generality, that the last $c-b$ rows of $H_0^{\mathrm{T}}(t)$, $t \in \mathbb{Z}$, are linearly independent. Furthermore, we assume that $H_{m_{\mathrm{s}}}^{\mathrm{T}}(t) \neq \mathbf{0}$, at least for one $t \in \mathbb{Z}$. A systematic encoder can be obtained if the first b symbols of $\boldsymbol{v}_t$ coincide with the information symbols, that is, $v_t^{(i)} = u_t^{(i)}$, $i = 1, 2, \ldots, b$. In the sequel we shall use the notation $\boldsymbol{H}_{[t_1,t_2]}^{\mathrm{T}}$ for the part of the syndrome former matrix (8.27) that starts with the $c-b$ columns at time t_1 and ends with the $c-b$ columns at time t_2.

Let $\boldsymbol{H}^{\mathrm{T}}_{[0,T)}$ denote one *period section* of the syndrome former $\boldsymbol{H}^{\mathrm{T}}$, that is, a vertical segment consisting of $(c-b)T$ columns of $\boldsymbol{H}^{\mathrm{T}}$:

$$\boldsymbol{H}^{\mathrm{T}}_{[0,T)} = \begin{pmatrix} H^{\mathrm{T}}_{m_{\mathrm{s}}}(0) & & & \\ \vdots & H^{\mathrm{T}}_{m_{\mathrm{s}}}(1) & & \\ H^{\mathrm{T}}_{1}(0) & \vdots & \ddots & \\ H^{\mathrm{T}}_{0}(0) & H^{\mathrm{T}}_{1}(1) & \ddots & H^{\mathrm{T}}_{m_{\mathrm{s}}}(T-1) \\ & H^{\mathrm{T}}_{0}(1) & \ddots & \vdots \\ & & \ddots & H^{\mathrm{T}}_{1}(T-1) \\ & & & H^{\mathrm{T}}_{0}(T-1) \end{pmatrix} \tag{8.29}$$

Then

$$\boldsymbol{v}_{[-m_{\mathrm{s}},T)}\boldsymbol{H}^{\mathrm{T}}_{[0,T)} = \boldsymbol{0}_{[0,T)} \tag{8.30}$$

where $\boldsymbol{v}_{[-m_{\mathrm{s}},T)}$ is a binary $c(m_{\mathrm{s}}+T)$-tuple and $\boldsymbol{0}_{[0,T)}$ is the $(c-b)T$-tuple of 0s.

The sets of symbols belonging to the rows of the bi-infinite syndrome former $\boldsymbol{H}^{\mathrm{T}}_{[-\infty,\infty]}$ can be compactly written as $(c-b)(m_{\mathrm{s}}+1)$-dimensional binary row vectors $\boldsymbol{h}_n$, $-\infty < n < \infty$, that is,

$$\boldsymbol{H}^{\mathrm{T}}_{[-\infty,\infty]} = \begin{pmatrix} \ddots & & & & \\ & \boldsymbol{h}_{-1} & & & \\ & & \boldsymbol{h}_0 & & \\ & & & \boldsymbol{h}_1 & \\ & & & & \ddots \end{pmatrix} \tag{8.31}$$

Definition Let the rate $R = b/c$ convolutional code $\mathcal{C}$ be defined by its bi-infinite syndrome former (8.31) of memory m_{s}. It is called a *low-density parity-check (LDPC) convolutional code* if the row vectors $\boldsymbol{h}_n$ are sparse for all n, that is, if

$$w_{\mathrm{H}}(\boldsymbol{h}_n) \ll (c-b)m_{\mathrm{s}}, \quad -\infty < n < \infty \tag{8.32}$$

Definition A *regular* (m_{s}, J, K) LDPC convolutional code is an LDPC convolutional code of syndrome memory m_{s}, determined by a sparse syndrome former (8.31), having exactly J 1s in each row, $J \ll (c-b)m_{\mathrm{s}}$, exactly K 1s in each column, $K \ll cm_{\mathrm{s}}$, and 0s elsewhere.

We shall consider regular periodically time-varying $(m_{\mathrm{s}}, J, K, T)$ LDPC convolutional codes, where m_{s} is the syndrome former memory, J is the number of 1s in each row of the syndrome former, K is the number of 1s in each column of the syndrome former, and T is the period of the code. We consider also the ensemble of regular *nonperiodical* (m_{s}, J, K) LDPC convolutional codes. By definition, the period of these codes is $T = \infty$.

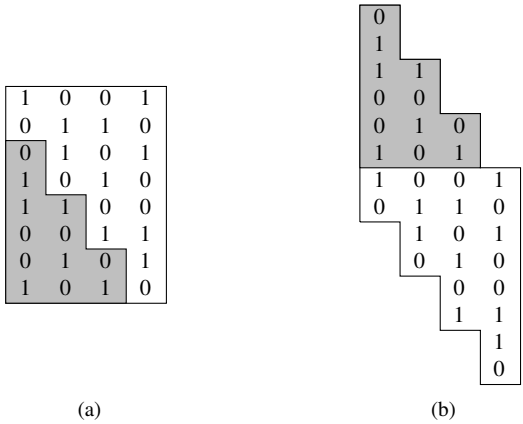


Figure 8.4 Illustration of the unwrapping method to construct the syndrome former of a bi-infinite regular periodically time-varying $(4, 3, 6)$ LDPC convolutional code. Steps (a) and (b).

We begin by describing an "unwrapping method" for constructing bi-infinite LDPC convolutional codes from LDPC block codes. Consider the following:

■ **EXAMPLE 8.4**

In order to construct a bi-infinite rate $R = 1/2$ regular LDPC convolutional code we first build an 8×4 transposed parity-check matrix of a regular $(8, 2, 4)$ LDPC block code, having exactly two 1s in each row and exactly four 1s in each column. This matrix is cut below the diagonal as illustrated in Fig. 8.4(a).

Then we unwrap the part which is below the diagonal as shown in Fig. 8.4(b). After that, four column vectors $(1\,1)^{\mathrm{T}}$ are appended from below (Fig. 8.5(c)). The resulting four columns represent one period of the bi-infinite syndrome former of an LDPC convolutional code. An indefinite sliding repetition of this pattern up to the left and down to the right (Fig. 8.5(d)) leads to a syndrome former $\boldsymbol{H}^{\mathrm{T}}$ with $J = 3$ 1s in each row and $K = 6$ 1s in each column yielding a rate $R = 1/2$ convolutional code with syndrome former memory $m_{\mathrm{s}} = 4$ and period $T = 4$. We say that this syndrome former determines a bi-infinite regular periodically time-varying $(m_{\mathrm{s}}, J, K, T)$ LDPC convolutional code.

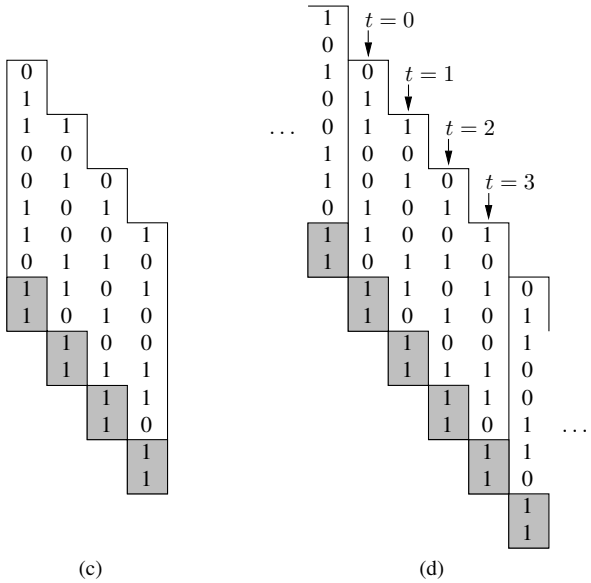


Figure 8.5 Illustration of the unwrapping method to construct the syndrome former of a bi-infinite regular periodically time-varying $(4, 3, 6)$ LDPC convolutional code. Steps (c) and (d).

The segment $\boldsymbol{H}^{\mathrm{T}}_{[0,4)}$ of its syndrome former is

$$\boldsymbol{H}^{\mathrm{T}}_{[0,4)} = \begin{pmatrix} 0 & & & \\ 1 & & & \\ 1 & 1 & & \\ 0 & 0 & & \\ 0 & 1 & 0 & \\ 1 & 0 & 1 & \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ & 1 & 0 & 0 \\ & 1 & 1 & 1 \\ & & 1 & 1 \\ & & 1 & 0 \\ & & & 1 \\ & & & 1 \end{pmatrix} \tag{8.33}$$

Moreover,

$$\boldsymbol{v}_{[-4,4)}\boldsymbol{H}^{\mathrm{T}}_{[0,4)} = \boldsymbol{0}_{[0,4)} \tag{8.34}$$

Next we consider general time-varying LDPC convolutional codes. Let

$$\boldsymbol{u} = \boldsymbol{u}_{[-\infty,\infty]} = \ldots \, \boldsymbol{u}_{-1} \, \boldsymbol{u}_0 \, \boldsymbol{u}_1 \, \ldots \, \boldsymbol{u}_t \, \ldots \tag{8.35}$$

where $\boldsymbol{u}_t = u_t^{(1)} \, u_t^{(2)} \, \ldots \, u_t^{(b)}$ and $u_t^{(i)} \in \mathbb{F}_2$, $i = 1, 2, \ldots, b$, be a bi-infinite information sequence which is mapped by a convolutional encoder of rate $R = b/c$, $b < c$, into the bi-infinite code sequence

$$\boldsymbol{v} = \boldsymbol{v}_{[-\infty,\infty]} = \ldots \, \boldsymbol{v}_{-1} \, \boldsymbol{v}_0 \, \boldsymbol{v}_1 \, \ldots \, \boldsymbol{v}_t \, \ldots \tag{8.36}$$

where $\boldsymbol{v}_t = v_t^{(1)} \, v_t^{(2)} \, \ldots \, v_t^{(c)}$ and $v_t^{(i)} \in \mathbb{F}_2$, $i = 1, 2, \ldots, c$. For convenience, we sometimes use the notation v_{ct+i-1} to denote $v_t^{(i)}$, $i = 1, 2, \ldots, c$.

Consider the following construction of a *systematic* time-varying encoder for an LDPC convolutional code. Let the last $c-b$ rows of $H_0^{\mathrm{T}}(t)$ form the $(c-b) \times (c-b)$ identity matrix. Then the code symbols at time t are determined as (cf. (2.478) where we consider a systematic time-invariant encoder)

$$v_t^{(j)} = u_t^{(j)}, \;\; j = 1, 2, \ldots, b \tag{8.37}$$

$$v_t^{(j)} = \sum_{k=1}^{b} v_t^{(k)} h_0^{(k,j-b)}(t) + \sum_{i=1}^{m_{\mathrm{s}}} \sum_{k=1}^{c} v_{t-i}^{(k)} h_i^{(k,j-b)}(t), \;\; j = b+1, b+2, \ldots, c \tag{8.38}$$

The encoder for this convolutional code can be implemented by c shift registers of length m_{s} with time-varying tap-weights corresponding to the matrix entries $h_i^{(k,j-b)}(t)$. It is a *syndrome former* realization of a convolutional encoder. The overall constraint length of the encoder, that is, the number of binary symbols which the encoder keeps in its memory, equals cm_{s}.

Such a shift-register realization of a systematic rate $R = 1/2$ encoder for the time-varying LDPC code given in Example 8.4 is shown in Fig. 8.6 for time instants $t = 0$ modulo T, where $T = 4$. It is based on a syndrome former $\boldsymbol{H}_{[-\infty,\infty]}^{\mathrm{T}}$ with memory $m_{\mathrm{s}} = 4$. The taps are given by the first column of (8.33). The next three columns determine the taps for the remaining time instants $t = 1, 2, 3$ modulo T.

We have described a particular syndrome former realization of the encoder. Another, less complex, realization of an LDPC convolutional encoder is the *partial syndrome* realization [PJS08].

A partial-syndrome encoder keeps at time t in its memory the *partial syndrome*

$$\boldsymbol{p}_t = \boldsymbol{p}_{t1} \, \boldsymbol{p}_{t2} \, \cdots \, \boldsymbol{p}_{tm_{\mathrm{s}}} \tag{8.39}$$

where $\boldsymbol{p}_{ti} = p_{ti}^{(1)} \, p_{ti}^{(2)} \, \ldots \, p_{ti}^{(c-b)}$, $i = 1, 2, \ldots, m_{\mathrm{s}}$, that is, only $(c-b)m_{\mathrm{s}}$ bits instead of cm_{s} bits for the syndrome encoder. We consider $\boldsymbol{p}_t$ to be the *state* of the partial-syndrome encoder at time t.

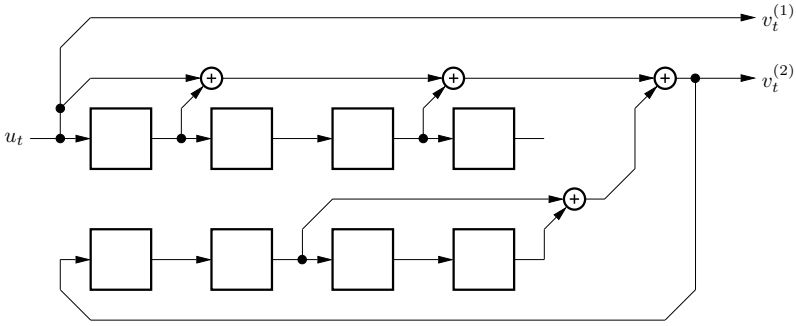


Figure 8.6 Realization of a systematic rate $R = 1/2$ syndrome former encoder for the LDPC convolutional code specified by (8.33). The taps are shown for the time instants $t = 0$ modulo T.

To obtain a recursion for the partial syndrome we truncate the bi-infinite syndrome former $\boldsymbol{H}^{\mathrm{T}}$ (8.27) after the row that starts with $H_0^{\mathrm{T}}(t)$ at time t, that is, we obtain

$$\boldsymbol{H}_{\mathrm{tr}}^{\mathrm{T}} = \begin{pmatrix} \ddots & H_{m_\mathrm{s}}^{\mathrm{T}}(t-1) & & & & & \\ \ddots & \vdots & H_{m_\mathrm{s}}^{\mathrm{T}}(t) & & & & \\ \ddots & \vdots & \dots & H_{m_\mathrm{s}}^{\mathrm{T}}(t+1) & & & \\ \ddots & H_1^{\mathrm{T}}(t-1) & H_2^{\mathrm{T}}(t) & \dots & \ddots & & \\ \ddots & H_0^{\mathrm{T}}(t-1) & H_1^{\mathrm{T}}(t) & H_2^{\mathrm{T}}(t+1) & \dots & H_{m_\mathrm{s}}^{\mathrm{T}}(t+m_\mathrm{s}-1) & \\ & & H_0^{\mathrm{T}}(t) & H_1^{\mathrm{T}}(t+1) & \dots & \dots & H_{m_\mathrm{s}}^{\mathrm{T}}(t+m_\mathrm{s}) \end{pmatrix} \tag{8.40}$$

Consider the triangle in (8.40). At time t we define the partial syndrome $\boldsymbol{p}_t = \boldsymbol{p}_{t1}\,\boldsymbol{p}_{t2}\,\cdots\,\boldsymbol{p}_{tm_\mathrm{s}}$ as

$$\begin{aligned} \boldsymbol{p}_{t1} &= \boldsymbol{v}_{t-1}H_1^{\mathrm{T}}(t) + \boldsymbol{v}_{t-2}H_2^{\mathrm{T}}(t) + \cdots + \boldsymbol{v}_{t-m_\mathrm{s}}H_{m_\mathrm{s}}^{\mathrm{T}}(t) \\ \boldsymbol{p}_{t2} &= \boldsymbol{v}_{t-1}H_2^{\mathrm{T}}(t+1) + \boldsymbol{v}_{t-2}H_3^{\mathrm{T}}(t+1) + \cdots + \boldsymbol{v}_{t-m_\mathrm{s}}H_{m_\mathrm{s}}^{\mathrm{T}}(t+1) \\ &\vdots \\ \boldsymbol{p}_{tm_\mathrm{s}} &= \boldsymbol{v}_{t-1}H_{m_\mathrm{s}}^{\mathrm{T}}(t+m_\mathrm{s}-1) \end{aligned} \tag{8.41}$$

If we slide the triangle one step back to time $t-1$, its bottom row will start with $H_1^{\mathrm{T}}(t-1)$. Then we obtain its second component of the partial syndrome $\boldsymbol{p}_{t-1}$ as

$$\boldsymbol{p}_{(t-1)2} = \boldsymbol{v}_{t-2}H_2^{\mathrm{T}}(t) + \boldsymbol{v}_{t-3}H_3^{\mathrm{T}}(t) + \cdots + \boldsymbol{v}_{t-m_\mathrm{s}}H_{m_\mathrm{s}}^{\mathrm{T}}(t) \tag{8.42}$$

Comparing (8.41) and (8.42) yields

$$\boldsymbol{p}_{t1} = \boldsymbol{v}_{t-1}H_1^{\mathrm{T}}(t) + \boldsymbol{p}_{(t-1)2} \tag{8.43}$$

We conclude that, in general, we have the following recursion:

$$\boldsymbol{p}_{ti} = \begin{cases} \boldsymbol{v}_{t-1}H_i^{\mathrm{T}}(t+i-1) + \boldsymbol{p}_{(t-1)(i+1)}, & t \in \mathbb{Z}^+, 1 \le i < m_\mathrm{s} \\ \boldsymbol{v}_{t-1}H_{m_\mathrm{s}}^{\mathrm{T}}(t+m_\mathrm{s}-1), & t \in \mathbb{Z}^+, i = m_\mathrm{s} \end{cases} \tag{8.44}$$

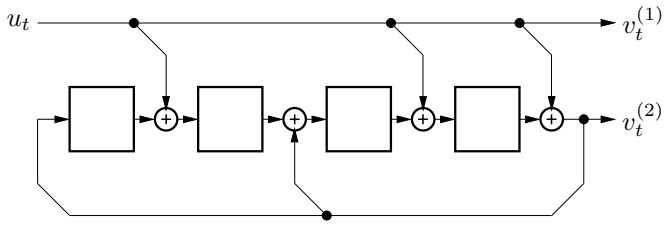


Figure 8.7 Realization of a systematic $R = 1/2$ partial-syndrome former encoder for a LDPC convolutional code specified by (8.33). The taps are shown for time instants $t = 0$ modulo T.

If we begin the encoding at time $t = 0$, say, then $\boldsymbol{p}_0$ is set equal to the allzero $(c-b)m_{\mathrm{s}}$-tuple.

Introduce the notation $\boldsymbol{v}_t = \boldsymbol{v}_t^{(1)}\,\boldsymbol{v}_t^{(2)}$ where $\boldsymbol{v}_t^{(1)} = \boldsymbol{u}_t$ is the subblock of b information symbols and $\boldsymbol{v}_t^{(2)}$ is the subblock of $c-b$ parity symbols and assume that the partial-syndrome former encoder in the systematic realization is in state $\boldsymbol{p}_t$ at time t. Then the subblock $\boldsymbol{v}_t^{(1)}$ enters the encoder. From (8.25) and (8.41) it follows that the subblock $\boldsymbol{v}_t^{(2)}$ can be obtained from the equation

$$\boldsymbol{v}_t H_0^{\mathrm{T}}(t) = \left(\boldsymbol{v}_t^{(1)}\,\boldsymbol{v}_t^{(2)}\right) H_0^{\mathrm{T}}(t) = \boldsymbol{p}_{t1} \tag{8.45}$$

Since the last $c-b$ rows of $H_0^{\mathrm{T}}(t)$ form the $(c-b)\times(c-b)$ identity matrix, the code symbols at time t can be determined as (cf. (2.488), which is the time-invariant counterpart)

$$v_t^{(j)} = u_t^{(j)}, \qquad j = 1, 2, \ldots, b \tag{8.46}$$

$$v_t^{(j)} = \sum_{k=1}^{b} v_t^{(k)} h_0^{(k,j-b)}(t) + p_{t1}^{(j-b)}, \quad j = b+1, b+2, \ldots, c-b \tag{8.47}$$

A partial-syndrome encoder can be implemented using $c-b$ shift registers, each with m_{s} memory units; see Fig. 8.7 for a partial-syndrome former encoder implementation of the syndrome former encoder shown in Fig. 8.6. The overall constraint length of the encoder is $(c-b)m_{\mathrm{s}}$. Thus, the encoder memory complexity of partial-syndrome realizations is in general less than that of the syndrome former realization.

The memory complexity grows linearly with the syndrome former memory m_{s} while the number of operations (multiplications and additions) per encoded information symbol grows linearly with K but it does not depend on m_{s} [PJS08].

In Example 8.4 we have considered the vertical unwrapping construction of the bi-infinite syndrome former of a time-varying LDPC convolutional code. Jimenez and Zigangirov [JiZ99] used the horizontal unwrapping method described in Fig. 8.8(a)–(d) when they introduced LDPC convolutional codes. In Fig. 8.8(d) we have shown the horizontal segment $\overrightarrow{\boldsymbol{H}}_{[0,T)}^{\mathrm{T}}$ of the syndrome former for a *semi-infinite* LDPC

convolutional code with period $T = 4$. Let the submatrix $\overrightarrow{\boldsymbol{H}}^{\mathrm{T}}_{[0,T)}$ consisting of cT rows be a horizontal segment of the semi-infinite syndrome former $\boldsymbol{H}^{\mathrm{T}}$. Then

$$\boldsymbol{v}_{[0,T)}\overrightarrow{\boldsymbol{H}}^{\mathrm{T}}_{[0,T)} = \mathbf{0}_{[0,T)}\boldsymbol{p}_T \tag{8.48}$$

where $\mathbf{0}_{[0,T)}$ is the $(c-b)T$-tuple of 0s and $\boldsymbol{p}_T$ is a binary $(c-b)m_{\mathrm{s}}$-tuple called the *partial* syndrome of $\boldsymbol{v}_{[0,T)}$.

Later we will analyze semi-infinite LDPC convolutional codes and will show that they have better performances than the corresponding LDPC block codes. Notice that the vertical and horizontal unwrapping methods result in LDPC convolutional codes that are shifted versions (one time unit) of each other.

The syndrome formers of LDPC convolutional codes constructed above can be considered as counterparts of the transposed parity-check matrices of LDPC block codes from the ensemble $\mathcal{B}_1(N, J, K)$ introduced in Section 8.1. Now we consider a construction of regular periodically time-varying LDPC convolutional codes, which is a convolutional counterpart of the LDPC block codes from the ensemble $\mathcal{B}_2(N, J, K)$ with the transposed parity-check matrix (8.10).

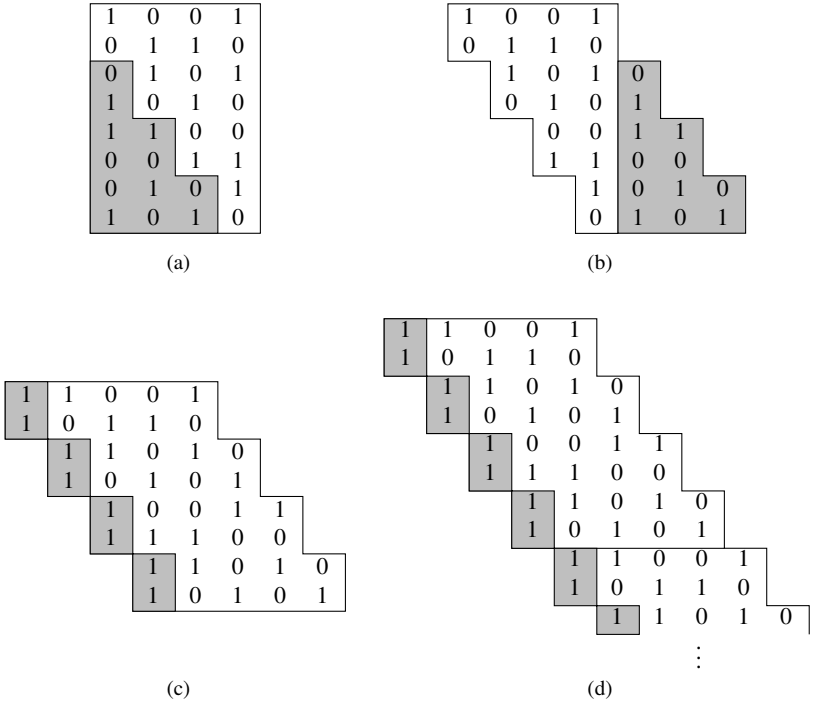


Figure 8.8 Illustration of the horizontal unwrapping method to construct the syndrome former of a semi-infinite regular periodically time-varying $(4, 3, 6)$ LDPC convolutional code.

In Fig. 8.9 we show one period of the syndrome former $\boldsymbol{H}^{\mathrm{T}}_{[0,T-1]}$ for a periodically time-varying convolutional code. The squares are $M \times M$ permutation matrices. In terms of the syndrome former (8.27) with submatrices (8.26) this code has the parameters $m_{\mathrm{s}} = 2$, $b = M$, and $c = 2M$. The design rate is $R_{\mathrm{d}} = M/2M$ and each $2M \times M$ matrix $H_i^{\mathrm{T}}(t+i)$ (8.26) consists of two $M \times M$ permutation matrices $P_i^{(1)}(t+i)$ and $P_i^{(2)}(t+i)$, that is,

$$H_i^{\mathrm{T}}(t+i) = \begin{pmatrix} P_i^{(1)}(t+i) \\ P_i^{(2)}(t+i) \end{pmatrix}, \quad i = 0, 1, 2, \quad t = 0, 1, \ldots, T-1 \tag{8.49}$$

The matrices (8.49) have full rank equal to M. The parameters of a regular periodically time-varying $(m_{\mathrm{s}}, J, K, T)$ LDPC convolutional code with its syndrome former given in Fig. 8.9 are $m_{\mathrm{s}} = J - 1 = 2$, $J = 3$, and $K = 6$. We assume that the first M symbols, $v_{2Mt}, v_{2Mt+1}, \ldots, v_{2Mt+M-1}$, of the subblock $\boldsymbol{v}_t = v_{2Mt}\, v_{2Mt+1} \ldots v_{2M(t+1)-1}$ are information symbols and the last M symbols, $v_{2Mt+M}\, v_{2Mt+M+1} \ldots v_{2M(t+1)-1}$, are parity symbols, that is, we use systematic encoding.

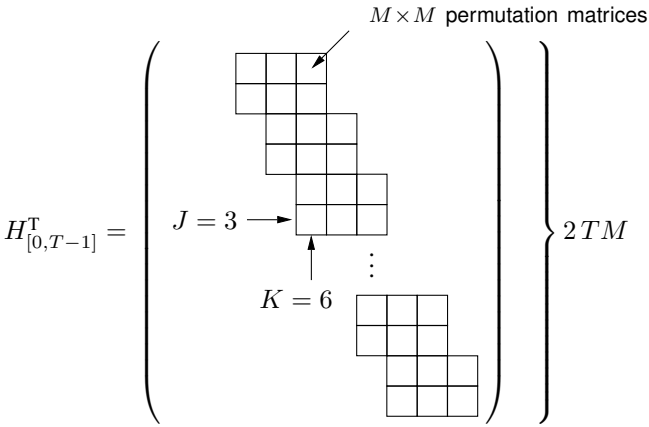


Figure 8.9 One period of a syndrome former for regular periodically time-varying LDPC convolutional codes from the ensemble $\mathcal{C}(M, J, K, T)$.

Consider one period of the syndrome former $\boldsymbol{H}^{\mathrm{T}}_{[0,T-1]}$ as presented in Fig. 8.9. If we would choose the permutation matrices $P_i^{(k)}(t+i)$, $k = 1, 2$, $i = 0, 1, \ldots, m_{\mathrm{s}}$, and $t = 0, 1, \ldots, T-1$, randomly, independently from each other such that all $M!$ values are equiprobable, we obtain the ensemble $\mathcal{C}(M, J, K, T)$ of regular periodically time-varying LDPC convolutional codes. We consider also the ensemble $\mathcal{C}(M, J, K)$ of regular nonperiodical LDPC convolutional codes (that is, $T = \infty$). We shall use these ensembles for analyses of the free distance and the thresholds of regular LDPC convolutional codes.

■ **EXAMPLE 8.5**

Consider one period of the syndrome former constructed from 3×3 permutation matrices with period $T = 1$,

$$\boldsymbol{H}^{\mathrm{T}}_{[0,0]} = \left(\begin{array}{ccc|ccc|ccc} 0&1&0&1&0&0&0&1&0\\ 0&0&1&0&0&1&0&0&1\\ 1&0&0&0&1&0&1&0&0\\ \hline 0&0&1&1&0&0&1&0&0\\ 0&1&0&0&0&1&0&1&0\\ 1&0&0&0&1&0&0&0&1 \end{array}\right) \tag{8.50}$$

This syndrome former has the following parameters: $M = 3$, $b = 3$, $c = 6$, $R_{\mathrm{d}} = 3/6$, $m_{\mathrm{s}} = 2$, and $T = 1$. It defines a regular $(m_{\mathrm{s}} = 2, J = 3, K = 6, T = 1)$ LDPC convolutional code. By permuting the rows in (8.50) we obtain the following syndrome former:

$$\boldsymbol{H}'^{\mathrm{T}}_{[0,0]} = \left(\begin{array}{ccc|ccc|ccc} 1&0&0&0&1&0&1&0&0\\ 0&1&0&1&0&0&0&1&0\\ 0&0&1&0&0&1&0&0&1\\ \hline 1&0&0&0&1&0&0&0&1\\ 0&1&0&0&0&1&0&1&0\\ 0&0&1&1&0&0&1&0&0 \end{array}\right) \tag{8.51}$$

By further permutation of the rows we obtain the period $T = 3$ syndrome former

$$\boldsymbol{H}''^{\mathrm{T}}_{[0,2]} = \left(\begin{array}{c|c|c|c|c|c|c|c|c|c|c|c} 1&0&0&0&1&0&1&0&0 & & & \\ 1&0&0&0&1&0&0&0&1 & & & \\ \hline & 1&0&1&0&0&0&1&0&0 & & \\ & 1&0&0&0&1&0&1&0&0 & & \\ \hline & & 1&0&0&1&0&0&1&0&0 & \\ & & 1&1&0&0&1&0&0&0&0 & \end{array}\right) \tag{8.52}$$

The syndrome formers (8.50), (8.51), and (8.52) define equivalent codes. The syndrome former (8.52) has the following parameters: memory $m_{\mathrm{s}} = 8$, period $T = 3$, $b = 1$, $c = 2$, and $R_{\mathrm{d}} = 1/2$. It defines a regular periodically time-varying ($m_{\mathrm{s}} = 8$, $J = 3$, $K = 6$, $T = 3$) LDPC convolutional code.

Note that in general, by permuting rows of the syndrome former in Fig. 8.9, an equivalent design rate $R_{\mathrm{d}} = 1/2$ regular periodically time-varying LDPC convolutional code can be obtained (see [STL07]). The syndrome memory is upper-bounded by the inequality $m_{\mathrm{s}} \leq JM - 1$.

There are two realizations of encoders for codes in Fig. 8.9. These are the syndrome former realization and the partial-syndrome realization. How can we give a meaningful definition of the overall constraint length of these encoders? Since in practice information symbols enter the encoder symbolwise, not by large blocks of length M, we define the overall constraint length of the encoder in syndrome former

realization as $\nu = cm_{\mathrm{s}} \leq c(JM-1)$. Similarly, the overall constraint length of the partial-syndrome realization of the encoder is defined as $\nu = m_{\mathrm{s}} \leq JM-1$.

This theory can be generalized to arbitrary codes from ensemble $\mathcal{C}(M, J, K, T)$. Let a be the greatest common divisor of J and K, that is, $a = \gcd(J, K)$. Then codes from this ensemble are equivalent to regular periodical (m_{s}, J, K) LDPC convolutional codes with syndrome memory $m_{\mathrm{s}} \leq JM-1$. The design rate of the code is $R_{\mathrm{d}} = b/c$, where $b = (K-J)/a$, $c = K/a$. It can be shown that the overall constraint length of the encoder in syndrome former realization and in partial-syndrome realization are upper-bounded as $\nu \leq K(JM-1)/a$ and $\nu \leq (K-J)(JM-1)/a$, respectively.

In order to simplify our analysis we consider also tailbiting LDPC codes. Then we can reduce our analysis of the free distance of an LDPC convolutional code to the analysis of a block code. In Fig. 8.10 we show the transposed parity-check matrix $\tilde{H}^{\mathrm{T}}_{[0,N-1]}$ of a tailbiting LDPC code. It is constructed from a period of length T $\boldsymbol{H}^{\mathrm{T}}_{[0,T-1]}$ of the syndrome former of a regular periodically time-varying LDPC convolutional code given in Fig. 8.9. For this construction we used the "wraparound" method described in Section 4.8.

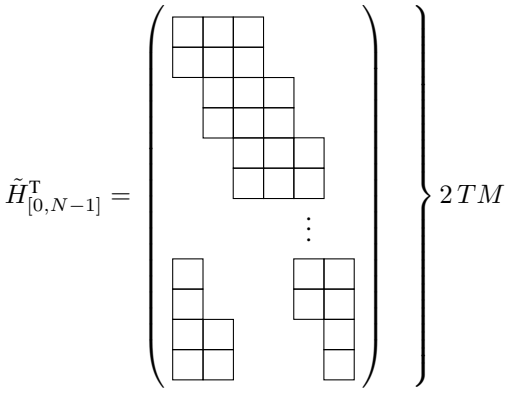


Figure 8.10 The transposed parity-check matrix of a tailbiting LDPC convolutional code of block length $N = 2TM$.

It is convenient to use the computational tree also for describing and analyzing LDPC convolutional codes. We can construct such a computational tree similarly to the one we constructed for the LDPC block code in Fig. 8.2. If the LDPC convolutional code is a regular (m_{s}, J, K) LDPC convolutional code and the codewords are bi-infinite, then the computational tree is also regular as the one given in Fig. 8.2. That means that a clan head has J families, the other clan members have $J-1$ families, and in each family there are $K-1$ children. But if the convolutional encoder starts to operate from the allzero state and ends also in the allzero state, then the computational tree becomes irregular, which means that the number of children varies between the families. As we shall see in Section 8.6, this irregularity of the

tree causes an essential improvement of the performance in comparison with the corresponding regular LDPC block codes.

All definitions related to the computational tree for regular LDPC convolutional codes are similar to the definitions introduced in Section 8.1 for regular LDPC block codes. Important examples are the definitions of the $2\ell_0$-cyclefree clan and the $2\ell_0$-cyclefree code. A lower bound on the parameter ℓ_0 characterizing the number of independent iterations during belief propagation decoding of regular LDPC convolutional codes was derived by Truhachev [Tru04]. It states that there exists a regular periodically time-varying $(m_{\rm s}, J, K, T)$ LDPC convolutional code with parameter ℓ_0 satisfying the inequality

$$\ell_0 > \frac{\log(m_{\rm s}+1)}{2\log((J-1)(K-1))} - \gamma \tag{8.53}$$

where γ is a constant depending on J and K but not on $m_{\rm s}$. This bound is similar to the bound (8.16) for (N, J, K) LDPC block codes.

8.3 BLOCK AND CONVOLUTIONAL PERMUTORS

An essential building block used in the construction of various LDPC codes, considered in this chapter, as well as of turbo codes, considered in Chapter 9, is the *permutor*. In this section we define the single and multiple block permutors and the single and multiple convolutional permutors. We introduce their parameters and discuss methods for their construction and implementation. We shall give two differerent descriptions of permutors, namely, the matrix and array descriptions. Finally we describe the multiple Markov permutor, which is a useful tool for analyzing distance properties.

Definition An $N \times N$ square matrix $P = (p_{nl})$, $n, l = 0, \ldots, N-1$, having one 1 in each row, one 1 in each column, and 0s elsewhere, is called *permutation matrix* or *single block permutor*.

There are $N!$ permutation matrices of size $N \times N$. A permutation matrix can be constructed from the $N \times N$ diagonal matrix by permuting the columns or the rows.

Let the N-dimensional vector $\boldsymbol{x} = (x_0\, x_1 \ldots x_{N-1})$ be an input of single permutor P. Then the vector $\boldsymbol{y} = (y_0\, y_1 \ldots y_{N-1})$,

$$\boldsymbol{y} = \boldsymbol{x}P \tag{8.54}$$

is the output of the permutor P.

Next we introduce *multiple block permutors (MBPs)*.

Definition An $N \times L$ binary matrix $P = (p_{nl})$, $n = 0, \ldots, N-1$, $l = 0, \ldots, L-1$, describes a multiple block (N, J, K)-permutor if it has J 1s in each row, K 1s in each column, such that $N/K = L/J$, and 0s elsewhere. The pair (J, K) is called the permutor's *multiplicity*.

If $L = N$, $J = K$, then the permutor is called *symmetric*.

It is important to emphasize that, in contrast to operations of single permutors, operations of multiple permutors can in general not be represented as the product of the input vector $\boldsymbol{x}$ and a permutation matrix P.

To describe the operation of multiple permutors we use, in addition to the matrix representation $P = (p_{nl})$, the array representation. In this representation, the MBP is regarded as a *memory array*, where each nonzero element in P represents a memory cell that can store an input symbol. Therefore, each column of this array can store K symbols and each row can store J symbols. The input sequence is the sequence of N J-dimensional column vectors $\boldsymbol{x}_n$, $n = 0, 1, \ldots, N-1$, where

$$\boldsymbol{x}_n = (x_{n0}\, x_{n1} \ldots x_{n(J-1)})^{\mathrm{T}} \tag{8.55}$$

These N column vectors can be represented as the $J \times N$ matrix $\boldsymbol{X} = (\boldsymbol{x}_0\, \boldsymbol{x}_1 \ldots \boldsymbol{x}_{N-1})$.

When an input sequence enters the permutor, the symbols of the N J-dimensional columns vectors are placed into the array memory positions row by row, starting from the top of the array. The output sequence is a sequence of L K-dimensional column vectors $\boldsymbol{y}_l$, where

$$\boldsymbol{y}_l = (y_{l0}\, y_{l1}, \ldots y_{l(K-1)})^{\mathrm{T}} \tag{8.56}$$

which can be represented as the $K \times L$ matrix $\boldsymbol{Y} = (\boldsymbol{y_0}\, \boldsymbol{y_1} \ldots \boldsymbol{y}_{L-1})$.

The MBP generates the output sequence $\boldsymbol{Y}$ by reading out the contents of the array memory positions column by column, starting from the left of the array. The multiple permutation operator $\circledast$ is defined by

$$\boldsymbol{Y} = \boldsymbol{X} \circledast P \tag{8.57}$$

In the inverse MBP, the symbols are inserted into the memory cells columnwise and read out rowwise. Thus, the inverse of a multiple block permutor is described by the transpose matrix P^{T}, that is,

$$\boldsymbol{X} = \boldsymbol{Y} \circledast P^{\mathrm{T}} \tag{8.58}$$

An MBP can be constructed by using operations of *rowwise* and *columnwise interleaving*:

Definition Consider K matrices $P^{(k)} = (p_{nl}^{(k)})$, $k = 1, 2, \ldots, K$, $n = 0, 1, \ldots, N-1$, and $l = 0, 1, \ldots, L-1$, of size $N \times L$. Then the matrix $P = (p_{nl})$, $n = 0, 1, \ldots, KN-1$, $l = 0, 1, \ldots, L-1$, of size $KN \times L$, where

$$p_{(nK+k-1)l} = p_{nl}^{(k)} \tag{8.59}$$

is called a *rowwise interleaving* of the matrix set $\{P^{(1)}, P^{(2)}, \ldots, P^{(K)}\}$ and denoted

$$P = \boxminus(P^{(1)}, P^{(2)}, \ldots, P^{(K)}) \tag{8.60}$$

Definition Consider J matrices $P^{(j)} = (p_{nl}^{(j)})$, $j = 1, 2, \ldots, J$, $n = 0, 1 \ldots, N-1$, and $l = 0, 1, \ldots, L-1$, of size $N \times L$. Then, the matrix $P = (p_{nl})$, $n = 0, 1, \ldots, N-1$, $l = 0, 1, \ldots, JL-1$, of size $N \times JL$ such that

$$p_{n(lJ+j-1)} = p_{nl}^{(j)} \tag{8.61}$$

is called a *columnwise interleaving* of the matrix set $\{P^{(1)}, P^{(2)}, \ldots, P^{(J)}\}$ and denoted

$$P = \boxplus(P^{(1)}, P^{(2)}, \ldots, P^{(J)}) \tag{8.62}$$

We can construct an $N \times L$, where $N = KM$, $L = JM$, multiple block permutor P of multiplicity (J, K) from a set of JK conventional permutation matrices, $P^{(kj)}$, $1 \le k \le K, 1 \le j \le J$, of size $M \times M$, as follows: First, using rowwise interleaving, we construct from the set of size $M \times M$ matrices $\{P^{(j1)}, P^{(j2)}, \ldots, P^{(jK)}\}$, $j = 1, 2, \ldots, J$, a set of $KM \times M$ matrices $P^{(j)}$, $j = 1, \ldots, J$,

$$P^{(j)} = \boxminus(P^{(j1)}, P^{(j2)}, \ldots, P^{(jK)}) \tag{8.63}$$

Then, by columnwise interleaving of the matrix set $\{P^{(1)}, P^{(2)}, \ldots, P^{(J)}\}$, we construct the final matrix

$$P = \boxplus\left(P^{(1)}, P^{(2)}, \ldots, P^{(J)}\right) \tag{8.64}$$

■ **EXAMPLE 8.6**

Using four 2×2 conventional permutation matrices,

$$P^{(11)} = P^{(12)} = P^{(22)} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \quad P^{(21)} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

we obtain after the rowwise interleaving step

$$P^{(1)} = \boxminus\left(P^{(11)}, P^{(12)}\right) = \begin{pmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \end{pmatrix}$$

$$P^{(2)} = \boxminus\left(P^{(21)}, P^{(22)}\right) = \begin{pmatrix} 0 & 1 \\ 1 & 0 \\ 1 & 0 \\ 0 & 1 \end{pmatrix}$$

After the columnwise interleaving step, the following symmetric MBP with multiplicity $(2, 2)$ is obtained:

$$P = \boxplus\left(P^{(1)}, P^{(2)}\right) = \begin{pmatrix} 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{pmatrix}$$

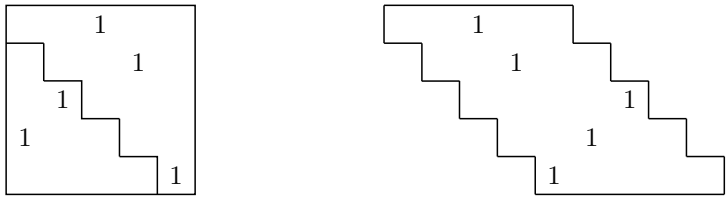


Figure 8.11 Unwrapping a 5×5 matrix.

Other multiple block permutors with multiplicity $(2, 2)$ are obtained using different initial conventional permutation matrices.

Definition A bi-infinite matrix $\boldsymbol{P} = (p_{nl})$, $n, l \in \mathbb{Z}$, that has one 1 in each row and one 1 in each column and that satisfies the causality condition

$$p_{nl} = 0, \quad n > l \tag{8.65}$$

is called a *single convolutional permutor*.

Let $\boldsymbol{x} = \ldots x_0 x_1 \ldots x_n \ldots$ denote the bi-infinite binary-input sequence of a single convolutional permutor and let $\boldsymbol{y} = \ldots y_0 y_1 \ldots y_l \ldots$,

$$\boldsymbol{y} = \boldsymbol{x}\boldsymbol{P} \tag{8.66}$$

denote the corresponding output sequence. The convolutional permutor permutes the symbols in the input sequence.

The *identity convolutional permutor*, whose matrix $\boldsymbol{P}$ has 1s only along the diagonal, that is, $p_{nn} = 1$, $n \in \mathbb{Z}$, is a special case of a single convolutional permutor. If

$$p_{n(n+\delta)} = 1, \quad n \in \mathbb{Z} \tag{8.67}$$

then the matrix $\boldsymbol{P}$ is a *delay permutor* with *delay* δ. The identity permutor is a delay permutor with delay $\delta = 0$.

For construction of convolutional permutors from block permutors we can use the unwrapping procedure described in Section 8.2.

■ **EXAMPLE 8.7**

Suppose that we have a 5×5 matrix $\boldsymbol{P}$ for a single block permutor (see Fig. 8.11). First, we cut this matrix just below its main diagonal. Next, we unwrap the submatrix which is below the diagonal.

Continuing the construction of convolutional permutor we repeat indefinitely the unwrapped matrix as shown in Fig. 8.12, where we also have shifted the matrix one position to the right and introduced an additional delay of one symbol by adding a diagonal of only 0s. Thus we avoid that an input symbol appears directly at the output. (All blanks denote 0s; the 0s along the diagonal are the only 0s actually written as 0s in the figure.)

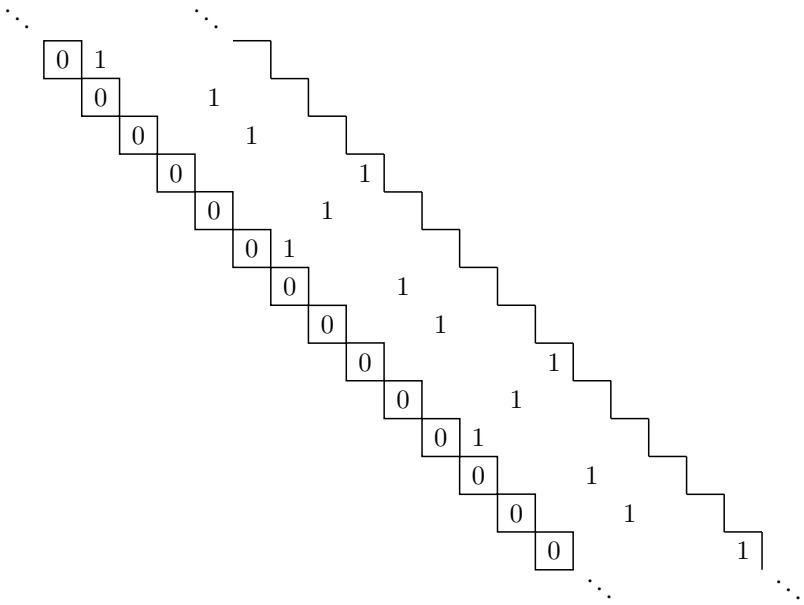


Figure 8.12 A single convolutional permutor obtained from the unwrapped matrix in Fig. 8.11.

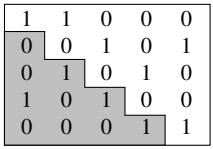
The unwrapping method can be generalized and used for construction of *multiple convolutional permutors (MCPs)* of multiplicity (J, K) from multiple block permutors of multiplicity (J, K). The following example illustrates the construction of a matrix $\boldsymbol{P}$ describing a symmetric multiple convolutional permutor.

■ **EXAMPLE 8.8**

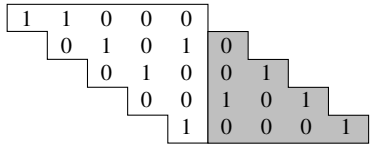
In Fig. 8.13 we present a construction of an MCP with multiplicity $J = K = 2$ and period $T = 5$. For the construction we use the unwrapping procedure.

First, we choose a symmetric 5×5 block permutor of multiplicity $(J = K = 2)$ (Fig. 8.13(a)). Then we cut this matrix just below its main diagonal. Next, the lower and upper parts are unwrapped (Fig. 8.13(b)) and repeated indefinitely to yield the multiple convolutional permutor of multiplicity $(J = K = 2)$ and period $T = 5$ as shown in Fig. 8.13(c).

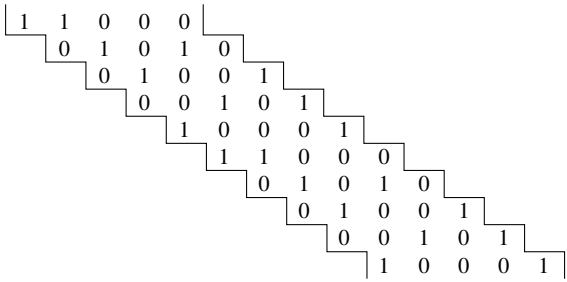
Next we define a general MCP.



(a) Multiple block permutor.



(b) Unwrapped block permutor.



(c) Multiple convolutional permutor.

Figure 8.13 Construction of a multiple periodic convolutional permutor.

Definition Consider the bi-infinite binary matrix

$$\boldsymbol{P} = \begin{pmatrix} \ddots & & \ddots & & & \\ p_{n(n+\delta)} & p_{n(n+\delta+1)} & \cdots & p_{n(n+\Delta)} & & \\ \ddots & p_{(n+1)(n+\delta+1)} & p_{(n+1)(n+\delta+2)} & \cdots & p_{(n+1)(n+\Delta+1)} & \\ \ddots & \ddots & p_{(n+2)(n+\delta+2)} & p_{(n+2)(n+\delta+3)} & \cdots & p_{(n+2)(n+\Delta+2)} \\ & & \ddots & & \ddots & \ddots \end{pmatrix} \tag{8.68}$$

The matrix (8.68) describes a *multiple convolutional permutor* with minimum delay $\delta \geq 0$, maximum delay $\Delta \geq \delta$, and multiplicity (J, K) if

- For all $l < n + \delta$, $n, l \in \mathbb{Z}$, the entries p_{nl} are equal to zero.
- For all $l > n + \Delta$, $n, l \in \mathbb{Z}$, the entries p_{nl} are equal to zero.
- For at least one n, $n \in \mathbb{Z}$, the entry $p_{n(n+\delta)}$ is nonzero.
- For at least one n, $n \in \mathbb{Z}$, the entry $p_{n(n+\Delta)}$ is nonzero.

- Each row of the matrix $\boldsymbol{P}$ has exactly J 1s.
- Each column of the matrix $\boldsymbol{P}$ has exactly K 1s.

The value $w = \Delta - \delta + 1$ is called the *width* of the convolutional permutor. If $J = K$, the permutor is symmetric.

Definition An MCP of multiplicity (J, K) is *periodic* if there is a positive integer T such that

$$p_{nl} = p_{(n+KT)(l+JT)}, \quad n, l \in \mathbb{Z} \tag{8.69}$$

The minimum T for which (8.69) is fulfilled is called the *period* of the multiple convolutional permutor.

Similarly to the MBP case, additionally to the matrix representation of an MCP we will use an array representation. In this representation, each nonzero element in $\boldsymbol{P}$ represents a memory cell that can store an input symbol. Each column of this memory array can store K symbols and each row J symbols. The input sequence is a bi-infinite sequence of J-dimensional column vectors, which can be represented as a bi-infinite matrix $\boldsymbol{X} = (\ldots \boldsymbol{x}_0 \, \boldsymbol{x}_1 \ldots \boldsymbol{x}_n \ldots)$, where $\boldsymbol{x}_n$ is defined by (8.55). When an input sequence enters the permutor, the symbols of the J-dimensional column vectors are placed into the memory array positions row by row.

The output sequence is a bi-infinite sequence of K-dimensional column vectors which can be represented as a bi-infinite matrix $\boldsymbol{Y} = (\ldots \boldsymbol{y}_0 \, \boldsymbol{y}_1 \ldots \boldsymbol{y}_l \ldots)$, where $\boldsymbol{y}_l$ is defined by (8.56). The MCP generates the output sequence $\boldsymbol{Y}$ by reading out the contents of the memory array positions column by column. The multiple convolutional permutation operator $\circledast$ is defined by

$$\boldsymbol{Y} = \boldsymbol{X} \circledast \boldsymbol{P} \tag{8.70}$$

In the inverse permutor, the symbols are placed into the memory cells columnwise and read out rowwise. Thus, the inverse of a symmetric multiple block permutor is described by the transpose matrix $\boldsymbol{P}^{\mathrm{T}}$.

The parameters δ, Δ, and w are important for characterizing a multiple convolutional permutor. Another important parameter is its *overall constraint length* ν. By definition, the overall constraint length ν of a permutor is equal to the maximum number of symbols that are stored in an array realization of the permutor. Particularly, to define formally the overall constraint length of a symmetric permutor $(J = K)$ we introduce the set

$$\mathcal{P}_t = \{p_{nl} \mid n \le t, \quad l > t\}, \quad n, l, t \in \mathbb{Z} \tag{8.71}$$

Let the Hamming weight $w_{\mathrm{H}}(\mathcal{P}_t)$ of a set $\mathcal{P}_t$ be the number of nonzero elements in the set. Then the overall constraint length of the multiple convolutional permutor is defined as

$$\nu = \max_t \{w_{\mathrm{H}}(\mathcal{P}_t)\} \tag{8.72}$$

Remark: Since a bi-infinite MCP constructed by the unwrapping method has the same number J of 1s on each row and same number K of 1s on each column, the weight of $\mathcal{P}_t$ does not depend on t. Therefore, the weight of $\mathcal{P}_t$ characterizes the permutor independently of t.

■ **EXAMPLE 8.6 (*Cont'd*)**

The parameters of the multiple convolutional permutor in Fig. 8.13(c) are $T = 5, \delta = 0, \Delta = 4, \nu = 6$.

Theorem 8.5 Consider the ensemble of symmetric MCPs of multiplicity $J = K$ constructed with the unwrapping procedure from random $T \times T$ interleaved MBPs of multiplicity $J = K$. Suppose these MBPs are built using the rowwise and columnwise interleaving procedure from a set of J^2 randomly chosen permutation matrices. Then, the mathematical expectation $E[\nu]$ of the overall constraint length of an MCP over the ensemble is

$$E[\nu] = J(T-1)/2 \tag{8.73}$$

Proof: An underlying random MBP has T^2 entries. Among the entries of an MBP, JT are 1s. Thus, the theorem follows from the fact that there are $T(T-1)/2$ entries below the main diagonal of the underlying MBP and that the probability that an entry is a 1 is J/T. ■

Theorem 8.5 and the T-periodic property of the MCPs due to the unwrapping construction procedure lead us to the following definition.

Definition Consider a symmetric MCP of multiplicity $J = K$ constructed with the unwrapping procedure using a $T \times T$ symmetric MBP of multiplicity $J = K$. The MCP is called *typical* if:

- It has period T.
- It has width $w = T$.
- It has overall constraint length $\nu = J(T-1)/2$.

We formulated the definition of a typical MCP in the symmetric case. The reformulation of the definition for the nonsymmetric case is left as an exercise (Problem 8.14).

If we transform a symmetric MCP $\boldsymbol{P} = (p_{nl})$ of multiplicity $J = K$ with minimum delay δ, maximum delay Δ, and overall constraint length ν using a shift $p_{nl} \to p_{n(l+a)}$, $a \in \mathbb{Z}^+$ of all its elements, we obtain a new MCP having minimum delay $\delta + a$, maximum delay $\Delta + a$, and overall constraint length $\nu + Ja$ (Problem 8.16).

To simplify our analysis we also consider the *ensemble of multiple Markov permutors (MMPs)*, introduced in [ELZ99, ELT00]. A multiple symmetric Markov

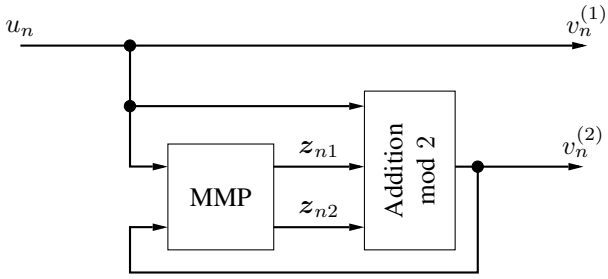


Figure 8.14 An example of a rate $R = 1/2$ systematic encoder for $(2, 4)$ LDPC convolutional codes based on multiple Markov permutors.

permutor of multiplicity J_{MMP} and overall constraint length ν can be described as a memory device which stores ν binary symbols 0 and 1. These ν binary symbols correspond to the constraint length ν positions in the permutor. At each time unit n, the permutor chooses J_{MMP} of these stored symbols with no preference and forms a block of output symbols $(y_{n1}y_{n2}\dots y_{nJ_{\text{MMP}}})$. Then it replaces them with J_{MMP} of its input symbols. The probability that a particular stored symbol is chosen is equal to J_{MMP}/ν. As a consequence, for any positive τ, the probability that a symbol will stay in the permutor more than τ time units is strictly positive, which implies that the maximum delay $\Delta = \infty$. The minimum delay is $\delta = 1$ and the average delay is equal to ν/J_{MMP}.

We can consider the ensemble of multiple Markov permutors as an infinite set of permutors; each of them is deterministic, but the choice of the permutor from the ensemble is random.

■ EXAMPLE 8.9

A rate $R = 1/2$ systematic encoder of a regular[16] $(2, 4)$ LDPC convolutional code is shown in Fig. 8.14. It uses an MMP with overall constraint length ν and multiplicity $J_{\text{MMP}} = 2$. The output of the permutor in the nth time moment are two symbols $\boldsymbol{z}_n = z_{n1}\, z_{n2}$ randomly chosen in the permutor. Adding these symbols and the input information symbol $u_n = v_n^{(1)}$ gives the parity symbol $v_n^{(2)}$. The output of the encoder in the nth time unit is $\boldsymbol{v}_n = v_n^{(1)}\, v_n^{(2)}$. The encoder replaces the two removed symbols in the permutor by the symbols $v_n^{(1)}$ and $v_n^{(2)}$ and then encodes the next information symbol.

[16]For LDPC convolutional codes using Markov permutors, the syndrome former memory m_{s} is not defined; we use simply the notation "regular (J, K) LDPC convolutional codes" without mentioning the memory.

For general rate $R = 1/2$ regular $(J_{\text{MMP}}, 2J_{\text{MMP}})$ LDPC convolutional codes using symmetric Markov permutors of multiplicity J_{MMP}, the encoder (see Fig. 8.15) chooses randomly a set of $2(J_{\text{MMP}} - 1)$ symbols from the permutor and calculates the parity symbol $v_n^{(2)}$ by adding these symbols and the input information symbol $u_t = v_n^{(1)}$. Then it replaces the $2(J_{\text{MMP}} - 1)$ early removed symbols in the permutor by $J_{\text{MMP}} - 1$ copies of $v_n^{(1)}$ and $J_{\text{MMP}} - 1$ copies of $v_n^{(2)}$ and comes to the next step of the encoding.

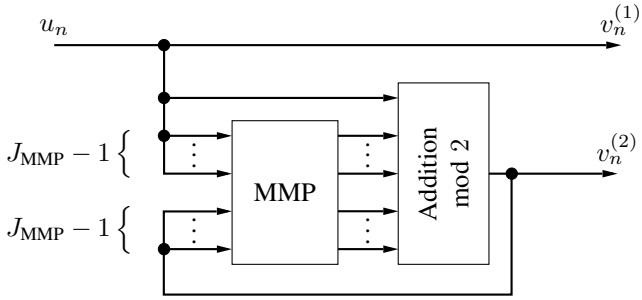


Figure 8.15 A systematic encoder using a multiple Markov permutor for a rate $R = 1/2$ regular $(J_{\text{MMP}}, 2J_{\text{MMP}})$ LDPC convolutional code .

8.4 LOWER BOUNDS ON DISTANCES OF LDPC CODES

In this section, we derive lower bounds for the minimum distance of regular LDPC block codes and for the free distance of regular LDPC convolutional codes. We start with the derivation of minimum distance bounds for LDPC block codes from the ensembles $\mathcal{B}_1(N, J, K)$ and $\mathcal{B}_2(N, J, K)$. To formulate the bounds we introduce the function

$$G(\lambda, \rho) = g(\lambda) - \lambda\rho - \frac{J-1}{J} h(\rho) \tag{8.74}$$

where $0 < \rho < 1/2$, $\lambda < 0$,

$$g(\lambda) = \frac{1}{K} \log\Big(\frac{\big(1 + 2^{\lambda}\big)^K + \big(1 - 2^{\lambda}\big)^K}{2} \Big) \tag{8.75}$$

and $h(\rho)$ is the binary entropy function (1.22). Then we introduce the function

$$F(\rho) = \min_{\lambda < 0} \{G(\lambda, \rho)\} \tag{8.76}$$

and the parameter ρ^* (if it exists[17]) such that

$$\rho^* = \sup_{0<\rho<1/2} \{\rho : F(\rho) < 0\} \tag{8.77}$$

Theorem 8.6 For $J \geq 3$, there exists a code in the ensemble $\mathcal{B}_1(N, J, K)$ with minimum distance lower-bounded by

$$d_{\min} \geq \rho_{JK} N + o(N) \tag{8.78}$$

where $o(N)/N \to 0$ as $N \to \infty$, $\rho_{JK} = \rho^* - \epsilon$, ρ^* satisfies to (8.77), and ϵ is an arbitrary small positive number.[18]

Theorem 8.7 For $J \geq 3$, there exists a code in the ensemble $\mathcal{B}_2(N, J, K)$ with minimum distance lower-bounded by

$$d_{\min} \geq \min\{\rho_{JK} N + o(N), 2N/K\} \tag{8.79}$$

where $o(N)$ and ρ_{JK} are defined in Theorem 8.6.

In Table 1.1 the parameters ρ_{JK} for $J = 3, 4, 5$ and $K = 4, 5, 6$ are given. They are compared with the corresponding Gilbert-Varshamov parameters.

Since the proofs of Theorems 8.6 and 8.7 are similar, we shall give a detailed proof only of Theorem 8.6 and then outline the proof of Theorem 8.7. The proof is based on the lemmas formulated and proved below.

According to Theorem 1.3 the N-tuple $\boldsymbol{v} = v_0\, v_1 \ldots v_{N-1}$ is a codeword if and only if

$$\boldsymbol{v} H^{\mathrm{T}} = \mathbf{0}_{[0,L-1]} \tag{8.80}$$

where H^{T} is an $N \times L$ transposed parity-check matrix and $\mathbf{0}_{[0,L-1]}$ is the length L allzero vector. From (8.5) and (8.80) it follows that for codes from the ensemble $\mathcal{B}_1(N, J, K)$

$$\boldsymbol{v} H_j^{\mathrm{T}} = \mathbf{0}_{[0,M-1]}, \quad j = 1, 2, \ldots, J \tag{8.81}$$

where $M = N/K$ and the submatrices H_j^{T} are defined in (8.5).

We denote the set of M parity-check equations (8.81) by $\mathcal{S}^{(j)}$, $j = 1, 2, \ldots, J$. Let $d = w_{\mathrm{H}}(\boldsymbol{v})$ be the Hamming weight of the codeword $\boldsymbol{v}$, and let $s_m^{(j)}$, $m = 1, 2, \ldots, M$, be the number of 1s in the N-tuple $\boldsymbol{v}$ included into the mth equation of $\mathcal{S}^{(j)}$ such that

$$d = s_1^{(j)} + s_2^{(j)} + \cdots + s_M^{(j)} \tag{8.82}$$

Since $\boldsymbol{v}$ is a codeword, the components $s_m^{(j)}$ are even, that is, $s_m^{(j)} \in \{0, 2, \ldots, 2K_0\}$, where $K_0 = \lfloor K/2 \rfloor$. The distribution of the nonzero symbols of the codeword $\boldsymbol{v}$ among the M parity-check equations of $\mathcal{S}^{(j)}$, that is, the vector $\boldsymbol{s}^{(j)} = \left(s_1^{(j)}\, s_2^{(j)}\right.$

[17] Numerical calculations show that for practically interesting cases it exists.

[18] We cannot numerically calculate the parameter $\rho*$, but we can calculate its lower bound ρ_{JK}.

$\ldots s_M^{(j)}\big)$, is called the *weight distribution* of the set $\mathcal{S}^{(j)}$. Let $\nu_{2i}^{(j)}$, $i = 0, 1, \ldots, K_0$, denote the number of components $\boldsymbol{s}^{(j)}$ which are equal to $2i$. The vector $\boldsymbol{\nu}^{(j)} = (\nu_0^{(j)} \nu_2^{(j)} \ldots \nu_{2K_0}^{(j)})$ is called the *constraint decomposition* of the weight distribution $\boldsymbol{s}^{(j)}$.

Lemma 8.8 The components of the constraint decomposition vector $\boldsymbol{\nu}^{(j)} = (\nu_0^{(j)} \nu_2^{(j)} \ldots \nu_{2K_0}^{(j)})$ satisfy the equalities

$$\sum_{i=0}^{K_0} \nu_{2i}^{(j)} = M \tag{8.83}$$

$$\sum_{i=0}^{K_0} 2i\nu_{2i}^{(j)} = d \tag{8.84}$$

Proof: The set of M parity-check equations $\mathcal{S}^{(j)}$ consists of $K_0 + 1$ subsets such that the ith subset, containing $2i$, $i = 0, 1, \ldots, K_0$, 1s, consists of $\nu_{2i}^{(j)}$ equations, which proves (8.83).

The sum in (8.84) counts the number of nonzero symbols of the codeword $\boldsymbol{v}$ in all check equations of $\mathcal{S}^{(j)}$. Since each symbol can participate only in one equation in $\mathcal{S}^{(j)}$, the sum equals the total number of nonzero symbols in the codeword. ∎

Lemma 8.9 Given a constraint decomposition $\boldsymbol{\nu}^{(j)} = (\nu_0^{(j)} \nu_2^{(j)} \ldots \nu_{2K_0}^{(j)})$, there are

$$\frac{M!}{\nu_0^{(j)}! \nu_2^{(j)}! \cdots \nu_{2K_0}^{(j)}!} \tag{8.85}$$

ways to choose the corresponding weight distribution $\boldsymbol{s}^{(j)}$.

Proof: The lemma follows directly from the definitions of the constraint decomposition $\boldsymbol{\nu}^{(j)}$ and the weight distribution $\boldsymbol{s}^{(j)}$. ∎

Lemma 8.10 For a given codeword $\boldsymbol{v}$ of weight d and a given weight distribution $\boldsymbol{s}^{(j)}$, the number of ways to choose the corresponding submatrix H_j^{T} in the ensemble $\mathcal{B}_1(N, J, K)$ is given by

$$\frac{d!(N-d)!}{\prod_{i=0}^{K_0} ((2i)!(K-2i)!)^{\nu_{2i}^{(j)}}} \tag{8.86}$$

This number depends only on the weight of the codeword $\boldsymbol{v}$ and the constraint decomposition $\boldsymbol{\nu}^{(j)}$ of the weight distribution $\boldsymbol{s}^{(j)}$.

Proof: The proof is similar to the proof of Lemma 8.1. Suppose that the weight distribution $\boldsymbol{s}^{(j)}$ is given. Given a codeword $\boldsymbol{v}$ of weight d, there are $\binom{d}{s_1^{(j)}} \binom{N-d}{K-s_1^{(j)}}$ ways to form the first column of H_j^{T}. The second column can then be chosen in

$\binom{d-s_1^{(j)}}{s_2^{(j)}}\binom{N-d-K+s_1^{(j)}}{K-s_2^{(j)}}$ different ways, and so on. We conclude that H_j^{T} can be chosen in

$$\binom{d}{s_1^{(j)}}\binom{N-d}{K-s_1^{(j)}}\binom{d-s_1^{(j)}}{s_2^{(j)}}\binom{N-d-K+s_1^{(j)}}{K-s_2^{(j)}} \cdots \binom{s_M^{(j)}}{s_M^{(j)}}\binom{K-s_M^{(j)}}{K-s_M^{(j)}} = \frac{d!(N-d)!}{\prod_{i=0}^{K_0}\left((2i)!(K-2i)!\right)^{\nu_{2i}^{(j)}}} \tag{8.87}$$

different ways. ∎

Lemma 8.11 Given a codeword $\boldsymbol{v}$ of weight d and the jth constraint decomposition $\boldsymbol{\nu}^{(j)} = (\nu_0^{(j)}\ \nu_2^{(j)}\ \ldots\ \nu_{2K_0}^{(j)})$, the probability $f_d(\nu_0^{(j)}, \nu_2^{(j)}, \ldots, \nu_{2K_0}^{(j)})$ for the ensemble $\mathcal{B}_1(N, J, K)$ of choosing the corresponding submatrix H_j^{T} equals

$$f_d(\nu_0^{(j)}, \nu_2^{(j)}, \ldots, \nu_{2K_0}^{(j)}) = \frac{\frac{M!}{\nu_0^{(j)}!\,\nu_2^{(j)}!\cdots\nu_{2K_0}^{(j)}!}\prod_{i=0}^{K_0}\left(\frac{K!}{(2i)!(K-2i)!}\right)^{\nu_{2i}^{(j)}}}{\frac{N!}{d!(N-d)!}} \tag{8.88}$$

Proof: The total number of ways to choose the matrix H_j in $\mathcal{B}_1(N, J, K)$ is given by Lemma 8.1. Then it follows that the number of ways to choose H_j^{T} for a given codeword $\boldsymbol{v}$ of weight d and its jth weight distribution $\boldsymbol{s}^{(j)}$ is given by (8.86). The number of ways to choose the jth weight distribution $\boldsymbol{s}^{(j)}$ is given by (8.85). Then we have

$$\begin{aligned} f_d(\nu_0^{(j)}, \nu_2^{(j)}, \ldots, \nu_{2K_0}^{(j)}) &= \frac{\frac{M!}{\nu_0^{(j)}!\nu_2^{(j)}!\cdots\nu_{2K_0}^{(j)}!}\frac{d!(N-d)!}{\prod_{i=0}^{K_0}((2i)!(K-2i)!)^{\nu_{2i}^{(j)}}}}{\frac{N!}{(K!)^M}} \\ &= \frac{\frac{M!}{\nu_0^{(j)}!\,\nu_2^{(j)}!\cdots\nu_{2K_0}^{(j)}!}\prod_{i=0}^{K_0}\left(\frac{K!}{(2i)!(K-2i)!}\right)^{\nu_{2i}^{(j)}}}{\frac{N!}{d!(N-d)!}} \end{aligned} \tag{8.89}$$

∎

We say that an N-tuple $\boldsymbol{v}$ of weight d satisfies the set of equations $\mathcal{S}^{(j)}$, $j = 1, 2, \ldots, J$, if in each of the M equations of $\mathcal{S}^{(j)}$ an even number of nonzero symbols of $\boldsymbol{v}$ are included. In the ensemble $\mathcal{B}_1(N, J, K)$, the probability that an N-tuple $\boldsymbol{v}$ of weight d satisfies the set of equations $\mathcal{S}^{(j)}$ is

$$P_d^{(j)} = \sum_{\boldsymbol{\nu}^{(j)}:\sum_{i=0}^{K_0} 2i\nu_{2i}^{(j)}=d} f_d(\nu_0^{(j)}, \nu_2^{(j)}, \ldots, \nu_{2K_0}^{(j)}) \tag{8.90}$$

If $\boldsymbol{v}$ satisfies the J sets of equations $\mathcal{S}^{(j)}$, $j = 1, 2, \ldots, J$, it is a codeword. The probability P_d that an N-tuple $\boldsymbol{v}$ of weight d is a codeword of a randomly chosen

code in the ensemble $\mathcal{B}_1(N, J, K)$ equals

$$P_d = \prod_{j=1}^{J} P_d^{(j)} \tag{8.91}$$

Now we are ready to prove Theorem 8.6:

Proof (Theorem 8.6): From Lemma 8.1, Lemmas 8.8–8.11, and (8.90) we obtain the following upper bound for $P_d^{(j)}$:

$$P_d^{(j)} = \sum_{\boldsymbol{\nu}^{(j)}:\sum_{i=0}^{K_0} 2i\nu_{2i}^{(j)}=d} f_d(\nu_0^{(j)}, \nu_2^{(j)}, \ldots, \nu_{2K}^{(j)}) \tag{8.92}$$

$$= \sum_{\boldsymbol{\nu}^{(j)}:\sum_{i=0}^{K_0} 2i\nu_{2i}^{(j)}=d} 2^{\lambda(\sum_{i=0}^{2K_0} 2i\nu_{2i}^{(j)}-d)} f_d(\nu_0^{(j)}, \nu_2^{(j)}, \ldots, \nu_{2K_0}^{(j)}) \tag{8.93}$$

$$\leq \sum_{\forall \boldsymbol{\nu}^{(j)}} 2^{\lambda(\sum_{i=0}^{K_0} 2i\nu_{2i}^{(j)}-d)} f_d(\nu_0^{(j)}, \nu_2^{(j)}, \ldots, \nu_{2K_0}^{(j)}) \tag{8.94}$$

$$= 2^{-\lambda d} \sum_{\forall \boldsymbol{\nu}^{(j)}} \frac{\frac{M!}{\nu_0^{(j)}!\nu_2^{(j)}!\cdots\nu_{2K_0}^{(j)}!} \prod_{i=0}^{K_0} \left(\frac{K!2^{2\lambda i}}{2i!(K-2i)!}\right)^{\nu_{2i}^{(j)}}}{\frac{N!}{d!(N-d)!}} \tag{8.95}$$

$$= 2^{-\lambda d} \binom{N}{d}^{-1} \left(\frac{\left(1+2^{\lambda}\right)^K + \left(1-2^{\lambda}\right)^K}{2}\right)^M \tag{8.96}$$

where λ is a negative constant. In going from (8.92) to (8.96), we first multiply the terms by $2^{\lambda(\sum_{i=0}^{K_0} 2i\nu_{2i}^{(j)}-d)}$, which does not change the sum since the summation is over the constraint decompositions $\boldsymbol{\nu}^{(j)}$ satisfying the condition $\sum_{i=0}^{K_0} 2i\nu_{2i}^{(j)} = d$. We then replace the summation over $\boldsymbol{\nu}^{(j)}$ satisfying the condition $\sum_{i=0}^{K_0} 2i\nu_{2i}^{(j)} = d$ by a summation over all $\boldsymbol{\nu}^{(j)}$, which can only increase the sum. In transition from (8.95) to (8.96) we use the formula of polynomial decomposition.

From (8.92)-(8.96) we obtain

$$P_d^{(j)} \leq 2^{(g(\lambda)-\lambda\rho-h(\rho))N+o(N)}, \quad j = 1, 2, \ldots, J \tag{8.97}$$

where $g(\lambda)$ is defined by (8.75), $\rho = d/N$ is the normalized Hamming weight, and $h(\rho)$ is the binary entropy function (1.22). From (8.91) and (8.97) it follows that

$$P_d \leq 2^{(g(\lambda)-\lambda\rho-h(\rho))JN+o(N)} \tag{8.98}$$

Since the number of N-tuples having weight d equals $\binom{N}{d}$, the mathematical expectation M_d over the ensemble $\mathcal{B}_1(N, J, K)$ of the number of codewords having weight

d is upper-bounded by the inequality

$$M_d = \binom{N}{d} P_d \le 2^{(g(\lambda)-\lambda\rho-\frac{J-1}{J}h(\rho))JN+o(N)} = 2^{G(\lambda,\rho)JN+o(N)} \tag{8.99}$$

where $G(\lambda, \rho)$ is defined by (8.74). From (8.76) and (8.99) we obtain

$$M_d \le 2^{F(\rho)JN+o(N)} \tag{8.100}$$

where $F(\rho)$ is defined by (8.76). From (8.100) it follows that for $\rho = \rho^* - \epsilon$, $\epsilon > 0$, the mathematical expectation of the number of codewords of weight d in the ensemble goes to zero if $N \to \infty$, which completes the proof. ∎

Next we shall emphasize a few important steps in the proof of Theorem 8.7.

The transposed parity-check matrix H^{T} of a code in the ensemble $\mathcal{B}_2(N, J, K)$ is defined by (8.10). It can also be given by (8.5) where

$$H_j^{\mathrm{T}} = \begin{pmatrix} P^{(1j)} \\ P^{(2j)} \\ \vdots \\ P^{(Kj)} \end{pmatrix} \tag{8.101}$$

and $P^{(kj)}$, $j = 1, 2, \ldots, J$, $k = 1, 2, \ldots, K$, are $M \times M$ permutation matrices.

Consider an N-tuple $\boldsymbol{v}$, where $N = KM$, consisting of K M-tuples, that is, $\boldsymbol{v} = \boldsymbol{v}_1\boldsymbol{v}_2 \ldots \boldsymbol{v}_K$, where $\boldsymbol{v}_k = v_{k1}\, v_{k2} \ldots v_{kM}$, $k = 1, 2, \ldots, K$. Let $d_1, d_2, \ldots, d_K$ be the Hamming weights of $\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_K$, respectively. We say that $\boldsymbol{v}$ has *weight decomposition* $\boldsymbol{d} = (d_1, d_2, \ldots, d_K)$. The Hamming weight d of $\boldsymbol{v}$ with weight decomposition $\boldsymbol{d}$ is $d = d_1 + d_2 + \cdots + d_K$. There exist $\prod_{k=1}^{K} \binom{M}{d_k}$ different N-tuples $\boldsymbol{v}$ with the weight composition $\boldsymbol{d}$.

We have proved (Theorem 8.4) that codes in the ensemble $\mathcal{B}_2(N, J, K)$ always have at least one codeword $\boldsymbol{v} = \boldsymbol{v}_1\boldsymbol{v}_2 \ldots \boldsymbol{v}_K$ of weight $2N/K$ that contains two M-tuples, $\boldsymbol{v}_{k_1}$ and $\boldsymbol{v}_{k_2}$, say, which are allone M-tuples and all other M-tuples $\boldsymbol{v}_k$, $k \ne k_1, k_2$ are allzero M-tuples. We denote the set of such codewords $\mathcal{V}$. Below we exclude the codewords $\boldsymbol{v} \in \mathcal{V}$ from consideration and consider only codewords $\boldsymbol{v} \notin \mathcal{V}$.

Similarly to codes from the ensemble $\mathcal{B}_1(N, J, K)$, an N-tuple $\boldsymbol{v}$, $\boldsymbol{v} \notin \mathcal{V}$, is a codeword if and only if it satisfies the J sets $\mathcal{S}^{(j)}$, $j = 1, 2, \ldots, J$, of the equations (constraints) (8.81) where the matrices H_j^{T} are defined by (8.101). Each set consists of M equations.

Consider the mth parity-check equation of $\mathcal{S}^{(j)}$. Instead of the variable $s_m^{(j)}$ used in the analysis of the ensemble $\mathcal{B}_1(N, J, K)$ we introduce the binary K-tuple $\boldsymbol{\sigma}_m^{(j)} = \sigma_{m1}^{(j)}\, \sigma_{m2}^{(j)} \ldots \sigma_{mK}^{(j)}$ where $\sigma_{mk}^{(j)} = 1$ if one of the nonzero symbols of $\boldsymbol{v}_k$, $k = 1, 2, \ldots, K$, is included in the mth parity-check equation of $\mathcal{S}^{(j)}$ and $\sigma_{mk}^{(j)} = 0$ otherwise. If $\boldsymbol{v}$ is a codeword, the vectors $\boldsymbol{\sigma}_m^{(j)}$ have even Hamming weight. The

jth weight distribution for codes of the ensemble $\mathcal{B}_2(N, J, K)$ is defined as the set $\boldsymbol{s}^{(j)} = \left\{\boldsymbol{\sigma}_1^{(j)}, \boldsymbol{\sigma}_2^{(j)}, \ldots, \boldsymbol{\sigma}_M^{(j)}\right\}$.

Now we define the jth constraint composition $\boldsymbol{\nu}^{(j)}$. Let $\nu_0^{(j)}$ be the number of K-tuples $\boldsymbol{\sigma}_m^{(j)}$, $m = 1, 2, \ldots, M$, which consists of K 0s. Then, let $\nu_2^{(j)}(k_1, k_2)$ be the number of K-tuples $\boldsymbol{\sigma}_m^{(j)}$ involving two 1s, one from $\boldsymbol{v}_{k_1}$ and one from $\boldsymbol{v}_{k_2}$; the other $(K-2)$ components of $\boldsymbol{\sigma}_m^{(j)}$ are 0s. In general, let $\nu_{2i}^{(j)}(k_1, k_2, \ldots, k_{2i})$, $i = 1, \ldots, K_0$, be the number of K-tuples $\boldsymbol{\sigma}_m^{(j)}$ involving $2i$ 1s, a single 1 from each of the M-tuples $\boldsymbol{v}_{k_1}, \boldsymbol{v}_{k_2}, \ldots, \boldsymbol{v}_{k_{2i}}$, and 0s from the remaining $(K-2i)$ components of $\boldsymbol{\sigma}_m^{(j)}$. Observe that the arguments of $\nu_{2i}^{(j)}(k_1, k_2, \ldots, k_{2i})$ are distinct, that is, $k_i \neq k_j$ if $i \neq j$. Furthermore, $\nu_{2i}^{(j)}(k_1, k_2, \ldots, k_{2i})$ is invariant to permutations of the arguments, for example, $\nu_{2i}^{(j)}(k_1, k_2, \ldots, k_{2i}) = \nu_{2i}^{(j)}(k_2, k_1, \ldots, k_{2i})$. In other words, $\nu_{2i}^{(j)}(k_1, k_2, \ldots, k_{2i})$ is a function of the set $\{k_1, k_2, \ldots, k_{2i}\}$. To emphasize this fact, we henceforth write $\nu_{2i}^{(j)}(\{k_1, k_2, \ldots, k_{2i}\})$ for $\nu_{2i}^{(j)}(k_1, k_2, \ldots, k_{2i})$. Then the jth constraint composition vector $\boldsymbol{\nu}^{(j)}$, $j = 1, 2, \ldots, J$, is defined as

$$\boldsymbol{\nu}^{(j)} = \left(\nu_0^{(j)} \;\; \{\nu_2^{(j)}(\{k_1, k_2\})\} \ldots \;\; \{\nu_{2K_0}^{(j)}(\{k_1, \ldots, k_{2K_0}\})\}\right) \tag{8.102}$$

where $\{\nu_{2i}^{(j)}(\{k_1, \ldots, k_{2i}\})\}$, $i = 0, 2, \ldots, K_0$, is the set of all $\binom{K}{2i}$ components $\nu_{2i}^{(j)}(\{k_1, \ldots, k_{2i}\})$.

The following lemma is a counterpart to Lemma 8.8.

Lemma 8.12 The components of the constraint decomposition $\boldsymbol{\nu}^{(j)}$ satisfy the equalities

$$\begin{aligned}\nu_0^{(j)} + \sum_{\{k_1,k_2\}} \nu_2^{(j)}(\{k_1, k_2\}) + \sum_{\{k_1,k_2,k_3,k_4\}} \nu_4^{(j)}(\{k_1, k_2, k_3, k_4\}) + \cdots \\ + \sum_{\{k_1,\ldots,k_{2K_0}\}} \nu_{2K_0}^{(j)}(\{k_1, \ldots, k_{2K_0}\}) = M\end{aligned} \tag{8.103}$$

and

$$\begin{aligned}\sum_{\{k_2\}} \nu_2^{(j)}(\{k_1, k_2\}) + \sum_{\{k_2,k_3,k_4\}} \nu_4^{(j)}(\{k_1, k_2, k_3, k_4\}) + \cdots \\ + \sum_{\{k_2,\ldots,k_{2K_0}\}} \nu_{2K_0}^{(j)}(\{k_1, \ldots, k_{2K_0}\}) = d_{k_1}, \;\; k_1 = 1, 2, \ldots, K\end{aligned} \tag{8.104}$$

Equations (8.103) and (8.104) for the constraint composition $\boldsymbol{\nu}^{(j)}$ of codes of the ensemble $\mathcal{B}_2(N, J, K)$ are generalizations of equations (8.83) and (8.84) for the constraint composition $\boldsymbol{\nu}^{(j)}$ of codes in the ensemble $\mathcal{B}_1(N, J, K)$. The set of $\boldsymbol{\nu}^{(j)}$ that satisfies equation (8.104) is denoted $\mathcal{D}_{\boldsymbol{d}}^{(j)}$.

The next three lemmas are similar to Lemmas 8.9–8.11.

Lemma 8.13 Given a constraint composition $\boldsymbol{\nu}^{(j)}$ defined by (8.102), there are

$$\frac{M!}{\nu_0^{(j)}! \prod_{\{k_1,k_2\}} \nu_2^{(j)}(\{k_1,k_2\})! \ldots \prod_{\{k_1\ldots,k_{2K_0}\}} \nu_{2K_0}^{(j)}(\{k_1,\ldots,k_{2K_0}\})!} \tag{8.105}$$

ways to choose the corresponding weight distribution $\boldsymbol{s}^{(j)}$.

Lemma 8.14 For a given codeword $\boldsymbol{v}$ of weight composition $\boldsymbol{d}$ and a given weight distribution $\boldsymbol{s}^{(j)}$, the number of ways to choose the corresponding submatrix H_j^{T} in the ensemble $\mathcal{B}_2(N,J,K)$ is given by

$$\prod_{k=1}^{K} d_k!(M-d_k)! \tag{8.106}$$

Equation (8.106) is a generalization of (8.86) to the ensemble $\mathcal{B}_2(N,J,K)$.

Lemma 8.15 Given a codeword $\boldsymbol{v}$ with weight composition $\boldsymbol{d}$ and jth constraint composition $\boldsymbol{\nu}^{(j)}$, the probability $f_d(\nu_0^{(j)}, \nu_2^{(j)}, \ldots, \nu_{2K_0}^{(j)})$ for the ensemble $\mathcal{B}_2(N,J,K)$ of choosing the corresponding submatrix H_j^{T} equals (cf. (8.88))

$$f_d(\nu_0^{(j)}, \nu_2^{(j)}, \ldots, \nu_{2K_0}^{(j)}) = \frac{\frac{M!}{\nu_0^{(j)}! \prod_{\{k_1,k_2\}} \nu_2^{(j)}(\{k_1,k_2\})! \ldots \prod_{\{k_1,\ldots,k_{2K_0}\}} \nu_{2K_0}^{(j)}(\{k_1,\ldots,k_{2K_0}\})!}}{\prod_{k=1}^{K} \binom{M}{d_k}} \tag{8.107}$$

Lemma 8.15 follows from Lemmas 8.13 and 8.14 and from the fact that there are $(M!)^K$ different submatrices H_j^{T}.

The following formulas, generalizing formulas (8.90) and (8.92)-(8.96), determine the probability $P_{\boldsymbol{d}}^{(j)}$ that the K-tuple $\boldsymbol{v} = \boldsymbol{v}_1\boldsymbol{v}_2\ldots\boldsymbol{v}_K, \boldsymbol{v} \notin \mathcal{V}$, with weight composition $\boldsymbol{d}$ satisfies the set of equations $\mathcal{S}^{(j)}$, $j = 1,2,\ldots,J$, and yields the upper bound

$$\begin{aligned} P_{\boldsymbol{d}}^{(j)} &= \sum_{\boldsymbol{\nu}^{(j)} \in \mathcal{D}_{\boldsymbol{d}}^{(j)}} f_d(\nu_0^{(j)}, \nu_2^{(j)}, \ldots, \nu_{2K}^{(j)}) \\ &\le 2^{-\sum_{k=1}^{K} \lambda_k d_k} \prod_{k=1}^{K} \binom{M}{d_k} \left(\frac{\prod_{k=1}^{K}(1+2^{\lambda_k}) + \prod_{k=1}^{K}(1-2^{\lambda_k})}{2} \right)^M \end{aligned} \tag{8.108}$$

where $\lambda_k < 0$, $k = 1,2,\ldots,K$. To prove (8.108) we first multiply the terms by

$$\prod_{k_1=1}^{K} 2^{\lambda_{k_1}\left(\sum_{\{k_2\}} \nu_2^{(j)}(\{k_1,k_2\}) + \cdots + \sum_{\{k_2,\ldots,k_{2K_0}\}} \nu_{2K_0}^{(j)}(\{k_1,\ldots,k_{2K_0}\}) - d_{k_1}\right)} \tag{8.109}$$

and then summarize over all $\boldsymbol{\nu}^{(j)}$.

Note that the upper bound (8.108) for the probability $P_{\boldsymbol{d}}^{(j)}$ depends not on a single auxiliary variable λ as in the case of the ensemble $\mathcal{B}_1(N, J, K)$ but on a set $\{\lambda_1, \lambda_2, \ldots, \lambda_K\}$ of auxiliary variables.

We can rewrite (8.108) as

$$P_{\boldsymbol{d}}^{(j)} \le 2^{\left(K\tilde{g}(\lambda_1,\lambda_2,\ldots,\lambda_K)-\sum_{k=1}^{K}\lambda_k\rho_k-\sum_{k=1}^{K}h(\rho_k)\right)M+o(M)} \tag{8.110}$$

where $\lambda_k < 0$, $k = 1, 2, \ldots, K$, $\rho_k = d_k/M$, $k = 1, 2, \ldots, K$, are normalized Hamming weights of the M-tuples $\boldsymbol{v}_k$, $h(\rho)$ is the binary entropy function (1.22), and

$$\tilde{g}(\lambda_1, \lambda_2, \ldots, \lambda_K) = \frac{1}{K} \log \frac{\prod_{k=1}^{K}(1 + 2^{\lambda_k}) + \prod_{k=1}^{K}(1 - 2^{\lambda_k})}{2} \tag{8.111}$$

From (8.110) we obtain similarly to (8.99) that the mathematical expectation over the ensemble $\mathcal{B}_2(N, J, K)$ of the number of codewords with weight composition $\boldsymbol{d}$ is upper-bounded by

$$M_{\boldsymbol{d}} \le 2^{\tilde{G}(\boldsymbol{\lambda},\boldsymbol{\rho})KJM+o(M)} \tag{8.112}$$

where $\boldsymbol{\lambda} = (\lambda_1\ \lambda_2\ \ldots\ \lambda_K)$, $\boldsymbol{\rho} = (\rho_1\ \rho_2\ \ldots\ \rho_K)$,

$$\tilde{G}(\boldsymbol{\lambda}, \boldsymbol{\rho}) = \tilde{g}(\lambda_1, \lambda_2, \ldots, \lambda_K) - \frac{1}{K}\sum_{k=1}^{K}\lambda_k\rho_k - \frac{J-1}{JK}\sum_{k=1}^{K}h(\rho_k) \tag{8.113}$$

Let

$$\tilde{F}(\rho) = \max_{\boldsymbol{\rho}:\frac{1}{K}\sum\rho_k=\rho} \min_{\boldsymbol{\lambda}} \{\tilde{G}(\boldsymbol{\lambda}, \boldsymbol{\rho})\} \tag{8.114}$$

Lower-bounding the minimum distance for block codes from the ensemble $\mathcal{B}_2(N, J, K)$ requires operations with K auxiliary variables $\lambda_1, \lambda_2, \ldots, \lambda_K$ and K variables $\rho_1, \rho_2, \ldots, \rho_K$. But, because of the symmetry of the problem with respect to the considered variables, it can be reduced to the analysis of a function of the two auxiliary variables λ and ρ. This follows from the following lemma proved in [STL07].

Lemma 8.16 If $J \ge 3$ and $\rho = (1/K)\sum\rho_k$ is fixed, $\rho < 2/K$, then the conditional extremum (8.114) of the function $\tilde{G}(\boldsymbol{\lambda}, \boldsymbol{\rho})$ is attained at the points $\boldsymbol{\rho} = \boldsymbol{\rho}_0 = (\rho\,\rho\ \ldots\ \rho)$, $\boldsymbol{\lambda} = \boldsymbol{\lambda}_0 = (\lambda\,\lambda\ \ldots\ \lambda)$, where all coordinates of the vector $\boldsymbol{\rho}_0$ are equal and all coordinates of the vector $\boldsymbol{\lambda}_0$ are equal.

Note that

$$\tilde{G}(\boldsymbol{\lambda}_0, \boldsymbol{\rho}_0) = G(\lambda, \rho) \tag{8.115}$$

where $G(\lambda, \rho)$ is defined by (8.74) and

$$\tilde{F}(\boldsymbol{\rho}_0) = F(\rho) \tag{8.116}$$

where $F(\rho)$ is defined by (8.76).

From Lemma 8.16, Theorem 8.7 follows.

Consider now lower-bounding the free distance of LDPC convolutional codes. There are two different approaches to this problem considered in [STL07] and [TZC10], respectively. They yield the same numerical results. In the first case, nonperiodically time-varying regular LDPC convolutional codes from ensemble $\mathcal{C}(M, J, K)$ with "expurgated" information bits and the corresponding segments of a code sequence were considered. It means that we expurgate low-weight segments of the code sequences by fixing one of the information bits in the initial information block to zero. If the size M of the permutation matrices of the syndrome former goes to infinity, then the loss in rate is negligible.

In the second case, regular periodically time-varying LDPC convolutional codes from the ensemble $\mathcal{C}(M, J, K, T)$, which we introduced in Section 8.2, were considered. Their analysis uses the same ideas as lower-bounding the minimum distance of codes from the ensemble $\mathcal{B}_2(N, J, K)$ but involves even more auxiliary variables. We shall follow this approach. To avoid cumbersome notations, we focus on the analysis of the case $J = 3$ and $K = 6$. The corresponding syndrome former is presented in Fig. 8.9.

Instead of lower-bounding the free distance d_{free} for the codes from the ensemble $\mathcal{C}(M, J, K, T)\big|_{J=3,K=6}$ we shall lower-bound the minimum distance $\tilde{d}_{[0,T-1]}$ of the corresponding tailbiting LDPC code with the transposed parity-check matrix $\tilde{H}^{\mathrm{T}}_{[0,T-1]}$ given in Fig. 8.10. The block length is $N = 2MT$. The ensemble of the corresponding tailbiting LDPC codes is denoted $\tilde{\mathcal{C}}(M, J, K, T)\big|_{J=3,K=6}$. For this analysis of tailbiting codes we can use essentially the same method as the one we used for analyzing $d_{\min}$ of LDPC block codes from the ensemble $\mathcal{B}_2(N, J, K)$.

Lemma 8.17 The free distance d_{free} of a regular periodically time-varying LDPC convolutional code from the ensemble $\mathcal{C}(M, J, K, T)$ is lower-bounded by the minimum distance $\tilde{d}_{[0,T-1]}$ of the corresponding tailbiting LDPC code from the ensemble $\tilde{\mathcal{C}}(M, J, K, T)$ with block length $N = 2MT$, that is,

$$d_{\text{free}} \geq \tilde{d}_{[0,T-1]} \tag{8.117}$$

Lemma 8.17 is proved in [TZC10].

Consider a length $2TM$ T-tuple $\tilde{\boldsymbol{v}} = \tilde{\boldsymbol{v}}_0^{(1)}\tilde{\boldsymbol{v}}_0^{(2)} \ldots \tilde{\boldsymbol{v}}_{T-1}^{(1)}\tilde{\boldsymbol{v}}_{T-1}^{(2)}$, where $\tilde{\boldsymbol{v}}_t^{(i)} = \tilde{v}_{t1}^{(i)}\, \tilde{v}_{t2}^{(i)} \ldots \tilde{v}_{tM}^{(i)}$, $i = 1, 2$, $t = 0, 1, \ldots, T-1$. A T-tuple $\tilde{\boldsymbol{v}}_{[0,T-1]}$ is a codeword of a regular tailbiting LDPC code if and only if it satisfies the TM equations (constraints) defined by the transposed parity-check matrix $\tilde{H}^{\mathrm{T}}_{[0,T-1]}$ of the code, that is,

$$\tilde{\boldsymbol{v}}_{[0,T-1]}\tilde{H}^{\mathrm{T}}_{[0,T-1]} = \mathbf{0} \tag{8.118}$$

For the ensemble $\tilde{\mathcal{C}}(M, J, K, T)|_{J=3,K=6}$, these TM parity-check equations can be divided into T sets where the tth set $\mathcal{S}^{(t)}$, $t = 0, 1, \ldots, T-1$, consists of the M parity-check equations determined by the six permutation matrices located in the tth column, $t = 0, 1, \ldots, T-1$, of the syndrome former $\tilde{H}^{\mathrm{T}}_{[0,T-1]}$ in Fig. 8.10.

Let $\tilde{d}_t^{(i)}$ be the Hamming weight of the T-tuple $\tilde{\boldsymbol{v}}_t^{(i)}$, $i = 1, 2$, $t = 0, 1, \ldots, T-1$. We then say that $\tilde{\boldsymbol{v}}_{[0,T-1]}$ has the weight composition

$$\tilde{\boldsymbol{d}}_{[0,T-1]} = \left(\tilde{d}_0^{(1)}\tilde{d}_0^{(2)}\tilde{d}_1^{(1)}\tilde{d}_1^{(2)}\ldots\tilde{d}_{T-1}^{(1)}\tilde{d}_{T-1}^{(2)}\right) \tag{8.119}$$

The Hamming weight of the T-tuple $\tilde{\boldsymbol{v}}_{[0,T-1]}$ with weight composition $\tilde{\boldsymbol{d}}_{[0,T-1]}$ is $\tilde{d}_{[0,T-1]} = \tilde{d}_0^{(1)} + \tilde{d}_0^{(2)} + \cdots + \tilde{d}_{T-1}^{(1)} + \tilde{d}_{T-1}^{(2)}$. Note that there exists $\prod_{t=0}^{T-1}\binom{M}{\tilde{d}_t^{(1)}} \times \prod_{t=0}^{T-1}\binom{M}{\tilde{d}_t^{(2)}}$ T-tuples $\tilde{\boldsymbol{v}}_{[0,T-1]}$ with weight composition $\tilde{\boldsymbol{d}}_{[0,T-1]}$. In the asymptotic case, $M \to \infty$, it is more convenient to operate with the normalized weight composition $\tilde{\boldsymbol{\rho}}_{[0,T-1]} = \left(\tilde{\rho}_0^{(1)}\tilde{\rho}_0^{(2)}\ldots\tilde{\rho}_{T-1}^{(1)}\tilde{\rho}_{T-1}^{(2)}\right)$, where $\tilde{\rho}_t^{(i)} = \tilde{d}_t^{(i)}/M$.

We introduce the additional notations

$$\tilde{\boldsymbol{\rho}}^{(t)} = \left(\tilde{\rho}_{t-2}^{(1)}\tilde{\rho}_{t-2}^{(2)}\tilde{\rho}_{t-1}^{(1)}\tilde{\rho}_{t-1}^{(2)}\tilde{\rho}_t^{(1)}\tilde{\rho}_t^{(2)}\right), \quad t = 0, 1, \ldots, T-1 \tag{8.120}$$

where by definition $\tilde{\rho}_{-2}^{(i)} = \tilde{\rho}_{T-2}^{(i)}$ and $\tilde{\rho}_{-1}^{(i)} = \tilde{\rho}_{T-1}^{(i)}$ for $i = 1, 2$, and

$$\boldsymbol{\lambda}^{(t)} = \left(\lambda_1^{(t)}\lambda_2^{(t)}\ldots\lambda_6^{(t)}\right), \quad t = 0, 1, \ldots, T-1 \tag{8.121}$$

To formulate our result we introduce the three functions[19]

$$\begin{aligned}\tilde{G}(\boldsymbol{\lambda}^{(t)}, \tilde{\boldsymbol{\rho}}^{(t)}) &= \tilde{g}(\lambda_1^{(t)}, \lambda_2^{(t)}, \ldots, \lambda_6^{(t)}) \\ &-\frac{1}{6}\left(\lambda_1^{(t)}\tilde{\rho}_{t-2}^{(0)} + \lambda_2^{(t)}\tilde{\rho}_{t-2}^{(1)} + \lambda_3^{(t)}\tilde{\rho}_{t-1}^{(0)} + \lambda_4^{(t)}\tilde{\rho}_{t-1}^{(1)} + \lambda_5^{(t)}\tilde{\rho}_t^{(0)} + \lambda_6^{(t)}\tilde{\rho}_t^{(1)}\right) \\ &-\frac{2}{18}\left(h(\tilde{\rho}_{t-2}^{(0)}) + h(\tilde{\rho}_{t-2}^{(1)}) + h(\tilde{\rho}_{t-1}^{(0)}) + h(\tilde{\rho}_{t-1}^{(1)}) + h(\tilde{\rho}_t^{(0)}) + h(\tilde{\rho}_t^{(1)})\right)\end{aligned} \tag{8.122}$$

which is a generalization of the function $\tilde{G}(\boldsymbol{\lambda}, \boldsymbol{\rho})$ defined by (8.113),

$$\tilde{H}(\boldsymbol{\rho}^{(t)}) = \min_{\boldsymbol{\lambda}^{(t)}}\left\{\tilde{G}(\boldsymbol{\lambda}^{(t)}, \tilde{\boldsymbol{\rho}}^{(t)})\right\} \tag{8.123}$$

and

$$\Phi(\tilde{\boldsymbol{\rho}}_{[0,T-1]}) = \sum_{t=0}^{T-1}\tilde{H}(\boldsymbol{\rho}^{(t)}) \tag{8.124}$$

Next we introduce the region $\mathcal{R}_T(\rho)$. By definition, the vector $\tilde{\boldsymbol{\rho}}_{[0,T-1]} \in \mathcal{R}_T(\rho)$ if

$$\frac{1}{2T}\left(\sum_{t=0}^{T-1}\rho_t^{(1)} + \sum_{t=0}^{T-1}\rho_t^{(2)}\right) = \rho \tag{8.125}$$

The function $\tilde{F}_T(\rho)$ generalizes the function $\tilde{F}(\rho)$ defined by (8.114),

$$\tilde{F}_T(\rho) = \max_{\tilde{\boldsymbol{\rho}}_{[0,T-1]} \in \mathcal{R}_T(\rho)}\{\Phi(\tilde{\boldsymbol{\rho}}_{[0,T-1]})\} \tag{8.126}$$

[19]Note that we obtain coefficients $1/6$ and $2/18$ in (8.122) because, for $J = 3$, $K = 6$, we have $1/K = 1/6$ and $(J-1)/JK = 2/18$.

Table 8.1 Numerically calculated parameters ρ_{JK} amd $\rho_{JK}^{(\text{conv})}$ of the distance bounds for LDPC block and convolutional codes, respectively.

(J, K)	ρ_{JK}	$\rho_{JK}^{(\text{conv})}$
$(3, 6)$	0.023	0.0833
$(4, 8)$	0.0627	0.1908

Then we introduce the parameter ρ_T^* such that

$$\rho_T^* = \sup_{0<\rho<1/2} \{\rho : \tilde{F}_T(\rho) < 0\} \tag{8.127}$$

Theorem 8.18 For $J \geq 3$, there exists a convolutional code in the ensemble $\mathcal{C}(m_{\text{s}}, J, K, T)$ with free distance lower-bounded by

$$d_{\text{free}} \geq \rho_{JK}^{(\text{conv})} \nu R^{-1} + o(N) \tag{8.128}$$

where $o(N)/N \to 0$ when $N \to \infty$, $\rho_{JK}^{(\text{conv})} = \rho_T^* - \epsilon$, where ρ_T^* satisfies (8.127), ϵ is an arbitrary small positive number, and $\nu \leq 3M - 1$ is the overall constraint length of the encoder in the partial-syndrome realization.

Remark: The parameter $\nu R^{-1} \leq 2(3M - 1)$ on the right-hand side of (8.128) equals the total number of code symbols involved in the parity-check constraints at any time instant t. This also corresponds to the total number of encoder output symbols that directly depend on a given block of b input information symbols.

Theorem 8.18 reduced the lower-bounding of d_{free} for regular periodically time-varying LDPC convolutional codes to a numerical calculation of the parameter ρ_T^*. This parameter is a nondecreasing function of the period T, and to get a reasonable lower bound on d_{free} it is sufficient to choose T of the order 15. In Table 8.1 we give the parameter $\rho_{JK}^{(\text{conv})}$ characterizing the lower bound on the free distance of regular periodically time-varying (m_{s}, J, K, T) LDPC convolutional codes and the parameter ρ_{JK} characterizing Gallager's lower bound on the minimum distance of regular (N, J, K) LDPC block codes (see Table 1.1). For $J = 3, K = 6$, the parameter $\rho_{JK}^{(\text{conv})} = 0.0833$ is about three and half times larger than the parameter $\rho_{JK} = 0.023$ characterizing Gallager's lower bound for the minimum distance of regular $(N, 3, 6)$ LDPC block codes. Interestingly, the parameter $\rho_{\text{C}} = -R/\log(2^{1-R} - 1)$ characterizing the Costello lower bound (3.162) on the free distance of general rate $R = 1/2$ convolutional codes, $\rho_{\text{C}} = 0.39$, is also about three and half times larger than the parameter ρ_{GV} characterizing the Gilbert-Varshamov lower bound on the minimum distance of general rate $R = 1/2$ block codes, that is, $\rho_{\text{GV}} = 0.11$. From Table 8.1 it follows that also in the case when regular LDPC codes have parameters $J = 4, K = 8$ we have this effect.

In [Len03] there is an analysis of the free distance for regular $(3, 6)$ LDPC convolutional codes using the Markov permutor that yields the value $\rho_{JK}^{(\text{conv})} = 0.125$.

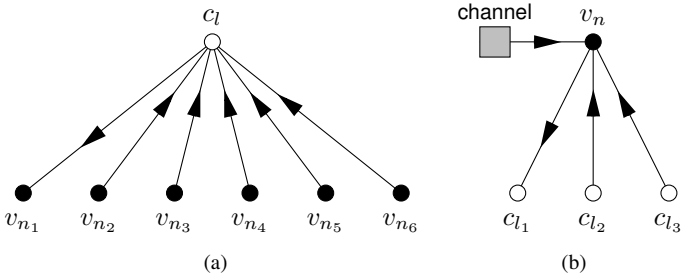


Figure 8.16 Illustration of belief propagation decoding for an $(N, 3, 6)$ LDPC code. In (a) an update at the constraint node c_l (first phase) is shown and in (b) an update at the symbol node v_n (second phase) is shown.

8.5 ITERATIVE DECODING OF LDPC CODES

In this section, we begin by explaining the belief propagation decoding of LDPC block codes using the Tanner graph introduced in Section 1.3. Then we generalize the algorithm to LDPC convolutional codes.

Let $\boldsymbol{v} = v_0 v_1 \ldots v_{N-1}$ be a codeword of an LDPC block code transmitted over a binary-input memoryless channel and let $\boldsymbol{r} = r_0 r_1 \ldots r_{N-1}$ be the received sequence. The belief propagation algorithm is based on the computation of the *a posteriori* probabilities (APPs),

$$P(v_n = 0 \mid \boldsymbol{r}) \tag{8.129}$$

for the code symbols v_n, $n = 0, 1, \ldots, N-1$.

First we describe the belief propagation decoding algorithm for regular (N, J, K) LDPC codes in the case when transmission is over the binary erasure channel (BEC) considered in Problem 1.27 (Fig. 1.22). In this case, $r_n \in \{0, 1, \Delta\}$, $n = 0, 1, \ldots, N-1$, where Δ is the erasure symbol.

The belief propagation decoding can be considered as *message passing* between the symbol and constraint nodes of the Tanner graph (see Fig. 8.16). We assume that in the beginning of the decoding process the symbols of the received sequence $\boldsymbol{r}$ are associated with their corresponding symbol nodes. After each iteration, the decoder reconstructs, if possible, some erased symbols and then, when we consider the BEC, prescribes symbols of the obtained (modified) tentative sequence $\boldsymbol{r}' = r'_0 r'_1 \ldots r'_{N-1}$ to the corresponding symbol nodes.

The operations of the message-passing decoder on the ith step, $i = 1, 2, \ldots, I$, can be split into two phases, as illustrated in Fig. 8.16. We recall that $l \in \mathcal{L}(n)$ enumerates the set of indices of the constraint nodes that are connected with the nth symbol node v_n and that $n \in \mathcal{N}(l)$ enumerates the set of indices of the symbol nodes connected with the lth constraint node c_l. In both iterative decoding phases, the nodes receive messages from the adjacent connected nodes and combine them using a computational rule to form new messages, which then are sent back to the corresponding nodes during the next phase of the iterative process.

In the first phase, constraint node c_l receives messages from the connected symbol nodes and performs an update as illustrated in Fig. 8.16(a). It forms messages which will be sent to the connected symbol nodes during the next phase of the process. The message that the constraint node c_l sends during the second phase to a *given* connected symbol node, for example, v_{n_1} in Fig. 8.16(a), depends on the messages that this constraint node receives from all the *other* $K-1$ connected symbol nodes, that is, $v_{n_2}, v_{n_3}, \ldots, v_{n_6}$, during the first phase of the iteration. If at least one of these messages was the erasure symbol Δ, then the constraint node also sends an erasure symbol to the given symbol node. If a constraint node receives no erasure symbols from these other nodes, that is, all these received symbols are either 0 or 1, the constraint node sends during the second phase the modulo-2 sum of these symbols to the given symbol node. The update at each constraint node c_l is performed for each symbol node $v_{n_i}, n_i \in \mathcal{N}(l)$, being the given node. This procedure is repeated for all constraint nodes.

During the second phase the constraint node c_l sends a message to a given connected symbol node, for example, v_{n_1} in Fig. 8.16(b). Symbol node v_{n_1} forms messages which will be sent to the connected constraint nodes during the first phase of the next iteration step. The message addressed to constraint node c_{l_1} depends not only on the messages that this symbol node receives from all the other $J-1$ connected constraint nodes, that is, c_{l_2} and c_{l_3}, but also on the tentative symbol r'_n. This update at the symbol node is performed for all constraint nodes $c_{l_i}, l_i \in \mathcal{L}(n)$. This procedure is repeated for all symbol nodes.

During the decoding process, the symbol nodes corresponding to the nonerased symbols retain these nonerased symbols independently of the messages received from their connected constraint nodes and continue to pass on the same messages. If a symbol node keeping an erasure symbol receives from at least one connected constraint node a 0 or a 1, it replaces the erasure symbol by this symbol and on all the following iterations it passes on this symbol to its connected constraint nodes.

To formulate the decoding stopping rule, we introduce the definition of the *stopping set* [RUr08]:

Definition A *stopping set* $\mathcal{S}$ is a subset of the set of symbol nodes such that all constraint nodes which are connected to $\mathcal{S}$ are connected at least twice.

It can be shown that a symbol of a stopping set is included in at least two parity-check equations.

Now we formulate the *decoding stopping rule*.

The decoding continues until either all symbol nodes have been assigned a 0 or a 1 or the set of erasure symbols in $\boldsymbol{r}'$ becomes a stopping set. In the first case, the decoding is *successful*, that is, the tentative sequence $\boldsymbol{r}'$ does not contain any erasure symbols. In the latter case, the decoder declares a *decoding failure*.

The decoding BP algorithm for the BEC can be formulated as follows. We introduce the statistics $z_n^{(0)}$, $n = 0, 1, \ldots, N-1$, such that

$$z_n^{(0)} = \begin{cases} 1 & \text{if } r_n = 0 \\ -1 & \text{if } r_n = 1 \\ 0 & \text{if } r_n = \Delta \end{cases} \tag{8.130}$$

Algorithm BPBEC (The BP algorithm for decoding the output of a BEC)

BPBEC1. Initialize $i = 0$ and $z_n^{(0)}$, $n = 0, 1, \ldots, N-1$.

BPBEC2. While $i > 0$, for all $n \in \mathcal{N}(l)$ and $l = 0, 1, \ldots, L-1$, compute

$$y_{ln}^{(i)} = \prod_{n' \in \mathcal{N}(l) \setminus n} z_{n'}^{(i-1)}$$

BPBEC3. For $n = 0, 1, \ldots N-1$, compute $z_n^{(i)}$ using the following rule:

- Set $z_n^{(i)} = a, a \in \{1, -1\}$, if $z_n^{(i-1)} = a$ or at least one $y_{ln}^{(i)} = a$, $l \in \mathcal{L}(n)$
- Set $z_n^{(i)} = 0$ if $z_n^{(i-1)} = 0$ and all $y_{ln}^{(i)} = 0$, $l \in \mathcal{L}(n)$

BPBEC4. The decoding is stopped at the ith step if $z_n^{(i)} = z_n^{(i-1)}$ for all $n = 0, 1, \ldots, N-1$; the decoder output is

$$\widehat{v}_n = \begin{cases} 0 & \text{if } z_n^{(i)} = 1 \\ 1 & \text{if } z_n^{(i)} = -1 \\ \Delta & \text{if } z_n^{(i)} = 0 \end{cases}$$

Both the description and analysis of the belief propagation decoding algorithms for LDPC codes for the BSC and AWGN channels are more complicated than for the BEC. The algorithm is still founded on the message-passing principle in combination with the APP decoding of the constituent codes, which are single-error-detecting codes. We describe the message-passing algorithm for a binary-input discrete memoryless channel (DMC). The description of the decoding algorithm for the binary-input AWGN channel is straightforward.

Suppose that a codeword $\boldsymbol{v} = v_0 v_1 \ldots v_{N-1}$ of a regular (N, J, K) LDPC block code is transmitted over a binary-input DMC. Let $\boldsymbol{r} = r_0 r_1 \ldots r_{N-1}$ denote the received sequence and let

$$\pi_n(0) \stackrel{\text{def}}{=} P(v_n = 0), n = 0, 1, \ldots, N-1 \tag{8.131}$$

denote the *a priori* probability that v_n equals zero. In the sequel we will assume that $P(v_n = 0) = 1/2$.

We introduce the log-likelihood ratio $z_n^{(0)}$ for the code symbol v_n given the received symbol r_n. Assuming that $P(v_n = 0) = \pi_n(0) = 1/2$, we obtain

$$z_n^{(0)} \stackrel{\text{def}}{=} \log\left(\frac{P\left(r_n \mid v_n = 0\right)}{P\left(r_n \mid v_n = 1\right)} \frac{\pi_n(0)}{1 - \pi_n(0)}\right) = \log \frac{P\left(v_n = 0 \mid r_n\right)}{P\left(v_n = 1 \mid r_n\right)} \tag{8.132}$$

The statistics $z_n^{(0)}$ is called the *intrinsic information* about the code symbol v_n. We assume that in the beginning of the iterative decoding process each symbol node memorizes the corresponding intrinsic information and keeps this information during the decoding process.

Each symbol node is connected with J constraint nodes. In the first phase of the first step of the iteration process, the nth symbol node, $n = 0, 1, \ldots, N-1$, sends the log-likelihood ratio $z_{nl}^{(0)} = z_n^{(0)}$, called "message," to the connected constraint nodes $l \in \mathcal{L}(n)$ (Fig. 8.16(a)). Then the lth constraint node, which receives K messages $z_{nl}^{(0)}$, $n \in \mathcal{N}(l)$, calculates the K log-likelihood ratios $y_{ln}^{(1)}$, $n \in \mathcal{N}(l)$,

$$y_{ln}^{(1)} \stackrel{\text{def}}{=} \log \frac{P\left(v_n = 0 \mid \{r_{n'}, n' \in \mathcal{N}(l) \setminus \{n\}\}\right)}{P\left(v_n = 1 \mid \{r_{n'}, n' \in \mathcal{N}(l) \setminus \{n\}\}\right)} \tag{8.133}$$

where $P\left(v_n = 0 \mid \{r_{n'}, n' \in \mathcal{N}(l) \setminus \{n\}\}\right)$ and $P\left(v_n = 1 \mid \{r_{n'}, n' \in \mathcal{N}(l) \setminus \{n\}\}\right)$ are the conditional probabilities that $v_n = 0$ and $v_n = 1$, respectively, given the set of the received symbols $\{r_{n'}, n' \in \mathcal{N}(l) \setminus \{n\}\}$. Since

$$\begin{aligned} &P\left(v_n = 0 \mid \{r_{n'}, n' \in \mathcal{N}(l) \setminus \{n\}\}\right) \\ &\quad = P\left(\sum_{n' \in \mathcal{N}(l) \setminus \{n\}} v_{n'} = 0 \,\middle|\, \{r_{n'}, n' \in \mathcal{N}(l) \setminus \{n\}\}\right) \end{aligned} \tag{8.134}$$

and

$$\begin{aligned} &P\left(v_n = 1 \mid \{r_{n'}, n' \in \mathcal{N}(l) \setminus \{n\}\}\right) \\ &\quad = P\left(\sum_{n' \in \mathcal{N}(l) \setminus \{n\}} v_{n'} = 1 \,\middle|\, \{r_{n'}, n' \in \mathcal{N}(l) \setminus \{n\}\}\right) \end{aligned} \tag{8.135}$$

we have

$$\begin{aligned} &P\left(v_n = 0 \mid \{r_{n'}, n' \in \mathcal{N}(l) \setminus \{n\}\}\right) \\ &\quad = \frac{1}{2}\left(\prod_{n' \in \mathcal{N}(l) \setminus \{n\}} \left(P\left(v_{n'} = 1 \mid r_{n'}\right) + P\left(v_{n'} = 0 \mid r_{n'}\right)\right)\right. \\ &\qquad \left. + \prod_{n' \in \mathcal{N}(l) \setminus \{n\}} \left(P\left(v_{n'} = 1 \mid r_{n'}\right) - P\left(v_{n'} = 0 \mid r_{n'}\right)\right)\right) \end{aligned} \tag{8.136}$$

and

$$P\left(v_n = 1 \mid \{r'_n, n' \in \mathcal{N}(l) \setminus \{n\}\}\right)$$
$$= \frac{1}{2}\left(\prod_{n' \in \mathcal{N}(l)\setminus\{n\}} \left(P\left(v_{n'} = 1 \mid r_{n'}\right) + P\left(v_{n'} = 0 \mid r_{n'}\right)\right) \right.$$
$$\left. - \prod_{n' \in \mathcal{N}(l)\setminus\{n\}} \left(P\left(v_{n'} = 1 \mid r_{n'}\right) - P\left(v_{n'} = 0 \mid r_{n'}\right)\right) \right) \quad (8.137)$$

The log-likelihood ratio $y_{ln}^{(1)}$ can be expressed as a function of the $(K-1)$ log-likelihood ratios $z_{n'}^{(0)}$, $n' \in \mathcal{N}(l) \setminus \{n\}$. Using the equality $(e^{z_{n'}^{(0)}} - 1)/(e^{z_{n'}^{(0)}} + 1) = \tanh(z_{n'}^{(0)}/2)$ we obtain, from (8.133), (8.136), and (8.137),

$$y_{ln}^{(1)} \stackrel{\text{def}}{=} \log \frac{P\left(v_n = 0 \mid \{r_{n'}, n' \in \mathcal{N}(l) \setminus \{n\}\}\right)}{P\left(v_n = 1 \mid \{r_{n'}, n' \in \mathcal{N}(l) \setminus \{n\}\}\right)}$$
$$= \log \frac{1 + \prod_{n' \in \mathcal{N}(l)\setminus\{n\}} \tanh(z_{n'l}^{(0)}/2)}{1 - \prod_{n' \in \mathcal{N}(l)\setminus\{n\}} \tanh(z_{n'l}^{(0)}/2)} \quad (8.138)$$

The calculation of the statistics $y_{ln}^{(1)}$, $l = 0, 1, \ldots, L-1$, $n \in \mathcal{N}(l)$, completes the first phase of the first step of the iterative decoding process.

During the second phase of the first step of the iterative decoding process, the lth, $l = 0, 1, \ldots, L-1$, constraint node sends log-likelihood ratios $y_{ln}^{(1)}$, $n \in \mathcal{N}(l)$, to the corresponding connected symbol nodes. Then the nth symbol node, which receive messages from the connected constraint nodes, calculates J log-likelihood ratios $z_{nl}^{(1)}$, $l \in \mathcal{L}(n)$,

$$z_{nl}^{(1)} \stackrel{\text{def}}{=} z_n^{(0)} + \sum_{l' \in \mathcal{L}(n)\setminus\{l\}} y_{l'n}^{(1)}$$
$$= z_n^{(0)} + \sum_{l' \in \mathcal{L}(n)\setminus\{l\}} \log \frac{1 + \prod_{n' \in \mathcal{N}(l')\setminus\{n\}} \tanh(z_{n'l'}^{(0)}/2)}{1 - \prod_{n' \in \mathcal{N}(l')\setminus\{n\}} \tanh(z_{n'l'}^{(0)}/2)} \quad (8.139)$$

Here $y_{l'n}^{(1)}$, $l' \in \mathcal{L}(n) \setminus \{l\}$, are messages, called *extrinsic information*, which the nth symbol node receives from the connected constraint nodes. Calculating the statistics $z_{nl}^{(1)}$, $l \in \mathcal{L}(n)$, completes the second phase of the first step of the iteration.

Next we describe the ith step of the iterative process, $i = 1, 2, \ldots, I-1$. During the first phase of the ith iteration, the nth symbol node sends messages $z_{nl}^{(i-1)}$, which were calculated in the previous step, to the connected constraint nodes $l \in \mathcal{L}(n)$. Then the lth constraint node, $l = 0, 1, \ldots, L-1$, which receives K messages $z_{nl}^{(i-1)}$, $n \in \mathcal{N}(l)$, from the connected symbol nodes, calculates the statistics $y_{ln}^{(i)}$, $n \in \mathcal{N}(l)$ (cf. (8.138))

$$y_{ln}^{(i)} = \log \frac{1 + \prod_{n' \in \mathcal{N}(l)\setminus\{n\}} \tanh(z_{n'l}^{(i-1)}/2)}{1 - \prod_{n' \in \mathcal{N}(l)\setminus\{n\}} \tanh(z_{n'l}^{(i-1)}/2)} \quad (8.140)$$

During the second phase of the ith step, $i = 1, 2, \ldots, I-1$, of the iterative decoding process the constraint node l, $l = 0, 1, \ldots, L-1$, sends $y_{ln}^{(i)}$ as the message to the connected nth symbol node, $n \in \mathcal{N}(l)$. Then the symbol node n calculates the statistics (cf. (8.139))

$$\begin{aligned} z_{nl}^{(i)} &= z_n^{(0)} + \sum_{l' \in \mathcal{L}(n) \setminus \{l\}} y_{l'n}^{(i)} \\ &= z_n^{(0)} + \sum_{l' \in \mathcal{L}(n) \setminus \{l\}} \log \frac{1 + \prod_{n' \in \mathcal{N}(l') \setminus \{n\}} \tanh(z_{n'l'}^{(i-1)}/2)}{1 - \prod_{n' \in \mathcal{N}(l') \setminus \{n\}} \tanh(z_{n'l'}^{(i-1)}/2)} \end{aligned} \tag{8.141}$$

which this symbol node will use as messages in the following step of the iterative process. This completes the second phase of the ith step of iteration for $i = 1, 2, \ldots, I-1$.

Note that (8.141) is used for calculation of $z_{nl}^{(i)}$ for all steps of the decoding, except the final Ith step. In the last step the decoder calculates only one log-likelihood ratio $z_n^{(I)}$ for the nth symbol node using

$$z_n^{(I)} = z_n^{(0)} + \sum_{l \in \mathcal{L}(n)} y_{ln}^{(I)} \tag{8.142}$$

$$= z_n^{(0)} + \sum_{l \in \mathcal{L}(n)} \log \frac{1 + \prod_{n' \in \mathcal{N}(l) \setminus \{n\}} \tanh(z_{n'l}^{(I-1)}/2)}{1 - \prod_{n' \in \mathcal{N}(l) \setminus \{n\}} \tanh(z_{n'l}^{(I-1)}/2)} \tag{8.143}$$

Then it makes a hard decision $\widehat{v}_n$ about the symbol v_n using the rule

$$\widehat{v}_n = \begin{cases} 0 & \text{if } z_n^{(I)} > 0 \\ 1 & \text{if } z_n^{(I)} < 0 \end{cases} \tag{8.144}$$

(If $z_n^{(I)} = 0$ the decoder flips a coin, choosing $\widehat{v}_n = 0$ or $\widehat{v}_n = 1$ with probability $1/2$). Since the decoding algorithm can be reduced to alternatively evaluating sums and products, the algorithm is sometimes called the *sum-product algorithm* [KFL01].

We have described the belief propagation algorithm in the case of a binary-input DMC. Generalizations of the algorithm on other binary-input memoryless channels, such as the binary-input additive white Gaussian noise channel, are straightforward. In the latter case, instead of (8.132) we operate with the intrinsic information

$$z_n^{(0)} = \frac{f_0(r_n)}{f_1(r_n)} \tag{8.145}$$

where $f_0(r_n)$ and $f_1(r_n)$ are the conditional probability density functions of the received symbol r_n given that the transmitted symbol is equal to 0 and 1, respectively. The BP algorithm for the DMC can be summarized as follows:

Algorithm BPDMC (The BP algorithm for decoding the output of a DMC)

BPDMC1. Initialize $i = 0$ and $z_n^{(0)}$, $n = 0, 1, \ldots, N-1$.

BPDMC2. For all $l \in \mathcal{L}(n)$ and $n = 0, 1, \ldots, N-1$, set $z_{nl}^{(0)} = z_n^{(0)}$.

BPDMC3. While $0 < i < I$, for all $n \in \mathcal{N}(l)$ and $l = 0, 1, \ldots, L-1$, compute

$$y_{ln}^{(i)} = \log \frac{1 + \prod_{n' \in \mathcal{N}(l) \setminus \{n\}} \tanh(z_{n'l}^{(i-1)}/2)}{1 - \prod_{n' \in \mathcal{N}(l) \setminus \{n\}} \tanh(z_{n'l}^{(i-1)}/2)}$$

BPDMC4. For all $l \in \mathcal{L}(n)$ and $n = 0, 1, \ldots N-1$, compute

$$z_{nl}^{(i)} = z_n^{(0)} + \sum_{l' \in \mathcal{L}(n) \setminus \{l\}} y_{l'n}^{(i)}$$

BPDMC5. For $i = I$, all $n \in \mathcal{N}(l)$, and $l = 0, 1, \ldots, L-1$, compute

$$y_{ln}^{(I)} = \log \frac{1 + \prod_{n' \in \mathcal{N}(l) \setminus \{n\}} \tanh(z_{n'l}^{(I-1)}/2)}{1 - \prod_{n' \in \mathcal{N}(l) \setminus \{n\}} \tanh(z_{n'l}^{(I-1)}/2)}$$

BPDMC6. For all $n = 0, 1, \ldots, N-1$, compute

$$z_n^{(I)} = z_n^{(0)} + \sum_{l \in \mathcal{L}(n)} y_{ln}^{(I)}$$

BPDMC7. For all $n = 0, 1, \ldots, N-1$, choose the estimation $\widehat{v}_n$ according to (8.144).

Consider iterative decoding of (m_{s}, J, K) LDPC convolutional codes for binary-input DMC or AWGN channels. We assume that the information sequence

$$\boldsymbol{u} = \boldsymbol{u}_{[0,\infty)} = \boldsymbol{u}_0 \boldsymbol{u}_1 \ldots \boldsymbol{u}_t \ldots \tag{8.146}$$

and the code sequence

$$\boldsymbol{v} = \boldsymbol{v}_{[0,\infty)} = \boldsymbol{v}_0 \boldsymbol{v}_1 \ldots \boldsymbol{v}_t \ldots \tag{8.147}$$

are semi-infinite sequences and that the syndrome former is a semi-infinite matrix

$$\boldsymbol{H}_{[0,\infty)}^{\mathrm{T}} = \begin{pmatrix} H_0^{\mathrm{T}}(0) & H_1^{\mathrm{T}}(1) & \ldots & H_{m_{\mathrm{s}}}^{\mathrm{T}}(m_{\mathrm{s}}) & & \\ & H_0^{\mathrm{T}}(1) & H_1^{\mathrm{T}}(2) & \ldots & H_{m_{\mathrm{s}}}^{\mathrm{T}}(m_{\mathrm{s}}+1) & \\ & & \ddots & \ddots & & \ddots \end{pmatrix}$$

We shall restrict ourselves to describe decoding of a regular (m_{s}, J, K) LDPC convolutional code encoded by a systematic encoder of rate $R = 1/2$ with parameters

$b = 1$ and $c = 2$ when $H_0^{\mathrm{T}}(t)$, $t = 0, 1, \ldots$, is the two-dimensional column vector $(1\,1)^{\mathrm{T}}$. The first symbol v_{2t} of the tuple $\boldsymbol{v}_t = v_{2t}v_{2t+1}$ is an information symbol and the second symbol v_{2t+1} is a parity symbol. Since each parity-check equation is associated with one of the columns of the syndrome former, we can enumerate constraint nodes according to the enumeration of the columns of $\boldsymbol{H}^{\mathrm{T}}_{[0,\infty)}$. Particularly, symbol nodes corresponding to v_{2t} and v_{2t+1} are always connected with the tth constraint node. In other words, $t \in \mathcal{L}(2t)$, $t \in \mathcal{L}(2t+1)$, $2t \in \mathcal{N}(t)$, and $2t+1 \in \mathcal{N}(t)$. Although we consider a particular structure of LDPC convolutional codes, the algorithm can be easily modified to general LDPC convolutional codes.

There are two variants of the belief propagation decoding algorithm for LDPC convolutional codes, an *algorithm with parallel updating* and an *algorithm with on-demand updating*. Both algorithms assume *pipeline implementations*. We shall describe the algorithm with parallel updating in detail. For our description we shall use the Tanner graph.

Although the Tanner graph of the LDPC convolutional code has an infinite number of nodes, the distance between two symbol nodes that are connected to the same constraint node is limited by the code memory. This allows continuous decoding with a decoder that operates on a finite window sliding along the received sequence, similar to a Viterbi decoder with finite back-search limit (see Section 5.5). The decoding of two symbols that are at least $2(m_{\mathrm{s}}+1)$ time instants apart can be performed independently, since these symbols cannot participate in the same parity-check equation.

A pipeline decoder of a regular (m_{s}, J, K) LDPC convolutional code consists of set of I separate processors $\{\mathcal{D}_i\}_{i=1}^{I}$. At the tth time instant the processors of the pipeline decoder operate within the set of symbol nodes v_n, $n \in \mathcal{T}(t) = \{2t, 2t+1, \ldots, 2t-2I(m_{\mathrm{s}}+1), 2t-2I(m_{\mathrm{s}}+1)+1\}$ located in a window of width $2I(m_{\mathrm{s}}+1)$. We assume that the intrinsic information $z_n^{(0)}$ (see (8.132) and (8.145)) for the symbols v_n, $n \in \mathcal{T}(t)$, is available at the decoder. The ith processor, $i = 1, 2, \ldots, I$, operates at the tth time instant with the t_ith constraint node, where $t_i = t - (i-1)(m_{\mathrm{s}}+1)$, and the K symbol nodes connected with this constraint node. Each processor operates in a window of width (in symbols) $2(m_{\mathrm{s}}+1)$ such that the windows in which two processors operate do not overlap and each symbol is covered by one window. Particularly, the window in which the ith processor operates at the tth time instant includes the symbols v_{2t_i} and v_{2t_i+1}.

Let $\mathcal{T}_i(t)$ denote the set of K symbol nodes with which processor $\mathcal{D}_i$ operates at the tth time instant. We shall say that at the tth time instant processor $\mathcal{D}_i$ *activates* these symbol nodes. Simultaneously, $\mathcal{D}_i$ activates the constraint node c_{t_i}.

At the tth time instant, the decoder keeps in memory two sets of decision statistics. The first set, $\mathcal{Z}(t) = \{z_{nl}(t),\ n \in \mathcal{T}(t),\ l \in \mathcal{L}(n)\}$, consists of $2J(m_{\mathrm{s}}+1)I$ statistics, a counterpart to the statistics (8.141). The second set, $\mathcal{Y}(t) = \{y_{ln}(t),\ l = t, t-1, \ldots, t-(I-1)(m_{\mathrm{s}}+1),\ n \in \mathcal{N}(l)\}$, consists of $K(m_{\mathrm{s}}+1)I$ statistics, a counterpart of statistics (8.140). Particularly, at $t = 0$, we have

$$z_{nl}(0) = \infty,\ \ n < 0,\ \ l \in \mathcal{L}(n) \tag{8.148}$$

(that is, the corresponding symbols v_n are known to be 0).

At time instant t the decoder updates the sets of the statistics $\mathcal{Z}(t-1)$ and $\mathcal{Y}(t-1)$ and replaces them by the sets of statistics $\mathcal{Z}(t)$ and $\mathcal{Y}(t)$ according to an algorithm which we shall describe later. Particularly, the decoder adds $2J$ new statistics $z_{nl}(t) = z_n^{(0)}$, $n = 2t, 2t+1$, $l \in \mathcal{L}(n)$, and excludes $2J$ statistics $z_{nl}(t-1)$, $n = 2t - 2I(m_s+1) - 2, 2t - 2I(m_s+1) - 1$, $l \in \mathcal{L}(n)$, which will not participate in the future decoding process. The decoder also sets $y_{ln}(t-1) = 0$, $n = 2t, 2t+1$, $l \in \mathcal{L}(n)$.

Next we describe the updating process at the tth time instant, $t = 1, 2, \ldots$, for the ith processor, $i = 1, 2, \ldots, I$. It consists of two phases. During the first phase, the vertical step, the symbol nodes $n \in \mathcal{T}_i(t)$ send messages (log-likelihood ratios $z_{nt_i}(t-1)$) to the constraint node c_{t_i}. These log-likelihood ratios were calculated by these symbol nodes in the previous steps of the iterative decoding process analogously to the statistics $z_{nl}^{(i)}$ in the iterative decoding of LDPC block codes (cf. (8.141)). Then the constraint node c_{t_i} calculates the statistics (cf. (8.140))

$$y_{t_i n}(t) = \log \frac{1 + \prod_{n' \in \mathcal{N}(t_i) \setminus n} \tanh(z_{n' t_i}(t-1)/2)}{1 - \prod_{n' \in \mathcal{N}(t_i) \setminus n} \tanh(z_{n' t_i}(t-1)/2)}, \quad n \in \mathcal{T}_i(t) \tag{8.149}$$

and replaces the statistics $y_{t_i n}(t-1)$, $i = 1, 2, \ldots, I$, in $\mathcal{Y}$ by the statistics $y_{t_i n}(t)$. These statistics will be used in the following decoding process.

During the second phase, that is, the horizontal step, of the updating process at the tth time instant, $t = 1, 2, \ldots$, the ith processor, $i = 1, 2, \ldots, I-1$, calculates new statistics $z_{2t_i l}(t)$, $l \in \mathcal{L}(2t_i)$, $l \neq t_i$, and $z_{2t_i+1,l}(t)$, $l \in \mathcal{L}(2t_i+1)$, $l \neq t_i$, according to

$$z_{nl}(t) = z_n^{(0)} + \sum_{l' \in \mathcal{L}(n) \setminus l} y_{l' n}(t-1), \quad n = 2t_i, 2t_i + 1 \tag{8.150}$$

The other symbol and constraint nodes, $n \in \mathcal{T}(t)$, $l \in \mathcal{L}(n)$, do not change their statistics z_{nl} and y_{ln},

$$\begin{aligned} z_{nl}(t) &= z_{nl}(t-1),\ n \in \mathcal{T}(t),\ n \notin \bigcup_{i=1}^{I} \mathcal{T}_i(t), \qquad l \in \mathcal{L}(n) \\ y_{ln}(t) &= y_{ln}(t-1),\ n \in \mathcal{T}(t),\ n \notin \bigcup_{i=1}^{I} \mathcal{T}_i(t), \qquad l \in \mathcal{L}(n) \end{aligned} \tag{8.151}$$

The decoder will use the newly calculated statistics in the following steps of the iterative decoding process.

During the second phase of the updating process the Ith processor calculates the statistics

$$z_n(t) = z_n^{(0)} + \sum_{l \in \mathcal{L}(n)} y_{ln}(t-1), \quad n = 2t_I \tag{8.152}$$

Then it makes the hard decision with respect to the information symbol v_{2t_I} (cf. (8.144)):

$$\widehat{v}_{2t_I} = \begin{cases} 0 & \text{if } z_{2t_I} \geq 0, \\ 1 & \text{otherwise} \end{cases} \tag{8.153}$$

and output this symbol.

We have described a variant of pipeline decoding of regular LDPC convolutional codes with parallel updating. More complicated and more effective variants of pipeline decoding with on-demand updating are considered in [EnZ99, PJS08]. In this case the updating of the statistics $y_{t_i n}(t)$ by the ith processor, $i = 1, 2, \ldots, I$, at the tth time instant is the same as for the decoding algorithm with parallel updating, that is, it follows (8.149). But instead of (8.150) for calculation of the statistics $z_{nl}(t)$ we should use[20]

$$z_{nl}(t) = \begin{cases} z_{nl}(t-1) & l = t_i \\ z_{nl}(t-1) + y_{t_i n}(t) - y_{t_i n}(t-1) & l \in \mathcal{L}(n),\ \ l \neq t_i \end{cases} \tag{8.154}$$

In other words, the activated symbol nodes receive messages only from the activated constraint nodes (with parallel updating, the activated symbol nodes receive messages from all connected constraint nodes; nonactivated constraint nodes send messages based on nonupdated statistics calculated during the previous steps).

Equation (8.154) assumes that the constraint node c_{t_i} does not send statistics $y_{t_i n}(t)$ to the connected symbol nodes as in the basic case. Instead it sends the difference between the new and old statistics y, that is, $y_{t_i n}(t) - y_{t_i n}(t-1)$. The Ith processor makes the hard decision with respect to the information symbol v_{2t_I} (cf. (8.144)) using the rule

$$\widehat{v}_{2t_I} = \begin{cases} 0 & \text{if } z_{2t_I, t_I}(t-1) + y_{t_I, 2t_I}(t) - y_{t_I, 2t_I}(t-1) \geq 0, \\ 1 & \text{otherwise} \end{cases} \tag{8.155}$$

and output this symbol.

8.6 ITERATIVE LIMITS AND THRESHOLDS

In Section 1.1 we introduced the Shannon limit for communication with rate R over the AWGN channel. It is defined as the lowest signal-to-noise ratio per bit, $E_{\mathrm{b}}/N_0 = \eta_{\mathrm{sh}}(R)$, for which reliable communication over the channel is possible. For the BEC/BSC, the Shannon limit is defined as the largest erasure/crossover probability for which it is possible to communicate reliably over the channel with code rate R. In all cases we assume no limitations on the used coding or decoding methods.

[20]Note that the symbol node c_n, $n \in \mathcal{T}_i(t)$, receives only one message, $y_{t_i n}(t)$, not J as in decoding of LDPC block codes.

Suppose that we study communication over the AWGN channel and that we are restricted to use a certain class of codes, for example, LDPC block or convolutional codes, and a class of iterative decoding algorithms, for example, belief propagation decoding algorithms. We denote the family of LDPC block codes of length N by $\mathcal{F}_N$ and the family of iterative decoding algorithms for these codes by $\mathcal{D}_{\text{block}}$.

Let $P_{\text{b}}^{(I)}(\mathcal{C}, \mathcal{A}_{\text{b}})$ denote the decoding bit error probability of a code $\mathcal{C}$, $\mathcal{C} \in \mathcal{F}_N$, after I decoding iterations using a decoding algorithm $\mathcal{A}_{\text{b}}$, $\mathcal{A}_{\text{b}} \in \mathcal{D}_{\text{block}}$.

Definition The *iterative decoding limit* for the code family $\mathcal{F}_N$ and the decoding algorithm family $\mathcal{D}_{\text{block}}$ used to communicate over the binary-input AWGN channel is η_{id} if, for all $E_{\text{b}}/N_0 \geq \eta_{\text{id}}$, any $\varepsilon > 0$, and sufficiently large positive integers N and I, there exists a code $\mathcal{C}$, $\mathcal{C} \in \mathcal{F}_N$, and an iterative decoding algorithm $\mathcal{A}_{\text{b}}$, $\mathcal{A}_{\text{b}} \in \mathcal{D}_{\text{block}}$, such that

$$P_{\text{b}}^{(I)}(\mathcal{C}, \mathcal{A}_{\text{b}}) < \varepsilon \tag{8.156}$$

Furthermore, if $E_{\text{b}}/N_0 < \eta_{\text{id}}$, then there exists an $\varepsilon > 0$ such that for any N and I, any code $\mathcal{C}$, $\mathcal{C} \in \mathcal{F}_N$, and any decoding algorithm $\mathcal{A}_{\text{b}}$, $\mathcal{A}_{\text{b}} \in \mathcal{D}_{\text{block}}$, we have

$$P_{\text{b}}^{(I)}(\mathcal{C}, \mathcal{A}_{\text{b}}) \geq \varepsilon \tag{8.157}$$

We denote a family of LDPC convolutional codes of syndrome former memory m_{s} by $\mathcal{F}_{m_{\text{s}}}$, a family of iterative decoding algorithms for these codes by $\mathcal{D}_{\text{conv}}$. Let $P_{\text{b}}^{(I)}(\mathcal{C}, \mathcal{A}_{\text{c}})$ denote the decoding bit error probability of a convolutional code $\mathcal{C}$, $\mathcal{C} \in \mathcal{F}_{m_{\text{s}}}$, after I decoding iterations using decoding algorithm $\mathcal{A}_{\text{c}}$, $\mathcal{A}_{\text{c}} \in \mathcal{D}_{\text{conv}}$. Then the iterative decoding limit for LDPC convolutional codes can be defined analogously to the iterative decoding limit for LDPC block codes for the binary-input AWGN channel (see Problem 8.21).

In contrast to the Shannon limits, which are well known for most memoryless channels, iterative decoding limits are presently not known for LDPC codes with belief propagation decoding. Because of this, we can only characterize these iterative decoding limits by lower and upper bounds. Obviously, the Shannon limit is a lower bound for the iterative decoding limit for the AWGN channel. An upper bound for the iterative decoding limit when we communicate over the AWGN channel can be calculated using *density evolution*. Such an upper bound is called a *threshold* and denoted η_{th}; thus, we have

$$\eta_{\text{sh}} \leq \eta_{\text{id}} \leq \eta_{\text{th}} \tag{8.158}$$

It is straightforward to generalize the definitions of the iterative limit and threshold to the BEC and BSC. Since the definitions are essentially identical for both channels, we formulate the definitions only for the BEC.

Consider the BEC with erasure probability δ and a family $\mathcal{F}_N$ of binary LDPC block codes of length N. Let $\mathcal{D}_{\text{block}}$ denote a family of iterative decoding algorithms for the BEC. Let $P_{\text{b}}^{(I)}(\mathcal{C}, \mathcal{A}_{\text{b}})$ denote the bit error probability after I decoding iterations for a code $\mathcal{C}$, $\mathcal{C} \in \mathcal{F}_N$, and for a decoding algorithm $\mathcal{A}_{\text{b}}$, $\mathcal{A}_{\text{b}} \in \mathcal{D}_{\text{block}}$.

Definition The δ_{id} is called the *iterative decoding limit* for the code family $\mathcal{F}_N$ and decoding algorithm family $\mathcal{D}_{\text{block}}$ used to communicate over the BEC if for all erasure probabilities $\delta \leq \delta_{\text{id}}$, any $\varepsilon > 0$, and sufficiently large positive integers N and I there exists a code $\mathcal{C}$, $\mathcal{C} \in \mathcal{F}_N$, and decoding algorithm $\mathcal{A}_{\text{b}}$, $\mathcal{A}_{\text{b}} \in \mathcal{D}_{\text{block}}$, such that

$$P_{\text{b}}^{(I)}(\mathcal{C}, \mathcal{A}_{\text{b}}) < \varepsilon \tag{8.159}$$

Furthermore, if $\delta > \delta_{\text{id}}$, there exists an $\varepsilon > 0$ such that for any N and I, any code $\mathcal{C}$, $\mathcal{C} \in \mathcal{F}_N$, and any decoding algorithm $\mathcal{A}_{\text{b}}$, $\mathcal{A}_{\text{b}} \in \mathcal{D}_{\text{block}}$, we have

$$P_{\text{b}}^{(I)}(\mathcal{C}, \mathcal{A}_{\text{b}}) \geq \varepsilon \tag{8.160}$$

The *threshold* δ_{th} for an LDPC code used together with the BEC such that

$$\delta_{\text{th}} \leq \delta_{\text{id}} \tag{8.161}$$

can be calculated by the density evolution method, which in the case of belief propagation decoding for the BEC is called *probability evolution*.

In the previous section we described the belief propagation decoding of regular (N, J, K) LDPC codes for the BEC using a Tanner graph. Now we explain this decoding method using the computational tree introduced in Section 8.1 (cf. Fig. 8.2).

Consider an arbitrary $2\ell_0$-cyclefree regular (N, J, K) LDPC block code. We describe the decoding of a symbol v_n using the tree presentation of a v_n-clan (cf. Fig. 8.2). We can assume that the decoding of a symbol v_n starts from level $2\ell_0$ (the ℓ_0th generation) of the tree and that for the symbols of this generation only the intrinsic information from the channel is available, that is, a symbol is known only if it has not been erased by the channel. A symbol of the previous $(\ell_0 - 1)$th generation is known not only if it is not erased by the channel but also if, in at least one of its families, no symbols has been erased. Therefore, in the first decoding iteration we can reconstruct some of the erased symbols in the $(\ell_0 - 1)$th generation. The same method can be used in the ith iteration for reconstruction of symbols in the $(\ell_0 - i)$th generation. The probability $p^{(i)}$ that after the ith iteration a symbol in the $(\ell_0 - i)$th generation remains erased is given by

$$\begin{aligned} p^{(0)} &= \delta \\ p^{(i)} &= f(p^{(i-1)}) \stackrel{\text{def}}{=} \delta(1 - (1 - p^{(i-1)})^{K-1})^{J-1}, \quad i > 0 \end{aligned} \tag{8.162}$$

Note that (8.162) is valid only if the iterations are independent, that is, if the number of iterations I does not exceed ℓ_0.

The process of calculating the erasure probabilities $\delta = p^{(0)} \to p^{(1)} \to \cdots p^{(i)} \to \cdots \to p^{(I)}$ described by equation (8.162) is called *probability evolution*. This process can be used for analysing the asymptotic behavior of the belief propagation algorithm when $N \to \infty$, $I = \ell_0 \to \infty$. In fact, suppose that for given $\delta = \delta_1$ the probability $p^{(I)} \to 0$ when $I \to \infty$. Then δ_1 is a lower bound on δ_{id}, that is, $\delta_1 \leq \delta_{\text{id}}$. To find the threshold δ_{th}, we have to find the maximum value δ_1 for which $p^{(I)} \to 0$ when $I \to \infty$.

The following example illustrates the calculation of the threshold for an LDPC block code used to communicate over BEC.

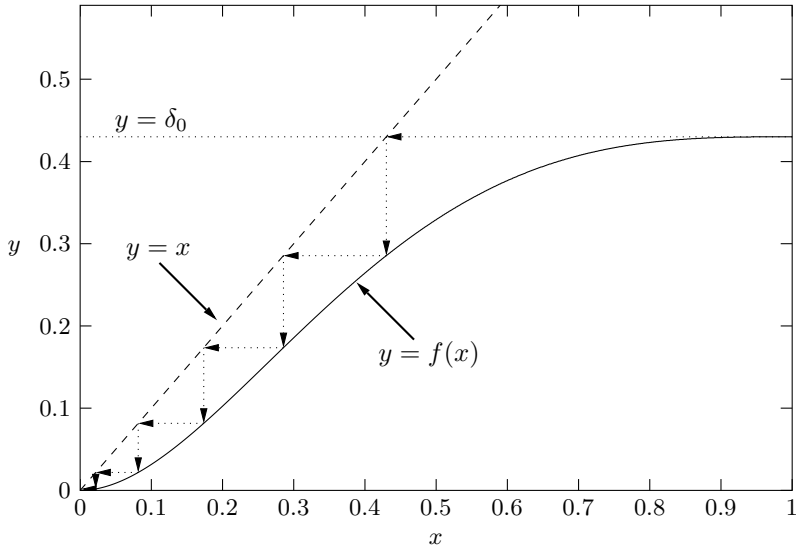


Figure 8.17 Illustration of the evolution of the probability $p^{(i)}$.

■ **EXAMPLE 8.10**

Figure 8.17 shows the function $y = f(x)$ defined by (8.162) for $J = 3, K = 4$, and $\delta = 0.43$. This function is below the main diagonal $y = x$. Then $p^{(I)} \to 0$, when $I \to \infty$ and we conclude that $\delta_{\text{th}} \geq \delta$.

Since the probability evolution process cannot be infinite, we have to stop the process either when $p^{(i)}$ becomes sufficiently small or when it remains larger than some positive constant after sufficiently many iterations. In the first case, we may either increase δ and repeat the process or declare current value δ as the threshold. In the second case, we decrease δ and repeat the process.

Next we shall establish what is meant by "sufficiently small $p^{(i)}$" and what is meant by "sufficiently many iterations."

Theorem 8.19 Consider the probability evolution process for a regular (N, J, K) LDPC code with $J > 2$ used together with the BEC with erasure probability δ. Suppose that during the i_0th step of the process the probability $p^{(i_0)}$ becomes less than the *breakout value*

$$p_{\text{br}} \stackrel{\text{def}}{=} \delta^{-1/(J-2)}(K-1)^{-(J-1)/(J-2)} \tag{8.163}$$

Then $p^{(I)} \to 0$ when $I \to \infty$.

Proof: Consider (8.162) for $J > 2$. From the inequality

$$1 - (1-x)^{K-1} < (K-1)x, \;\; x \in (0,1) \tag{8.164}$$

it follows that

$$p^{(i)} < \delta((K-1)p^{(i-1)})^{(J-1)} = (\gamma p^{(i-1)})^{J-1} \tag{8.165}$$

where $\gamma = \delta^{1/(J-1)}(K-1)$. From (8.165) it follows that

$$\begin{aligned} p^{(I)} &< (\gamma p^{(I-1)})^{J-1} < \gamma^{(J-1)}(\gamma p^{(I-2)})^{(J-1)^2} \\ &< \cdots < (\gamma^{\frac{J-1}{J-2}} p^{(i_0)})^{(J-1)^{(I-i_0)}} \end{aligned} \tag{8.166}$$

If $p^{(i_0)} < p_{\text{br}} = \delta^{-1/(J-2)}(K-1)^{-(J-1)/(J-2)}$, then $p^{(I)} \to 0$ when $I \to \infty$. Particularly, $\delta = p^{(0)} < p_{\text{br}} = \delta^{-1/(J-2)}(K-1)^{-(J-1)/(J-2)}$ if $\delta < 1/(K-1)$. ■

From Theorem 8.19 it follows that a "sufficiently small" $p^{(i)}$ is p_{br}. "Sufficiently many iterations I" can be determined as follows. For example, if we let $I = 1000$ and after 1000 steps of the probability evolution process have $p^{(I)} > p_{\text{br}}$ then we stop the process, decrease δ, and repeat the process with a smaller value of δ.

For $J = 2$ the breakout value p_{br} can be found analytically from (8.165). It is equal to $1/(K-1)$ (see Problem 8.18).

Next we describe the density evolution process for a $2\ell_0$-cyclefree regular (N, J, K) LDPC code used together with the AWGN channel (the density evolution process for a DMC can be described analogously). Let $\boldsymbol{v} = v_0 v_1 \ldots v_{N-1}$ denote a block of the transmitted code symbols and $\boldsymbol{r} = r_0 r_1 \ldots r_{N-1}$ denote the corresponding block of the received symbols. Assume that the belief propagation iterative decoding algorithm described in Section 8.5 is used with a total number of iterations $I = \ell_0$. As for the BEC, it is convenient to study the iterative decoding process of a symbol v_{n_0} in the tree-shaped graph, grown from the symbol v_{n_0}.

Let $\mathcal{N}_\ell$ denote the set of symbol nodes at level 2ℓ, $\ell = 0, 1, \ldots, \ell_0$, that is, the ℓth generation of the v_{n_0}-clan, and let $\mathcal{L}_\ell$ denote the set of constraint nodes at level $2\ell + 1$, that is, the families of symbols of the ℓ generation. Let $\mathcal{L}_\ell(n)$ denote the set of families of a symbol v_n of the ℓth generation and let $\mathcal{N}_{\ell+1}(l)$ denote the set of direct descendants of a symbol node v_n of the ℓth generation which belongs to family $l, l \in \mathcal{L}_\ell(n)$. The intrinsic information $z_n^{(0)}$ about the symbol v_n is defined as

$$z_n^{(0)} = \log \frac{f_0(r_n)}{f_1(r_n)} \tag{8.167}$$

where

$$\begin{aligned} f_0(r) &= \frac{1}{\sqrt{\pi N_0}} e^{-\frac{(r-\sqrt{E_{\text{s}}})^2}{N_0}} \\ f_1(r) &= \frac{1}{\sqrt{\pi N_0}} e^{-\frac{(r+\sqrt{E_{\text{s}}})^2}{N_0}} \end{aligned} \tag{8.168}$$

and E_{s} is energy per symbol. Then the probability density functions $\varphi^{(0)}(z)$ and $\psi^{(0)}(z)$ of the random variable $z_n^{(0)}$, given that $v_n = 0$ and $v_n = 1$, respectively, are

$$\begin{aligned} \varphi^{(0)}(z) &= \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(z-\mu)^2}{2\sigma^2}} \\ \psi^{(0)}(z) &= \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(z+\mu)^2}{2\sigma^2}} \end{aligned} \tag{8.169}$$

where $\mu = 4E_s/N_0$, $\sigma^2 = 16E_s/N_0$. From (8.169) it follows that

$$\varphi^{(0)}(z) = \psi^{(0)}(-z) \tag{8.170}$$

and

$$\varphi^{(0)}(z) \begin{cases} > \psi^{(0)}(z) & \text{if } z > 0 \\ = \psi^{(0)}(z) & \text{if } z = 0 \\ < \psi^{(0)}(z) & \text{if } z < 0 \end{cases} \tag{8.171}$$

The decoder starts from the ℓ_0th generation of the v_{n_0}-clan and proceeds level by level up towards the clan head. In the ith iteration, $i = 1, 2, \ldots, \ell_0$, all symbol nodes n, $n \in \mathcal{N}_{\ell_0 - i + 1}$, send messages $z_{nl}^{(i-1)}$ to the corresponding constraint nodes of the level $2(\ell_0 - i) + 1$ such that the symbol nodes n, $n \in \mathcal{N}_{\ell_0 - i + 1}(l)$, send messages $z_{nl}^{(i-1)}$ to constraint node l. The constraint node l, $l \in \mathcal{L}_{\ell_0 - i}$, receives messages $z_{nl}^{(i-1)}$ from $K - 1$ nodes $n \in \mathcal{N}_{\ell_0 - i + 1}(l)$ and calculates the statistics (cf. (8.140))

$$y_{ln}^{(i)} = \log \frac{1 + \prod_{n \in \mathcal{N}_{\ell_0 - i + 1}(l)} \tanh(z_{nl}^{(i-1)}/2)}{1 - \prod_{n \in \mathcal{N}_{\ell_0 - i + 1}(l)} \tanh(z_{nl}^{(i-1)}/2)}, \quad l \in \mathcal{L}_{\ell_0 - i}(n), \quad n \in \mathcal{N}_{\ell_0 - i} \tag{8.172}$$

Then it sends the message $y_{ln}^{(i)}$ to the node n, $n \in \mathcal{N}_{\ell_0 - i}$, such that $l, l \in \mathcal{L}_{\ell_0 - i}(n)$. Symbol node n, $n \in \mathcal{N}_{\ell_0 - i}$, calculates the log-likelihood ratios

$$z_{nl}^{(i)} = z_n^{(0)} + \sum_{l' \in \mathcal{L}_{\ell_0 - i}(n)} y_{l'n}^{(i)}, \quad n \in \mathcal{N}_{\ell_0 - i}(l), \quad l \in \mathcal{L}_{\ell_0 - i - 1} \tag{8.173}$$

which will be used during the following step of iterations. This concludes the ith step of iterations.

The probability distributions of the random variables $z_{nl}^{(i)}$ depends on i and v_n but not on n and l. Let $\varphi^{(i)}(z)$ and $\psi^{(i)}(z)$ be the probability density functions of the random variable $z_{nl}^{(i)}$ conditioned on $v_n = 0$ and $v_n = 1$, respectively. We know that $\varphi^{(0)}(z) = \psi^{(0)}(-z)$ and that $\varphi^{(0)}(z) \geq \psi^{(0)}(z)$ if $z \geq 0$ and $\varphi^{(0)}(z) < \psi^{(0)}(z)$ if $z < 0$. Because of symmetry with respect to the two hypotheses, $v_n = 0$ and $v_n = 1$, the analogous equations are valid for $i > 0$, namely $\varphi^{(i)}(z) = \psi^{(i)}(-z)$, $i = 1, 2, \ldots$, and $\varphi^{(i)}(z) \geq \psi^{(i)}(z)$ if $z \geq 0$, $\varphi^{(i)}(z) < \psi_1^{(i)}(z)$, if $z < 0$. Note that the ML hard decision with respect to symbol v_n, $n \in \mathcal{N}_{l_0 - i}$, during the ith step of the iteration process is

$$\widehat{v}_n = \begin{cases} 0 & \text{if } z_n^{(i)} \geq 0 \\ 1 & \text{otherwise} \end{cases} \tag{8.174}$$

The density evolution process consists of consecutive numerical calculations of the probability density functions $\varphi^{(0)}(z) \to \varphi^{(1)}(z) \to \cdots \to \varphi^{(I)}(z)$.

We can ask the same question as in the case of the probability evolution process for the BEC: When do we have to stop the density evolution process? Suppose that in parallel to observing the density evolution process we observe the evolution of the

two functionals connected with the density function $\varphi^{(i)}(z)$. The first one is the error probability $\pi_e^{(i)}$ of hard decision[21] $\widehat{v}_n$ with respect to a symbol v_n, $n \in \mathcal{N}_{l_0-i}$, during the ith step of the iteration process,

$$\pi_e^{(i)} = \int_{-\infty}^{0} \varphi^{(i)}(z)dz = \int_{0}^{\infty} \psi^{(i)}(z)dz \tag{8.175}$$

We can observe the hard decision error probability evolution process $\pi_e^{(0)} \to \pi_e^{(1)} \to \cdots \to \pi_e^{(i)} \to \cdots \to \pi_e^{(I)}$. If we would prove that for a given E_b/N_0 we have $\pi_e^{(I)} \to 0$ when $I \to \infty$, then this E_b/N_0 is an upper bound on the iterative decoding limit η_{id}. Analogously to the BEC case, we may stop the density evolution process when $\pi_e^{(i)}$ became sufficiently small.

The second functional is the Bhattacharyya parameter of the random variable $z_{nl}^{(i)}$,

$$B^{(i)} = \int_{-\infty}^{\infty} \sqrt{\varphi^{(i)}(z)\psi^{(i)}(z)}dz = \int_{-\infty}^{\infty} \sqrt{\varphi^{(i)}(z)\varphi^{(i)}(-z)}dz, \quad i = 0, 1, \ldots \tag{8.176}$$

From (8.169) and (8.176) we conclude that

$$B^{(0)} = e^{-\frac{E_\mathrm{s}}{N_0}} \tag{8.177}$$

where E_s/N_0 is the signal-to-noise ratio per symbol, that is,

$$\frac{E_\mathrm{s}}{N_0} = R\frac{E_\mathrm{b}}{N_0} \tag{8.178}$$

The evolution process of the Bhattacharyya parameter is $B^{(0)} \to B^{(1)} \to \cdots \to B^{(i)} \to \cdots \to B^{(I)}$.

From the definition of the Bhattacharyya parameter it follows that $B^{(i)}$ is an upper bound on the error probability $\pi_e^{(i)}$,

$$\pi_e^{(i)} = P\left(z_n^{(i)} \le 0 \;\middle|\; v_n = 0\right) = \int_{-\infty}^{0} \varphi^{(i)}(z)dz \tag{8.179}$$

$$\le \int_{-\infty}^{0} \sqrt{\frac{\varphi^{(i)}(-z)}{\varphi^{(i)}(z)}}\varphi^{(i)}(z)dz \tag{8.180}$$

$$\le \int_{-\infty}^{\infty} \sqrt{\varphi^{(i)}(z)\varphi^{(i)}(-z)}dz = B^{(i)} \tag{8.181}$$

where we use the inequality $\varphi^{(i)}(-z)/\varphi^{(i)}(z) \ge 1$ for $-\infty < z < 0$. From (8.181) it follows that we may stop the density evolution process not only when $\pi_e^{(i)}$ becomes

[21] Strictly speaking, $\pi_e^{(i)}$ is the probability of a hard decision error made on the basis of a message (8.173) sent by a symbol node to one of its neighboring constraint nodes; that is, it does not include the information sent by this constraint node to the symbol node, but $\pi_e^{(i)}$ can be easily calculated in the process of density evolution. It is an upper bound on the error probability of a hard decision made on the basis of all messages from all connected constraint nodes.

sufficiently small but also when $B^{(i)}$ becomes sufficiently small. The following theorem establishes what is a "sufficiently small" value of the Bhattacharyya parameter.

Theorem 8.20 Consider communication over the binary-input AWGN channel with signal-to-noise ratio per symbol E_s/N_0. Suppose that a regular (N, J, K) LDPC block code with $J > 2$ is used and that we observe the Bhattacharyya parameter evolution process $B^{(0)} \to B^{(1)} \to \cdots \to B^{(i)} \to \cdots \to B^{(I)}$. Suppose that during the i_0th step of the process the Bhattacharyya parameter $B^{(i_0)}$ becomes less than the *breakout value*

$$B_{\text{br}} = e^{\frac{E_s}{N_0(J-2)}}(K-1)^{-(J-1)/(J-2)} \tag{8.182}$$

Then $B^{(I)} \to 0$ when $I \to \infty$.

The proof of the theorem is analogous to the proof of Theorem 8.19 and follows from the inequality

$$B^{(i)} < B^{(0)} \left((K-1)B^{(i-1)}\right)^{J-1} \tag{8.183}$$

similarly to (8.165) and proved[22] in [LTZ05].

Theorem 8.20 establishes the breakout value for the Bhattacharyya parameter $B^{(i)}$. Using this theorem we can establish the breakout value directly for the error probability $\pi_e^{(i)}$. In fact,

$$\begin{aligned} B^{(i)} &= 2\int_0^\infty \sqrt{\varphi^{(i)}(z)\varphi^{(i)}(-z)}dz \\ &\le 2\sqrt{\int_0^\infty \varphi^{(i)}(z)\,dz \int_0^\infty \varphi^{(i)}(-z)\,dz} \\ &= 2\sqrt{\pi_e^{(i)}(1-\pi_e^{(i)})}, \quad i = 0, 1, \ldots \end{aligned} \tag{8.184}$$

where we have used Cauchy's inequality. From (8.184), it follows that it is sufficient to observe the evolution of $\pi_e^{(i)}$ up to the moment when it becomes less than the breakout value π_{br} for the error probability of hard decision $\widehat{v}_n$,

$$\pi_{\text{br}} = \frac{B_{\text{br}}^2}{4} = \frac{e^{-\frac{2E_s}{N_0(J-2)}}}{4(K-1)^{2(J-1)/(J-2)}} \tag{8.185}$$

We shortly summarize our description of the density evolution process for a regular (N, J, K) LDPC code. First we fix the maximum number of iterations I. During each step of the density evolution process we choose a signal-to-noise ratio $E_b/N_0 = E_s/N_0R$. Then we perform the density evolution process $\varphi^{(i)}(z)$ with maximum number of iterations I. In parallel we observe the evolution of the error

[22] If we would replace the $B^{(0)}$ in (8.183) by δ and $B^{(i)}$ by $p^{(i)}$ we obtain the inequality (8.165) for the BEC. It is interesting to note that for the BEC the Bhattacharyya parameter $B^{(0)}$ equals the erasure probability δ. Thus, Theorem 8.20 is valid not only for the AWGN channel but also for the BEC if we would replace e^{-E_s/N_0} by δ.

probability for hard decision $\pi_e^{(i)}$. If during the ith step, $i \leq I$, we have $\pi_e^{(i)} \leq \pi_{\text{br}}$, then we stop the density evolution process and either declare the corresponding E_b/N_0 as the threshold or decrease E_b/N_0 and repeat the density evolution process for a new E_b/N_0. If $\pi_e^{(i)} > \pi_{\text{br}}$ for all $i \leq I$, then we increase E_b/N_0 and repeat the density evolution process for a new signal-to-noise ratio.

Calculating the thresholds for LDPC convolutional codes is more complicated. We shall restrict ourselves to an analysis of the thresholds of regular (m_s, J, K) LDPC convolutional codes from the ensembles $\mathcal{C}(M, J, K, T)$ and $\mathcal{C}(M, J, K)$. (The ensemble $\mathcal{C}(M, J, K) = \mathcal{C}(M, J, K, T)|_{T=\infty}$ was introduced in Section 8.2.)

If we study thresholds for regular (m_s, J, K) LDPC convolutional codes with bi-infinite codewords we will get the same thresholds as those considered above for regular (N, J, K) LDPC block codes. But we can improve these thresholds if we study *terminated* regular (m_s, J, K) LDPC convolutional codes.

The syndrome former of a code from $\mathcal{C}(M, J, K, T)$ is shown in Fig. 8.9. In terms of the syndrome former (8.27) the code has the parameters $m_\text{s} = J - 1 = 2$, $b = M$, $c = 2M$, $J = 3$, and $K = 6$. Generalization to more general LDPC convolutional codes is straightforward. The entries $H_i^\text{T}(t + i)$ of the syndrome former (8.27) are composed from permutation matrices,

$$H_i^\text{T}(t+i) = \begin{pmatrix} P_i^{(1)}(t+i) \\ P_i^{(2)}(t+i) \end{pmatrix}, \quad i = 0, 1, J-1, \quad t = 0, 1, \ldots, T-1 \tag{8.186}$$

where $P_i^{(k)}(t + i)$, $k = 1, 2$, are $M \times M$ permutation matrices. All other entries of the period section of the syndrome former are allzero matrices. By choosing the permutation matrices $P_i^{(k)}(t + i)$, $k = 1, 2$, $i = 0, 1, J - 1$, $t = 0, 1, \ldots, T - 1$, randomly and independently of each other such that all $M!$ values are equiprobable, we obtain the ensemble $\mathcal{C}(M, J, K, T)$ of time-varying LDPC convolutional codes.

To calculate the thresholds of the codes from $\mathcal{C}(M, J, K)$ we consider transmission of terminated code sequences over a channel with binary-input symbols. We assume that we use a code from $\mathcal{C}(M, J, K)$ and that the encoder starts from the zero state and then sends binary code sequences of block length $N = 2ML$,

$$\boldsymbol{v} = \boldsymbol{v}_{[0,L-1]} = \boldsymbol{v}_0 \boldsymbol{v}_1 \ldots \boldsymbol{v}_t \ldots \boldsymbol{v}_{L-1} \tag{8.187}$$

where $\boldsymbol{v}_t = (v_{2Mt}\, v_{2Mt+1} \ldots v_{2Mt+2M-1})$, $t = 0, 1, \ldots, L - 1$. We assume that the first M symbols, $\boldsymbol{v}_t^{(1)} = (v_{2Mt}\, v_{2Mt+1} \ldots v_{2Mt+M-1})$, of subblock $\boldsymbol{v}_t$ are the encoder input symbols and the last M symbols, $\boldsymbol{v}_t^{(2)} = (v_{2Mt+M}\, v_{2Mt+M+1} \ldots v_{2Mt+2M-1})$, are the parity-check symbols. Among the ML encoder input symbols, the $M(L - \tau)$ first symbols are the information symbols and the last $M\tau$ symbols form a zero-tail (ZT) termination (cf. Section 4.1). The symbols of the tail bring the encoder from any state at time $t = L - 1 - \tau$ to the allzero state at time $t = L - 1$.

The termination procedure described in [LSC10] leads to the following:

Theorem 8.21 Almost all codes in $\mathcal{C}(M, J, K)$ can be terminated with a tail of length $\tau = m_\text{s} + 1 = J$ blocks, that is, $2MJ$ bits.

A sketch of the proof is given in [LSC10].

Terminating LDPC convolutional codes is not only a practical method of transmitting finite blocks of data symbols but also a convenient instrument for theoretical performance assessment. By terminating LDPC convolutional codes we reduce the investigation of bi-infinite code sequences to the investigation of LDPC block codes and we can apply analysis methods developed for the latter.

It follows from Theorem 8.21 that in the ensemble $\mathcal{C}(M, J, K)$ almost all resulting terminated codes has rate

$$R = \frac{L-\tau}{2L} = \frac{1}{2}\left(1 - \frac{J}{L}\right) \tag{8.188}$$

Note that the rate loss is negligible for $L \gg J$. Although we consider a particular structure of LDPC convolutional codes, the terminating algorithm can be easily modified to general LDPC convolutional codes.

For the analysis of iterative decoding of a particular code symbol v_n, it is convenient to consider a tree-shaped graph for this symbol, showing how different code symbols contribute to the decoding of this symbol as the iterations proceed. In Fig. 8.18 we show the first three levels of the tree in the case of terminated LDPC convolutional codes from the ensemble $\mathcal{C}(M, J, K)$ when $J = 3$, $K = 6$. The figure illustrates the trees of the clan head v_n at time instant $t = 0$ when $n = 0, 1, \ldots, 2M-1$, at time instant $t = 1$ when $n = 2M, 2M+1, \ldots, 4M-1$, and at time instant $t = 2$ when $n = 4M, 4M+1, \ldots, 6M-1$. Note that for terminated LDPC convolutional codes the tree of the clan head v_n, $n = 2(L-1)M, \ldots, 2LM-1$, is similar to that of v_n, $n = 0, 1, \ldots, 2M-1$, and the tree of v_n, $n = 2(L-2)M, \ldots, 2(L-1)M-1$, is analogous to that of $n = 2M, 2M+1, \ldots, 4M-1$ since the encoder input sequence is a zero state driving sequence.

The difference between the computational tree for terminated LDPC convolutional codes from $\mathcal{C}(M, J, K)$ and the computational tree for bi-infinite regular LDPC convolutional codes from $\mathcal{C}(M, J, K)$ is the following. In the second case from all nodes at odd levels stem $K-1$ edges. In the first case there are nodes at odd levels from which stem fewer than $K-1$ edges, namely, $1, 3, \ldots$, or $K-2$ edges. This leads to irregularities in the Tanner graph of the terminated LDPC convolutional code. As a consequence, the constraint nodes of lower degrees provide better protection for the symbols at the beginning and the end of a block. Besides these irregularities the computational trees are identical. Particularly, we may apply the bound (8.53) for the number of independent iterations saying that $\ell_0 \to \infty$ when $M \to \infty$ since terminating LDPC convolutional codes can only increase ℓ_0.

In our threshold analysis of LDPC convolutional codes from $\mathcal{C}(M, J, K)$ we follow the principles described in the beginning of this section for regular LDPC block codes.

Consider an arbitrary $2\ell_0$-cyclefree code from $\mathcal{C}(M, J, K)$ and the tree of the v_{n_0}-clan. We assume that the decoding of the symbol v_{n_0} starts from level $2\ell_0$. Consider the ith iteration of the decoding procedure, where $1 \le i \le \ell_0$. In contrast to the tree description of a regular LDPC block code, considered in Section 8.1, where each

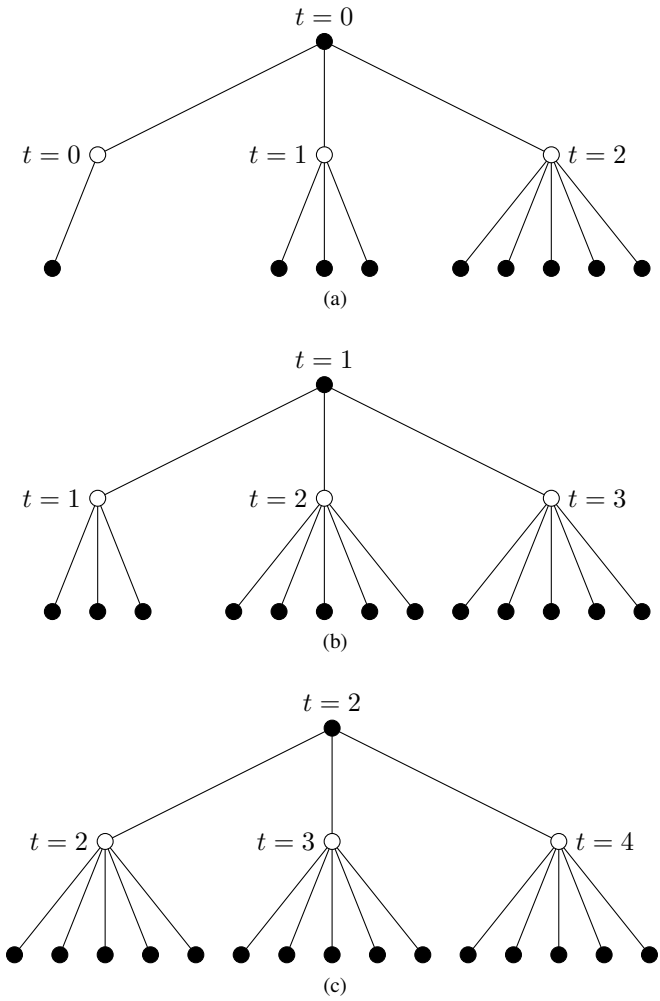


Figure 8.18 The first three levels of the computational trees for $t = 0$, $t = 1$, and $t = 2$ with $J = 3$.

symbol and constraint node was characterized by the level of the computational tree, in our case each node is also characterized by the instant t in the Tanner graph.

For terminated codes from $\mathcal{C}(M, J, K)$ all symbol nodes n can be divided into three groups. The symbol nodes n corresponding to the first $2M$ and the last $2M$ symbols v_n, $n \in \mathcal{G}_0$, $\mathcal{G}_0 = \{0, 1, \ldots, 2M-1,\ \ 2(L-1)M, \ldots, 2LM-1\}$, belong to the first group. The symbol nodes n corresponding to symbols v_n, $n \in \mathcal{G}_1$, $\mathcal{G}_1 = \{2M, 2M+1, \ldots, 4M-1,\ \ 2(L-2)M, \ldots, 2(L-1)M-1\}$, belong to the second group. All other symbol nodes correspond to $n \in \mathcal{G}_2$, $\mathcal{G}_2 = \{4M, 4M+1, \ldots, 2(L-2)M-1\}$, and belong to the third group.

Similarly, all constraint nodes c_ℓ of the Tanner graph for codes in $\mathcal{C}(M, J, K)$ can be divided into three groups. The $2M$ constraint nodes c_ℓ corresponding to the first M and the last M parity-check equations defining the code belong to the first group, that is, $\ell \in \mathcal{H}_0 = \{0, 1, \ldots, M-1, (L-1)M, \ldots, LM-1\}$. Each of these equations includes two symbols. The $2M$ constraint nodes c_ℓ corresponding to parity-check equations that include four symbols belong to the second group, that is, $\ell \in \mathcal{H}_1$, $\mathcal{H}_1 = \{M, M+1, \ldots, 2M-1, (L-2)M, \ldots, (L-1)M-1\}$. The remaining $(L-4)M$ constraint nodes c_ℓ belong to the third group, that is, $\ell \in \mathcal{H}_2$, $\mathcal{H}_2 = \{2M, 2M+1, \ldots, 3M-1, (L-3)M, \ldots, (L-2)M-1\}$. These latter constraint nodes correspond to parity-check equations including six symbols.

Next we analyze the BEC with the erasure probability δ. Let $p^{(i)}_{\{t\to(t+j)\}}$, $t = 0, 1, \ldots, L-1$, $j = 0, 1, \ldots, J-1$, be the probability that during the ith step, $1 \le i \le \ell_0$, of the iterative decoding process a symbol node at time instant t, $t = 0, 1, \ldots, L-1$, located at level $2(\ell_0 - i + 1)$ of a computational tree, sends the erasure symbol to a connected constraint node at time instant $t+j$ at the level $2(\ell_0 - i) + 1$. Let $q^{(i)}_{\{(t+j)\to t\}}$ be the probability that on the ith step, $1 \le i \le \ell_0$, of the iterative decoding process a constraint node at time instant $t+j$, $t = 0, 1, \ldots, L-1$, $j = 0, 1, \ldots, J-1$, located at level $2(\ell_0 - i) + 1$ of a computational tree sends the erasure symbol to a connected symbol node at time instant t located at level $2(\ell - i)$. Analogously to (8.162) we obtain

$$q^{(i)}_{\{(t+j)\to t\}} = 1 - \left((1 - p^{(i-1)}_{\{t\to(t+j)\}}) \prod_{j' \ne j} (1 - p^{(i-1)}_{\{(t+j-j')\to(t+j)\}})^2 \right) \tag{8.189}$$

and

$$p^{(i)}_{\{t\to(t+j)\}} = \delta \prod_{j' \ne j} q^{(i)}_{\{(t+j')\to t\}} \tag{8.190}$$

By definition, the boundary condition is

$$p^{(i)}_{\{t\to t'\}} = 0 \ \text{ if } t < 0 \text{ or } t > L-1 \tag{8.191}$$

where t' is any interger.

For regular LDPC block codes and bi-infinite regular LDPC convolutional codes, the distribution of the messages exchanged during the ith iteration is the same for all nodes regardless of their positions in the Tanner graph (cf. (8.162)). In the terminated LDPC convolutional case, while nodes at the same time instant behave identically, the messages from nodes at different time instants can behave differently and must be tracked separately.

Thus, the probability evolution procedure in our case involves calculating the probabilities $p^{(i)}_{\{t\to(t+j)\}}$ during the ith iteration, $i = 1, 2, \ldots$, for all time instances, $t = 0, 1, \ldots, L-1$, and $j = 0, 1, \ldots, J-1$. It follows from Theorem 8.19 that if $J > 2$, it is sufficient to observe the probability evolution up to iteration I, when the maximum of $p^{(I)}_{\{t\to(t+j)\}}$ over all t and j becomes less than the breakout value $p_{\text{br}} = \delta^{-1/(J-2)}(K-1)^{-(J-1)/(J-2)}$ defined by (8.163).

Table 8.2 BEC thresholds $\delta_{JK}^{(\text{conv})}$ of terminated LDPC convolutional codes for the ensembles $\mathcal{C}(M, J, K)$ with different J and K. For comparison the thresholds $\delta_{JK}^{(\text{block})}$ for the corresponding regular (N, J, K) LDPC block codes are given.

(J, K)	R	$\delta_{JK}^{(\text{conv})}$	$\delta_{JK}^{(\text{block})}$
$(3, 6)$	0.49	0.488	0.429
$(4, 8)$	0.49	0.497	0.383
$(5, 10)$	0.49	0.499	0.341

Analyses of the thresholds of LDPC convolutional codes for the BEC are given in [LSC10] and [KRU11]. In Table 8.2 the BEC thresholds, calculated in [LSC10], for terminated regular LDPC convolutional codes from the ensemble $\mathcal{C}(M, J, K)$ are given for different (J, K). In each case L is chosen so that there is a rate loss of 2%, that is , $R = 0.49$. The first column in Table 8.2 shows the values J and K of the underlying convolutional code, the second column the rate R of the terminated code, the third column the threshold $\delta_{JK}^{(\text{conv})}$ for terminated LDPC convolutional codes, and the fourth column the threshold $\delta_{JK}^{(\text{block})}$ for regular (N, J, K) LDPC block codes with the same J and K. We observe that the thresholds of the terminated convolutional codes are much better than those for the corresponding block codes. The reason is that the constraint nodes for either end of the Tanner graph in the terminated convolutional codes have lower degrees than those for the block codes; that is, the terminated convolutional codes have a structured irregularity.

Terminated convolutional codes with higher J have better thresholds than those with lower J. This interesting behavior is different from that of regular (N, J, K) LDPC block codes, where, for a fixed rate, increasing J typically results in worse thresholds, since it is necessary to increase K accordingly.[23] For regular (N, J, K) LDPC block codes, the loss due to the higher constraint node degrees outweighs the gain resulting from the higher symbol node degrees, adversely affecting performance. However, in the terminated regular LDPC convolutional code case, the codes with higher J still have strong constraint nodes with low degrees at either end of the Tanner graph. Thus the symbols at the ends are better protected for codes with larger symbol degrees, and this results in better thresholds. Furthermore, the thresholds of terminated regular LDPC convolutional codes practically coincide with the thresholds of the corresponding regular LDPC block codes for *maximum a posteriori probability (MAP)* decoding; these thresholds are called *MAP thresholds* (see [LSC10] and [KRU11]). If the density of the parity-check matrices increases, then the MAP threshold of LDPC codes approaches the Shannon limit of the given rate. In other words, by using LDPC convolutional codes we can reach the channel capacity even if the decoding complexity remains nonexponential.

[23] Clearly, symbol nodes with higher degrees are stronger than those with lower degrees, but higher degree constraint nodes are weaker than lower degree constraint nodes.

The threshold analysis of terminated regular LDPC convolutional codes used to communicate over the binary-input AWGN channel follows the same thread as the threshold analysis of terminated regular LDPC convolutional codes for the BEC channel. Namely, the density evolution procedure requires that we observe the probability density functions for nodes at different time instants.

Consider the ith decoding iteration, where $1 \leq i \leq \ell_0$, and a symbol node corresponding to symbol $v_n = 0$ at time instant t, $t = 0, 1, \ldots, L-1$, located at level $2(\ell_0 - i + 1)$ of a computational tree. Let $z_{nl}^{(i)}$ be the message (8.173) that this symbol node sends to a connected constraint node and let $\varphi_{\{t\to(t+j)\}}^{(i)}(z)$, $t = 0, 1, \ldots, L-1$, $j = 0, 1, \ldots, J-1$, be the probability density functions[24] of this message.

Analogously, consider the ith decoding iteration, where $1 \leq i \leq \ell_0$, and a symbol node corresponding to symbol $v_n = 1$ at time instant t, $t = 0, 1, \ldots, L-1$, located at level $2(\ell_0 - i + 1)$ of a computational tree. Let $\psi_{\{t\to(t+j)\}}^{(i)}(z)$, $t = 0, 1, \ldots, L-1$, $j = 0, 1, \ldots, J-1$, be the probability density functions of the message $z_{nl}^{(i)}$ that this symbol node sends to a connected constraint node. It follows from symmetry arguments that $\varphi_{\{t\to(t+j)\}}^{(i)}(z) = \psi_{\{t\to(t+j)\}}^{(i)}(-z)$. The Bhattacharyya parameter $B_{\{t\to t+j\}}^{(i)}$ of these messages, $j = 0, \ldots, J-1$, is equal to

$$B_{\{t\to t+j\}}^{(i)} = \int_{-\infty}^{\infty} \sqrt{\varphi_{\{t\to(t+j)\}}^{(i)}(z)\psi_{\{t\to(t+j)\}}^{(i)}(z)}dz \tag{8.192}$$

$$= \int_{-\infty}^{\infty} \sqrt{\varphi_{\{t\to(t+j)\}^{(i)}}(z)\varphi_{\{t\to(t+j)\}}^{(i)}(-z)}dz \tag{8.193}$$

The Bhattacharyya parameter $B^{(0)}$ of the intrinsic information $z_n^{(0)}$ is given by (8.177).

The density evolution process for terminated regular LDPC convolutional codes from $\mathcal{C}(M, J, K)$ consists of recurrent calculation sets of probability density functions $\{\varphi_{\{t\to(t+j)\}}^{(0)}(z)\} \to \{\varphi_{\{t\to(t+j)\}}^{(1)}(z),\} \to \cdots \to \{\varphi_{\{t\to(t+j)\}}^{(I)}(z),\}$ for all $t = 0, 1, \ldots, L-1$, $j = 0, 1, \ldots, J-1$.

In parallel to the density evolution we observe the Bhattacharyya parameter evolution $\{B_{\{t\to(t+j)\}}^{(0)}(z)\} \to \{B_{\{t\to(t+j)\}}^{(1)}(z)\} \to \cdots \to \{B_{\{t\to(t+j)\}}^{(I)}(z)\}$ for all $t = 0, 1, \ldots, L-1$, $j = 0, 1, \ldots, J-1$. Here $B_{\{t\to(t+j)\}}^{(0)}(z) = B^{(0)}$ is defined by (8.177).

Let $B_{\max}^{(i)}$ denote the largest value of $B_{\{t\to(t+j)\}}^{(i)}$ over all edges in the graph, that is,

$$B_{\max}^{(i)} = \max_{t,j}\{B_{\{t\to(t+j)\}}^{(i)}\}, \quad t = 0, 1, \ldots, L-1, \quad j \in \{0, \ldots, J-1\} \tag{8.194}$$

[24]Note that the probability density function of the messages sent by the symbol v_n to one of its neighboring constraint nodes at time instant $t+j$ depends only on the symbol, its position (time instant) t in the Tanner graph, the time instant increment j, and iteration i.

The inequalities

$$B^{(i)}_{\{t\to(t+j)\}} \le B^{(i)}_{\max} < B^{(0)} \left((K-1)B^{(i-1)}_{\max}\right)^{J-1} \tag{8.195}$$

are a generalization of (8.183).

Suppose now that, after the ith iteration, $i \le \ell_0$, all Bhattacharyya parameters $B^{(i)}_{\{t\to(t+j)\}}$ become smaller than the *breakout value* (8.182),

$$B_{\text{br}} = \left(B^{(0)}\right)^{-1/(J-2)} (K-1)^{-(J-1)/(J-2)} \tag{8.196}$$

Then it follows from (8.195) that if the number of independent iterations $I = \ell_0$ goes to infinity, the bit error probability of all symbols converges to zero.

Instead of observing the evolution of the Bhattacharyya parameter $B^{(i)}_{\{t\to(t+j)\}}$, we can also observe the evolution of the sets of the *probabilities of hard decision error* $\pi^{(i)}_{\{t\to(t+j)\}}$, $i = 0, 1, \ldots, I$, defined analogously; see (8.175). From (8.184), it follows that it is sufficient to observe the evolution of

$$\pi^{(i)}_{\max} = \max_{t,j}\{\pi^{(i)}_{\{t\to(t+j)\}}\}, \;\; t = 0, 1, \ldots, L-1, \;\; j \in \{0, \ldots, J-1\} \tag{8.197}$$

until it becomes less than the breakout value (8.185). This means that we can perform density evolution until the hard decision error probabilities $\pi^{(i)}_{t\to(t+j)}$ reach a sufficiently low level defined by (8.185).

In order to calculate numerically the thresholds of terminated codes from the ensemble $\mathcal{C}(M, J, K)$ it is possible, in principle, to use the standard parallel updating schedule. In this case we must evaluate numerically, iteration by iteration, the sets of probability density functions $\{\varphi^{(i)}_{\{t\to(t+j)\}}(z), \;\; t = 0, 1, \ldots, L-1, \;\; j = 0, 1, \ldots, J-1\}$. Note that, in addition to the node degrees J and K, the termination length L is another parameter that influences the result. While increasing L reduces the rate loss of the terminated code, the computational burden of performing density evolution becomes heavier for larger L.

Both the number of different probability density functions to be tracked and the number of iterations needed for the effect of the strong nodes at the ends of the Tanner graph to propagate to the time instants in the center increase with L. Also, for the AWGN channel, the complexity of the density evolution is much greater than for the BEC, where a simple one-dimensional recursion formula can be used. In [LSC10] a *sliding window updating schedule* was introduced. It reduces the number of operations required for the threshold computation.

Using the parallel updating schedule and sliding window updating schedule described in [LSC10] the thresholds of the terminated regular LDPC convolutional codes from $\mathcal{C}(M, J, 2J)$ for the binary-input AWGN channel were calculated. In Table 8.3, the thresholds $(E_{\text{b}}/N_0)^{(\text{conv})*}_{JK}$ calculated using the parallel updating schedule are given for different J. The values of L were chosen such that there is a rate loss of 2%, that is, $R = 0.49$. The same values can be obtained using the sliding window updating schedule. Also shown in the table are the threshold values $(E_{\text{b}}/N_0)^{(\text{conv})**}_{JK}$,

Table 8.3 Thresholds for the ensembles $\mathcal{C}(M, J, 2J)$ with different J for the binary-input AWGN channel.

(J, K)	R	$(E_\mathrm{b}/N_0)_{JK}^{(\mathrm{conv})*}$	R	$(E_\mathrm{b}/N_0)_{JK}^{(\mathrm{conv})**}$	$(E_\mathrm{b}/N_0)_{JK}^{(\mathrm{block})}$
$(3, 6)$	0.49	0.55 dB	0.5	0.46 dB	1.11 dB
$(4, 8)$	0.49	0.35 dB	0.5	0.26 dB	1.26 dB
$(5, 10)$	0.49	0.30 dB	0.5	0.21 dB	2.05 dB

calculated for the case $L \to \infty$, that is, $R = 1/2$, using the sliding window updating schedule. Finally, the rightmost column of the table shows the thresholds $(E_\mathrm{b}/N_0)_{JK}^{(\mathrm{block})}$ of the regular $(N, J, K)|_{K=2J}$ LDPC block codes with the same J and $K = 2J$.

The results in Table 8.3 for the AWGN channel are similar to the results in Table 8.2 for the BEC. In particular, we observe that the thresholds of regular LDPC convolutional codes are much better than those of the corresponding regular LDPC block codes and that the thresholds improve with increasing J.

Simulation results for the BEC of some randomly chosen terminated LDPC convolutional codes with $J = 3$ ($L = 100$) and $J = 4$ ($L = 200$) are given in Fig. 8.19 and compared to the corresponding regular LDPC block codes. The choice of L leads to an equal rate $R = 0.49$ for both LDPC convolutional codes[25]. The curves in Fig. 8.19 are obtained with randomly selected permutation matrices of size $M = 6000$ under standard belief propagation decoding with the parallel updating schedule. In order to demonstrate that low error rates are achievable close to the calculated thresholds, a maximum number of 5000 iterations were simulated to allow a progression of the reliable messages through all time instants $t = 1, \ldots, L$. Note that, although the overall block length $N = 2LM$ increases with L, for a fixed M the performance improves with decreasing L at the expense of a higher rate loss. On the other hand, after some sufficiently large L, convergence to a steady behavior is expected since the potential strength of a convolutional code in $\mathcal{C}(M, J, K)$ is determined by its overall constraint length $\nu = 2M(m_s + 1)$, which is independent of the termination length L. While the number of required decoding iterations during the parallel updating schedule increases significantly with L, the analysis suggests that a sliding window decoder can provide a good performance complexity trade-off with an iteration number per decoded symbol that is independent of the termination length L.

8.7 BRAIDED BLOCK CODES*

We begin with the definition of a subclass of *braided block codes (BBCs)* called *tightly braided block codes (TBBCs)* and then we consider *sparsely braided block codes (SBBCs)*. There are similarities between the TBBCs and Elias' product codes [Eli54], which are their block counterparts,[26] and similarities between SBBCs and

[25] Both selected codes can be terminated within $\tau = m_s = J - 1$ time instants.

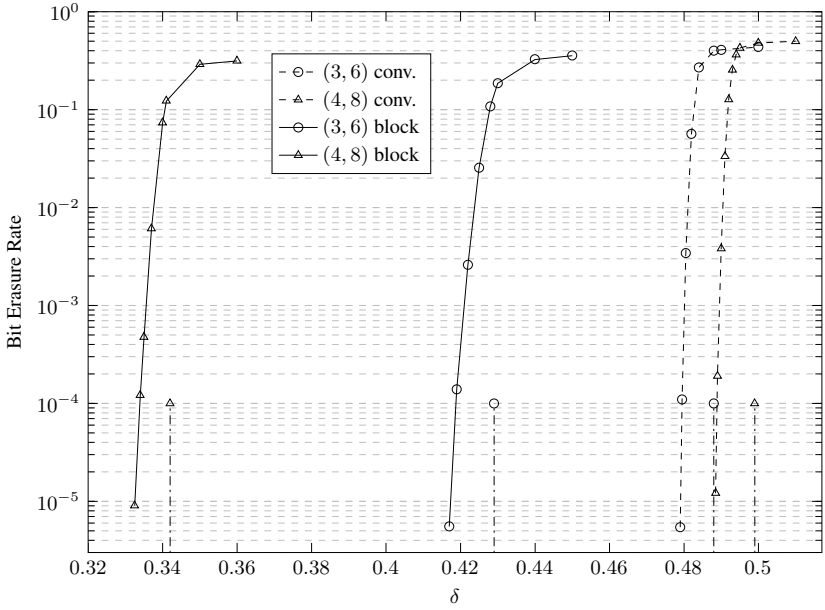


Figure 8.19 Simulation results for the BEC of terminated LDPC convolutional codes from the ensembles $\mathcal{C}(M, 3, 6)|_{M=6000}$ and $\mathcal{C}(M, 4, 8)|_{M=6000}$ (dashed lines) for $L = 100$ and $L = 200$, respectively. Simulation results of regular LDPC block codes composed of permutation matrices of the same size $M = 6000$ are shown for comparison (solid lines). The vertical dashed-dotted lines show the convergence thresholds for the different ensembles.

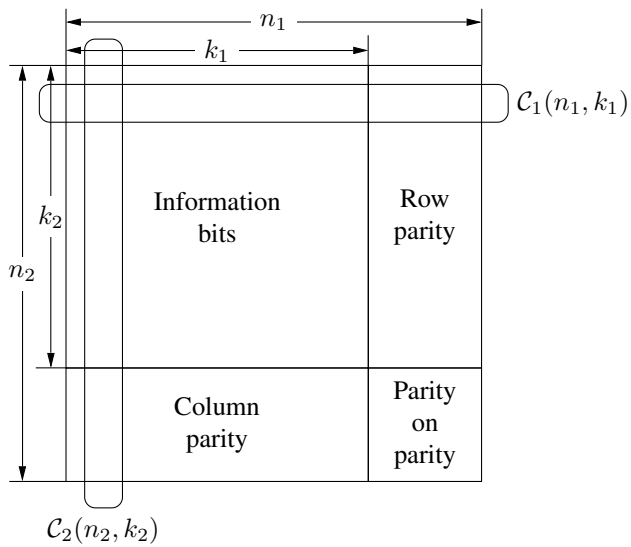


Figure 8.20 An $(n_1 n_2, k_1 k_2)$ product code.

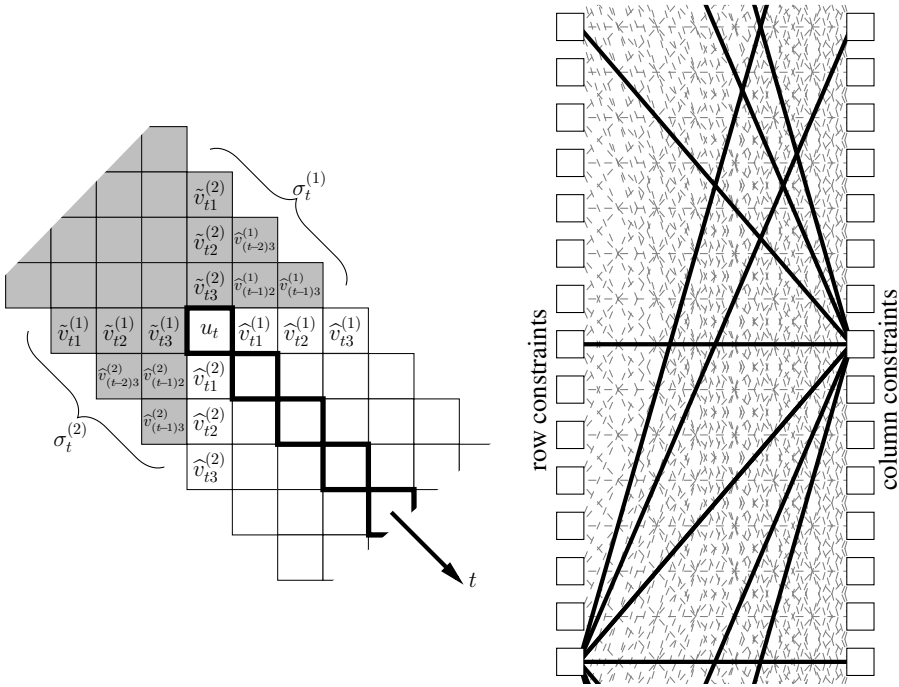


Figure 8.21 Array (a) and graph (b) representations of a tightly braided code with $(7, 4)$ Hamming constituent codes.

GLDPC codes considered in Section 8.1

A conventional product code can be defined by a rectangular $(n_2 \times n_1)$ array that stores the code symbols as illustrated in Fig. 8.20. The information bits are placed in the upper left corner of the array. The symbols stored in each row of the array form a codeword $\boldsymbol{v}^{(1)}$ of the *horizontal* (n_1, k_1) *constituent block code*. The symbols stored in each column of the array form a codeword $\boldsymbol{v}^{(2)}$ of the *vertical* (n_2, k_2) constituent block code.

We consider TBBCs using (μ, κ) Hamming code as constituent codes. The array representation of a TBBC using Hamming $(7, 4)$-codes, $\mu = 7$, $\kappa = 4$, as constituent codes is demonstrated in Fig. 8.21(a). The array is of infinite length and has a width equal to the length $\mu = 7$ of the constituent codes. The area with already encoded bits is shaded gray. There are two constituent codes, a *horizontal* code and a *vertical* code. We shall use double notation of the symbols in the array. The symbol $\tilde{v}_{ti}^{(1)}$ denotes the ith, $i = 1, 2, 3$, input symbol of the horizontal code at the tth time instant

[26]There exist several sliding counterparts of Elias product codes; see, for example, [BaT97, FHB89].

(the fourth input symbol is the information bit u_t), the symbol $\tilde{v}_{ti}^{(2)}$ denotes the ith, $i = 1, 2, 3$, input symbol of the vertical code at the tth time instant, the symbol $\widehat{v}_{ti}^{(1)}$ denotes the ith, $i = 1, 2, 3$, output symbol of the horizontal code at the tth time instant, and the symbol $\widehat{v}_{ti}^{(2)}$ denotes the ith, $i = 1, 2, 3$, output symbol of the vertical code at the tth time instant.

At every time instant t, $t = 0, 1, \ldots$, a new information bit u_t is placed on the main diagonal, indicated by the bold squares, and encoded. The input of the horizontal encoder is $\kappa = 4$ symbols. These are u_t and the three parity bits

$$\tilde{\boldsymbol{v}}_t^{(1)} = \left(\tilde{v}_{t1}^{(1)} \tilde{v}_{t2}^{(1)} \tilde{v}_{t3}^{(1)}\right) \tag{8.198}$$

previously encoded by the vertical encoder and located on the tth row on the left side of the main diagonal. Using these symbols, the horizontal encoder calculates three new parity bits

$$\widehat{\boldsymbol{v}}_t^{(1)} = \left(\widehat{v}_{t1}^{(1)} \widehat{v}_{t2}^{(1)} \widehat{v}_{t3}^{(1)}\right) \tag{8.199}$$

and places them on the tth row on the right side of the main diagonal. Similarly, the vertical encoder calculates the parity bits

$$\widehat{\boldsymbol{v}}_t^{(2)} = \left(\widehat{v}_{t1}^{(2)} \widehat{v}_{t2}^{(2)}, \widehat{v}_{t3}^{(2)}\right) \tag{8.200}$$

using u_t and the three parity bits of the horizontal code

$$\tilde{\boldsymbol{v}}_t^{(2)} = \left(\tilde{v}_{t1}^{(2)} \tilde{v}_{t2}^{(2)} \tilde{v}_{t3}^{(2)}\right) \tag{8.201}$$

previously calculated by the horizontal encoder and located in the tth column straight above the main diagonal. Then the encoder places them on the tth column below the main diagonal.

We describe the encoding process in more detail in the case of *direct encoder realization*. Both the horizontal $(e = 1)$ and vertical $(e = 2)$ constituent codes use the $(7, 4)$ Hamming code with parity-check matrix

$$H_{[1,7]} = \begin{pmatrix} 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{pmatrix} \tag{8.202}$$

Let $H_{[t_1,t_2]}$ denote the matrix composed of the columns $t_1, \ldots, t_2$ of the matrix (8.202), and let $H_{[t_1,t_2]}^{\mathrm{T}}$ be the transposed matrix $H_{[t_1,t_2]}$.

At time t, the encoder e keeps in memory the set of $\sigma_t^{(e)}$ enumerated according to the output of encoder e, where

$$\sigma_t^{(e)} = \{\widehat{v}_{(t-3)3}^{(e)}, \widehat{v}_{(t-2)2}^{(e)}, \widehat{v}_{(t-2)3}^{(e)}, \widehat{v}_{(t-1)1}^{(e)}, \widehat{v}_{(t-1)2}^{(e)}, \widehat{v}_{(t-1)3}^{(e)}\}, \quad e = 1, 2 \tag{8.203}$$

This is the set of parity-check symbols calculated by encoder e before. The set $\sigma_t^{(e)}$ is the *state* of the constituent encoder e at time t. When an information symbol u_t

enters the TBBC encoder, the constituent encoder e, $e = 1, 2$, calculates the parity triple $\widehat{\boldsymbol{v}}_t^{(e)} = (\widehat{v}_{t1}^{(e)} \widehat{v}_{t2}^{(e)} \widehat{v}_{t3}^{(e)})$ using the equation

$$\widehat{\boldsymbol{v}}_t^{(e)} = \tilde{\boldsymbol{v}}_t^{(e)} H_{[1,3]}^{\mathrm{T}} + u_t H_{[4,4]}^{\mathrm{T}}, \quad e = 1, 2 \tag{8.204}$$

where $\tilde{\boldsymbol{v}}_t^{(e)} = \left(\widehat{v}_{(t-3)3}^{(3-e)} \widehat{v}_{(t-2)2}^{(3-e)}, \widehat{v}_{(t-1)1}^{(3-e)} \right)$, $e = 1, 2$. Then, the sets $\sigma_{t+1}^{(e)}$, $e = 1, 2$, are updated as

$$\sigma_{t+1}^{(e)} = \{\widehat{v}_{(t-2)3}^{(e)}, \widehat{v}_{(t-1)2}^{(e)}, \widehat{v}_{(t-1)3}^{(e)}, \widehat{v}_{t1}^{(e)}, \widehat{v}_{t2}^{(e)}, \widehat{v}_{t3}^{(e)}\} \tag{8.205}$$

The output of the TBBC encoder is $\boldsymbol{v} = \boldsymbol{v}_1 \boldsymbol{v}_2 \ldots \boldsymbol{v}_t \ldots$, where $\boldsymbol{v}_t = v_{t1} v_{t2} \ldots v_{t7}$ and

$$v_{ti} = \begin{cases} u_t & i = 1 \\ \widehat{v}_{t(i-1)}^{(1)} & 2 \le i \le 4 \\ \widehat{v}_{t(i-4)}^{(2)} & 5 \le i \le 7 \end{cases} \tag{8.206}$$

Equation (8.204) describes the *direct realization* of the TBBC encoder in Fig. 8.21. It can be generalized to an arbitrary TBBC.

The array in Fig. 8.21(a) can be considered as composed of three ribbons (permutors):

(i) a central ribbon of width 1 (identity convolutional permutor),

(ii) an upper ribbon of width 3 (convolutional permutor with minimum delay $\delta = 1$ and maximum delay $\Delta = 3$),

(iii) a lower ribbon of width 3 (convolutional permutor with minimum delay $\delta = 1$ and maximum delay $\Delta = 3$).

The central ribbon is used for storing the information bits and its function is described by the identity convolutional permutor $\boldsymbol{P}^{(0)}$. The upper ribbon stores the horizontal parity-check symbols and its function is described by a (trivial[27]) multiple convolutional permutor $\boldsymbol{P}^{(1)}$ with width $w = 3$, multiplicity $(3, 3)$, and overall constraint length $\nu = 6$. The lower ribbon stores the vertical parity-check symbols and its function is described by a transposed (trivial) multiple convolutional permutor $(\boldsymbol{P}^{(2)})^{\mathrm{T}}$ of the same width, the same multiplicity, and the same overall constraint length.

In general, the rate of a TBBC is given by

$$R = R^{(1)} + R^{(2)} - 1 \tag{8.207}$$

where $R^{(1)}$ and $R^{(2)}$ are the rates of the horizontal and vertical constituent codes [Tru04]. In our case, the rates of the constituent codes are $R^{(1)} = R^{(2)} = 4/7$ and the rate of the TBBC is $R = 1/7$.

[27]We call the multiple convolutional permutors $\boldsymbol{P}^{(1)}$ and $\boldsymbol{P}^{(2)}$ trivial since all their array cells are storage symbols.

As convolutional codes, TBBCs are characterized by their encoder memory m and overall constraint length ν. In our example, the encoder needs to store 12 previously calculated horizontal and vertical parity-check symbols. Therefore, the overall constraint length of this encoder realization is $\nu = 12$, which is the sum of the overall constraint length of the MCPs. The minimum symbol delay is $\delta = 0$ and the maximum symbol delay $\Delta = 3$. The symbols stay in the encoder up to a maximum of three time instances; therefore the memory of the encoder is $m = 3$.

Along with the array representation of a TBBC, it is interesting to consider the Tanner graph representation. However, it is convenient to use the alternative graph representation of a TBBC given in Fig. 8.21(b). In this graph, the constraints imposed by the horizontal constituent codes are represented by the left nodes; the constraints imposed by the vertical codes are represented by the right nodes. Each edge of the graph corresponds to a code symbol. Therefore, seven edges are coming out of each node and they correspond to the 7 bits of a constituent code. From the array and graph representations of a TBBC in Fig. 8.21 we observe that none of the horizontal codewords contains more than one symbol of a vertical codeword and, vice versa, none of the vertical codewords contains more than one symbol of a horizontal codeword. This property is fulfilled for any BBCs, both tightly and sparsely braided. It guarantees that the girth of the Tanner graph of a code is lower-bounded by 8, not by 4, as it is for conventional LDPC codes.

The memory cells storing symbols in the array do not have to be adjacent and do not have to be close either. They can be spread apart from each other like the symbols in the arrays of the multiple convolutional permutors considered in Section 8.3. Codes described by sparse arrays we call *sparsely braided* block codes. Due to the larger memory and to the absence of short loops in their corresponding graph representation, they have better performance under iterative decoding than TBBC.

The array representation of a SBBC with Hamming $(7, 4)$-constituent codes is found in Fig. 8.22, where dark shaded squares indicate array cells with stored symbols. The array consists of three ribbons. The gray shaded cells indicate the cells keeping a code symbols. The central ribbon of width $w^{(0)} = 1$ is described by the identity permutor $\boldsymbol{P}^{(0)}$ and the upper and lower ribbons are described by the symmetric multiple convolutional permutors $\boldsymbol{P}^{(1)}$ and $(\boldsymbol{P}^{(2)})^{\mathrm{T}}$, respectively. The permutors have the same period $T = 7$, the same multiplicity $(3, 3)$, and the same memory $m = 7$. The overall width of the array is $w = 15$ and the overall constraint length of the SBBCs is $\nu = 27$.

In analogy with the TBBC encoder, the SBBC in Fig. 8.22 places the information symbols u_t on the central ribbon. The triple $\tilde{\boldsymbol{v}}_t^{(1)} = \tilde{v}_{t1}^{(1)} \tilde{v}_{t2}^{(1)} \tilde{v}_{t3}^{(1)}$ is stored in the three leftmost gray shaded cells of the tth row, and $\tilde{\boldsymbol{v}}_t^{(2)} = \tilde{v}_{t1}^{(2)} \tilde{v}_{t2}^{(2)} \tilde{v}_{t3}^{(2)}$ is stored in the three uppermost gray shaded cells of the tth column. The constituent encoders calculate the parity-check symbols $\widehat{\boldsymbol{v}}_t^{(1)}$ and $\widehat{\boldsymbol{v}}_t^{(2)}$ in analogy with the TBBC encoder. The symbols $\widehat{\boldsymbol{v}}_t^{(1)}$ are placed in the three rightmost gray shaded cells of the tth row and the symbols $\widehat{\boldsymbol{v}}_t^{(2)}$ are stored in the three lowermost gray shaded cells of the tth column.

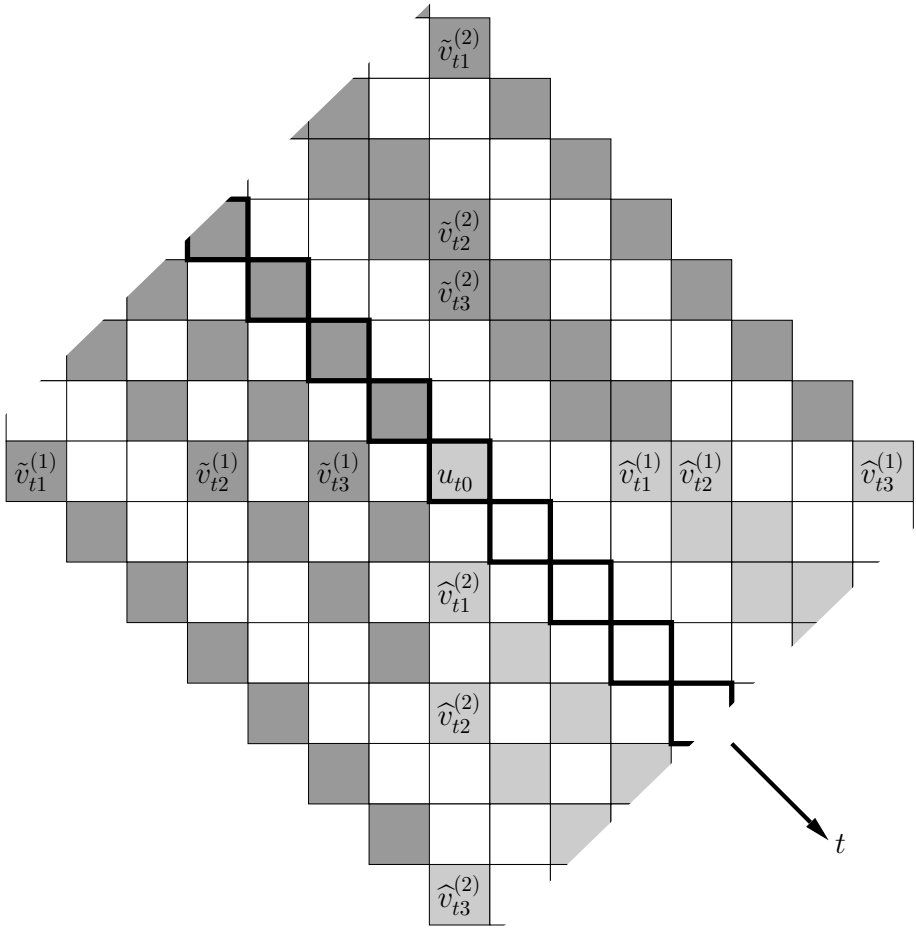


Figure 8.22 Array representation of a sparsely braided code with $(7, 4)$ Hamming constituent codes.

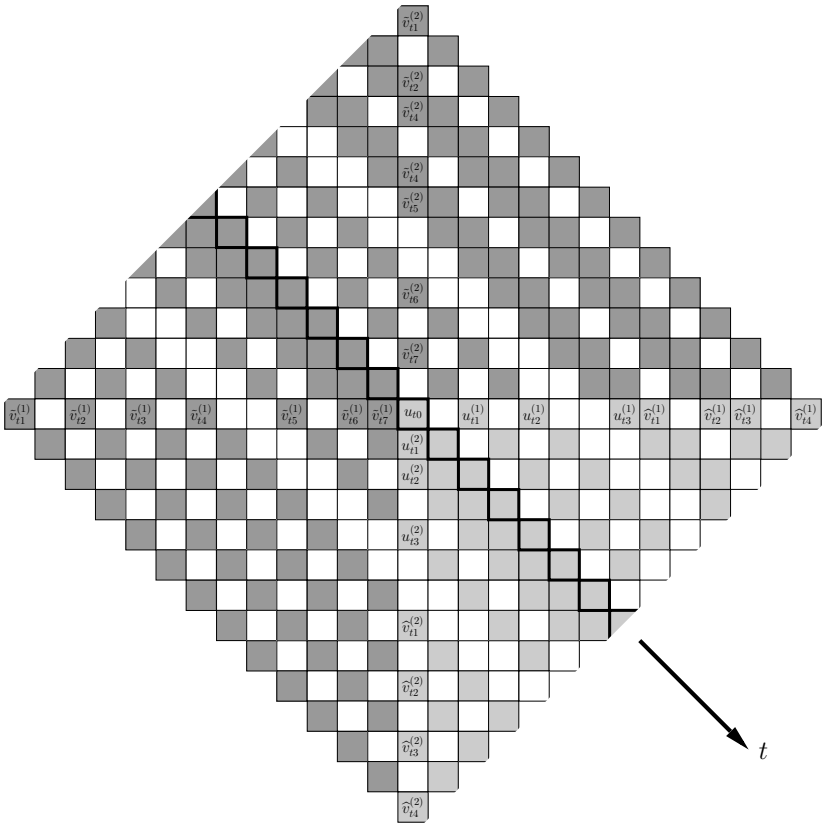


Figure 8.23 Array representation of a sparsely braided code with $(15, 11)$ Hamming constituent codes.

The output of the encoder at time t, $t = 0, 1, \ldots$, is again $\boldsymbol{v}_t = (v_{t1} v_{t2} \ldots v_{t7})$, where

$$v_{ti} = \begin{cases} u_t, & i = 1 \\ \widehat{v}_{t(i-1)}^{(1)}, & 2 \le i \le 4 \\ \widehat{v}_{t(i-4)}^{(2)}, & 5 \le i \le 7 \end{cases} \tag{8.208}$$

Figure 8.23 illustrates another array describing a SBBC with $(15, 11)$ Hamming codes as constituent codes, $\mu = 15$, $\kappa = 11$. The overall code rate R of this SBBC is given by (8.207) and equals $7/15$. Therefore the information block size is now 7 bits. As in Fig. 8.22, the array in Fig. 8.23 consists of three ribbons. The central ribbon has width $w^{(0)} = 1$ and is described by the identity permutor $\boldsymbol{P}^{(0)}$. The upper and lower ribbons have width $w^{(1)} = w^{(2)} = 13$ and are described by the MCPs $\boldsymbol{P}^{(1)}$ and $(\boldsymbol{P}^{(2)})^{\mathrm{T}}$, respectively. These permutors have the same memory $m = 13$, the same multiplicity $(7, 7)$, and the same width $w^{(1)} = w^{(2)} = m$. The encoder memory is

$m = 13$, the overall width of the array describing the SBBC is $2m + 1 = 27$, and the overall encoder constraint length is $\nu = 102$.

Suppose that, at the tth time instance, an information subblock

$$\boldsymbol{u}_t = u_{t0}\boldsymbol{u}_t^{(1)}\boldsymbol{u}_t^{(2)} \tag{8.209}$$

where $\boldsymbol{u}_t^{(e)} = u_{t1}^{(e)}u_{t2}^{(e)}u_{t3}^{(e)}$, enters the eth encoder, $e = 1, 2$. Then, the symbol u_{t0} is placed in the tth position of the central ribbon, the symbols $\boldsymbol{u}_t^{(1)}$ are placed in the three leftmost positions of the tth row of the upper ribbon, and the symbols $\boldsymbol{u}_t^{(2)}$ are placed in the three uppermost positions of the tth column of the lower ribbon. Next, the horizontal encoder reads the seven symbols $\tilde{\boldsymbol{v}}_t^{(1)} = \left(\tilde{v}_{t1}^{(1)}\tilde{v}_{t2}^{(1)}\ldots\tilde{v}_{t7}^{(1)}\right)$, located in the tth row of the lower ribbon, concatenates them with the information symbols $(u_{t0}\boldsymbol{u}_t^{(1)})$, calculates the parity-check bits $\widehat{\boldsymbol{v}}_t^{(1)} = \left(\widehat{v}_{t1}^{(1)}\widehat{v}_{t2}^{(1)}\widehat{v}_{t3}^{(1)}\widehat{v}_{t4}^{(1)}\right)$, and stores them in the four rightmost gray shaded positions of the tth row. Similarly, the vertical encoder reads $\tilde{\boldsymbol{v}}_t^{(2)} = \left(\tilde{v}_{t1}^{(2)}\tilde{v}_{t2}^{(2)}\ldots\tilde{v}_{t7}^{(2)}\right)$, located in the tth column of the upper ribbon, concatenates them with $(u_{t0}\boldsymbol{u}_t^{(2)})$, calculates the parity-check bits $\widehat{\boldsymbol{v}}_t^{(2)} = \left(\widehat{v}_{t1}^{(2)}\widehat{v}_{t2}^{(2)}\widehat{v}_{t3}^{(2)}\widehat{v}_{t4}^{(2)}\right)$, and stores them in the four lowermost gray shaded positions of the tth column. The encoded SBBC sequence is $\boldsymbol{v} = \boldsymbol{v}_1\boldsymbol{v}_2\ldots\boldsymbol{v}_t\ldots$, where $\boldsymbol{v}_t = (v_{t1}v_{t2}\ldots v_{t15})$ and

$$v_{ti} = \begin{cases} u_{t0} & i = 1 \\ u_{t(i-1)}^{(1)} & 2 \le i \le 4 \\ u_{t(i-4)}^{(2)} & 5 \le i \le 7 \\ \widehat{v}_{t(i-7)}^{(1)} & 8 \le i \le 11 \\ \widehat{v}_{t(i-4)}^{(2)} & 12 \le i \le 15 \end{cases} \tag{8.210}$$

We have described symmetric BBCs with $(7, 4)$ and $(15, 11)$ Hamming constituent codes. The generalization to other symmetric BBCs is straightforward and given in [FTL09].

Simulation results of SBBCs after $I = 50$ iterations using the described pipeline decoder taken from [FTL09] are shown in Fig. 8.24. There, the bit error rate P_b for rate $R = 7/15$ codes using $(15, 11)$ Hamming constituent codes with memory $m = 105$ and $m = 4004$ are shown. Their performance has a waterfall effect very close to the threshold for the $R = 7/15$ code, which is equal to 0.86 dB. For comparison, the bit error rates P_b of two $R = 1/2$ GLDPCs based on the $(15, 11)$ Hamming codes with block lengths $N = 3840$ and $N = 30720$ are shown [LeZ99].

The P_b for a rate $R = 16/32$, memory $m = 2244$ asymmetric SBBC using extended $(32, 16)$ and $(16, 11)$ Hamming codes as horizontal and vertical constituent codes is also shown in the middle. Also there, the performance at the threshold for the $R = 16/32$ code, which is equal to 1.13 dB, is given.

The most right curve corresponds to the rate $R = 21/31$, memory $m = 2250$ SBBC using identical $(31, 26)$ Hamming constituent codes. As the previous codes,

the performance has also the waterfall behavior at the threshold for rate $R = 21/31$, that is, at 1.65 dB.

8.8 COMMENTS

LDPC block codes were discovered by Gallager [Gal62, Gal63] in the early 1960s. Gallager's remarkable discovery was mostly ignored by coding researchers for almost 20 years. Important exceptions are two papers by Zyablov and Pinsker [ZyP74, ZyP75], where two low-complexity decoding algorithms for LDPC codes when communicating over the BEC and BSC were suggested, for example, the bit-flipping algorithm. In 1981 Tanner [Tan81] published a new interpretation of LDPC codes from a graphical point of view. Margulis described the first explicit construction of codes on graphs [Mar82]. The Tanner and Margulis papers were also ignored by coding researchers until the 1990s when coding theorists began to investigate codes on graphs and iterative decoding of these codes. MacKay and Neal rediscovered Gallager's LDPC codes [MaN96]. The LDPC codes became strong competitors for error control in many communication and digital storage systems.

Iterative decoding methods were introduced by Elias as early as 1954 [Eli54]. Gallager used an iterative APP decoding algorithm for decoding of low-density parity-check block codes [Gal62, Gal63]. He showed that good performances could be achieved with relatively simple decoding methods.

Jimenez and Zigangirov extended Gallager's concept of LDPC block codes to convolutional codes [JiZ99, Jim06]. Distances of LDPC convolutional codes were studied in [STL07, TZC10]. Iterative decoding threshold analyses for LDPC convolutional codes are given in [LSC10] and [KRU11]. Braided block codes were described and analyzed in [LTZ02, TLZ03], and [FTL09].

Finally, although they are beyond the scope of this book, we would like to mention the important contributions of Wiberg, Loeliger, and Kötter to iterative decoding of codes defined on general graphs [WLK95, Wib96].

PROBLEMS

8.1 Draw a Tanner graph of the code defined by the transposed parity-check matrix (syndrome former)

$$H^{\mathsf{T}} = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 \end{pmatrix}$$

What are the parameters of the LDPC code described by this syndrome former?

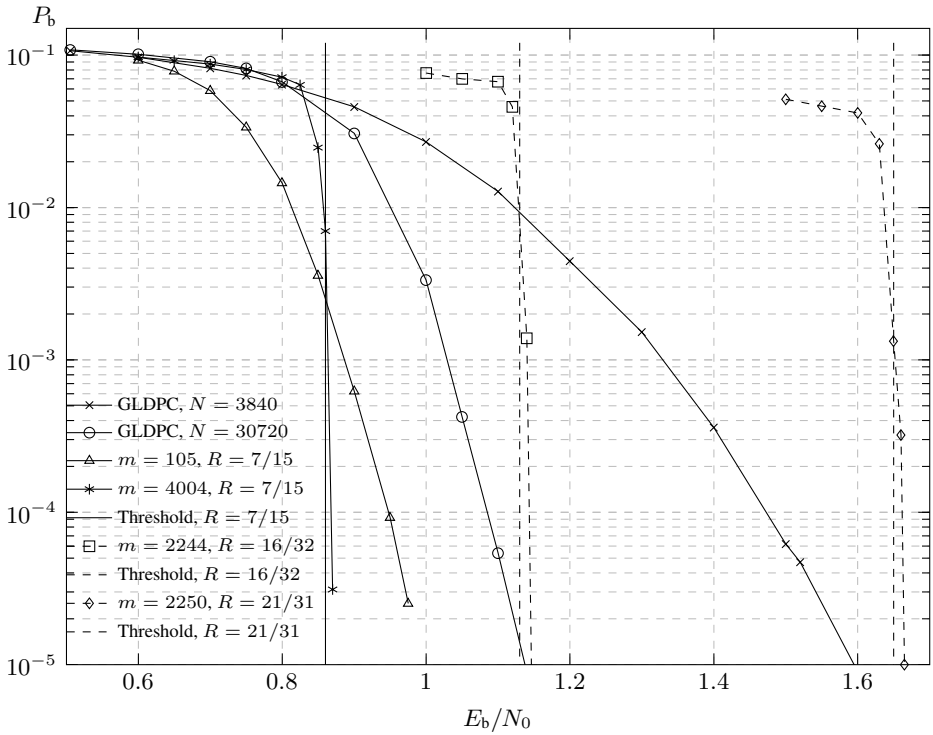


Figure 8.24 Simulation results of SBBCs for the AWGN channel. SBBCs of memory $m = 105$ and rate $R = 7/15$; memory $m = 4004$ and rate $R = 7/15$; memory $m = 2244$ and rate $R = 16/32$; memory $m = 2250$ and rate $R = 21/31$ are considered. For comparison the P_b of two GLDPCs based on $(15, 11)$ Hamming codes are included.

8.2 Draw a Tanner graph of the code from the ensemble $\mathcal{B}_2(12, 2, 3)$ considered in Example 8.2. Show that each constraint node is connected with exactly one symbol node from each of the three groups on which the symbol nodes are divided.

8.3 Consider the code given in Example 8.2 with parameters $N = 12$, $J = 2$, $K = 3$. Construct three codewords having weight $w_{\mathrm{H}} = 2N/K = 8$.

8.4 The transposed parity-check matrix H^{T} given by (8.12) can be written as a combination of the submatrices H_1^{T} and H_2^{T}, that is, $H^{\mathrm{T}} = (H_1^{\mathrm{T}}\ H_2^{\mathrm{T}})$.

a) Show that the submatrix H_1^{T} can be constructed by rowwise interleaving of the three 4×4 matrices

$$P^{(11)} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad P^{(12)} = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix} \quad P^{(13)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

b) Choose matrices $P^{(21)}$, $P^{(22)}$, and $P^{(23)}$ such that their rowwise interleaving gives the submatrix H_2^{T}.

8.5 Suppose that you are constructing a GLDPC code having (μ, κ) Hamming codes as the constituent code from an $(N, 2, K)$ LDPC code. How should the parameters N, K, μ, and κ be related?

8.6 Prove formula (8.23).

8.7 Construct a syndrome former of a $(4, 2, 4)$ LDPC convolutional code from the following transposed parity-check matrix of an LDPC block code:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

8.8 Use the unwrapping method and construct a syndrome former for an $(m_{\mathrm{s}} = 4, J = 3, K = 6)$ LDPC convolutional code from the transposed parity-check matrix

$$\begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \end{pmatrix}$$

of an $(N = 8, J = 2, K = 4)$ LDPC block code.

8.9 Construct a syndrome former of a rate $R = 4/8$ regular periodically time-varying LDPC convolutional code from the ensemble $\mathcal{C}(M = 4, J = 3, K = 6, T = 1)$ using the submatrices

$$P_0^{(1)} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \quad P_1^{(1)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad P_2^{(1)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

$$P_0^{(2)} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \quad P_1^{(2)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \quad P_2^{(2)} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

Transform by permutation of columns and rows the constructed syndrome former to a conventional syndrome former of a rate $R = 1/2$ LDPC convolutional code.

8.10 Combine the identity permutor and the delay permutor with $\delta = 5$ by columnwise interleaving and find the parameters of the combination.

8.11 Construct a multiple block permutor of multiplicity $(J = 3, K = 6)$ from the identity matrix

$$P^{(kj)} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \quad k = 1, \ldots, 6, \quad j = 1, 2, 3$$

using rowwise and columnwise interleaving.

8.12 Combine the identity permutor and the scrambler shown in Fig. 8.12 by columnwise interleaving and find the delay and size of the combination.

8.13 Use rowwise and columnwise interleaving to construct a multiple block permutor of multiplicity $(J = 3, K = 2)$ from the submatrices

$$P^{(11)} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \quad P^{(12)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad P^{(13)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

$$P^{(21)} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \quad P^{(22)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \quad P^{(23)} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

8.14 Formulate a definition of a typical MCP for the nonsymmetric case, $J \neq K$.

8.15 Using the unwrapping procedure, construct a multiple convolutional permutor from the block permutor given in Problem 8.11. Which are the parameters of the permutor?

8.16 Suppose that we have a symmetric MCP, $\boldsymbol{P} = (p_{nl})$, of multiplicity $J = K$ with minimum delay δ, maximum delay Δ, and overall constraint length ν. Transform it using a shift $p_{nl} \to p_{n,l+a}, a \in \mathbb{Z}^+$, on all its elements. Show that we obtain a new MCP having minimum delay $\delta + a$, maximum delay $\Delta + a$, and overall constraint length $\nu + Ja$.

8.17 Show that the belief propagation algorithm for the BEC described in Section 8.5 has a decoding complexity that is not larger than $O(N^2)$.

8.18 Show that the breakout value p_{br} for the BEC for a regular (N, J, K) LDPC block code with $J = 2$ equals $1/(K - 1)$.

8.19 Suppose that the LDPC block code, defined by the transposed parity-check matrix of Problem 8.1, is used together with the BEC and that the received sequence is $\boldsymbol{r} = 1\,\Delta\,0\,\Delta\,0\,\Delta\,0\,1\,1\,0$, where Δ is the erasure symbol. Use the belief propagation algorithm for decoding.

8.20 Repeat Problem 8.19 assuming that the received sequence is $\boldsymbol{r} = \Delta\,\Delta\,1\,\Delta\,\Delta\,0\,0\,1\,1\,0$.

8.21 Formulate a definition of the iterative decoding limit for an LDPC convolutional code for the binary-input AWGN channel.

CHAPTER 9

TURBO CODING

In the previous chapter we have considered a class of iteratively decodable error correcting codes—LDPC codes. In this chapter we consider another important class of iteratively decodable codes—*turbo codes* and some generalizations of these codes, such as *multiple turbo codes* and *braided convolutional codes*.

Turbo coding is based on two fundamental ideas: a design of a code with randomlike properties and a decoder design that exploits easily implementable iterative decoding. The codes have exceptionally good performance, particularly at moderate bit error rates and for large block lengths.

9.1 PARALLEL CONCATENATION OF TWO CONVOLUTIONAL CODES

Turbo codes were invented by Berrou, Glavieux, and Thitimajshima [BGT93, BeG96]. Different variants of turbo codes were suggested later. We begin with a description of the Berrou, Glavieux, and Thitimajshima scheme.

A turbo code can be defined as *parallel concatenation* of two convolutional codes. An example of a turbo encoder is given in Fig. 9.1.

Fundamentals of Convolutional Coding, 2. Edition. By Rolf Johannesson, Kamil Sh. Zigangirov

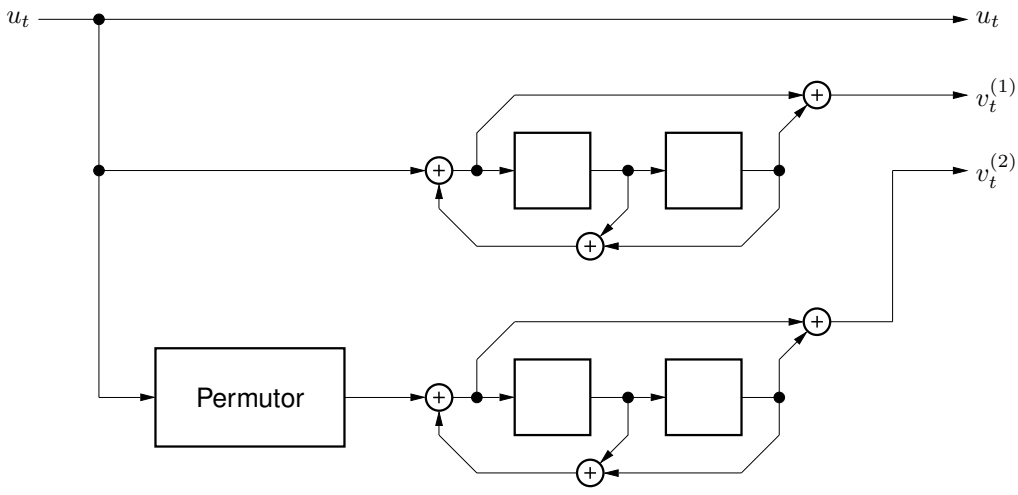


Figure 9.1 A rate $R = 1/3$, systematic, parallel concatenated convolutional encoder.

The encoder consists of two parallel systematic rate $R = 1/2$ memory $m = 2$ convolutional encoders (*constituent encoders*) with feedback (recursive encoders) having the rational generator matrix

$$G(D) = \left(\begin{array}{cc} 1 & \dfrac{1+D^2}{1+D+D^2} \end{array} \right) \tag{9.1}$$

and a single block permutor (SBP). The input of the turbo encoder is the length $\tilde{K}$ binary information sequence

$$\tilde{\boldsymbol{u}} = u_0 u_1 \ldots u_{\tilde{K}-1} \tag{9.2}$$

The input of the first constituent encoder and the permutor associated with the second constituent encoder is length $K = \tilde{K} + m$ sequence $\boldsymbol{u}$,

$$\boldsymbol{u} = u_0 u_1 \ldots u_{K-1} \tag{9.3}$$

where m is the memory of the constituent encoder. The first $\tilde{K}$ symbols of this sequence are symbols of the information sequence $\tilde{\boldsymbol{u}}$; the last m symbols form a tail such that the first constituent encoder returns to the allzero state. Since the encoding is systematic, the input sequence (including the tail)

$$\boldsymbol{v}^{(0)} = v_0^{(0)} v_1^{(0)} \ldots v_{K-1}^{(0)} \stackrel{\text{def}}{=} \boldsymbol{u} \tag{9.4}$$

is included in the codeword.

The first constituent encoder generates the parity sequence

$$\boldsymbol{v}^{(1)} = v_0^{(1)} v_1^{(1)} \ldots v_{K-1}^{(1)} \tag{9.5}$$

Since the tail forces the first constituent encoder to return to the allzero state, we can use the BCJR algorithm for *a posteriori* probability (APP) decoding, as described in Section 4.5, for decoding of the code sequence of the first constituent encoder (see below).

In parallel, the sequence $\boldsymbol{u}$ enters an SBP, defined by a $K \times K$ permutation matrix $P = (p_{ik})$, $i = 0, 1, \ldots, K-1$, $k = 0, 1, \ldots, K-1$. The output SBP sequence $\boldsymbol{u}^{(2)} = \boldsymbol{u}P$ enters the second constituent encoder which generates the parity sequence

$$\boldsymbol{v}^{(2)} = v_0^{(2)} v_1^{(2)} \ldots v_{K-1}^{(2)} \tag{9.6}$$

In general, the second encoder will be nonterminated. In this case we cannot use the BCJR algorithm for decoding, but we can use the one-way algorithm for APP decoding, as described in Section 4.6. This has little effect on the performance for large lengths K and causes only a small degradation of the performance.

We defined SBP used in the second encoder by the permutation matrix P. An alternative definition of the SBP is obtained by introducing the length-K *index permutation vector*

$$\boldsymbol{\pi} = \begin{pmatrix} \pi_0 & \pi_1 & \ldots & \pi_{K-1} \end{pmatrix} \tag{9.7}$$

where $0 \leq \pi_i \leq K-1$, $i = 0, 1, \ldots, K-1$, $\pi_i \neq \pi_{i'}$, if $i \neq i'$ and π_i is such that $p_{i\pi_i} = 1$, that is, $p_{i\pi_i}$ is the only nonzero entry in row i of the permutation matrix P. We also consider the *inverse index permutation vector*

$$\boldsymbol{\pi}^{-1} = \begin{pmatrix} \pi'_0 & \pi'_1 & \ldots & \pi'_{K-1} \end{pmatrix} \tag{9.8}$$

where $0 \leq \pi'_i \leq K-1$, $i = 0, 1, \ldots, K-1$, $\pi'_i \neq \pi'_{i'}$, if $i \neq i'$ and π'_i is such that $p_{\pi'_i i} = 1$, that is, $p_{\pi'_i i}$ is the only nonzero entry in column i of the permutation matrix P.

The combination of the three sequences $\boldsymbol{v}^{(0)}$, $\boldsymbol{v}^{(1)}$, and $\boldsymbol{v}^{(2)}$ gives the final transmitted sequence (codeword)

$$\boldsymbol{v} = \boldsymbol{v}_0 \boldsymbol{v}_1 \ldots \boldsymbol{v}_{K-1} \tag{9.9}$$

where $\boldsymbol{v}_n = v_n^{(0)} v_n^{(1)} v_n^{(2)}$, $n = 0, 1, \ldots, K-1$, such that the overall code has block length $N = 3K$. Since usually $K \gg m$, the code rate is $R \approx 1/3$.

As we pointed out above, the second encoder will not, in general, return to the allzero state. The important exception is the case when both constituent encoders are memory $m = 1$ recursive convolutional encoders with generator matrix

$$G(D) = \begin{pmatrix} 1 & \dfrac{1}{1+D} \end{pmatrix} \tag{9.10}$$

Then the tail consists of one symbol u_{K-1} which equals to 0 if the weight of the information sequence is even and equals to 1 if the weight of this sequence is odd. Such an input sequence brings both memory $m = 1$ constituent convolutional encoders to the allzero state (Problem 9.1). Because of this, the trellis of a constituent code starts and ends in an allzero state and we can use the BCJR algorithm for APP decoding of both constituent codes.

There is also a *tailbiting version* of turbo coding when both constituent encoders are tailbiting encoders and the input sequence consists only of the information sequence (9.2), that is, $K = \tilde{K}$.

We should also mention a variant of a turbo encoder which is described by Richardson and Urbanke [RUr08]. In this case, the input of the first constituent recursive encoder is the length $\tilde{K}$ information sequence $\tilde{\boldsymbol{u}}$ plus a tail of m 0s, where m is the memory of the constituent encoder. We assume that the feedback of the encoder is switched off during the last m encoding steps such that the first constituent encoder will return to the allzero state. The input of the second constituent recursive encoder is the permuted sequence $\tilde{\boldsymbol{u}}\tilde{P}$, where $\tilde{P}$ is a $\tilde{K} \times \tilde{K}$ permutation matrix, plus a tail of m 0s. Again we assume that the feedback of the encoder is switched off during the last m encoding steps and, hence, the second constituent encoder will return to the allzero state. Thus, we can use the BCJR algorithm for APP decoding of both constituent codes independently of their memory. Since we do not need to send allzero tails, the block length $N = 3K - m$; since $K \gg m$, the code rate is $R \approx 1/3$.

We can modify the original turbo codes to broaden the range of their performance. We can puncture the parity sequences to achieve higher code rates. For example, puncturing alternating bits from $\boldsymbol{v}^{(1)}$ and $\boldsymbol{v}^{(2)}$ produces a systematic rate $R = 1/2$ code. Adding constituent codes and permutors can be used to obtain low rate codes. In this case, the codes are called *multiple turbo codes*. We consider such codes in Section 9.3.

9.2 DISTANCE BOUNDS OF TURBO CODES

In this section, we shall analyze the minimum distance $d_{\min}$ of turbo codes. It was shown by Breiling [Bre04] that the minimum distance of a turbo code cannot grow faster than logarithmically with the block length N. In [TLZ01] it was shown that there exists a turbo code whose minimum distance grows logarithmically with N. Therefore, these two bounds show that the maximum attainable minimum distance $d_{\min}$ of turbo codes grows logarithmically with N.

First, we formulate Breiling's result as follows:

Theorem 9.1 The minimum distance of a turbo code is upper-bounded by a logarithmic function of its block length

$$d_{\min} \leq O(\log N) \tag{9.11}$$

The bound (9.11) is valid for conventional turbo codes when the number of constituent encoders equals 2. The proof of the theorem in the general case is quite complicated. Below we shall give a simple proof in the case when both constituent encoders are memory $m = 1$ recursive convolutional encoders with generator matrix (9.10). In this case, the proof is similar to the proof that the minimum distance of regular (N, J, K) LDPC block codes with $J = 2$ grows as $O(\log N)$ when $N \to \infty$

(Theorem 8.3). As we shall see in Section 9.3, increasing the number of constituent encoders improves asymptotically the behavior of the minimum distance.

We consider a turbo code with two memory $m = 1$ constituent encoders with generator matrix (9.10); let $\boldsymbol{u} = u_0 u_1 \dots u_{K-1}$ be the first constituent encoder input sequence consisting of $K - 1$ information symbols and one "tail" symbol. The sequence $\boldsymbol{u}$ has even weight. The permuted sequence $\boldsymbol{u}^{(2)}$, where $\boldsymbol{u}^{(2)} = \boldsymbol{u}P$, is the input sequence of the second constituent encoder. We shall study the minimum distance of turbo codes using a computational tree and shall enumerate the symbol nodes according to their positions in the sequence $\boldsymbol{u}$. This computational tree is similar to the computational tree of the LDPC block codes given in Fig. 8.2 and has the following peculiarities:

(i) the clan head has two families, other clan members have one family;

(ii) in each family there are not more than two children; there are families having one child.

Let the input information symbol u_n, $n = 0, 1, \dots, K - 1$, be a clan head. Thus, it has two families. The first family corresponds to the first constituent code and the second family corresponds to the second constituent code. By definition, the children of the clan head of the first family are neighboring information symbols in the sequence $\boldsymbol{u}$. Similarly, the children of the clan head in the second family are its neighboring information symbols in the sequence $\boldsymbol{u}^{(2)}$.

Consider the first family. If $n \neq 0$ and $n \neq K - 1$, then, by definition, the symbol u_n has in this family two children, u_{n-1} and u_{n+1}. The clan head u_0 has in the first family only one child, u_1. Similarly, the clan head u_{K-1} has in the first family one child, u_{K-2}. The number of children of the clan head u_n in the second family depends on the position of the symbol u_n in the permuted sequence, that is, it depends on the index of the permutation symbol π_n. If $\pi_n \neq 0$, $\pi_n \neq K - 1$, then the symbol has in this family two children, $u_{\pi'_{\pi_n - 1}}$ and $u_{\pi'_{\pi_n + 1}}$, where π'_n is defined in (9.8). If $\pi_n = 0$, or if $\pi_n = K - 1$, then u_n has in the second family one child $u_{\pi'_1}$ or $u_{\pi'_{K-2}}$, respectively.

The nodes of the following levels of the tree are enumerated similarly. Particularly, a child of the clan head u_i, $i \neq n$, $i = 0, 1, \dots, K - 1$, has only one family corresponding to the first or the second constituent code (this is one of the two constituent codes different from the code associated with the family of the clan head u_n to which u_i belongs). If the family corresponds to the first constituent code and $i \neq 0$ and $i \neq K - 1$, then the symbol u_i has in this family two children, u_{i-1} and u_{i+1}. If the family corresponds to the first constituent code and $i = 0$ or $i = K - 1$, then u_i has one child. If the family of u_i corresponds to the second constituent code and $\pi_i \neq 0$, $\pi_i \neq K - 1$, then the symbol u_i has in this family two children, $u_{\pi'_{\pi_i - 1}}$ and $u_{\pi'_{\pi_i + 1}}$. If the family of u_i corresponds to the second constituent code and $\pi_i = 0$ or $\pi_i = K - 1$, then u_i has one child.

According to this construction of the tree, the first generation of the v_n-clan can consist of two, three, or four nodes which are *direct descendants* (children) of the clan head (Problem 9.7).

The second generation of a clan has minimum two descendants. The ℓth, $\ell \geq 2$, generation has minimum $2^{\ell-1}$ descendants. Since the total number of clan members cannot exceed K, eventually different nodes in the tree will represent the same symbol, that is, the graph contains a cycle.

The definitions of a *cycle*, a $2\ell_0$-*cyclefree clan*, and a $2\ell_0$-*cyclefree code* for the computational tree of turbo code coincide with the corresponding definitions for the computational tree of an LDPC code given in Section 8.1. The following lemma is related to Theorem 8.2.

Lemma 9.2 Consider a turbo code of block length N, $N \geq 15$, with two recursive memory $m = 1$ convolutional encoders as constituent encoders. If the turbo code is $2\ell_0$-cyclefree, $\ell_0 \geq 2$, then

$$\ell_0 \leq \frac{\log N}{\log 2} - \frac{\log 3}{\log 2} \tag{9.12}$$

Proof: Consider an arbitrary u_n-clan, $n = 0, 1, \ldots, K-1$, of a length $N = 3K$ turbo code. To the root node (zeroth generation) corresponds a 1 symbol, u_n, in the first and the second generation (on levels 2 and 4) there are not less than two descendants, in the third generation there are not less than four descendants, ..., in the ℓ_0th generation, $\ell_0 \geq 2$, there are not less than 2^{ℓ_0-1} descendants. Correspondingly, the number of clan members up to the ℓ_0th generation is not less than 2^{ℓ_0}. If the clan is $2\ell_0$-cyclefree, this number does not exceed $K = N/3$, that is,

$$\frac{N}{3} \geq 2^{\ell_0} \tag{9.13}$$

symbols. Equation (9.13) is equivalent to (9.12). ∎

From Lemma 9.2 the following theorem follows. It is a partial case of Theorem 9.1.

Theorem 9.3 The minimum distance of a length N turbo code with two memory $m = 1$ recursive convolutional encoders as constituent encoders is upper-bounded by the inequality

$$d_{\min} < 6 \log_2 N \tag{9.14}$$

Proof: Consider again a $2\ell_0$-cyclefree v_n-clan graph and cycles including v_n. (We consider only cycles including the clan head v_n, although, in principle, there exist cycles not including the clan head.) According to the definition of a cyclefree clan, the v_n-clan graph has a cycle of length (in edges) not larger than $4\ell_0 + 4$, where ℓ_0 satisfies (9.12); it includes not more than $2\ell_0 + 2$ symbol nodes.

This cycle includes necessarily the children of the clan head. On each odd level, particularly level $2\ell_0 + 1$, the cycle has two constraint nodes associated with *different* constituent codes. On the last level $2\ell_0 + 2$, it has one symbol node which simultaneously belongs to both families on level $2\ell_0 + 1$.

Now we construct a codeword of low weight. We ascribe to each symbol node of the cycle the symbol 1 and ascribe to all other input symbols of the encoder the

symbol 0. Then the weight of the turbo encoder input sequence does not exceed twice the right-hand side of (9.12) plus 1, that is,

$$2\Big(\frac{\log N}{\log 2} - \frac{\log 3}{\log 2}\Big) + 1 \tag{9.15}$$

The parity sequences $\boldsymbol{v}^{(1)}$ and $\boldsymbol{v}^{(2)}$ consist of length 2 "bursts" of 1s. Since each burst of 1s is created by two neighboring 1s in the input of a constituent encoder, the weights of the parity sequences $\boldsymbol{v}^{(1)}$ and $\boldsymbol{v}^{(2)}$ do not exceed the weight of the input encoder sequence and is upper-bounded by (9.15). So, the total weight of the constructed codeword is upper-bounded by

$$6\Big(\frac{\log N}{\log 2} - \frac{\log 3}{\log 2} + 1\Big) \tag{9.16}$$

Since $\log 3/\log 2 > 1$ we proved inequality (9.14). ■

The following theorem [TLZ01] establishes a lower bound for the minimum distance of turbo code:

Theorem 9.4 There exists a block length N turbo code whose minimum distance is lower-bounded by the inequality

$$d_{\min} \geq \alpha \log N \tag{9.17}$$

where the constant $\alpha > 0$ depends on the type of constituent encoders only.

From Theorems 9.1 and 9.4 it follows that the minimum distance of a turbo code grows not faster than $O(\log N)$, and there exists a turbo code with minimum distance $d_{\min} = O(\log N)$. In the following section we show that the minimum distance of multiple turbo codes grows faster than the minimum distance of conventional turbo codes.

9.3 PARALLEL CONCATENATION OF THREE AND MORE CONVOLUTION CODES

The conventional turbo codes have relatively poor minimum distances, which causes their performance curves to flatten out at high signal-to-noise ratios. Although the minimum distance of a turbo code can be improved by the design of the permutor, it cannot grow faster than logarithmically with the block length, as has been shown in Section 9.2. As we will see, multiple turbo codes (MTCs) with encoders which consist of more than two parallel constituent encoders achieve better minimum distance than the conventional turbo codes.

An encoder of a J-dimensional MTC consists of J, $J \geq 3$, parallel constituent convolutional encoders and J single block permutors (SBPs) of size $K \times K$, as shown in Fig. 9.2. (We define the first SBC $P^{(1)}$ to be the $K \times K$ identity matrix.)

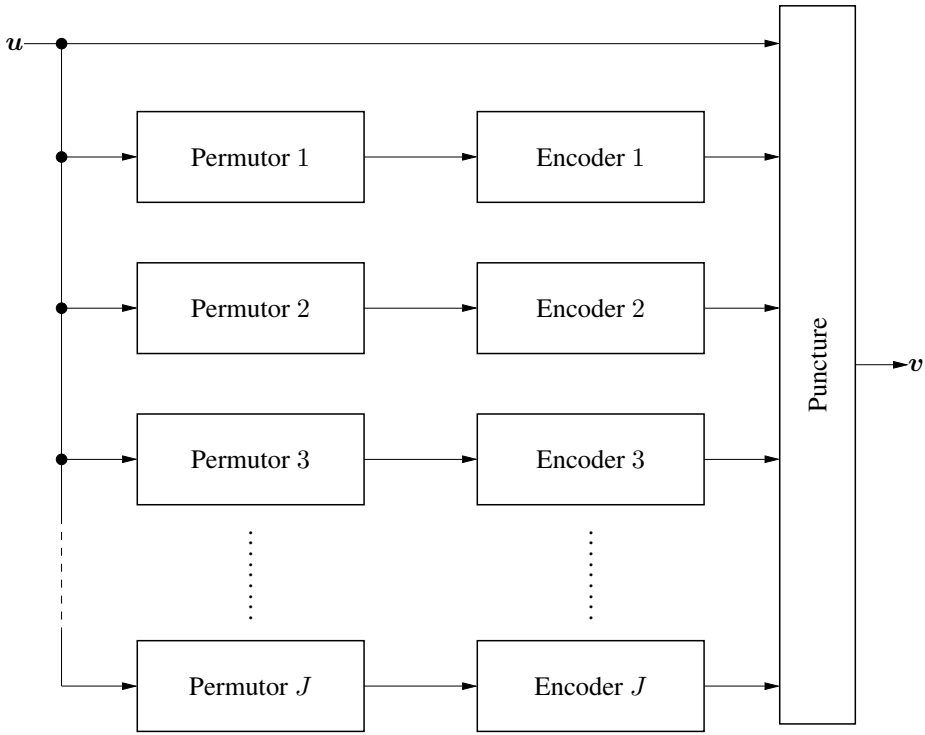


Figure 9.2 Encoder structure of MTCs.

We assume that all constituent encoders are rate $R = 1/2$ memory m recursive convolutional encoders.

A length K binary sequence

$$\boldsymbol{u} = u_0 u_1 \dots u_{K-1} \tag{9.18}$$

is the input to the MTC. It can be an information sequence; in this case all constituent encoders are tailbiting encoders. Similarly to conventional turbo codes, we consider the case when $\boldsymbol{u}$ consists on $\tilde{K}$ information symbols plus a tail from m binary symbols such that $K = \tilde{K} + m$. Particularly, if all constituent encoders are memory 1 recursive convolutional encoders, the tail consists of 1 symbol, as we saw in the previous section.

The sequence $\boldsymbol{u}$ is fed into J SBP described by $K \times K$ permutation matrices $P^{(j)} = (p_{ik}^{(j)})$, $j = 1, 2, \dots, J$, $i, k = 0, 1, \dots, K-1$. The permutors are followed by the constituent encoders such that the jth permuted version of $\boldsymbol{u}$, that is, $\boldsymbol{u}^{(j)} = \boldsymbol{u}P^{(j)}$, $j = 1, 2, \dots, J$, is fed to the jth constituent encoder.

The output sequence of the jth constituent encoder (a parity sequence) is

$$\boldsymbol{v}^{(j)} = v_0^{(j)} v_1^{(j)} \dots v_{K-1}^{(j)}, \quad j = 1, 2, \dots, J \tag{9.19}$$

The input MTC sequence

$$\boldsymbol{u} \stackrel{\text{def}}{=} \boldsymbol{v}^{(0)} = v_0^{(0)} v_1^{(0)} \dots v_{K-1}^{(0)}$$

is multiplexed with the output sequences of the J constituent encoders and sent to the channel. The overall codeword is

$$\boldsymbol{v} = \boldsymbol{v}_0 \boldsymbol{v}_1 \dots \boldsymbol{v}_{K-1} \tag{9.20}$$

where $\boldsymbol{v}_n = v_n^{(0)} v_n^{(1)} \dots v_n^{(J)}$, $n = 0, 1, \dots, K-1$, such that the overall code has block length $N = (J+1)K$ and the resulting rate is $R = 1/(J+1)$. Puncturing can be used to increase the code rate.

The combination of the first identity permutor and the other $J-1$ permutors is called a J-dimensional permutor. It can be represented by a J-dimensional array constructed such that its projection to the plane formed by the 1st and the jth dimensions represents the permutation matrix $P^{(j)}$, $j = 2, 3, \dots, J$.

Similarly to the consideration in Section 9.1, in parallel to the matrix presentation of the permutors we introduce the jth *index permutation vector*

$$\boldsymbol{\pi}^{(j)} = \left(\pi_0^{(j)} \quad \pi_1^{(j)} \quad \dots \quad \pi_{K-1}^{(j)} \right), \quad j = 1, 2, \dots, J \tag{9.21}$$

where $0 \le \pi_i^{(j)} \le K-1$, $i = 0, 1, \dots, K-1$, $\pi_i^{(j)} \ne \pi_{i'}^{(j)}$ if $i \ne i'$ and $\pi_i^{(j)}$ is such that $p_{i\pi_i^{(j)}}^{(j)} = 1$, that is, $p_{i\pi_i^{(j)}}^{(j)}$ is the nonzero entry in row i of the permutation matrix $P^{(j)}$.

The combination of the first identity permutor and the other $J-1$ permutors can be described by the *index permutation matrix*

$$\Pi = \begin{pmatrix} \pi_0^{(1)} & \pi_1^{(1)} & \dots & \pi_{K-1}^{(1)} \\ \pi_0^{(2)} & \pi_1^{(2)} & \dots & \pi_{K-1}^{(2)} \\ \dots & \dots & \dots & \dots \\ \pi_0^{(J)} & \pi_1^{(J)} & \dots & \pi_{K-1}^{(J)} \end{pmatrix} = \begin{pmatrix} 0 & 1 & \dots & K-1 \\ \pi_0^{(2)} & \pi_1^{(2)} & \dots & \pi_{K-1}^{(2)} \\ \dots & \dots & \dots & \dots \\ \pi_0^{(J)} & \pi_1^{(J)} & \dots & \pi_{K-1}^{(J)} \end{pmatrix} \tag{9.22}$$

The MTC is fully characterized by the J constituent codes and the index permutation matrix Π. We can also characterize the MTC by the set of K *position vectors*

$$\boldsymbol{\phi}_i = \begin{pmatrix} \pi_i^{(1)} & \pi_i^{(2)} & \dots & \pi_i^{(J)} \end{pmatrix}, \quad i = 0, 1 \dots, K-1 \tag{9.23}$$

which are the transposed columns of the matrix Π. The set of K position vectors for an index permutation matrix Π is denoted $\boldsymbol{\Phi}_K$, that is,

$$\boldsymbol{\Phi}_K = \{\boldsymbol{\phi}_0, \boldsymbol{\phi}_1, \dots, \boldsymbol{\phi}_{K-1}\} \tag{9.24}$$

Note that the components of the position vectors $\boldsymbol{\phi}_i$ are integer numbers modulo K, and, hence, $\boldsymbol{\phi}_i \in \mathbb{R}_K^J$. The following definition introduces a metric in $\mathbb{R}_K^J$, called an $L1$-*metric*.

Definition The distance $\|\boldsymbol{\phi}_i, \boldsymbol{\phi}_{i'}\|$ between the two vectors $\boldsymbol{\phi}_i, \boldsymbol{\phi}_{i'} \in \mathbb{R}_K^J$ is defined as

$$\|\boldsymbol{\phi}_i, \boldsymbol{\phi}_{i'}\| \stackrel{\text{def}}{=} \sum_{j=1}^{J} |\pi_i^{(j)} - \pi_{i'}^{(j)}| \tag{9.25}$$

Lemma 9.5 The minimum distance of a length $N = (J+1)K$ multiple turbo code with J constituent convolutional codes of memory $m = 1$ is upper-bounded by the inequality

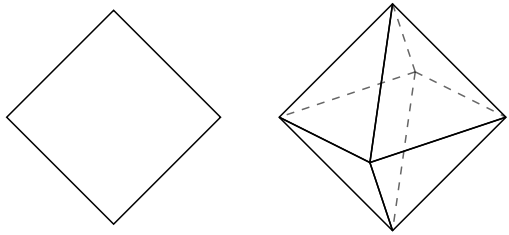
$$d_{\min} \leq 2 + J + \min_{i,i'} \{\|\boldsymbol{\phi}_i, \boldsymbol{\phi}_{i'}\|\} \tag{9.26}$$

Proof: Consider a weight 2 input encoder sequence having 1s in positions i and i', $0 \leq i < i' \leq K-1$. Then the weight of the parity sequence of the first constituent encoder equals $|\pi_i^{(1)} - \pi_{i'}^{(1)}| + 1 = |i - i'| + 1$, the weight of the parity sequence of the second constituent encoder equals $|\pi_i^{(2)} - \pi_{i'}^{(2)}| + 1$, ..., the weight of the parity sequence of the Jth constituent encoder equals $|\pi_i^{(J)} - \pi_{i'}^{(J)}| + 1$. The sum of the weights of the input encoder sequence and the J parity sequences equals

$$2 + J + \|\boldsymbol{\phi}_i, \boldsymbol{\phi}_{i'}\| \tag{9.27}$$

and Lemma 9.5 is proven. ∎

To upper-bound $d_{\min}$ we have to upper-bound $\min_{i,i'} \|\boldsymbol{\phi}_i, \boldsymbol{\phi}_{i'}\|$. We use *sphere packing* arguments based on considering the J-dimensional space with metric $L1$. In the proof of the block sphere-packing bound in Theorem 5.11 we have considered the Hamming metric.



(a) (b)

Figure 9.3 (a) Two-dimensional sphere: square; (b) three-dimensional sphere: octahedron.

Definition For any $\boldsymbol{\phi}_i \in \mathbb{R}_K^J$, the sphere $S_J(\boldsymbol{\phi}_i, r)$ with center $\boldsymbol{\phi}_i$ and radius r, where r is a positive integer, is defined as the set of points $\boldsymbol{\phi}_{i'} \in \mathbb{R}_K^J$ such that

$$\|\boldsymbol{\phi}_i, \boldsymbol{\phi}_{i'}\| \leq r \tag{9.28}$$

where $\|\boldsymbol{\phi}_i, \boldsymbol{\phi}_{i'}\|$ is defined in (9.25).

The number of points $\boldsymbol{\phi}_{i'}$ satisfying (9.28), denoted by $V^{(J)}(r)$, is called the *volume* of the sphere $S_J(\boldsymbol{\phi}_i, r)$.

■ **EXAMPLE 9.1**

Consider the one-dimensional space, $J = 1$. In this case $\mathbb{R}_K^J$ is a set of K nonnegative integer numbers $\{0, 1, \ldots, K-1\}$, that is, $\boldsymbol{\phi}_i = \pi_i^{(1)} = i$. If $r \leq i \leq K-1-r$, then the volume of the sphere $S_1(\boldsymbol{\phi}_i, r)$ is maximal, that is, $V^{(1)}(r) = V_{\max}^{(1)}(r) = 2r+1$. If $i < r$ or $i > K-1-r$, then $2r+1 > V^{(1)}(r) \geq r+1$. The minimal volume $V_{\min}^{(1)}(r) = r+1$ is obtained when $i = 0$ or $i = K-1$.

■ **EXAMPLE 9.2**

In the two-dimensional case, $J = 2$, $\mathbb{R}_K^J$ consists of K^2 points and $\boldsymbol{\phi}_i = (i, \pi_i^{(2)})$, $0 \leq i, \pi_i^{(2)} \leq K-1$. The volume $V^{(2)}(r)$ depends on the position of the center of the sphere and it has minimum $V_{\min}^{(2)}(r) = \sum_{k=0}^{r+1} k = (r+1)(r+2)/2 > r^2/2$ when it is located in vertices of the two-dimensional square, that is, in points $(i = 0, \pi_i^{(2)} = 0)$, or $(i = 0, \pi_i^{(2)} = K-1)$, or $(i = K-1, \pi_i^{(2)} = 0)$, or $(i = K-1, \pi_i^{(2)} = K-1)$. If $r \leq i$ and $\pi_i^{(2)} \leq K-1-r$, then the sphere

$S_2(\boldsymbol{\phi}_i, r)$ is a square (see Fig. 9.3(a)), and it has maximal volume

$$\begin{aligned} V_{\max}^{(2)}(r) &= 2\sum_{k=0}^{r-1} V_{\max}^{(1)}(k) + V_{\max}^{(1)}(r) \\ &= 2\sum_{k=0}^{r-1}(2k+1) + 2r + 1 = 2r^2 + 2r + 1 \end{aligned} \tag{9.29}$$

In general, we have $V_{\max}^{(2)}(r) = 2r^2 + 2r + 1 \geq V^{(2)}(r) \geq (r+1)(r+2)/2 = V_{\min}^{(2)}(r) > r^2/2$.

■ EXAMPLE 9.3

In the three-dimensional case, that is, $J = 3$, $\mathbb{R}_K^J$ consists of K^3 points and $\boldsymbol{\phi}_i = (i, \pi_i^{(2)}, \pi_i^{(3)})$, $0 \leq i$, $\pi_i^{(2)}$ and $\pi_i^{(3)} \leq K-1$. The volume $V^{(3)}(r)$ depends on the position of the center of the sphere and it has minimum volume when it is located in the vertices of the three-dimensional cube. If $r \leq i$, $\pi_i^{(2)}$, $\pi_i^{(3)} \leq K - 1 - r$, then the sphere $S_3(\boldsymbol{\phi}_i, r)$ is an octahedron (see Fig. 9.3(b)) and it has the maximal volume

$$\begin{aligned} V_{\max}^{(3)}(r) &= 2\sum_{k=0}^{r-1} V_{\max}^{(2)}(k) + V_{\max}^{(2)}(r) \\ &= 2\sum_{k=0}^{r-1}(2k^2 + 2k + 1) + 2r^2 + 2r + 1 \\ &= \frac{4}{3}r^3 + 2r^2 + \frac{8}{3}r + 1 \end{aligned} \tag{9.30}$$

The minimal volume is

$$\begin{aligned} V_{\min}^{(3)}(r) &= \sum_{k=0}^{r} V_{\min}^{(2)}(k) \\ &= \sum_{k=0}^{r} \frac{(k+1)(k+2)}{2} \\ &= \frac{1}{6}r^3 + r^2 + \frac{17}{6}r > \frac{1}{6}r^3 \end{aligned} \tag{9.31}$$

In general, a J-dimensional sphere can be viewed as a concatenation of $(J-1)$-dimensional spheres, and the maximal and minimal volumes of $S_J(\boldsymbol{\phi}_i, r)$ can be found by using the recursion

$$V_{\max}^{(J)}(r) = 2\sum_{k=0}^{r-1} V_{\max}^{(J-1)}(k) + V_{\max}^{(J-1)}(r) \tag{9.32}$$

and

$$V_{\min}^{(J)}(r) = \sum_{k=0}^{r-1} V_{\min}^{(J-1)}(k) \tag{9.33}$$

It follows by induction from (9.33) that in the general case we have

$$V_{\min}^{(J)}(r) > \frac{r^J}{J!} = O(r^J) \tag{9.34}$$

Now we are ready to prove the sphere-packing bound for the minimum distance of a multiple turbo code with J constituent convolutional codes of memory $m = 1$.

Theorem 9.6 The minimum distance of a length $N = (J+1)K$, $J \geq 2$, multiple turbo code with J constituent encoders of memory $m = 1$ is upper-bounded by the inequality

$$d_{\min} < O(N^{\frac{J-1}{J}}) \tag{9.35}$$

Proof: Consider the set of codewords which are encoded by the input sequences $\boldsymbol{u}$ of weight 2. The summarized weight of the input sequence $\boldsymbol{u}$ and the J parity-check sequences obtained from the sequence $\boldsymbol{u}$, having 1s in positions i and i', is given by formula (9.27). Then from Lemma 9.5 it follows that the minimum distance of the code is upper-bounded by the inequality $d_{\min} \leq 2 + J + 2r + 1$, where $2r + 1$ is the minimal distance between the two vectors $\boldsymbol{\phi}_i, \boldsymbol{\phi}_{i'} \in \mathbb{R}_K^J$. To optimize this bound we have to place $K = N/(J+1)$ spheres of radius r in $\mathbb{R}_K^J$ such that no two spheres intersect. This requires that

$$KV_{\min}^{(J)}(r) \leq K^J \tag{9.36}$$

where K^J is the number of points in $\mathbb{R}_K^J$.

From (9.34) and (9.36) it follows that

$$\frac{r^J}{J!} < K^{J-1} \tag{9.37}$$

or

$$r < (J!)^{\frac{1}{J}} K^{\frac{J-1}{J}} \tag{9.38}$$

From (9.22) and (9.34) it follows that

$$d_{\min} < 2 + J + 2(J!)^{\frac{1}{J}} K^{\frac{J-1}{J}} + 1 = O(N^{\frac{J-1}{J}}) \tag{9.39}$$

and the proof is complete. ■

■ **EXAMPLE 9.4**

In the two-dimensional case, $J = 2$, it follows from (9.39) that

$$d_{\min} < O(\sqrt{N}) \tag{9.40}$$

This bound is weaker than the bound (9.14) on the minimum distance of the conventional turbo codes with two constituent convolutional codes of memory $m = 1$. In the three-dimensional case, $J = 3$, we have from (9.39) that

$$d_{\min} < O(N^{\frac{2}{3}}) \tag{9.41}$$

Next we derive a lower (existence) bound on $d_{\min}$ of the multiple turbo codes.

Theorem 9.7 There exists a block length $N = (J + 1)K$ multiple turbo code with J constituent codes, $J \geq 3$, of memory 1 with minimum distance lower-bounded by the inequality

$$d_{\min} > O(N^{\frac{J-2}{J}-\epsilon}) \tag{9.42}$$

where $\epsilon > 0$ is arbitrarily small.

Proof: Consider the ensemble of turbo codes having randomly chosen $K \times K$ single block permutors $P^{(j)}$, $j = 1, 2, \ldots, J$, and J memory 1 constituent encoders[28]. Let the even number $2w$, $w \geq 1$, be the Hamming weight of the input encoder sequence $\boldsymbol{u} \stackrel{\text{def}}{=} \boldsymbol{v}^{(0)}$ and choose $\widehat{d} \stackrel{\text{def}}{=} \lfloor K^{\frac{J-2}{J}-\epsilon} \rfloor = O(K^{\frac{J-2}{J}-\epsilon}) = O(N^{\frac{J-2}{J}-\epsilon})$. We consider only the case $2w \leq \widehat{d}$ since if $2w > \widehat{d}$, then $d_{\min} > \widehat{d} = O(N^{\frac{J-2}{J}-\epsilon})$.

The number of length K sequences $\boldsymbol{v}^{(0)}$ having weight $2w$ equals $\binom{K}{2w}$. Let $M_w^{(j)}(d)$, $j = 1, 2, \ldots, J$, be the number of parity-check sequences $\boldsymbol{v}^{(j)}$ of weight d caused by a sequence $\boldsymbol{v}^{(0)}$ of weight $2w$. The sequence $\boldsymbol{v}^{(j)}$, $j = 1, 2, \ldots, J$, can be presented as the set of w "bursts" of 1s separated by "bursts" of 0s. Since a positive integer d can be represented as the sum of w positive integers by $\binom{d-1}{w-1}$ ways and a set of $K - d$ 0s can be represented as a set of $w + 1$ "bursts" of 0s by $\binom{K-d}{w}$ ways, we have

$$M_w^{(j)}(d) = \binom{d-1}{w-1}\binom{K-d}{w} \tag{9.43}$$

The probability $P_w^{(j)}(d)$ that a weight $2w$ input sequence of the jth constituent encoder causes parity-check sequence $\boldsymbol{v}^{(j)}$ of weight d is

$$P_w^{(j)}(d) = \frac{\binom{d-1}{w-1}\binom{K-d}{w}}{\binom{K}{2w}} \tag{9.44}$$

The probability $P_w^{(j)}(d \leq \widehat{d})$ that a weight $2w$ input sequence of the jth constituent encoder causes the parity-check sequence $\boldsymbol{v}^{(j)}$ of weight $d \leq \widehat{d}$ is

$$P_w^{(j)}(d \leq \widehat{d}) = \sum_{d=w}^{\lfloor \widehat{d}/2 \rfloor} \frac{\binom{d-1}{w-1}\binom{K-d}{w}}{\binom{K}{2w}} \tag{9.45}$$

[28]To simplify the proof we consider the case when the first permutor is not necessarily the identity permutor. We can do it since the symbols of input encoder sequence can have any order.

The probability $P_w(d \le \widehat{d})$ that a weight $2w$ input sequence of a turbo code with J constituent encoders causes parity-check sequences $\boldsymbol{v}^{(j)}, j = 1, 2, \ldots, J$, such that the weight of each of them does not exceed $\widehat{d}$ is

$$P_w(d \le \widehat{d}) = \Big(\sum_{d=w}^{\lfloor \widehat{d}/2 \rfloor} \frac{\binom{d-1}{w-1}\binom{K-d}{w}}{\binom{K}{2w}} \Big)^J \tag{9.46}$$

There are $\binom{K}{2w}$ input sequences $\boldsymbol{v}^{(0)}$ of weight $2w$. Then in the ensemble of turbo codes, the mathematical expectation $M(\widehat{d})$ of the number of codes with $J+1$ code subsequences $\boldsymbol{v}^{(j)}$, $j = 0, 1, \ldots, J$, having weights not exceeding $\widehat{d}$ is

$$M(\widehat{d}) = \sum_{w=1}^{\lfloor \widehat{d}/2 \rfloor} \binom{K}{2w} \Big(\sum_{d=w}^{\widehat{d}/2} \frac{\binom{d-1}{w-1}\binom{K-d}{w}}{\binom{K}{2w}} \Big)^J \tag{9.47}$$

In the ensemble of turbo codes, the mathematical expectation $M(\widehat{d})$ is an upper bound for the mathematical expectation of codes with minimum distance not exceeding $\widehat{d}$. Then we have for $\widehat{d} = \lfloor K^{\frac{J-2}{J}-\epsilon} \rfloor$

$$\begin{aligned} M(\widehat{d}) &< \sum_{w=1}^{\lfloor \widehat{d}/2 \rfloor} \binom{K}{2w}^{1-J} \Big(\widehat{d} \binom{\widehat{d}-1}{w-1} \binom{K-\widehat{d}}{w} \Big)^J \\ &= \sum_{w=1}^{\lfloor \widehat{d}/2 \rfloor} \binom{K}{2w}^{1-J} \Big(w \binom{\widehat{d}}{w} \binom{K-\widehat{d}}{w} \Big)^J \\ &< \sum_{w=1}^{\lfloor \widehat{d}/2 \rfloor} w^J \binom{2w}{w}^J \Big(\frac{K!}{(K-2w)!} \Big)^{1-J} \times \\ &\quad \Big(\frac{\widehat{d}!}{(\widehat{d}-w)!} \Big)^J \Big(\frac{(K-\widehat{d})!}{(K-\widehat{d}-w)!} \Big)^J \end{aligned} \tag{9.48}$$

Using the following formulas valid for $K \to \infty$

$$\binom{2w}{w} = O\Big(2^{2w}\Big) \tag{9.49}$$

$$\frac{K!}{(K-2w)!} = O\Big(K^{2w}\Big) \tag{9.50}$$

$$\frac{\widehat{d}!}{(\widehat{d}-w)!} = O\Big(\widehat{d}^w\Big) \tag{9.51}$$

$$\frac{(K-\widehat{d})!}{(K-\widehat{d}-w)!} = O\Big(K-\widehat{d}\Big)^w \tag{9.52}$$

we obtain from (9.48)

$$M(\widehat{d}) < O\Big(K^{-\epsilon J}\Big) \tag{9.53}$$

From (9.53) follows that in the ensemble of the turbo codes there exists a code with minimum distance satisfying (9.42). ∎

9.4 ITERATIVE DECODING OF TURBO CODES

In this section, we first describe iterative decoding of the turbo code introduced in Section 9.1 and then modify the algorithm for iterative decoding of MTC introduced in Section 9.3.

Consider decoding of turbo codes. We assume that the code sequence $\boldsymbol{v} = \boldsymbol{v}_0\boldsymbol{v}_1 \dots \boldsymbol{v}_{K-1}$ given by (9.9) and constructed from three sequences, $\boldsymbol{v}^{(0)} \stackrel{\text{def}}{=} \boldsymbol{u}$, $\boldsymbol{v}^{(1)}$, and $\boldsymbol{v}^{(2)}$ is transmitted over a discrete memoryless channel. Let

$$\boldsymbol{r} = \boldsymbol{r}_0\, \boldsymbol{r}_1 \dots \boldsymbol{r}_{K-1} \tag{9.54}$$

where $\boldsymbol{r}_n = r_n^{(0)} r_n^{(1)} r_n^{(2)}$, $n = 0, 1, \dots, K-1$, denote the received sequence and let

$$\pi_n^{(0)}(0) \stackrel{\text{def}}{=} P(u_n = 0) \tag{9.55}$$

denote the *a priori* probability that the information symbol $u_n = 0, n = 0, 1, \dots, K-1$. In practice, we often have $P(u_n = 0) = 1/2$.

For decoding a turbo code an iterative decoding algorithm similar to the belief propagation decoding algorithm of LDPC codes is used. The decoder consists of two constituent *a posteriori* probability (APP) decoders. Such decoders were described in Chapter 4. The iterative decoding process consists of I iteration steps; each iteration is executed in two phases. The result of the eth phase, $e = 1, 2$, of the ith iteration, $i = 1, 2, \dots, I$, is the sequence

$$\boldsymbol{\pi}^{(e)}(i) = \pi_0^{(e)}(i)\, \pi_1^{(e)}(i) \dots \pi_{K-1}^{(e)}(i) \tag{9.56}$$

where $\pi_n^{(e)}(i)$ is the *a posteriori* probability[29] that information symbol $u_n = 0$, $n = 0, 1, \dots, K-1$. It is convenient instead of $\pi_n^{(e)}(i)$ to operate with the corresponding log-likelihood ratio, that is,

$$z_n^{(e)}(i) \stackrel{\text{def}}{=} \log \frac{\pi_n^{(e)}(i)}{1 - \pi_n^{(e)}(i)} \tag{9.57}$$

and instead of $\boldsymbol{\pi}^{(e)}(i)$ to operate with the sequence

$$\boldsymbol{z}^{(e)}(i) = z_0^{(e)}(i)\, z_1^{(e)}(i) \dots z_{K-1}^{(e)}(i) \tag{9.58}$$

We will also use the notation

$$z_n^{(0)}(0) \stackrel{\text{def}}{=} \log \frac{P\left(u_n = 0 \,\middle|\, r_n^{(0)}\right)}{P\left(u_n = 1 \,\middle|\, r_n^{(0)}\right)} = \log \frac{\pi_n(0)}{1 - \pi_n(0)} + \log \frac{P\left(r_n^{(0)} \,\middle|\, u_n = 0\right)}{P\left(r_n^{(0)} \,\middle|\, u_n = 1\right)} \tag{9.59}$$

[29] Actually, the decoder operates only with approximated values of the *a posteriori* probabilities since we do not have independence between the data used in the different iterations.

for the log-likelihood ratio of the *a posteriori* probability for the symbol u_n, given the received symbol r_n. It is called the *intrinsic information* about the information symbol $u_n = v_n^{(0)}$. We assume that both constituent decoders keep the sequence $\boldsymbol{z}^{(0)}(0) = z_0^{(0)}(0)\, z_1^{(0)}(0) \ldots z_{K-1}^{(0)}(0)$ in the memory and use it on each step of the iterative decoding process.

Let $\boldsymbol{r}^{(0)}, \boldsymbol{r}^{(1)}$, and $\boldsymbol{r}^{(2)}$ denote the sequences of the received symbols corresponding to the information sequence $\boldsymbol{v}^{(0)} \stackrel{\text{def}}{=} \boldsymbol{u}$, the first parity sequence $\boldsymbol{v}^{(1)}$, and the second parity sequence $\boldsymbol{v}^{(2)}$, respectively. Then we let $\boldsymbol{r}_{\not n}^{(0)}$ denote the sequence of the received symbols corresponding to the sequence $\boldsymbol{u}$ *except* the received symbol $r_n^{(0)}$; that is,

$$\boldsymbol{r}_{\not n}^{(0)} = r_0^{(0)} r_1^{(0)} \ldots r_{n-1}^{(0)} r_{n+1}^{(0)} \ldots \tag{9.60}$$

Similarly, we let the sequence $\boldsymbol{\pi}_{\not n}^{(e)}(i)$ denote the sequence of the *a posteriori* probabilities of the symbols of the sequence $\boldsymbol{u}$ *except* $\pi_n^{(e)}(i)$ after the eth phase of the ith iteration; that is,

$$\boldsymbol{\pi}_{\not n}^{(e)}(i) = \pi_0^{(e)}(i)\pi_1^{(e)}(i) \ldots \pi_{n-1}^{(e)}(i)\pi_{n+1}^{(e)}(i) \ldots \tag{9.61}$$

On the first phase of the first iteration the first constituent encoder is using $\boldsymbol{r}_{\not n}^{(0)}$, $\boldsymbol{r}^{(1)}$, and

$$\boldsymbol{\pi}_{\not n}^{(0)}(0) = \pi_0^{(0)}(0)\pi_1^{(0)}(0) \ldots \pi_{n-1}^{(0)}(0)\pi_{n+1}^{(0)}(0) \ldots \tag{9.62}$$

to calculate the log-likelihood ratios

$$y_n^{(1)}(1) \stackrel{\text{def}}{=} \log \frac{P\left(\boldsymbol{r}_{\not n}^{(0)}, \boldsymbol{r}^{(1)} \;\middle|\; u_n = 0\right)}{P\left(\boldsymbol{r}_{\not n}^{(0)}, \boldsymbol{r}^{(1)} \;\middle|\; u_n = 1\right)} \tag{9.63}$$

for the information symbols $u_n = v_n^{(0)}$, $n = 0, 1, \ldots, K-1$.

For calculation of $y_n^{(1)}(1)$ the decoder uses the BCJR algorithm of APP decoding described in Section 4.5. The variable $y_n^{(1)}(1)$ is called the *extrinsic information* of the information symbol u_n on the first iteration for the first constituent decoder. Then the first constituent decoder calculates the sequence

$$\boldsymbol{z}^{(1)}(1) = z_0^{(1)}(1) z_1^{(1)}(1) \ldots z_{K-1}^{(1)}(1) \tag{9.64}$$

using the formula

$$z_n^{(1)}(1) = z_n^{(0)}(0) + y_n^{(1)}(1) \tag{9.65}$$

and sends it to the second constituent decoder.

On the second phase of the first iteration, the second constituent decoder, knowing sequence $\boldsymbol{z}_n^{(1)}(1)$ and using formula (9.57), can calculate the *a posteriori* probabilities $\pi_n^{(1)}(1)$, $n = 0, 1, \ldots, K-1$, and, hence, $\boldsymbol{\pi}_{\not n}^{(1)}(1)$. Using $\boldsymbol{r}_{\not n}^{(0)}$, $\boldsymbol{r}^{(2)}$, and $\boldsymbol{\pi}_{\not n}^{(1)}(1)$,

the decoder, similarly to (9.63), calculates the extrinsic information $y_n^{(2)}(1)$:

$$y_n^{(2)}(1) \stackrel{\text{def}}{=} \log \frac{P\left(\boldsymbol{r}_{\not\prec}^{(0)}, \boldsymbol{r}^{(2)} \;\middle|\; u_n = 0\right)}{P\left(\boldsymbol{r}_{\not\prec}^{(0)}, \boldsymbol{r}^{(2)} \;\middle|\; u_n = 1\right)} \tag{9.66}$$

The difference between (9.66) and (9.63) is that the decoder is used together with $\boldsymbol{r}_{\not\prec}^{(0)}$, not $\boldsymbol{r}^{(1)}$ but $\boldsymbol{r}^{(2)}$, and as the *a priori* probabilities it uses not $\boldsymbol{\pi}^{(0)}(0)$ but $\boldsymbol{\pi}^{(1)}(1)$ calculated by the first constituent decoder on the first phase of the first iteration. Then the decoder calculates the log-likelihood ratio

$$z_n^{(2)}(1) = z_n^{(0)}(0) + y_n^{(2)}(1) \tag{9.67}$$

and sends

$$\boldsymbol{z}^{(2)}(1) = z_0^{(2)}(1)\, z_1^{(2)}(1) \ldots z_{K-1}^{(2)}(1) \tag{9.68}$$

to the first constituent decoder. This completes the first iteration.

Now consider the ith iteration, $i = 2, 3, \ldots, I$. On the first phase of the ith iteration, the first constituent decoder, knowing the sequence of the log-likelihood ratios $\boldsymbol{z}^{(2)}(i-1)$, can calculate the sequence of the *a posteriori* probabilities $\boldsymbol{\pi}^{(2)}(i-1)$ and, hence, $\boldsymbol{\pi}_{\not\prec}^{(2)}(i-1)$. Using $\boldsymbol{r}_{\not\prec}^{(0)}$, $\boldsymbol{r}^{(1)}$, and $\boldsymbol{\pi}_{\not\prec}^{(2)}(i-1)$, the first constituent decoder calculates $y_n^{(1)}(i)$, $n = 0, 1, \ldots, K-1$, similarly to (9.63), and $z_n^{(1)}(i)$, $n = 0, 1, \ldots, K-1$, similarly to (9.65). Then it sends $\boldsymbol{z}^{(1)}(i)$ to the second constituent decoder etc. The idea is that on each phase of the iterative decoding process an active constituent decoder uses (an approximation of) the *a posteriori* probabilities, calculated by another constituent encoder on the previous phase of decoding, as the *a priori* probabilities and calculates a new approximation of the *a posteriori* probabilities.

The decoder alternates in this way during I iterations. The decision of $\widehat{u}_n$, $n = 0, 1, \ldots, K-1$, is obtained by comparing the final log-likelihood ratio $z_n^{(2)}(I)$ with the threshold 0:

$$\widehat{u}_n = \begin{cases} 0 & \text{if } z_n^{(2)}(I) > 0 \\ 1 & \text{if } z_n^{(2)}(I) < 0 \end{cases} \tag{9.69}$$

(If $z_n^{(I)} = 0$, then the decoder flips a coin and decides $\widehat{u}_n = 0$ or $\widehat{u}_n = 1$ with probability $1/2$.) An iterative decoder is illustrated in Fig. 9.4, where $\widehat{\boldsymbol{u}} = \widehat{u}_0\, \widehat{u}_1 \ldots \widehat{u}_{K-1}$.

The iterative decoding of multiple turbo codes is similar to the iterative decoding of the conventional turbo codes. In general, for a multiple turbo code with J constituent codes, each decoding iteration consists of J phases. On the first phase ($e = 1$) of the first iteration the decoder calculates similarly to (9.65) the statistics $z_n^{(1)}(1)$ for $n = 0, 1, \ldots, K-1$. The second phase ($e = 2$) is also similar to the second phase of the first iteration of the conventional turbo code. The decoder calculates the statistics $z_n^{(2)}(1)$ for $n = 0, 1, \ldots, K-1$, similarly to (9.67). On the following phases, $e = 3, \ldots, J$, the decoder calculates the statistics $z_n^{(e)}(1)$.

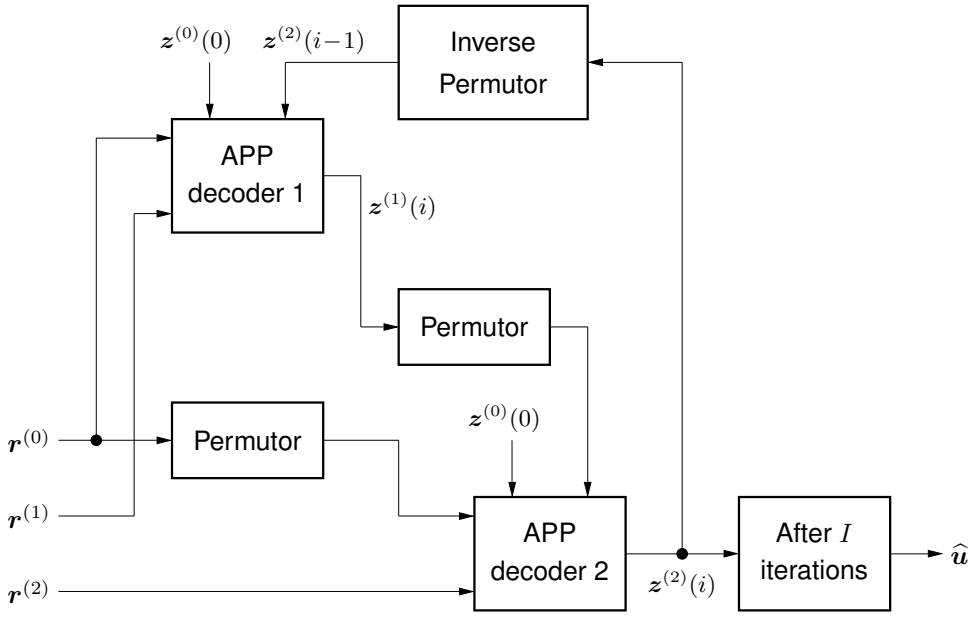


Figure 9.4 Iterative decoder for turbo codes.

After calculation of the Jth statistic $z_n^{(J)}(1)$ the decoder executes the second iteration consisting of J phases and recursively calculates the statistics $z_n^{(e)}(2)$ for $e = 1, 2, \ldots, J$. Then the decoder executes the 3th, 4th, . . ., Ith iterations and makes the decision of $\widehat{u}_n$ similarly to the conventional turbo code case.

We have described the iterative decoding of turbo codes for the discrete memoryless channel (DMC). The decoding of turbo codes for the AWGN channel is similar but we have to replace the conditional probabilities $P\left(r_n^{(0)} \,\middle|\, u_n\right)$ by the corresponding probability density functions.

Using turbo codes we can obtain a performance within less than 1 dB from the Shannon limit. Particularly, the Shannon limit for communication with $R = 1/3$ codes over the binary-input AWGN channel is -0.495 dB. The thresholds for rate $R = 1/3$ turbo codes with constituent convolutional encoders of memories $m = 2, 3, 4, 5$ when used with this channel are 0.05 dB, -0.08 dB, 0.00 dB, and 0.01 dB, respectively [LTZ01].

Simulation results of transmission over the AWGN channel for the conventional turbo code ($J = 2$) and the three-dimensional MTC ($J = 3$) are shown in Fig. 9.5 [HLC06]. Both codes have rate $R \approx 1/3$ (in the case of MTC puncturing was used to increase the rate). The length of the encoder input sequence is $K = 320$ and the permutor size is $K \times K$. The block lengths are 960. The constituent encoders are the four-state $\left(1 \quad (1 + D^2)/(1 + D + D^2)\right)$ encoders (9.1). The minimum distance of

the chosen codes is 29 for the conventional turbo code and 44 for the MTC. In Fig. 9.5 the bit error probability P_{b} and block error probability P_{B} are given as functions of the signal-to-noise ratio per bit E_{b}/N_0 (in dB). Note that for relatively low signal-to-noise ratios the performances of the two codes are very close (cf. Section 4.8), while for $E_{\mathrm{b}}/N_0 = 2.4$ dB the performance of the MTC is better than the performance of the conventional turbo code by more than one order. This reflects the fact that the MTC has larger minimum distance.

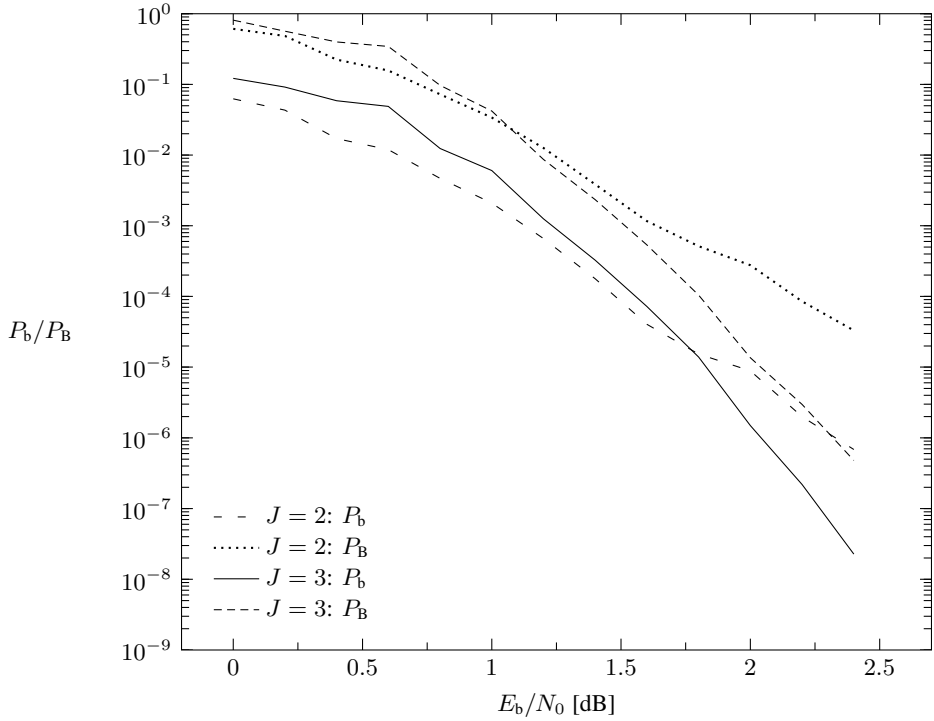


Figure 9.5 P_{b} and P_{B} as functions of the signal-to-noise ratio E_{b}/N_0 for a rate $R = 1/3$ conventional turbo code ($J = 2$) and a rate $R = 1/3$ three-dimensional MTC ($J = 3$).

9.5 BRAIDED CONVOLUTIONAL CODES*

In this section, we consider a class of turbo-like codes, *braided convolutional codes* (BCCs) [ZLZ10]. Similarly to braided block codes (BBCs), which are described in Section 8.7, braided convolutional codes use convolutional permutors, introduced in Section 8.3, as an important constituent element of the code construction, but in contrast to BBC they do use not block codes but convolutional codes as constituent codes.

An example of a rate $R = 1/3$ BCC encoder is shown in Fig. 9.6. The encoder consists of three single convolutional permutors (SCPs) $\boldsymbol{P}^{(0)}$, $\boldsymbol{P}^{(1)}$, and $\boldsymbol{P}^{(2)}$ and two constituent convolutional encoders, the horizontal encoder (encoder 1) of rate $R^{(1)}$ and the vertical encoder (encoder 2) of rate $R^{(2)}$ (we shall consider the case when $R^{(1)} = R^{(2)} = 2/3$ in more detail).

An information sequence $\boldsymbol{u} = u_0 u_1 \ldots u_t \ldots \stackrel{\text{def}}{=} \boldsymbol{v}^{(0)} = v_0^{(0)} v_1^{(0)} \ldots v_t^{(0)} \ldots$ enters the first input of encoder 1 directly, and the permuted information sequence $\tilde{\boldsymbol{v}}^{(0)} = \tilde{v}_0^{(0)} \tilde{v}_1^{(0)} \ldots \tilde{v}_t^{(0)} \ldots$, such that $\tilde{\boldsymbol{v}}^{(0)} = \boldsymbol{v}^{(0)} \boldsymbol{P}^{(0)}$ at the output of convolutional permutor $\boldsymbol{P}^{(0)}$ enters the first input of encoder 2. The encoder 1 generates the parity sequence $\boldsymbol{v}^{(1)} = v_0^{(1)} v_1^{(1)} \ldots v_t^{(1)} \ldots$ and encoder 2 generates the parity sequence $\boldsymbol{v}^{(2)} = v_0^{(2)} v_1^{(2)} \ldots v_t^{(2)} \ldots$. The permuted parity sequence $\tilde{\boldsymbol{v}}^{(2)} = \tilde{v}_0^{(2)} \tilde{v}_1^{(2)} \ldots \tilde{v}_t^{(2)} \ldots$, such that $\tilde{\boldsymbol{v}}^{(2)} = \boldsymbol{v}^{(1)} \boldsymbol{P}^{(1)}$, at the output of convolutional permutor $\boldsymbol{P}^{(1)}$, is fed back to the second input of encoder 2, and the permuted parity sequence $\tilde{\boldsymbol{v}}^{(1)} = \tilde{v}_0^{(1)} \tilde{v}_1^{(1)} \ldots \tilde{v}_t^{(1)} \ldots$, such that $\tilde{\boldsymbol{v}}^{(1)} = \boldsymbol{v}^{(2)} \boldsymbol{P}^{(2)}$ at the output of the convolutional permutor $\boldsymbol{P}^{(2)}$, is fed back to the second input of encoder 1.

Remark: In the array representation of the BCC we shall denote the second permutor as the transposed convolutional permutor $(\boldsymbol{P}^{(2)})^{\mathrm{T}}$ since the input symbols of the permutor are inserted columnwise and read out rowwise.

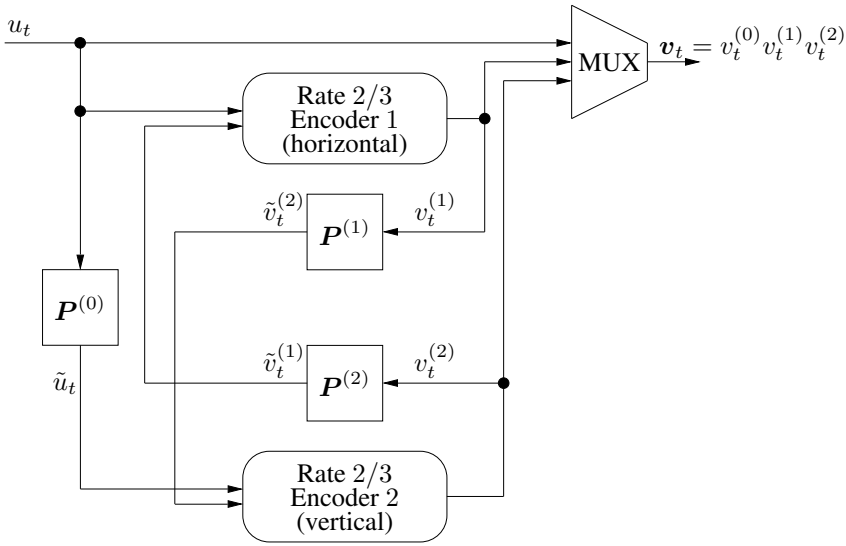


Figure 9.6 Encoder for a rate $R = 1/3$ braided convolutional code.

The information sequence $\boldsymbol{u} = \boldsymbol{v}^{(0)}$ and the parity sequences $\boldsymbol{v}^{(1)}$ and $\boldsymbol{v}^{(2)}$ are multiplexed into the output sequence of the BCC encoder $\boldsymbol{v} = \boldsymbol{v}_0 \boldsymbol{v}_1 \ldots \boldsymbol{v}_t \ldots$, where $\boldsymbol{v}_t = v_t^{(0)} v_t^{(1)} v_t^{(2)}$.

The BCC encoder is a convolutional-type encoder; the turbo encoder, described in Section 9.1, is a block-type encoder. If we would compare the BCC encoder in Fig. 9.6 with the turbo encoder in Fig. 9.1 we can see that, in contrast to the turbo encoder, whose constituent encoders have one input sequence, the BCC encoder has constituent encoders with two input sequences. Because of this, the free distance of the BBC grows linearly with the overall constraint length, while the minimum distance of the turbo code grows logarithmically with the overall constraint length.

An array representation of a rate $R = 1/3$ BCC with the encoder given in Fig. 9.6 is illustrated in Fig. 9.7. Each row and column of the array contains one information symbol, one parity symbol of the vertical encoder, and one parity symbol of the horizontal encoder. The sparse array retains the three-ribbon structure corresponding to three convolutional permutors. We assume that the SCPs $\boldsymbol{P}^{(j)} = (p_{nl}^{(j)})$, $j = 0, 1, 2$, are periodic with periods T_j and are constructed using the unwrapping procedure described in Section 8.3 with the width of each ribbon equal to the period of the corresponding permutor. Thus, the widths of the central, upper, and lower ribbons are T_0, T_1, and T_2, respectively.

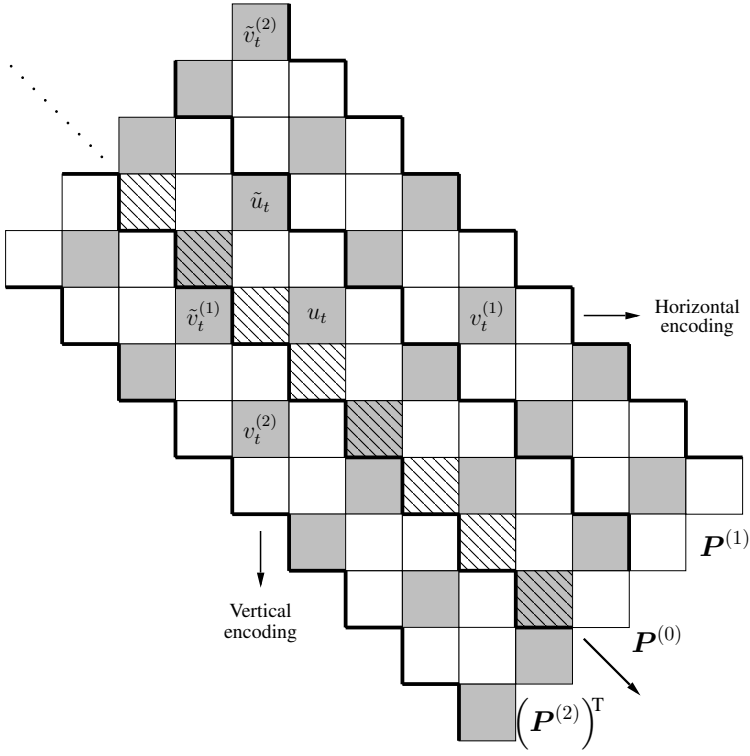


Figure 9.7 Array representation of a BCC. Notice that $\tilde{u}_t = u_{t-2}$.

The gray shaded squares in the array indicate cells with stored symbols. The information symbols are enumerated according to their positions in the sequence $\boldsymbol{u} = \boldsymbol{v}^{(0)}$; the parity symbols are enumerated according to their positions in the sequences $\boldsymbol{v}^{(1)}$ and $\boldsymbol{v}^{(2)}$. The information symbols u_t are placed in the central ribbon. The structure of the central ribbon is defined by the convolutional permutor $\boldsymbol{P}^{(0)}$. Based on the analysis in Section 8.3, a *typical* convolutional permutor $\boldsymbol{P}^{(0)}$ has an overall constraint length of $\nu_0 = (T_0 - 1)/2$. The parity symbols $v_t^{(1)}$ of the horizontal encoder are placed in the tth row of the upper ribbon. The structure of the upper ribbon is defined by the typical SCP $\boldsymbol{P}^{(1)}$. To match the ribbon structure of the array, this permutor has an additional delay of T_0 symbols, and its overall constraint length is $\nu_1 = T_0 + (T_1 - 1)/2$.

The parity symbols $v_t^{(2)}$ of the vertical encoder are placed in the tth column of the lower ribbon, whose structure depends on a typical convolutional permutor $(\boldsymbol{P}^{(2)})^{\mathrm{T}}$. To match the array structure, $(\boldsymbol{P}^{(2)})^{\mathrm{T}}$ has minimal delay 1, maximal delay T_2, and overall constraint length $\nu_2 = (T_2 - 1)/2 + 1$.

The memory of the encoder is defined as the maximum number of time units that a symbol stays in the encoder. The *overall constraint length* ν of a BCC encoder is defined as the maximal total number of symbols stored in the encoder. Thus, if all permutors $\boldsymbol{P}^{(0)}$, $\boldsymbol{P}^{(1)}$, and $(\boldsymbol{P}^{(2)})^{\mathrm{T}}$ are *typical* (see Section 8.3), the overall constraint length of the BCC encoder is

$$\nu = \frac{T_0 - 1}{2} + \frac{T_1 - 1}{2} + \frac{T_2 - 1}{2} + T_0 + 1 \tag{9.70}$$

If $T_0 = T_1 = T_2 = T$, the total width of the three ribbons in a BCC is $3T$, and the total number of symbols stored in the memory of the permutors is given by

$$\nu = 5(T - 1)/2 + 2 \tag{9.71}$$

Generalizations of the rate $R = 1/3$ BCC given in Fig. 9.6 to other rates and constructions are straightforward. We can use different constituent encoders for the horizontal and vertical encodings. Moreover, in general, $\boldsymbol{P}^{(0)}$, $\boldsymbol{P}^{(1)}$, and $(\boldsymbol{P}^{(2)})^{\mathrm{T}}$ can be multiple convolutional permutors (MCPs).

In Fig. 9.8, the results of some simulations of transmissions using a BCC over the binary-input AWGN channel are presented. The BCC uses two identical rate $2/3$, memory $m = 2$ systematic constituent encoders with feedback. The generator matrix of the constituent codes is

$$G(D) = \begin{pmatrix} 1 & 0 & \dfrac{1}{1 + D + D^2} \\ 0 & 1 & \dfrac{1 + D^2}{1 + D + D^2} \end{pmatrix} \tag{9.72}$$

The three convolutional permutors $\boldsymbol{P}^{(0)}$, $\boldsymbol{P}^{(1)}$, and $\boldsymbol{P}^{(2)}$ used in the encoder were constructed randomly with the same period T. We assumed that the transmitted sequence consists of an information sequence of length $50T$ and a tail of $2T$ zero tail bits. Thus, we have a rate loss of 2.67%, that is, the effective rate is about 0.325. In

the one-way APP decoder, a delay value τ was chosen to be equal to the period T and $I = 100$ decoding iterations were performed.[30]

In Fig. 9.8, we show the effect of the period T of the convolutional permutors on the bit error probability as a function of the signal-to-noise ratio E_b/N_0. We see that the performance of iterative decoding improves dramatically with increasing permutor period. The BCC achieves a bit error probability 10^{-6} at an E_b/N_0 of 0.2 dB with permutor period $T = 8000$, which is only 0.8 dB from the Shannon limit[31] of the binary-input AWGN channel with code rate 0.325. The pipeline iterative one-way decoder does not exhibit an error floor for any of the permutor periods used in the simulations.

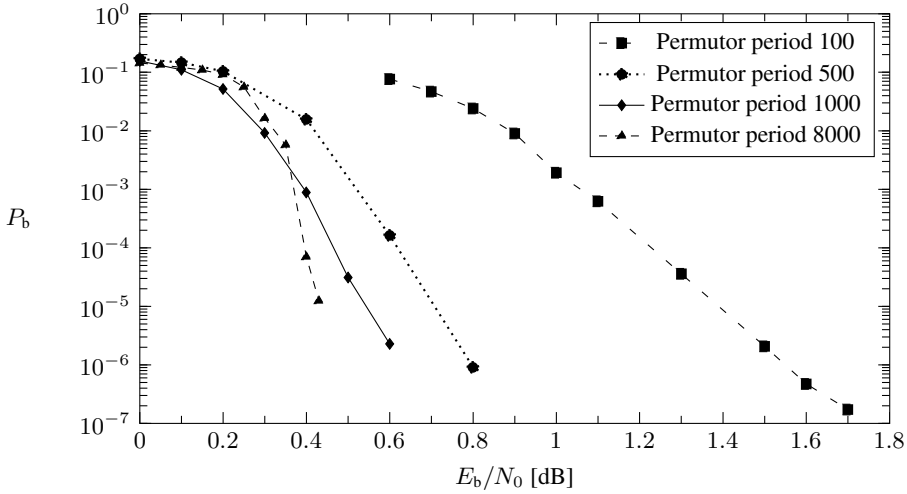


Figure 9.8 Error performance of a rate $R = 1/3$ terminated BCC on an AWGN channel.

In [ZLZ10] an analysis of the free distance is given for the rate $R = 1/3$ braided convolutional code using a Markov permutor (see Section 8.3) with overall constraint length ν. The encoder uses two identical rate $R = 2/3$ constituent encoders of memories $m = 2, 3, 4$. A lower bound on the free distance growing linearly with ν such that

$$d_\mathrm{free} \geq \rho_\mathrm{BCC} \nu R^{-1} + o(\nu) \tag{9.73}$$

was proved. The asymptotical coefficient ρ_BCC is given in Table 9.1.

[30]The value of $\tau = T$ was chosen for convenience, but in a practical implementation, a much smaller value of τ can be chosen to minimize latency.

[31]The Shannon limit for $R = 1/3$ in the binary-input AWGN channel equals -0.495 dB.

Table 9.1 The asymptotic coefficient ρ_{BCC} for the lower bound on the free distance (9.73) of rate $R = 1/3$ BCCs with constituent codes of various memories, $m = 2, 3, 4$.

Memory	Generator matrix	Coefficient ρ_{BCC}
$m = 2$	$\begin{pmatrix} 1 & 0 & D^2/(1+D+D^2) \\ 0 & 1 & (1+D^2)/(1+D+D^2) \end{pmatrix}$	0.2023
$m = 3$	$\begin{pmatrix} 1 & 0 & (1+D+D^2+D^3)/(1+D^2+D^3) \\ 0 & 1 & (1+D+D^3)/(1+D^2+D^3) \end{pmatrix}$	0.2410
$m = 4$	$\begin{pmatrix} 1 & 0 & (1+D^2+D^4)/(1+D^2+D^3+D^4) \\ 0 & 1 & (1+D+D^4)/(1+D^2+D^3+D^4) \end{pmatrix}$	0.2447

9.6 COMMENTS

As mentioned before, the original concept of turbo coding was introduced in a talk by Berrou, Glavieux, and Thitimajshima [BGT93] at the IEEE International Conference on Communication (ICC) held in Geneva, Switzerland, in May 1993. This presentation set off a revolution in coding and led, among many other important things, to the rediscovery of Gallager's LDPC codes.

Multiple turbo codes were introduced by Divsalar and Pollara [DiP95]. Analyses of multiple turbo codes were given in [KaU98, HLC06]. Laminated turbo codes [HZC08] is another class of turbo-like codes, similar to multiple turbo codes.

Braided convolutional codes were introduced in [ZLZ10] as a variant of turbo-like codes.

PROBLEMS

9.1 Show that if the input information sequence has even weight, then the memory $m = 1$ convolutional encoder with generator matrix (9.10) terminates in the allzero state.

9.2 Consider a turbo code with an SBP of length $K = 15$ described by the index permutation vector

$$\boldsymbol{\pi} = \begin{pmatrix} 5 & 8 & 11 & 14 & 2 & 6 & 9 & 12 & 0 & 3 & 10 & 13 & 1 & 4 & 7 \end{pmatrix}$$

Which permutor P corresponds to this index permutation vector? Construct the inverse of the permutor P.

9.3 Suppose that the constituent codes of the turbo code in Problem 9.2 have memory $m = 1$ and generator matrices (9.10). Show that the minimum distance of this turbo code is upper-bounded by 6, $d_{\text{min}} \leq 6$. Can you find a tighter upper bound?

9.4 Describe the Richardson-Urbanke construction of a turbo encoder with two constituent encoders of memory $m = 2$.

9.5 Consider a three-dimensional MTC with two SBP of length $K = 15$. They are described by the index permutation vectors

$$\boldsymbol{\pi}^{(2)} = \begin{pmatrix} 5 & 8 & 11 & 14 & 2 & 6 & 9 & 12 & 0 & 3 & 10 & 13 & 1 & 4 & 7 \end{pmatrix}$$

and

$$\boldsymbol{\pi}^{(3)} = \begin{pmatrix} 1 & 10 & 4 & 13 & 7 & 2 & 11 & 5 & 14 & 8 & 3 & 12 & 9 & 0 & 6 \end{pmatrix}$$

Which permutors $P^{(j)}$, $j = 2, 3$, correspond to these index permutation vectors?

9.6 Calculate numerically the upper bound (9.26) on the minimum distance of the three-dimensional MTC with the two SBCs described in Problem 9.5 and with three constituent recursive convolutional codes of memory $m = 1$ having generator matrices (9.10).

9.7 Consider a turbo code with two memory $m = 1$ constituent convolutional codes having generator matrix (9.10). Construct the computational tree for an arbitrary input information symbol u_n. Show that the clan head can have two, three, or four direct descendants in the first generation.

9.8 Consider a rate $R = 1/3$, length $N = 18$ turbo code with two memory $m = 1$ constituent convolutional codes both having generator matrix (9.10). The encoder of the code uses an SBC defined by the index permutation vector $\boldsymbol{\pi} = \begin{pmatrix} 1 & 4 & 2 & 5 & 3 & 0 \end{pmatrix}$. Suppose that this turbo code is used to communicate over a BSC with crossover probability $\epsilon = 0.045$. Assume that the received sequence is $\boldsymbol{r} = 100\ 010\ 011\ 011\ 011\ 111$.

Use the algorithm described in Section 9.4 to decode the received sequence when the information symbols are *a priori* equiprobable.

9.9 Consider the turbo encoder given in Fig. 9.1. Assume that the Richardson-Urbanke construction for the code termination is used and that the length of the encoder input information sequence is $\tilde{K} = 4$. The information symbols u_n, $n = 0, 1, 2, 3$, are *a priori* equiprobable. The index permutation vector for the SBP is $\boldsymbol{\pi} = \begin{pmatrix} 1 & 4 & 2 & 5 & 3 & 0 \end{pmatrix}$.

a) What is the length of the codeword?

b) Use the algorithm described in Section 9.4 to decode the received sequence $\boldsymbol{r} = 101\ 010\ 010\ 011\ 01\ 11$.

CHAPTER 10

CONVOLUTIONAL CODES WITH GOOD DISTANCE PROPERTIES

We have previously shown that when convolutional codes are used to communicate over channels at rather low signal-to-noise ratios the Viterbi spectrum is the principal determiner of the burst error probability when maximum-likelihood (or nearly so) decoding is used. We have also seen that an optimum distance profile is desirable to obtain a good computational performance with sequential decoding. Thus, it is important to find methods for constructing convolutional encoders with both a good distance spectrum and a good distance profile.

So far there has been little success in finding very good convolutional encoders by algebraic methods. Most encoders used in practice have been found by computer search.

In this chapter we discuss two algorithms: FAST for computing the Viterbi spectrum for convolutional encoders and BEAST for computing the weight spectrum for block codes and the Viterbi spectrum for convolutional encoders. Extensive tables of good convolutional encoders are given.

Fundamentals of Convolutional Coding, 2. Edition. By Rolf Johannesson, Kamil Sh. Zigangirov

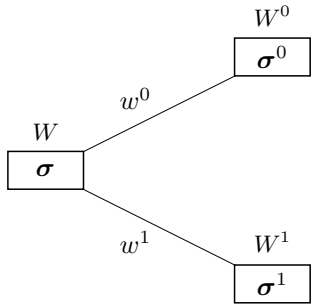


Figure 10.1 Successor nodes at time t.

10.1 COMPUTING THE VITERBI SPECTRUM USING FAST

To compute the *Viterbi spectrum* for a convolutional code encoded by a noncatastrophic generator matrix, that is, the number of paths with a given distance, d say, to the allzero path we exploit the linearity of the code and count the number of weight d sequences with $\boldsymbol{u}_0 \neq \mathbf{0}$ stemming from the zero state and terminating for the first time at the zero state. For simplicity we limit our discussion to binary codes of rate $R = 1/2$. The extension to rate $R = 1/c$ is trivial and to rate $R = b/c$ is straightforward. We assume that the rate $R = 1/2$ convolutional code is encoded by a minimal-memory m generator matrix that is realized in controller canonical form. As usual u_t and $\boldsymbol{v}_t$ denote the input and output at time t, respectively.

Suppose we are in an arbitrary node at depth t in the code tree and that we have produced channel symbols whose total weight is W_t. Then, in each subtree stemming from this node we have to *spend* the weight $(d - W_t)$. Hence, let us label each node with the *state* of the encoder and the *remaining weight*, that is, $W = d - W_t$.

Let $\boldsymbol{\sigma}_t = (\sigma_t^{(1)}\, \sigma_t^{(2)} \ldots \sigma_t^{(m)})$, where the *state variable* $\sigma_t^{(n)} = u_{t-n}$ for $n = 1, 2, \ldots, m$ and $u_t = 0$ for $t < 0$ denote the *state* of the encoder. From each state we have two successor states, $\boldsymbol{\sigma}_{t+1}^0 = (0\, u_{t-1} \ldots u_{t-m-1})$ and $\boldsymbol{\sigma}_{t+1}^1 = (1\, u_{t-1} \ldots u_{t-m-1})$, corresponding to information symbol u_t equal to zero and unity, respectively. To simplify the notations we suppress the index t in the sequel. For given encoders we can of course use the state of a node to determine the weights w^0 and w^1 of the branches stemming from that node. By using these branch weights together with the node weight W we can determine the two new node weights $W^0 = W - w^0$ and $W^1 = W - w^1$; see Fig. 10.1.

When searching for a path in the code tree with a given weight we explore a subtree if and only if the new node weight, W^u, is nonnegative and if the state of the new node, $\boldsymbol{\sigma}^u$, differs from the zero state. Let us arbitrarily give priority to the zero branch when we have to select between two new possible nodes.

A straightforward algorithm for determining the number of paths of a given weight d can be formulated as follows:

Start at state $\boldsymbol{\sigma} = (1\, 0 \ldots 0)$ with weight $W = d - d_0^{\mathrm{c}}$, where d_0^{c} is the 0th order column distance, and move forwards in the code tree. If $\boldsymbol{\sigma} = (0\, 0 \ldots 0\, 1)$ and

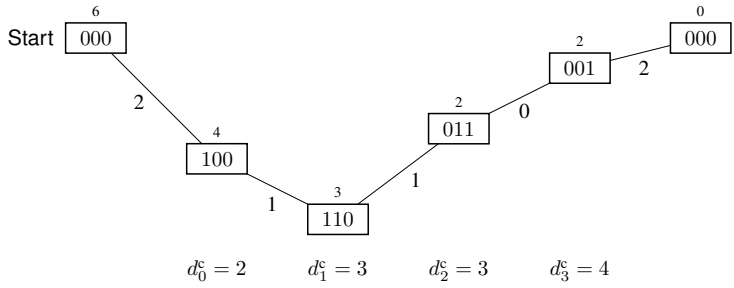


Figure 10.2 An example of a weight d_{free} path.

$W^0 = 0$, then increase the path counter. If the new node weight is negative or if the new node is in its zero state, then we will move *backwards*. Thus, we have to remember all of the previous information symbols so that we can move backwards until we find a new "one"-branch with a nonnegative node weight. Then we move forwards again. A *stop condition* appears when we reach the root.

This basic algorithm is of course very time consuming. To measure the performance of the algorithm we count the total number of nodes visited. Each visit to a node, regardless if we have been there before or not, is counted as a visit.

As an example we can use this basic algorithm to verify that the memory $m = 3$ encoder with encoding matrix $G(D) = (g_{11}(D) \;\; g_{12}(D)) = (1 + D + D^2 + D^3 \quad 1 + D^2 + D^3)$, or in octal notation $G = ((g_{11} \;\; g_{12}) = g_{11}^{(0)} \;\; g_{11}^{(1)} \;\; g_{11}^{(2)} \;\; g_{11}^{(3)} \;\; g_{12}^{(0)} \;\; g_{12}^{(1)} \;\; g_{12}^{(2)} \;\; g_{12}^{(3)}) = (1111 \;\; 1011) = (74 \;\; 54)$, has one path of weight $d_{\text{free}} = 6$. (The binary digits are collected in groups of three starting from the *left*.) We visit as many as 121 nodes in the explored tree.

Now we shall by an illuminative example show how we can obtain a substantial reduction in the number of nodes we have to visit.

Our encoding matrix $G = (74 \;\; 54)$ has an optimum distance profile $\boldsymbol{d}^{\text{p}} = (2, 3, 3, 4)$. In Fig. 10.2 we show only that part of its trellis which contains the weight 6 ($= d_{\text{free}}$) path. This path corresponds to the information sequence 11000, that is, to the encoded sequence $\boldsymbol{v}_{[0,4]} = 11\,01\,01\,00\,11$. Since the column distance is the *minimum* of the Hamming weights of the paths with $u_0 = 1$, the distance profile can be used as a *lower* bound on the *decrease of the node weight* along the path. In steps 1, 2, and 4 in Fig. 10.2 this bound is tight.

If we traverse this path in the opposite direction we will of course get the same total weight but different node weights. In Fig. 10.3 we can use the distance profile as a *lower* bound on the *node weights* along the path. Notice that if a node has weight less than this bound, then every path leading backwards to the zero state will give a negative node weight at the root node; for example, if the node weight in state (001) is less than $d_3^{\text{c}} = d_m^{\text{c}} = 4$ we must not extend this node when we are traversing the trellis backwards. More general, the weight of a backwards path stemming from a node in state $\boldsymbol{\sigma} \neq (00\ldots0)$, *starting with a one-branch*, and eventually leading to the root node (zero state) is lower-bounded by d_m^{c}.

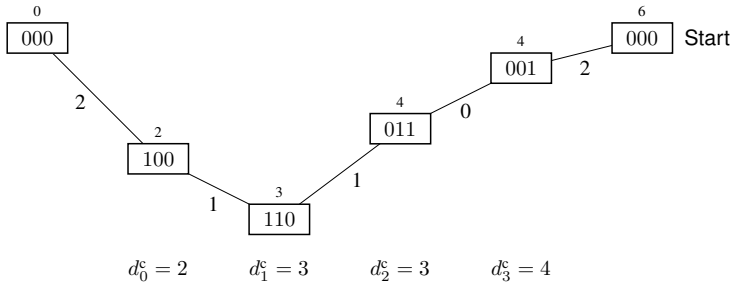


Figure 10.3 The weight d_{free} path traversed backward.

In Fig. 10.3 we notice that, for example, if the node weight in state (110) would be less than $d_1^c = 3$, then we must not extend this node.

The use of the distance profile as a lower bound works for every path from the end to the root node. Moving backwards from state $\boldsymbol{\sigma}$ we can reach the states $\boldsymbol{\sigma}^{-0}$ and $\boldsymbol{\sigma}^{-1}$, where

$$\boldsymbol{\sigma} = (?\ldots??1\underbrace{0\ldots00}_{\ell-1 \text{ zeros}}) \tag{10.1}$$

$$\boldsymbol{\sigma}^{-0} = (?\ldots?1\underbrace{00\ldots00}_{\ell \text{ zeros}}) \tag{10.2}$$

$$\boldsymbol{\sigma}^{-1} = (?\ldots?1\underbrace{00\ldots01}_{\ell-1 \text{ zeros}}) \tag{10.3}$$

The minimum weights of backwards paths stemming from the states $\boldsymbol{\sigma}^{-0}$ and $\boldsymbol{\sigma}^{-1}$ are lower-bounded by $d_{m-\ell-1}^c$ and d_{m-1}^c, respectively.

Instead of moving backwards in the trellis we can of course *reverse* the entries of the generator matrix and move forwards in the corresponding tree and use the distance profile (of the nonreciprocal generator matrix) to effectively limit the part of the tree that must be explored.

We shall now describe a *F*ast *A*lgorithm for *S*earching a code *T*ree *(FAST)* in order to determine the distance spectrum for a convolutional code with generator matrix $G = (g_{11}\ \ g_{12})$ and distance profile $\boldsymbol{d}^p$[CeJ89]. Let

$$\tilde{g}_{1k} = \left(\begin{array}{cccc} g_{1k}^{(m)} & g_{1k}^{(m-1)} & \ldots & g_{1k}^{(0)} \end{array} \right), \tag{10.4}$$

$k = 1, 2$, denote the generators for the *reversed* convolutional code encoded by the reciprocal generator matrix $\widetilde{G} = (\tilde{g}_{11}\ \ \tilde{g}_{12})$. The distance profile of the reciprocal generator matrix is denoted $\tilde{\boldsymbol{d}}^p = (\tilde{d}_0^c, \tilde{d}_1^c, \ldots, \tilde{d}_m^c)$. To calculate the ith spectral component we start at state $\boldsymbol{\sigma} = (10\ldots0)$ with weight $W = d_{\text{free}} + i - \tilde{d}_0^c$ in the code tree generated by the reciprocal generator matrix $\widetilde{G}$. Then we *reduce* this weight by the weights of the branches that we traverse when the code tree is searched for nodes with both node weight and state equal to zero. For the state of each explored node we use the column distances d_{m-l-1}^c or d_{m-1}^c to lower-bound the weight of

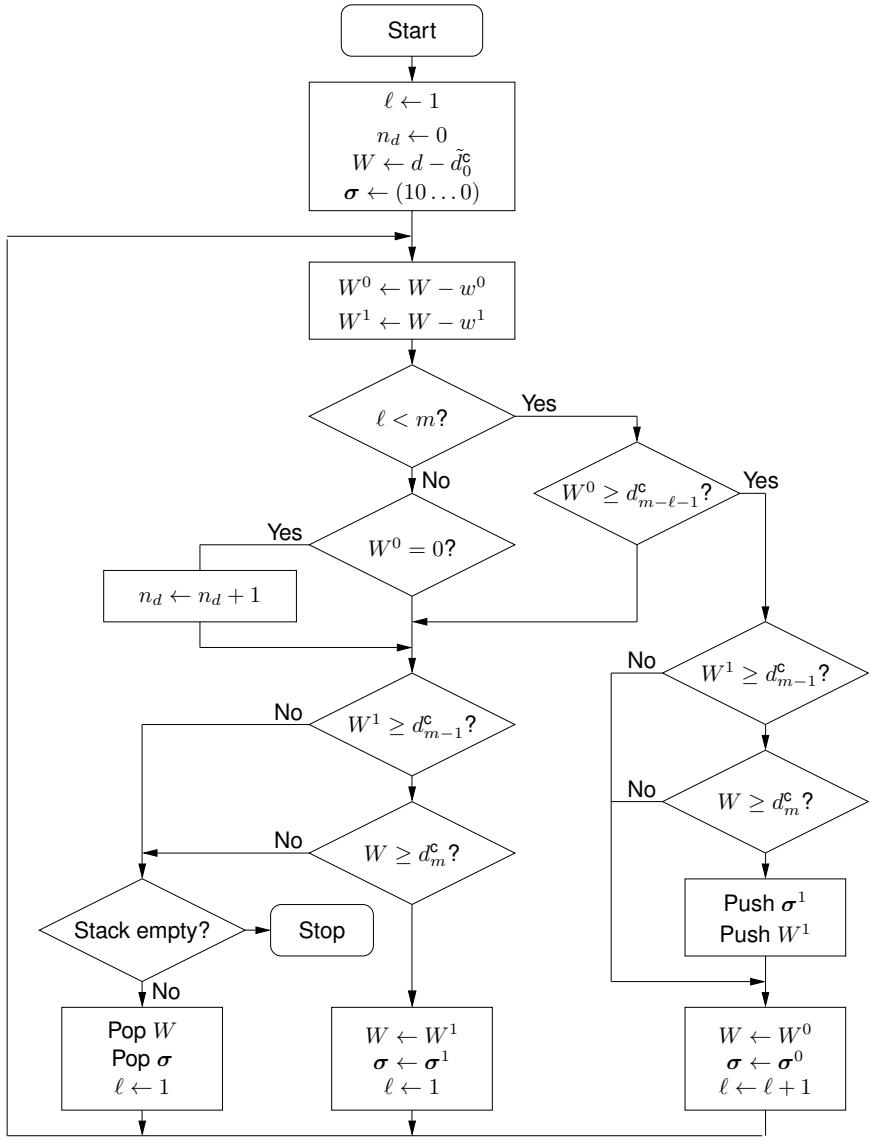


Figure 10.4 Flowchart of the FAST algorithm. Notice that w^i is calculated using the reciprocal generator matrix $\widetilde{G} = (\tilde{g}_{11}\ \ \tilde{g}_{12})$.

any path leading to a zero state. If the weight is less than this bound we will always reach a weight that is zero or negative at a nonzero state. Hence, it is only necessary to extend a node if the node weight is larger than or equal to this bound!

If both successor nodes are achievable, then we follow the zero branch and save (push) the one-branch node (state $\boldsymbol{\sigma}^1$ and weight W^1) on a stack. Thus we can avoid

calculating the node weight for the same node twice, and the algorithm will be twice as fast. (The basic algorithm should of course also be implemented with a stack.)

Our FAST algorithm is shown in Fig. 10.4. If $\boldsymbol{d}^{\mathrm{p}} \geq \tilde{\boldsymbol{d}}^{\mathrm{p}}$, then the *reciprocal* entries in the generator matrix, else the entries in the generator matrix, are used as inputs to FAST.

Using FAST to verify that $n_{d_{\mathrm{free}}} = 1$ for our encoding matrix $G = (74\ 54)$ we visit only five nodes!

Since we are interested in the spectral components for encoders with optimum (or good) distance profiles, it is interesting to notice that *the better the distance profile is, the faster* FAST *runs!*

10.2 THE MAGNIFICIENT BEAST

In Section 4.10 we discussed BEAST decoding for tailbing codes. BEAST was actually developed in order to speed up the calculations of the Viterbi spectra for convolutional encoders and the weight spectra for block codes [BHJ04].

Consider a rate $R = K/N$ binary block code whose codewords are binary N-tuples $\boldsymbol{v} = (v_0\, v_1 \ldots v_{N-1})$. Such a code can be described either by a forward code tree of length N starting from the root node (at depth 0) and going forwards or by a backward code tree starting from the toor node (at depth N) and going backwards. In the following, we shall distinguish between the forward and the backward code tree by the subscripts F and B, respectively.

Every node ξ in such a code tree has a unique parent node ξ^{p}, assuming only one binary digit per branch at most two children nodes ξ^{c}, and is characterized by three parameters: its state $\boldsymbol{\sigma}(\xi)$, weight $\omega(\xi)$, and depth $\ell(\xi)$. Its depth is equal to the length (in branches) of the path arriving at node ξ and starting from either the root node ξ_{root} or toor node ξ_{toor}, while its weight is determined by the accumulated Hamming weight of the corresponding path.

Consider, for example, the path $\xi_{\mathrm{root}} \to \xi$ in the forward code tree corresponding to the codeword segment $\boldsymbol{v}_{[0,\ell_{\mathsf{F}}(\xi))}$. Its accumulated Hamming weight is given by

$$\omega_{\mathsf{F}}(\xi) = w_{\mathrm{H}}\left(\boldsymbol{v}_{[0,\ell_{\mathsf{F}}(\xi))}\right) = \sum_{i=0}^{\ell_{\mathsf{F}}(\xi)-1} w_{\mathrm{H}}\left(v_i\right) \tag{10.5}$$

where $\ell_{\mathsf{F}}(\xi_{\mathrm{root}}) = 0$, $\omega_{\mathsf{F}}(\xi_{\mathrm{root}}) = 0$, and $\boldsymbol{\sigma}(\xi_{\mathrm{root}}) = \mathbf{0}$.

Similarly, the path $\xi_{\mathrm{toor}} \to \xi$ in the backward code tree corresponds to the codeword segment $\boldsymbol{v}_{[N-\ell_{\mathsf{B}}(\xi),N)}$ and yields the accumulated Hamming weight

$$\omega_{\mathsf{B}}(\xi) = w_{\mathrm{H}}\left(\boldsymbol{v}_{[N-\ell_{\mathsf{B}}(\xi),N)}\right) = \sum_{i=N-\ell_{\mathsf{B}}(\xi)}^{N-1} w_{\mathrm{H}}\left(v_i\right) \tag{10.6}$$

where $\ell_{\mathsf{B}}(\xi_{\mathrm{toor}}) = 0$, $\omega_{\mathsf{B}}(\xi_{\mathrm{toor}}) = 0$, and $\boldsymbol{\sigma}(\xi_{\mathrm{toor}}) = \mathbf{0}$.

Clearly, for every codeword $\boldsymbol{v}$ with Hamming weight w, there exists a path $\xi_{\mathrm{root}} \to \xi_{\mathrm{toor}}$ in the code tree with an intermediate node ξ such that

$$\omega_{\mathsf{F}}(\xi) = \left\lfloor \frac{w}{2} \right\rfloor, \qquad \omega_{\mathsf{B}}(\xi) = \left\lceil \frac{w}{2} \right\rceil, \qquad \ell_{\mathsf{F}}(\xi) + \ell_{\mathsf{B}}(\xi) = N \tag{10.7}$$

Hence, searching for all such paths (codewords) can be split up into two separate and independent steps: a *forward search* for all path segments $\xi_{\text{root}} \to \xi$ with Hamming weight w_{F} and a *backward search* for all path segments $\xi_{\text{toor}} \to \xi$ with Hamming weight w_{B}, where the *forward* and *backward weights* w_{F} and w_{B} can be chosen freely[32] as long as

$$w_{\mathsf{F}} + w_{\mathsf{B}} = w \tag{10.8}$$

Since every branch in a bit-level trellis is labeled by exactly one code symbol, the length of any such path segment has to satisfy

$$\omega_{\mathsf{F}}(\xi) \le \ell_{\mathsf{F}}(\xi) \le N - \omega_{\mathsf{B}}(\xi) \tag{10.9}$$

$$\omega_{\mathsf{B}}(\xi) \le \ell_{\mathsf{B}}(\xi) \le N - \omega_{\mathsf{F}}(\xi) \tag{10.10}$$

In other words, the maximum depth of the forward and backward code tree is limited by $N - w_{\mathsf{B}}$ and $N - w_{\mathsf{F}}$, respectively.

Algorithm BSB (BEAST for finding a spectral component for a block code)

BSB1. *Forward search:* Starting at the root node ξ_{root}, grow a forward code tree to obtain the set of nodes[33]

$$\mathcal{F} = \{\xi \mid \omega_{\mathsf{F}}(\zeta) = w_{\mathsf{F}},\, \omega_{\mathsf{F}}(\xi^{\text{P}}) < w_{\mathsf{F}},\, \ell_{\mathsf{F}}(\xi) \le N - w_{\mathsf{B}}\}$$

where w_{F} and w_{B} are chosen according to (10.8). The set $\mathcal{F}$ contains the leaves of the partially explored forward code tree, whose accumulated Hamming weights are equal to the forward weight w_{F}.

BSB2. *Backward search:* Starting at the toor node ξ_{toor}, grow a backward code tree to obtain the set of nodes[33]

$$\mathcal{B} = \{\xi \mid \omega_{\mathsf{B}}(\zeta) = w_{\mathsf{B}},\, \omega_{\mathsf{B}}(\xi^{\text{C}}) > w_{\mathsf{B}},\, \ell_{\mathsf{B}}(\xi) \le N - w_{\mathsf{F}}\}$$

Similar to step **BSB1**, the set $\mathcal{B}$ contains the last interior nodes of the partially explored backward code tree, before their accumulated Hamming weights exceed the backward weight w_{B}.

BSB3. *Matching:* Find all pairs of nodes $(\xi, \xi') \in \mathcal{F} \times \mathcal{B}$ such that

$$\boldsymbol{\sigma}(\xi) = \boldsymbol{\sigma}(\xi'), \qquad \ell_{\mathsf{F}}(\xi) + \ell_{\mathsf{B}}(\xi') = N \tag{10.11}$$

[32] In order to efficiently exploit the bidirectional idea behind BEAST, the size of the forward and backward code trees should be balanced, that is, the forward and backward weights w_{F} and w_{B} should be approximately equal.

[33] The conditions $\omega_{\mathsf{F}}(\xi^{\text{P}}) < w_{\mathsf{F}}$ and $\omega_{\mathsf{B}}(\xi^{\text{C}}) > w_{\mathsf{B}}$ in the forward and backward sets $\mathcal{F}$ and $\mathcal{B}$, respectively, are necessary to avoid multiple matches corresponding to the same codeword. While one of these two conditions would be sufficient to avoid such multiple matches, the second condition helps to reduce the number of stored nodes.

Each such match describes a unique codeword with Hamming weight $w = \omega_{\mathsf{F}}(\xi) + \omega_{\mathsf{B}}(\xi') = w_{\mathsf{F}} + w_{\mathsf{B}}$. Thus, the number of codewords with Hamming weight w, that is, the spectral component A_w, follows as

$$A_w = \sum_{(\xi,\xi') \in \mathcal{F} \times \mathcal{B}} \chi(\xi, \xi')$$

where χ is the match-indicator function, defined as

$$\chi(\xi, \xi') = \begin{cases} 1 & \text{if (10.11) holds} \\ 0 & \text{otherwise} \end{cases}$$

EXAMPLE 10.1

Consider the $(6, 3, 3)$ shortened Hamming block code with parity-check matrix

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

Assume that we want to find the number of codewords with Hamming weight $w = d_{\text{min}} = 3$, that is, A_3. According to (10.8) possible choices for the forward and backward weights are given by $w_{\mathsf{F}} = 2$ and $w_{\mathsf{B}} = 1$.

The corresponding, partially explored, forward and backward code trees are illustrated in Fig. 10.5. The nodes stored in the forward set $\mathcal{F}$ and in the backward set $\mathcal{B}$ are marked by squares, while dashed lines in the backward code tree indicate branches to children nodes exceeding the backward weight w_{B}. In total, the

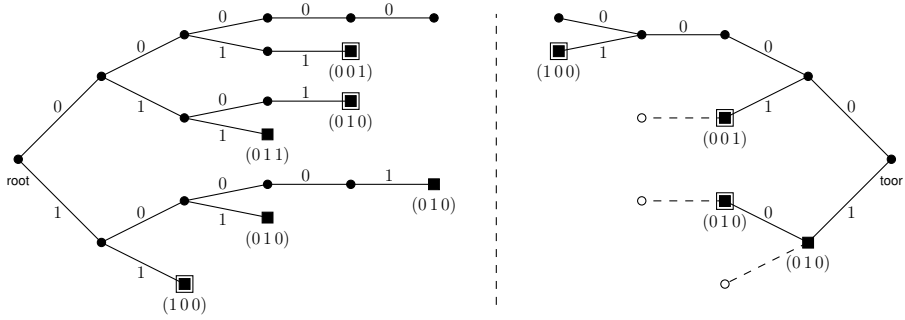


Figure 10.5 Partially explored forward and backward code trees used by BEAST to determine the number of codewords with Hamming weight $w = 3$ for the $(6, 3, 3)$ shortened Hamming code.

forward and backward sets contain $|\mathcal{F}| = 6$ and $|\mathcal{B}| = 4$ nodes, respectively, which are given below:

$$\mathcal{F} = \Big\{\Big(\boldsymbol{\sigma} = (1\,0\,0), \ell = 2\Big), \Big(\boldsymbol{\sigma} = (0\,1\,0), \ell = 3\Big), \Big(\boldsymbol{\sigma} = (0\,1\,1), \ell = 3\Big), \Big(\boldsymbol{\sigma} = (0\,1\,0), \ell = 4\Big), \Big(\boldsymbol{\sigma} = (0\,0\,1), \ell = 4\Big), \Big(\boldsymbol{\sigma} = (0\,1\,0), \ell = 5\Big)\Big\}$$

$$\mathcal{B} = \Big\{\Big(\boldsymbol{\sigma} = (0\,1\,0), \ell = 1\Big), \Big(\boldsymbol{\sigma} = (0\,1\,0), \ell = 2\Big), \Big(\boldsymbol{\sigma} = (0\,0\,1), \ell = 2\Big), \Big(\boldsymbol{\sigma} = (1\,0\,0), \ell = 4\Big)\Big\}$$

Comparing these nodes, we find three node pairs satisfying the condition (10.11), which are highlighted in Fig. 10.5: state $\boldsymbol{\sigma} = (1\,0\,0)$ with $\ell_{\mathsf{F}} = 2$ and $\ell_{\mathsf{B}} = 4$ as well as states $\boldsymbol{\sigma} = (0\,1\,0)$ and $\boldsymbol{\sigma} = (0\,0\,1)$, both with $\ell_{\mathsf{F}} = 4$ and $\ell_{\mathsf{B}} = 2$. Hence, the $(6, 3, 3)$ shortened Hamming code contains $A_3 = 3$ codewords of Hamming weight $w = 3$.

Next we shall calculate the Viterbi spectrum.

Consider a rate $R = b/c$ convolutional code with free distance d_{free}. Recall, that its $(i+1)$th Viterbi spectral component is denoted by $n_{d_{\text{free}}+i}$ and is defined as the number of paths with Hamming weight $d_{\text{free}} + i$, which diverge from the allzero path at the root of the code trellis and terminate in the allzero encoder state, but do not merge with the allzero path until their termini.

Similar to the case when finding the spectral component for a given block code, there exists an intermediate node $\boldsymbol{\sigma}(\xi) \neq \mathbf{0}$ for every path with Hamming weight w in the code tree which satisfies exactly one of the following c conditions:

$$\omega_{\mathsf{F}}(\xi) = \left\lfloor \frac{w}{2} \right\rfloor + j, \qquad \omega_{\mathsf{B}}(\xi) = \left\lceil \frac{w}{2} \right\rceil - j, \qquad j = 0,\, 1, \ldots,\, c-1 \tag{10.12}$$

where the additional term $j = 0,\, 1, \ldots,\, c-1$ originates from the fact that every branch in the corresponding trellis or code tree is labeled by a code c-tuple.

Since the length of the detour from the allzero path varies among different code sequences, several toor nodes have to be taken into account, one for each possible length of the detour of its code sequences. However, due to the regular and time-invariant structure of the trellis for convolutional codes, it is sufficient to only consider a single toor node and instead omit the restriction to a specific depth (length) in (10.7).

Algorithm BSC (BEAST for finding a spectral component for a convolutional encoder)

BSC1. *Forward search:* Starting at the root node ξ_{root}, extend the forward code tree to obtain c sets indexed by $j = 0,\, 1, \ldots,\, c-1$ containing only the states $\boldsymbol{\sigma}(\xi)$ of all nodes ξ satisfying

$$\mathcal{F}_{+j} = \{\xi \mid \omega_{\mathsf{F}}(\xi) = w_{\mathsf{F}} + j,\, \omega_{\mathsf{F}}(\xi^{\mathrm{p}}) < w_{\mathsf{F}},\, \boldsymbol{\sigma}(\xi) \neq \mathbf{0}\}$$

where w_{F} and hence w_{B} are chosen according to (10.8).

BSC2. *Backward search:* Starting at the toor node ξ_{toor}, extend the backward code tree to obtain c sets indexed by $j = 0,\, 1, \ldots,\, c-1$ containing only

the states $\boldsymbol{\sigma}(\xi)$ of all nodes ξ satisfying

$$\mathcal{B}_{-j} = \{\xi \mid \omega_{\mathsf{B}}(\xi) = w_{\mathsf{B}} - j,\, \omega_{\mathsf{B}}(\xi^{\mathrm{c}}) > w_{\mathsf{B}},\, \boldsymbol{\sigma}(\xi) \neq \mathbf{0}\}$$

BSC3. *Matching:* For every pair $\{\mathcal{F}_{+j}, \mathcal{B}_{-j}\}$, $j = 0,\, 1, \ldots,\, c-1$, find all pairs of nodes $(\xi, \xi') \in \mathcal{F}_{+j} \times \mathcal{B}_{-j}$ with equal states $\boldsymbol{\sigma}(\xi) = \boldsymbol{\sigma}(\xi')$. Then the number of convolutional code sequences with Hamming weight w follows as

$$n_w = \sum_{j=0}^{c-1} \sum_{(\xi,\xi') \in \mathcal{F}_{+j} \times \mathcal{B}_{-j}} \chi(\xi, \xi')$$

In [BHJ04] a comparison of FAST and BEAST is discussed. For example, when determining the spectral component n_{34} for the rate $R = 1/2$ convolutional encoder $G(D) = (6717423 \quad 5056615)$ of memory $m = 20$ and $d_{\text{free}} = 24$, FAST visited more than 1575 times as many nodes as BEAST. Moreover, BEAST was more than 600 times faster than FAST when programmed in C and run on a Pentium II 400 MHz computer (this test was done around year 2000!). In that paper a comparison with Rouanne's and Costello's bidirectional [RoC89] showed that their algorithm needed $14,000$ times as many comparisons as BEAST.

The ultimate test of BEAST is calculating the free distance of a rate $R = 5/20$ hypergraph-based woven convolutional code [HBJ10]. An implementation of its encoder is illustrated in Fig. 10.6. The input 5-tuple $u_i^{(1)}\, u_i^{(2)}\, \ldots\, u_i^{(5)}$, $i = 0, 1, 2, \ldots$, enters the encoder every fifth clock pulse. The output connections with modulo-2 adders of each of the registers 1–4 (counted from left to right) are time-varying and determined by the matrices $G_0, G_1, \ldots, G_4$. The connections of register 5 are time-invariant and are determined by the matrix G_5. During each round of five clock pulses we begin with four circular shifts and obtain the four output 4-tuples $v_i^{(1+4l)}\, v_i^{(2+4l)}\, \ldots\, v_i^{(4+4l)}$, $l = 0, 1, 2, 3$, by adding the 4-tuples of the outputs from the registers 1–4 to the 4-tuples of outputs from register 5. After these four circular shifts, $u_i^{(1)}$ is back at register 1 and the 4-tuple $v_i^{(1+4l)}\, v_i^{(2+4l)}\, \ldots\, v_i^{(4+4l)}$ is generated for $l = 4$. The corresponding time-varying connections are described by Table 10.1.

All registers can be considered as enlarged delay elements of the encoder of a tail-biting code. After one clock cycle of five clock pulses a 20-tuple, $v_i^{(1)}\, v_i^{(2)}\, \ldots\, v_i^{(20)}$, of the rate $R = 5/20$ convolutional code is generated. Then we shift a new input 5-tuple into the five registers without a circular shift and the next 20-tuple of output symbols is generated similarly.

Applying the Griesmer bound (3.122) to any rate $R = 5/20$, memory $m = 14$ convolutional code, we obtain $d_{\text{free}} \leq 154$.

Another approach is based on the row distances of convolutional encoders. For our encoder we obtain $d_0^{\mathrm{r}} = d_1^{\mathrm{r}} = d_2^{\mathrm{r}} = 130$ and $d_3^{\mathrm{r}} = \cdots = d_6^{\mathrm{r}} = 120$, and thus, we have $d_{\text{free}} \leq 120$. The row-distance approach yields a much stronger upper bound at the cost of rather heavy computer computations.

Finally, the BEAST algorithm is used for the code analysis. Finding the free distance for such a code would take a prohibitively long time without using parallel

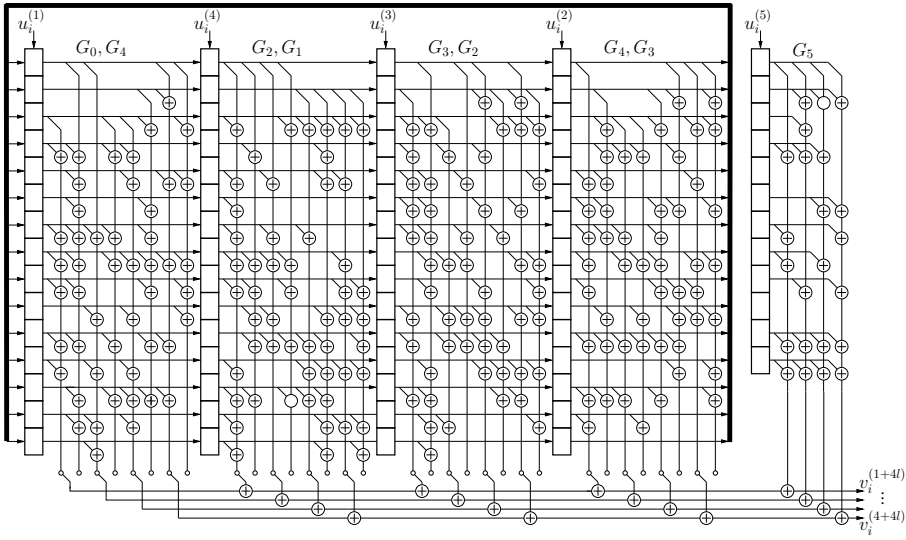


Figure 10.6 Implementation of an $R = 5/20$ hypergraph-based woven convolutional encoder.

computations on many processors. Using about 100 processors in parallel yields the free distance, $d_{\text{free}} = 120$, of this woven hypergraph-based convolutional code, where the individual forward and backward sets of the BEAST algorithm are sorted and merged by individual processors. We thereby obtain that there exists only one codeword of weight 120 with corresponding length $(1 + 3 + 11)20 = 300$.

For verifying such a huge free distance, it was crucial to have an extremely powerful algorithm. We used the BEAST algorithm where the sorting and merging of the individual forward and backward sets were distributed among several processors,

Table 10.1 Time-varying connections.

	Register			
Clock pulse	1	2	3	4
1	G_0	G_2	G_3	G_4
2	G_0	G_1	G_3	G_4
3	G_0	G_1	G_2	G_4
4	G_0	G_1	G_2	G_3
5	G_4	G_1	G_2	G_3

but we would also like to mention that by running the BEAST algorithm on a single laptop we could, in a few minutes, obtain that the free distance was at least 80. To verify $d_{\text{free}} = 120$ was a task worthy the magnificient BEAST.

It is interesting to notice that Costello's asymptotic lower bound on the free distance (3.161) establishes the existence of a code which for the parameters $R = 5/20$ and memory $m = 14$ has $d_{\text{free}} \geq 109$.

10.3 SOME CLASSES OF RATE $R = 1/2$ CONVOLUTIONAL CODES

An exhaustive search for convolutional codes with large d_{free} is practically impossible even for relatively short memories. Therefore we need efficient rejecting rules that limit the computation of d_{free} to a small fraction of the complete ensemble of encoders. As we mentioned in Chapter 3 the row distances can be used as a rejecting rule. Their efficiency is shown by the following example.

Let

$$G(D) = G_0 + G_1 D + \cdots + G_m D^m \tag{10.13}$$

The total number of rate $R = 1/2$, memory $m = 16$ ($G_m = (10)$, (01), or (11)) encoding matrices with $G_0 = (11)$ is $3 \cdot 2^{2m-2} = 3,221,225,472$. A simple way to generate all the encoding matrices $G = (g_{11} \;\; g_{12})$ and eliminate the encoding matrices $G' = (g_{12} \;\; g_{11})$ is: for each g_{11} test only those g_{12} for which $\tilde{g}_{12} < \tilde{g}_{11}$ (in obvious binary notation). The number of encoding matrices is reduced to $3 \cdot 2^{2m-3} - 2^{m-2}$ and, thus, we have $1,610,596,352$ encoding matrices left to test. Hoping to find an encoding matrix with $d_{\text{free}} = 20$ we reject successively all encoding matrices with $d_j^{\text{r}} < 20$, $j = 0, 1, \ldots, 15$, where 15 is arbitrarily chosen (see Fig. 10.7). After having used the row distance d_{15}^{r}, as a rejecting rule only 1034 candidates are left. Another 123 of these can be rejected since they suffer from *catastrophic error propagation*. Of the remaining 911 encoding matrices 200 have $d_{\text{free}} = 20$. The best one is given in Table 10.4. One might suspect that there exists a memory $m = 17$, $R = 1/2$, encoding matrix with $d_{\text{free}} = 21$. However, all candidates have row distance $d_{10}^{\text{r}} < 21$. The efficiency of using the row distances as rejecting rules in this case is shown in Fig. 10.7.

In Table 10.2 we give an extensive list of nonsystematic rate $R = 1/2$ ODP encoding matrices and in Table 10.3 we list the total number of bit errors for the codewords of weight d_{free} for nonsystematic rate $R = 1/2$ ODP encoding matrices. In Table 10.4 we give some encoding matrices that have d_{free} superior to or have better Viterbi spectra than those of the corresponding ODP encoding matrices in Table 10.2.

Massey and Costello [MaC71] introduced a class of rate $R = 1/2$ nonsystematic convolutional encoding matrices called *quick-look-in* (QLI) encoding matrices in which the two entries in each encoding matrix differ only in the second position:

$$G_{\text{QLI}}(D) = \begin{pmatrix} g_{11}(D) & g_{11}(D) + D \end{pmatrix} \tag{10.14}$$

Table 10.2 Viterbi spectra $n_{d_{\text{free}}+i}$, $i = 0, 1, \ldots, 5$, for nonsystematic rate $R = 1/2$ ODP encoding matrices $G = (g_{11} \quad g_{12})$.

				i					
m	g_{11}	g_{12}	d_{free}	0	1	2	3	4	5
2	7	5	5	1	2	4	8	16	32
3	74	54	6	1	3	5	11	25	55
4	62	56	7	2	3	4	16	37	68
5	77	45	8	2	3	8	15	41	90
6	634	564	10	12	0	53	0	234	0
7	626	572	10	1	6	13	20	64	123
8	751	557	12	10	9	30	51	156	340
9	7664	5714	12	1	8	8	31	73	150
10	7512	5562	14	19	0	80	0	450	0
11	6643	5175	14	1	10	25	46	105	258
12	63374	47244	15	2	10	29	55	138	301
13	45332	77136	16	5	15	21	56	161	381
14	65231	43677	17	3	16	44	62	172	455
15	727144	424374	18	5	15	21	56	161	381
16	717066	522702	19	9	16	48	112	259	596
17	745705	546153	20	6	31	58	125	314	711
18	6302164	5634554	21	13	34	72	161	369	914
19	5122642	7315626	22	26	0	160	0	916	0
20	7375407	4313045	22	1	17	49	108	234	521
21	67520654	50371444	24	40	0	251	0	1379	0
22	64553062	42533736	24	4	27	75	147	331	817
23	55076157	75501351	26	65	0	331	0	2014	0
24	744537344	472606614	26	10	45	91	235	465	1186
25	746411326	544134532	27	14	58	120	264	569	1406
26	525626523	645055711	28	24	56	131	273	736	1723
27	7270510714	5002176664	28	1	28	66	138	366	789
28	7605117332	5743521516	30	54	0	356	0	2148	0
29	7306324763	5136046755	30	5	47	97	211	514	1171
30	60425367524	45542642234	32	143	0	240	0	3870	0
31	51703207732	66455246536	32	14	65	136	336	753	1860
32	41273467427	70160662325	33	28	61	167	372	898	2168
33	407346436304	711526703754	34	44	0	338	0	2081	0
34	410174456276	702647441572	34	5	35	84	229	532	1320
35	627327244767	463171036121	36	111	0	553	0	3309	0
36	7664063056054	5707165143064	36	12	53	146	360	783	1917
37	7267577012232	5011131253046	37	18	73	163	381	884	2232
38	6660216760717	4131271202755	38	30	83	225	524	1152	2761
39	42576550101264	66340614757214	38	2	38	97	219	575	1324
40	26204724041271	37146123573117	40	78	0	532	0	6040	0

Table 10.3 Total number of bit errors for the codewords of weight d_{free} for nonsystematic rate $R = 1/2$ ODP encoding matrices $G = (g_{11} \quad g_{12})$.

m	g_{11}	g_{12}	d_{free}	$n_{d_{\text{free}}}$	# bit errors
1	6	4	3	1	1
2	7	5	5	1	1
3	74	54	6	1	2
4	62	56	7	2	4
5	77	45	8	2	4
6	634	564	10	12	46
7	626	572	10	1	6
8	751	557	12	10	40
9	7664	5714	12	1	2
10	7512	5562	14	19	82
11	6643	5175	14	1	4
12	63374	47244	15	2	6
13	77442	56506	16	5	16
14	65231	43677	17	3	17
15	764474	573304	18	5	20
16	717066	522702	19	9	55
	663256	513502	19	11	53
17	745705	546153	20	6	30
18	7746714	5634664	21	13	73
19	7315626	5122642	22	26	130
20	7375407	4313045	22	1	2
21	67520654	50371444	24	40	260
22	75457402	46705066	24	4	16
23	75501351	55076157	26	65	498
24	744537344	472606614	26	10	82

Table 10.4 Viterbi spectra $n_{d_{\text{free}}+i}$, $i = 0, 1, \ldots, 9$, for nonsystematic rate $R = 1/2$ OFD (optimum free distance) encoding matrices $G = (g_{11} \quad g_{12})$.

				i					
m	g_{11}	g_{12}	d_{free}	0	1	2	3	4	5
11	7173	5261	15	14	21	34	101	249	597
12	53734	72304	16	14	38	35	108	342	724
13	63676	45272	16	1	17	38	69	158	414
14	75063	56711	18	26	0	165	0	845	0
15	533514	653444	19	30	67	54	167	632	1402
16	626656	463642	20	43	0	265	0	1341	0
17	611675	550363	20	4	24	76	150	354	826
18	4551474	6354344	22	65	0	349	0	1903	0
19	7504432	4625676	22	5	52	116	163	456	1135
20	6717423	5056615	24	145	0	225	0	3473	0
21	63646524	57112134	24	17	95	136	138	679	2149
22	64353362	41471446	25	47	88	137	313	912	2172
23	75420671	45452137	26	45	0	364	0	1968	0
24	766446634	540125704	27	50	135	118	294	1481	3299
25	662537146	505722162	28	71	196	112	339	2053	4548
26	637044367	450762321	28	9	66	152	307	757	1823

The main feature of QLI encoding matrices is that they have a feedforward right pseudo inverse, viz.,

$$\widetilde{G_{\text{QLI}}^{-1}}(D) = \begin{pmatrix} 1 \\ 1 \end{pmatrix} \tag{10.15}$$

which can be implemented by a simple modulo 2 adder. Clearly,

$$G_{\text{QLI}}(D)\widetilde{G_{\text{QLI}}^{-1}}(D) = D \tag{10.16}$$

This makes it easy to extract an estimate of the information digits from the hard-decisioned received sequences. Furthermore, since the feedforward right inverse has "weight" 2, the *error amplification factor* $A = 2$ is the smallest possible for nonsystematic encoders.

j	# encoders with $d_j^{\text{r}} \geq 20$
0	543537361
1	267253166
2	84145636
3	19788663
4	4764506
5	1138502
6	309889
7	96872
8	35853
9	14974
10	7167
11	3954
12	2488
13	1650
14	1233
15	1034

j	# encoders with $d_j^{\text{r}} \geq 21$
0	2204679293
1	791375586
2	160725370
3	16854476
4	1471120
5	101684
6	5098
7	236
8	16
9	2
10	0

Figure 10.7 Two examples of using the row distance as a rejection rule.

In Tables 10.5 and 10.6 we list some QLI encoding matrices.

Extensive lists of systematic ODP encoding matrices $G = (4 \quad g_{12})$ are given in Tables 10.7–10.10.

10.4 LOW RATE CONVOLUTIONAL CODES

In Tables 10.11–10.14 we list rate $R = 1/3$ and $R = 1/4$ systematic as well as nonsystematic ODP convolutional encoding matrices. Their free distances are compared with Heller's and Griesmer's upper bounds in Figs. 10.8 and 10.9.

Table 10.5 Viterbi spectra $n_{d_{\text{free}}+i}$, $i = 0, 1, \ldots, 9$, for QLI rate $R = 1/2$ encoding matrices.

			i									
m	g_{11}	d_{free}	0	1	2	3	4	5	6	7	8	9
2	5	5	1	2	4	8	16	32	64	128	256	512
3	54	6	1	3	5	11	25	55	121	267	589	1299
4	46	7	2	4	6	15	37	83	191	442	1015	2334
5	55	8	2	7	10	18	49	124	292	678	1576	3694
6	454	9	4	8	11	25	70	181	405	945	2279	5414
7	542	9	1	4	13	25	51	115	270	686	1663	3955
8	551	10	1	9	18	30	73	172	379	992	2495	5735
9	5664	11	3	6	19	37	83	207	450	1146	2719	6631
10	5506	12	3	11	23	47	99	234	587	1474	3535	8363
11	5503	13	8	16	26	67	146	361	870	2128	5205	12510
12	56414	14	10	21	47	90	210	520	1311	3096	7458	17856
13	46716	14	3	12	32	71	141	335	877	1991	4852	11775
14	51503	15	6	14	36	68	176	469	1006	2390	5924	14285
15	510474	16	11	29	50	122	269	688	1637	3955	9574	22960
16	522416	16	2	21	35	51	155	376	898	2164	5337	12891
17	454643	17	5	23	38	90	230	499	1227	2994	7233	17526
18	5522214	18	6	26	62	139	326	727	1614	4070	10189	24338
19	4517006	18	2	16	42	78	173	445	1120	2610	6158	14933
20	5036543	19	4	24	50	97	253	586	1441	3525	8526	20482
21	47653514	20	7	26	69	138	349	867	1965	4803	11405	27759
22	51102726	20	1	17	42	93	215	476	1096	2733	6640	16127
23	53171663	21	3	31	70	136	327	754	1866	4531	10676	26209
24	510676714	22	7	38	77	171	423	948	2231	5469	13466	32186

Table 10.6 Viterbi spectra $n_{d_{\text{free}}+i}$, $i = 0, 1, \ldots, 9$, for QLI rate $R = 1/2$ ODP encoding matrices.

			i									
m	g_{12}	d_{free}	0	1	2	3	4	5	6	7	8	9
2	7	5	1	2	4	8	16	32	64	128	256	512
3	74	6	1	3	5	11	25	55	121	267	589	1299
4	76	6	1	1	3	7	18	40	87	209	476	1096
5	75	8	2	7	10	18	49	124	292	678	1576	3694
6	714	8	1	2	5	14	27	68	157	366	914	2161
7	742	9	1	4	13	25	51	115	270	686	1663	3955
8	743	9	1	1	5	12	21	51	127	316	780	1886
9	7434	10	2	1	6	14	31	112	219	492	1205	2846
10	7422	11	2	5	14	26	57	146	345	841	2070	4956
11	7435	12	5	3	10	45	81	183	427	1020	2593	6186
12	74044	11	1	1	5	18	33	62	162	377	930	2352
13	74046	13	2	6	7	19	48	115	278	676	1726	4070
14	74047	14	2	8	12	32	71	184	402	981	2391	5589
15	740464	14	2	1	6	18	61	89	260	633	1466	3560
16	740462	15	3	5	11	33	67	168	404	992	2470	5903
17	740463	16	2	9	15	46	114	231	585	1344	3179	7850
18	7404634	16	1	2	13	24	43	139	283	741	1717	4040
19	7404242	15	1	0	2	9	19	48	143	315	725	1825
20	7404155	18	2	12	15	45	126	226	552	1412	3329	8109
21	74041544	18	2	4	6	36	78	183	439	1026	2419	6049
22	74042436	19	2	9	13	28	96	225	539	1283	3131	7534
23	74041567	19	1	2	8	26	50	105	302	722	1702	4064
24	740415664	20	1	8	11	29	67	170	427	939	2325	5702
25	740424366	20	1	3	6	19	54	117	242	567	1447	3525
26	740424175	22	8	7	38	52	164	311	806	1996	4828	12103
27	7404155634	22	2	6	14	31	93	186	467	1141	2658	6545
28	7404241726	23	2	7	24	38	105	270	685	1589	3936	9611
29	7404154035	24	6	24	32	84	202	473	1195	2653	6687	16203
30	74041567514	23	1	1	6	10	34	88	208	559	1293	3051
31	74041567512	25	5	11	15	54	134	332	841	2072	4878	11683

Table 10.7 Viterbi spectra $n_{d_{\text{free}}+i}$, $i = 0, 1, \ldots, 9$, for systematic rate $R = 1/2$ ODP encoding matrices $G = (4 \quad g_{12})$.

			i									
m	g_{12}	d_{free}	0	1	2	3	4	5	6	7	8	9
1	6	3	1	1	1	1	1	1	1	1	1	1
2	7	4	2	0	5	0	13	0	34	0	89	0
3	64	4	1	0	6	0	16	0	69	0	232	0
4	66	5	2	2	1	10	21	29	77	180	332	711
5	73	6	3	0	13	0	55	0	298	0	1401	0
6	674	6	1	3	4	11	25	53	118	274	654	1430
7	714	6	2	0	9	0	46	0	248	0	1289	0
8	671	7	1	5	5	17	35	70	173	452	993	2415
9	7154	8	4	0	19	0	94	0	542	0	3159	0
10	7152	8	3	0	16	0	79	0	457	0	2618	0
11	7153	9	3	5	11	26	52	124	317	821	1870	4364
12	67114	9	1	4	10	15	46	104	224	576	1368	3322
13	67116	10	5	0	27	0	124	0	777	0	4529	0
14	71447	10	4	0	12	0	105	0	517	0	3138	0
15	671174	10	1	0	16	0	78	0	437	0	2391	0
16	671166	12	13	0	46	0	263	0	1486	0	9019	0
17	671166	12	13	0	46	0	263	0	1486	0	9019	0
18	6711454	12	4	0	23	0	154	0	817	0	4896	0
19	7144616	12	3	0	23	0	92	0	556	0	3472	0
20	7144761	12	1	3	10	25	53	110	263	676	1593	3838
21	71447614	12	1	0	7	0	66	0	314	0	1842	0
22	71446166	14	6	0	44	0	189	0	1132	0	6570	0
23	67115143	14	2	0	38	0	168	0	947	0	5726	0
24	714461654	15	5	7	23	62	115	256	669	1648	3999	9703
25	671145536	15	3	7	16	44	112	244	578	1312	3267	8097
26	714476053	16	8	0	54	0	289	0	1691	0	9609	0
27	7144760524	16	7	0	73	0	350	0	1971	0	11624	0
28	7144616566	16	3	0	38	0	134	0	834	0	5052	0
29	7144760535	18	22	0	118	0	695	0	3926	0	22788	0
30	67114543064	16	1	1	10	15	36	101	225	596	1342	3298
31	67114543066	18	11	0	53	0	307	0	1742	0	10218	0

Table 10.8 Total number of bit errors for the codewords of weight d_{free} for systematic rate $R = 1/2$ ODP encoding matrices $G = (4 \quad g_{12})$.

m	g_{12}	d_{free}	$n_{d_{\text{free}}}$	# bit errors
1	6	3	1	1
2	7	4	2	3
3	64	4	1	1
4	66	5	2	4
5	73	6	3	6
6	674	6	1	2
7	714	6	2	3
8	671	7	1	1
9	7154	8	4	11
10	7152	8	3	8
11	7153	9	3	9
12	67114	9	1	1
13	67116	10	5	13
14	71447	10	4	9
15	671174	10	1	2
16	671166	12	13	44
17	671166	12	13	44
18	6711514	12	4	9
19	7144616	12	3	7
20	7144761	12	1	2
21	71447614	12	1	2
22	71446166	14	6	20
23	67115143	14	2	4
24	714476124	15	5	17
25	671145536	15	3	11
26	714476053	16	8	28
27	7144760524	16	7	28
28	7144616566	16	3	11
29	7144760535	18	22	89
30	71446165670	16	1	2
31	67114543066	18	11	50

Table 10.9 Viterbi spectra $n_{d_{\text{free}}+i}$, $i = 0, 1, \ldots, 4$, for systematic rate $R = 1/2$ OFD encoding matrices $G = (4 \quad g_{12})$.

			i				
m	g_{12}	d_{free}	0	1	2	3	4
2	7	4	2	0	5	0	13
3	54	4	1	0	6	0	16
4	66	5	2	2	1	10	21
6	674	6	1	3	4	11	25
7	556	7	3	4	7	20	35
8	727	8	7	0	23	0	133
9	5754	8	3	6	8	23	55
10	6716	8	1	4	6	12	39
11	6457	9	2	7	13	27	60
12	65474	10	7	0	43	0	190
13	63656	10	2	8	15	36	70
14	65347	11	6	13	18	50	118
15	647654	12	16	0	71	0	404
16	676456	12	7	21	23	51	146
17	647653	12	2	9	22	43	103
18	7124674	13	7	21	39	66	156
19	6754516	14	21	0	114	0	581
20	6476267	14	9	24	35	84	195
21	67612634	14	1	17	29	61	162
22	65471166	15	10	26	50	79	239
23	67612547	16	34	0	150	0	858
24	647626354	16	11	34	58	132	288
25	674124766	16	4	25	43	90	212
26	732624767	17	13	32	75	169	351
27	7123744254	17	4	27	50	104	266
28	6563731156	18	25	0	165	0	
29	7256157123	18	8	36	71	128	

Table 10.10 Minimum distances d_m and # truncated codewords $\boldsymbol{v}_{[0,m]}$ of weight d_m for systematic rate $R = 1/2$ ODP encoding matrices $G = (4 \quad g_{12})$.

m	g_{12}	d_m	$\#\boldsymbol{v}_{[0,m]}$ of weight d_m
1	6	3	1
2	6	3	1
3	64	4	3
4	64	4	1
5	65	5	5
6	650	5	2
7	670	6	11
8	670	6	5
9	6710	6	1
10	6710	7	12
11	6711	7	5
12	67114	8	29
13	67114	8	12
14	67115	8	6
15	671150	8	1
16	671144	9	18
17	671151	9	7
18	6711514	9	3
19	6711454	10	31
20	6711454	10	13
21	67114544	10	4
22	67115142	10	1
23	67114543	11	27
24	671145430	11	11
25	671151572	11	5
26	671151505	11	1
27	6711454574	12	21
28	6711454306	12	8
29	6711454311	12	2
30	67114545754	13	43
31	67114545754	13	15
32	67114545755	13	4
33	671145457554	13	1
34	671145457556	14	34
35	671145454470	14	14
36	6711454544704	14	5
37	6711454544676	14	2
38	6711454575564	15	31
39	67114545755644	15	12
40	67114545755712	15	3
41	67114545755713	15	1

Table 10.10 (cont'd) Minimum distances d_m and # truncated codewords $\boldsymbol{v}_{[0,m]}$ of weight d_m for systematic rate $R = 1/2$ ODP encoding matrices $G = (4 \quad g_{12})$.

m	g_{12}	d_m	$\#\boldsymbol{v}_{[0,m]}$ of weight d_m
42	671145457556464	16	31
43	671145457556464	16	14
44	671145457556153	16	5
45	6711454575561314	16	1
46	6711454575564666	17	39
47	6711454575564667	17	13
48	67114545755646674	17	4
49	67114545755646676	17	1
50	67114545755646676	18	38
51	671145457556466760	18	16
52	671145457556466760	18	7
53	671145457550027077	18	2
54	6711454575571301174	19	43
55	6711454575571301176	19	20
56	6711454575571301176	19	7
57	6711454755713011760	19	2
58	6711454755713011760	20	60
59	6711454755646670367	20	25
60	67114545755646703670	20	10
61	67114545755713011760	20	2
62	67114545755713011761	20	1
63	671145457557130117611	21	25
64	671145457557130117611	21	10
65	671145457557130117611	21	2
66	671145457557130117611	22	71
67	671145457556466703670	22	29
68	671145457556466703671	22	9
69	671145457556466703670	22	4
70	671145457556466703670	22	1
71	671145457557130117611463	23	46
72	671145457556466703670144	23	16
73	671145457556466703670146	23	5
74	671145457556466703670147	23	2
75	671145457557130117611463	24	56
76	671145457557130117611463	24	20
77	671145457557130117611463	24	8
78	671145457557130117611463	24	3
79	671145457557130117611463	25	74
80	671145457557130117611463	25	33
81	671145457557130117611463	25	16
82	671145457556466703670144	25	4

Table 10.10 (cont'd) Minimum distances d_m and # truncated codewords $\boldsymbol{v}_{[0,m]}$ of weight d_m for systematic rate $R = 1/2$ ODP encoding matrices $G = (4 \quad g_{12})$.

m	g_{12}	d_m	$\#\boldsymbol{v}_{[0,m]}$ of weight d_m
83	671145457556466703670144727 7	25	1
84	6711454575564667036701447273 0	26	41
85	6711454575571301176114634336 2	26	20
86	6711454575571301176114634336 3	26	6
87	67114545755713011761146343363 4	26	2
88	671145457556466703670144727304	27	62
89	671145457556466703670144727305	27	28
90	671145457556466703670144727305 4	27	11
91	671145457556466703670144727305 6	27	5
92	671145457556466703670144727335 7	27	1
93	671145457556466703670144727305 10	28	42
94	671145457556466703670144727305 12	28	20
95	671145457556466703670144727305 11	28	5
96	671145457556466703670144727305 110	28	1

Table 10.11 Viterbi spectra $n_{d_{\text{free}}+i}$, $i = 0, 1, \ldots, 5$, for systematic rate $R = 1/3$ ODP encoding matrices $G = (4 \quad g_{12} \quad g_{13})$.

				i					
m	g_{12}	g_{13}	d_{free}	0	1	2	3	4	5
1	6	6	5	1	1	1	1	1	1
2	5	7	6	1	0	2	0	4	0
3	64	74	8	2	0	3	0	8	0
4	56	72	9	1	2	3	3	1	2
5	57	73	10	1	3	1	0	4	10
6	564	754	12	4	0	8	0	12	0
7	516	676	12	1	2	1	7	8	7
8	531	676	13	1	3	3	6	8	9
9	5314	6764	15	3	5	2	4	11	11
10	5312	6766	16	4	0	8	0	20	0
11	5317	6767	16	1	2	4	7	7	15
12	65304	71274	17	1	2	6	8	4	12
13	65306	71276	18	1	3	4	7	5	9
14	65305	71273	19	2	2	3	5	9	23
15	653764	712614	20	2	0	8	0	19	0
16	531206	676672	20	1	0	5	0	11	0
17	653055	712737	22	2	0	7	0	20	0
18	5144574	7325154	24	6	0	19	0	40	0
19	6530576	7127306	24	2	0	11	0	27	0
20	6530547	7127375	26	4	0	21	0	43	0
21	65376114	71261054	26	3	0	11	0	23	0
22	51445036	73251266	26	1	0	4	0	18	0
23	65305477	71273753	28	3	4	3	9	17	25
24	514453214	732513134	28	1	0	8	0	29	0
25	653761172	712610566	30	2	0	14	0	46	0
26	514450363	732512675	31	2	5	9	14	17	25
27	6537616604	7126106264	32	10	0	19	0	43	0
28	6537616606	7126106264	33	5	13	12	13	23	54
29	5312071307	6766735721	33	3	2	5	12	16	31
30	51445320354	73251313564	34	1	6	6	5	14	31

Table 10.12 Viterbi spectra $n_{d_{\text{free}}+i}$, $i = 0, 1, \ldots, 5$, for nonsystematic rate $R = 1/3$ ODP encoding matrices $G = (g_{11} \quad g_{12} \quad g_{13})$.

					i					
m	g_{11}	g_{12}	g_{13}	d_∞	0	1	2	3	4	5
1	4	6	6	5	1	1	1	1	1	1
2	5	7	7	8	2	0	5	0	13	0
3	54	64	74	10	3	0	2	0	15	0
4	52	66	76	12	5	0	3	0	13	0
5	47	53	75	13	1	3	6	4	5	12
6	574	664	744	14	1	0	8	0	11	0
7	536	656	722	16	1	5	2	6	14	18
8	435	526	717	17	1	2	6	7	6	13
9	5674	6304	7524	20	7	0	19	0	40	0
10	5136	6642	7166	21	4	1	4	14	18	28
11	4653	5435	6257	22	3	0	9	0	32	0
12	47164	57254	76304	24	2	8	10	15	18	29
13	47326	61372	74322	26	7	0	23	0	64	0
14	47671	55245	63217	27	6	4	6	21	24	37
15	447454	632734	766164	28	1	6	5	17	24	34
16	552334	614426	772722	30	3	9	20	21	29	49
17	552137	614671	772233	32	7	15	11	21	58	82
18	4550704	6246334	7731724	34	28	0	53	0	112	0
19	5531236	6151572	7731724	35	8	18	29	32	54	78

Table 10.13 Viterbi spectra $n_{d_{\text{free}}+i}$, $i = 0, 1, \ldots, 5$, for systematic rate $R = 1/4$ ODP encoding matrices $G = (4 \quad g_{12} \quad g_{13} \quad g_{14})$.

					i					
m	g_{12}	g_{13}	g_{14}	d_{free}	0	1	2	3	4	5
1	4	6	6	6	1	0	1	0	1	0
2	5	6	7	8	1	0	1	0	2	0
3	54	64	74	11	1	0	1	0	5	0
4	56	62	72	12	1	0	1	0	5	0
5	51	67	73	14	1	0	2	0	5	0
6	534	634	754	16	1	3	0	0	3	5
7	516	676	732	18	1	3	1	2	3	3
8	535	637	755	20	2	4	0	3	2	4
9	5350	6370	7554	20	2	0	2	0	9	0
10	5156	6272	7404	20	1	0	1	0	4	0
11	5351	6371	7557	24	1	0	7	0	7	0
12	53514	63714	75574	24	1	0	1	2	1	8
13	51056	63116	76472	26	2	0	2	0	5	0
14	51055	63117	76473	28	1	0	5	0	4	0
15	515630	627350	740424	27	1	0	0	2	3	2
16	530036	611516	747332	30	3	0	2	0	6	0
17	535154	637141	755775	30	1	0	1	0	5	0
18	5105444	6311614	7647074	32	1	0	1	0	3	0
19	5105446	6311616	7647072	34	2	0	3	0	4	0
20	5105447	6311617	7647073	36	2	2	3	5	1	4
21	51054474	63116164	76470730	36	1	0	3	0	4	0
22	51563362	62735066	74040356	38	2	0	3	0	5	0
23	51054477	63116167	76470731	40	1	1	3	4	1	3
24	510544764	631161674	764707304	42	2	0	7	0	5	0
25	510544770	631161666	764707302	42	2	0	3	0	10	0
26	510544771	631161667	764707303	44	1	0	1	3	2	12
27	5105447710	6311616664	7647073024	47	3	3	6	9	13	9
28	5105447714	6311616664	7647073032	48	3	4	2	7	8	17
29	5105447715	6311616671	7647073025	51	6	3	4	8	15	25
30	51054477154	63116166734	76470730324	52	3	2	7	6	9	13

Table 10.14 Viterbi spectra $n_{d_{\text{free}}+i}$, $i = 0, 1, \ldots, 5$, for nonsystematic rate $R = 1/4$ ODP encoding matrices $G = (g_{11} \quad g_{12} \quad g_{13} \quad g_{14})$.

						i					
m	g_{11}	g_{12}	g_{13}	g_{14}	d_{free}	0	1	2	3	4	5
1	4	4	6	6	6	1	0	1	0	1	0
2	4	5	6	7	8	1	0	1	0	2	0
3	44	54	64	70	11	1	0	0	1	1	2
4	46	52	66	76	15	1	2	1	1	1	3
5	47	53	67	75	18	3	0	5	0	8	0
6	454	574	664	724	20	3	0	4	0	7	0
7	476	556	672	712	22	1	5	2	2	4	4
8	457	575	663	723	24	1	3	4	7	2	1
9	4730	5574	6564	7104	26	3	0	4	0	12	0
10	4266	5362	6136	7722	28	4	0	5	0	10	0
11	4227	5177	6225	7723	30	4	0	4	0	11	0
12	46554	56174	66450	72374	32	1	3	6	9	6	13
13	45562	57052	64732	73176	34	1	0	11	0	11	0
14	47633	57505	66535	71145	37	3	5	6	10	11	11
15	454374	574624	662564	723354	39	5	7	10	4	5	10
16	463712	566132	661562	727446	41	3	7	7	10	19	11
17	415727	523133	624577	744355	42	1	0	14	0	17	0
18	4653444	5426714	6477354	7036504	45	3	5	8	13	16	14
19	4654522	5617436	6645066	7237532	46	1	0	13	0	20	0
20	4712241	5763615	6765523	7330467	50	13	0	18	0	39	0
21	45724414	55057474	65556514	72624710	50	1	7	6	13	15	13

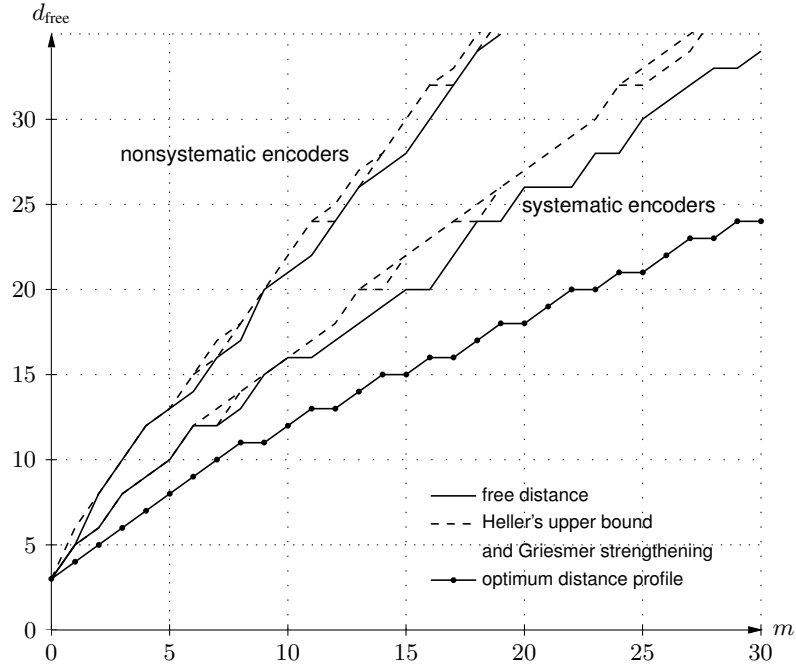


Figure 10.8 The free distances for rate $R = 1/3$ systematic and nonsystematic ODP convolutional codes and comparisons with Heller's and Griesmer's upper bounds.

10.5 HIGH RATE CONVOLUTIONAL CODES

In Table 10.15 we list rate $R = 2/3$ systematic, polynomial, ODP convolutional encoding matrices

$$G = \begin{pmatrix} 1 & 0 & g_{13} \\ 0 & 1 & g_{23} \end{pmatrix} \tag{10.17}$$

Rate $R = 2/3$ nonsystematic, polynomial ODP convolutional encoding matrices

$$G = \begin{pmatrix} g_{11} & g_{12} & g_{13} \\ g_{21} & g_{22} & g_{23} \end{pmatrix} \tag{10.18}$$

are listed in Table 10.16.

In Section 2.9 (Theorem 2.6) we proved that if an encoding matrix $G(D)$ and the encoding matrix $H(D)$ of the convolutional dual code both are minimal-basic, then they have the same overall constraint length ν. Furthermore, in Section 2.10 we showed how to use $H(D)$ to build systematic encoders with feedback of memory $\nu = m$. Thus, it is natural and quite useful when we search for encoding matrices to extend the concept of distance profile to $\boldsymbol{d}_\nu^{\text{p}} = (d_0^{\text{c}}, d_1^{\text{c}}, \ldots, d_\nu^{\text{c}})$. The encoding matrices in Table 10.17 are ODP in this extended sense [JoP78]. In Table 10.18 we

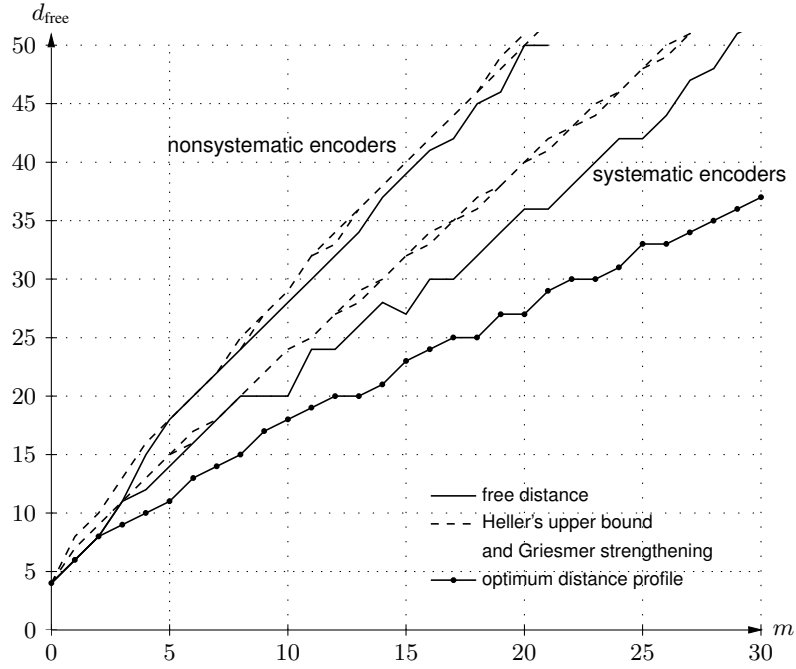


Figure 10.9 The free distances for rate $R = 1/4$ systematic and nonsystematic ODP convolutional codes and comparisons with Heller's and Griesmer's upper bounds.

give the encoding matrices for the convolutional duals of the codes encoded by the encoding matrices in Table 10.17.

10.6 TAILBITING TRELLIS ENCODERS

Tailbiting encoders can generate many of the most powerful binary block codes. Extensive lists of the best tailbiting encoding matrices can be found in [SAJ99, BJK02, BHJ02, BHJ04].

10.7 COMMENTS

Our search for good convolutional encoding matrices started with the introduction of the distance profile, which could be used to give an essential reduction of the set of potential candidates that we had to investigate [Joh75]. It turned out that for short rate $R = 1/2$ encoding matrices the ODP condition could be imposed at no cost in free distance. Additional ODP encoding matrices were reported in [Joh76, Joh77a, JoP78].

Table 10.15 Viterbi spectra $n_{d_{\text{free}}+i}$, $i = 0, 1, \ldots, 5$, for systematic rate $R = 2/3$ ODP encoding matrices.

				i					
m	g_{13}	g_{23}	d_{free}	0	1	2	3	4	5
1	6	6	2	1	2	5	12	25	54
2	5	7	3	2	5	15	45	127	370
3	54	64	4	7	0	76	0	820	0
4	56	62	4	2	10	22	92	343	1109
5	57	63	5	6	20	55	207	799	2896
6	554	704	5	2	16	46	126	554	2144
7	664	742	6	24	0	237	0	3608	0
8	665	743	6	5	19	52	208	789	3008
9	5504	7064	6	1	12	50	175	605	2346
10	5736	6322	7	8	24	84	323	1211	4718
11	5736	6323	8	44	0	505	0	7586	0
12	66414	74334	8	16	0	260	0	3668	0
13	57372	63226	8	3	27	82	318	1099	4254
14	55175	70223	8	2	14	57	184	705	2821
15	664074	743344	8	1	9	36	122	477	1760
16	664072	743346	10	40	0	588	0	8203	0
17	573713	632255	10	15	52	170	695	2685	10131
18	6641054	7431164	10	18	0	249	0	3638	0
19	5514632	7023726	10	2	19	74	297	1029	
20	5514633	7023725	11	8	46	196	696		

With the development of the FAST algorithm we achieved a huge improvement in the speed at which we could determine the Viterbi spectra for convolutional encoders. Together with extensive tables of rate $R = 1/2$ convolutional encoding matrices e the FAST algorithm was reported in [CeJ89].

The BEAST algorithm was first presented at the Allerton Conf. [BHJ01]. Later it appeared in *IEEE Trans. Inform. Theory* [BHJ04].

The tables of encoding matrices in the first edition were exended by Florian Hug [Hug12], who used BEAST and obtained remarkable improvements in the search for new encoding matrices compared to what could be obtained by FAST.

Table 10.16 Viterbi spectra $n_{d_{\text{free}}+i}$, $i = 0, 1, \ldots, 5$, for nonsystematic rate $R = 2/3$ ODP encoding matrices.

									i					
m	ν	g_{11}	g_{12}	g_{13}	g_{21}	g_{22}	g_{23}	d_{free}	0	1	2	3	4	5
1	2	2	6	6	4	2	6	3	1	4	14	40	116	339
2	3	6	2	4	1	4	7	4	1	5	24	71	239	871
2	4	3	4	5	4	3	7	5	2	13	45	143	534	2014
3	5	60	70	30	64	00	74	6	6	27	70	285	1103	4063
3	6	30	64	64	54	10	54	7	17	53	133	569	2327	8624
4	7	54	30	54	64	62	12	8	41	0	528	0	7497	0
4	8	46	50	26	64	32	52	8	6	43	154	493	1860	7406
5	9	34	46	56	41	17	46	9	17	81	228	933	3469	13203
5	10	43	46	33	55	25	76	10	69	0	925	0	13189	0

Table 10.17 Viterbi spectra $n_{d_{\text{free}}+i}$, $i = 0, 1, \ldots, 5$, for nonsystematic rate $R = 2/3$ ODP encoding matrices. These encoding matrices are ODP in the $\boldsymbol{d}_\nu^{\text{p}}$ sense.

										i				
ν	g_{11}	g_{12}	g_{13}	g_{21}	g_{22}	g_{23}	d_ν	$\sharp d_\nu$-paths	d_{free}	0	1	2	3	4	5
3	6	2	4	1	4	7	3	2	4	1	5	24	71	239	871
4	6	3	7	1	5	5	4	17	5	7	23	59	240	912	2986
5	60	30	70	34	74	40	4	7	6	9	19	80	276	1122	4121
6	50	24	54	24	70	54	4	2	6	1	17	47	145	613	2269
7	54	30	64	00	46	66	5	30	7	6	26	105	378	1320	5089
8	64	12	52	26	66	44	5	15	8	8	40	157	598	1987	7761
9	54	16	66	25	71	60	5	9	8	1	20	75	243	904	3522
10	53	23	51	36	53	67	6	54	9	9	45	166	593	2353	8879
11	710	260	670	320	404	714	6	29	10	29	0	473	0	6711	0
12	740	260	520	367	414	515	6	27	10	4	34	127	450	1657	6565
13	710	260	670	140	545	533	6	5	11	9	72	222	824	3192	12457
14	676	046	704	256	470	442	7	65	12	58	0	847	0	12416	0
15	722	054	642	302	457	435	7	38	12	25	0	462	0	6583	0
16	7640	2460	7560	0724	5164	4260	7	14	12	7	26	120	437	1695	6470
17	5330	3250	5340	0600	7650	5434	7	7	13	18	73	247	983	3861	14616
18	6734	1734	4330	1574	5140	7014	8	106	14	50	0	764	0	11499	0
19	5044	3570	4734	1024	5712	5622	8	43	14	24	0	425	0	5801	0
20	7030	3452	7566	0012	6756	5100	8	23	14	15	0	185	0	2837	0
21	6562	2316	4160	0431	4454	7225	8	11	15	14	59	187	780	3004	11534
22	57720	12140	63260	15244	70044	47730	9	144	16	50	0	740	0	10665	0
23	51630	25240	42050	05460	61234	44334	9	60	16	10	53	181	661	2555	9981

Table 10.18 Encoding matrices $H = (h_1\, h_2\, h_3)$ for the convolutional dual of the codes encoded by the encoding matrices in Table 10.17.

ν	h_1	h_2	h_3
3	74	54	64
4	50	62	72
5	65	45	53
6	424	644	764
7	472	752	532
8	635	403	571
9	5014	4634	6664
10	7164	4136	5416
11	5755	7767	6601
12	70414	52464	60244
13	56502	76346	67772
14	71433	53241	61175
15	660004	575734	776554
16	461656	700006	630732
17	544463	433501	615256
18	4114444	5433454	7152024
19	6171512	5475256	4301002
20	7500021	6742327	4162245
21	72164254	45126324	61662214
22	55422416	42035332	60362506
23	45416327	51203765	76300111

APPENDIX A

MINIMAL ENCODERS

In this appendix we show how to obtain the minimal encoders in Examples 2.22 and 2.37. (The readers who would like to learn more about realizations of sequential circuits are referred to standard textbooks, for example, [Lee78, Gil66].)

■ **EXAMPLE A.1**

In Example 2.21 we showed that since the encoding matrix

$$G'(D) = \begin{pmatrix} 1+D & D & 1 \\ 1+D^2+D^3 & 1+D+D^2+D^3 & 0 \end{pmatrix} \tag{A.1}$$

is minimal and $\mu = 3$ there exists a minimal realization with only three memory elements. In order to obtain such a realization we introduce the state variables shown in Fig. A.1.

Fundamentals of Convolutional Coding, 2. Edition. By Rolf Johannesson, Kamil Sh. Zigangirov

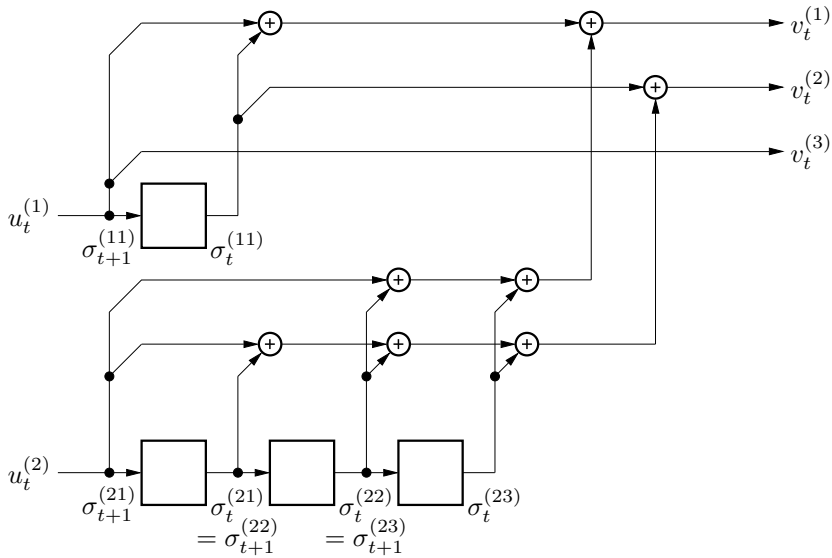


Figure A.1 The controller canonical form of the encoding matrix $G'(D)$ in Example A.1.

First we notice that the input of a memory element whose output is $\sigma_t^{(ij)}$ is $\sigma_{t+1}^{(ij)}$. Then we have

$$\begin{aligned}
\sigma_{t+1}^{(11)} &= u_t^{(1)} \\
\sigma_{t+1}^{(21)} &= u_t^{(2)} \\
\sigma_{t+1}^{(22)} &= \sigma_t^{(21)} \\
\sigma_{t+1}^{(23)} &= \sigma_t^{(22)} \\
v_t^{(1)} &= \sigma_t^{(11)} + \sigma_t^{(22)} + \sigma_t^{(23)} + u_t^{(1)} + u_t^{(2)} \\
v_t^{(2)} &= \sigma_t^{(11)} + \sigma_t^{(21)} + \sigma_t^{(22)} + \sigma_t^{(23)} + u_t^{(2)} \\
v_t^{(3)} &= u_t^{(1)}
\end{aligned} \tag{A.2}$$

For simplicity we use hexadecimal notations for the state-4-tuples, $\left(\sigma_t^{(11)}\ \sigma_t^{(21)}\ \sigma_t^{(22)}\ \sigma_t^{(23)}\right)$. Then, from (A.2) we obtain Table A.1 for the behavior of the encoder in Fig. A.1.

In order to find which states are equivalent we start by merging those states that correspond to the same outputs. Hence, we obtain partition P1:

$$\text{P1:}\quad \{0,3,9,A\},\{1,2,8,B\},\{4,7,D,E\},\{5,6,C,F\}$$

The states that belong to the same set in P1 are said to be 1-equivalent.

Two states are 2-equivalent if they are 1-equivalent and their successor states are 1-equivalent. For example, the states 0 and 3 are not 2-equivalent since their successor states are not 1-equivalent. From Table A.1 and P1 we obtain

$$\text{P2:}\quad \{0,9\},\{3,A\},\{1,8\},\{2,B\},\{4,D\},\{7,E\},\{5,C\},\{6,F\}$$

Table A.1 Successor state and output $\left(\sigma_{t+1}^{(11)}\,\sigma_{t+1}^{(21)}\,\sigma_{t+1}^{(22)}\,\sigma_{t+1}^{(23)}/v_t^{(1)}\,v_t^{(2)}\,v_t^{(3)}\right)$ as a function of the present state $\left(\sigma_t^{(11)}\,\sigma_t^{(21)}\,\sigma_t^{(22)}\,\sigma_t^{(23)}\right)$ and the input $\left(u_t^{(1)}\,u_t^{(2)}\right)$.

Present state	Input 00	01	10	11
0	0/0	4/6	8/5	C/3
1	0/6	4/0	8/3	C/5
2	1/6	5/0	9/3	D/5
3	1/0	5/6	9/5	D/3
4	2/2	6/4	A/7	E/1
5	2/4	6/2	A/1	E/7
6	3/4	7/2	B/1	F/7
7	3/2	7/4	B/7	F/1
8	0/6	4/0	8/3	C/5
9	0/0	4/6	8/5	C/3
A	1/0	5/6	9/5	D/3
B	1/6	5/0	9/3	D/5
C	2/4	6/2	A/1	E/7
D	2/2	6/4	A/7	E/1
E	3/2	7/4	B/7	F/1
F	3/4	7/2	B/1	F/7

The states that belong to the same set in P2 are 2-equivalent.

Two states are 3-equivalent if they are 2-equivalent and their successor states are 2-equivalent. Hence, we see that in this example P3 = P2. Thus, we can stop the procedure and the eight sets in P2 represent the eight states of the minimal encoder. (In general we proceed until $P(k+1) = P(k)$.)

We let the first octal digit in each set represent the states $\left(\sigma_t^{(1)}\,\sigma_t^{(2)}\,\sigma_t^{(3)}\right)$ of the minimal encoder. (In order to obtain a linear realization of the minimal encoder we must let (000) represent the state $\{0, 9\}$!) Then we obtain Table A.2.

From Table A.2 we obtain the following table for $\sigma_{t+1}^{(1)}$:

Table A.2 Successor state and output $\left(\sigma_{t+1}^{(1)}\,\sigma_{t+1}^{(2)}\,\sigma_{t+1}^{(3)}/v_t^{(1)}\,v_t^{(2)}\,v_t^{(3)}\right)$ as a function of the present state $\left(\sigma_t^{(1)}\,\sigma_t^{(2)}\,\sigma_t^{(3)}\right)$ and the input $\left(u_t^{(1)}\,u_t^{(2)}\right)$.

Present state	Input 00	01	10	11
0	0/0	4/6	1/5	5/3
1	0/6	4/0	1/3	5/5
2	1/6	5/0	0/3	4/5
3	1/0	5/6	0/5	4/3
4	2/2	6/4	3/7	7/1
5	2/4	6/2	3/1	7/7
6	3/4	7/2	2/1	6/7
7	3/2	7/4	2/7	6/1

$\sigma_t^{(1)}\,\sigma_t^{(2)}\,\sigma_t^{(3)}$	$u_t^{(1)}\,u_t^{(2)}$ 00	01	10	11
000	0	1	0	1
001	0	1	0	1
010	0	1	0	1
011	0	1	0	1
100	0	1	0	1
101	0	1	0	1
110	0	1	0	1
111	0	1	0	1

Clearly,

$$\sigma_{t+1}^{(1)} = u_t^{(2)} \tag{A.3}$$

For $\sigma_{t+1}^{(2)}$ we have:

$\sigma_t^{(1)}\,\sigma_t^{(2)}\,\sigma_t^{(3)}$	$u_t^{(1)}\,u_t^{(2)}$ 00	01	10	11
000	0	0	0	0
001	0	0	0	0
010	0	0	0	0
011	0	0	0	0
100	1	1	1	1
101	1	1	1	1
110	1	1	1	1
111	1	1	1	1

Thus,

$$\sigma_{t+1}^{(2)} = \sigma_t^{(1)} \tag{A.4}$$

For $\sigma_{t+1}^{(3)}$ we have:

$\sigma_t^{(1)}\,\sigma_t^{(2)}\,\sigma_t^{(3)}$	$u_t^{(1)}\,u_t^{(2)}$			
	00	01	10	11
000	0	0	1	1
001	0	0	1	1
010	1	1	0	0
011	1	1	0	0
100	0	0	1	1
101	0	0	1	1
110	1	1	0	0
111	1	1	0	0

It is easily seen that

$$\sigma_{t+1}^{(3)} = \sigma_t^{(2)} + u_t^{(1)} \tag{A.5}$$

Repeating this procedure for each of the outputs we obtain

$$\begin{aligned} v_t^{(1)} &= \sigma_t^{(2)} + \sigma_t^{(3)} + u_t^{(1)} + u_t^{(2)} \\ v_t^{(2)} &= \sigma_t^{(1)} + \sigma_t^{(2)} + \sigma_t^{(3)} + u_t^{(2)} \\ v_t^{(3)} &= u_t^{(1)} \end{aligned} \tag{A.6}$$

The minimal encoder is shown in Fig. 2.16.

■ **EXAMPLE A.2**

In Example 2.37 we showed that

$$G_{\text{sys}}(D) = \begin{pmatrix} 1 & 0 & \dfrac{1+D^2}{1+D+D^2} & \dfrac{D^2}{1+D+D^2} \\ 0 & 1 & \dfrac{D^2}{1+D+D^2} & \dfrac{1}{1+D+D^2} \end{pmatrix} \tag{A.7}$$

is equivalent to a minimal-basic encoding matrix with $\mu = \nu = 2$. Since $G_{\text{sys}}(D)$ is systematic it is minimal and, hence, it is also realizable with two memory elements, albeit not on controller canonical form.

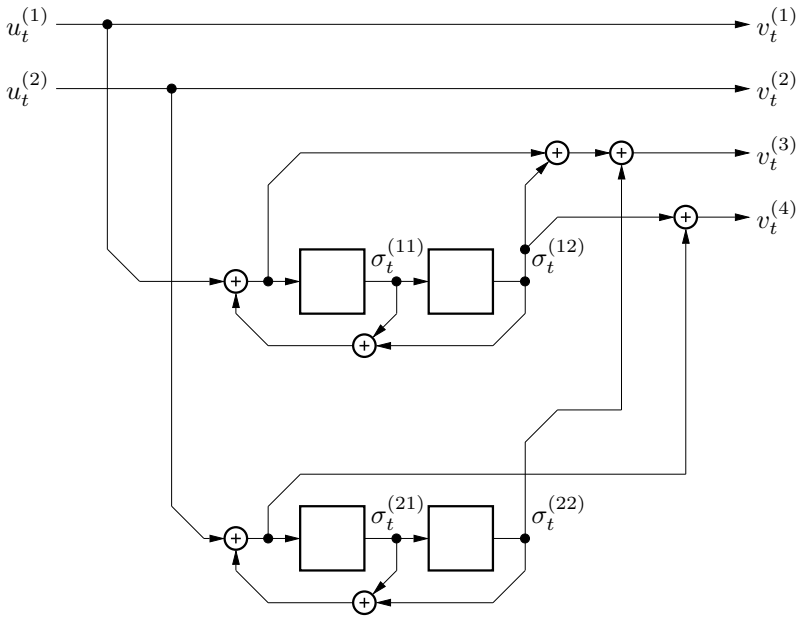


Figure A.2 A realization of the systematic encoding matrix in controller canonical form.

Its realization in controller canonical form uses four memory elements and is shown in Fig. A.2. From the figure follows immediately that

$$\begin{aligned}
\sigma_{t+1}^{(11)} &= \sigma_t^{(11)} + \sigma_t^{(12)} + u_t^{(1)} \\
\sigma_{t+1}^{(12)} &= \sigma_t^{(11)} \\
\sigma_{t+1}^{(21)} &= \sigma_t^{(21)} + \sigma_t^{(22)} + u_t^{(2)} \\
\sigma_{t+1}^{(22)} &= \sigma_t^{(21)} \\
v_t^{(1)} &= u_t^{(1)} \\
v_t^{(2)} &= u_t^{(2)} \\
v_t^{(3)} &= \sigma_t^{(11)} + \sigma_t^{(22)} + u_t^{(1)} \\
v_t^{(4)} &= \sigma_t^{(12)} + \sigma_t^{(21)} + \sigma_t^{(22)} + u_t^{(2)}
\end{aligned} \tag{A.8}$$

Clearly we can ignore the outputs $v_t^{(1)}$ and $v_t^{(2)}$ when we minimize the encoder. By repeating the state minimization procedure described in Example A.1 we obtain the following table for the successor state and output $\left(\sigma_{t+1}^{(1)}\, \sigma_{t+1}^{(2)} / v_t^{(3)}\, v_t^{(4)}\right)$ as a function of the present state and present input:

$\sigma_t^{(1)}\,\sigma_t^{(2)}$	$u_t^{(1)}\,u_t^{(2)}$			
	00	01	10	11
00	00/00	01/01	10/10	11/11
01	10/01	11/00	00/11	01/10
10	11/10	10/11	01/00	00/01
11	01/11	00/10	11/01	10/00

From the table we obtain

$$\begin{aligned} \sigma_{t+1}^{(1)} &= \sigma_t^{(1)} + \sigma_t^{(2)} + u_t^{(1)} \\ \sigma_{t+1}^{(2)} &= \sigma_t^{(1)} + u_t^{(2)} \\ v_t^{(3)} &= \sigma_t^{(1)} + u_t^{(1)} \\ v_t^{(4)} &= \sigma_t^{(2)} + u_t^{(2)} \end{aligned} \tag{A.9}$$

The minimal realization is shown in Fig. 2.23.

Finally, we remark that different mappings between the states and their binary representations will give different realizations that in general are nonlinear. However, there exist always mappings such that the encoders can be realized by linear sequential circuits consisting of only memory elements and modulo 2 adders. For the systematic encoder in Example A.2 there exist six different linear realizations. Two of them require five adders, three six adders, and one as many as eight adders.

APPENDIX B

WALD'S IDENTITY

Let a random walk be represented by the sequence of random variables $S_0 = 0, S_1, S_2, \ldots$, where

$$S_j = \sum_{i=0}^{j-1} Z_i \tag{B.1}$$

for $j \geq 1$ and where the Z_i's are independent, identically distributed random variables such that

$$P(Z_i > 0) > 0 \tag{B.2}$$

and

$$P(Z_i < 0) > 0 \tag{B.3}$$

We introduce an absorbing barrier at $u < 0$ such that the random walk will stop whenever it crosses the barrier from above, that is, as soon as $S_j < u$.

Let

$$g(\lambda) = E[2^{\lambda Z_i}] \tag{B.4}$$

be the moment-generating function of Z_i. From (B.2) and (B.3) it follows that $g(\lambda) \to \infty$ both when $\lambda \to \infty$ and when $\lambda \to -\infty$. Furthermore,

$$g(0) = 1 \tag{B.5}$$

Fundamentals of Convolutional Coding, 2. Edition. By Rolf Johannesson, Kamil Sh. Zigangirov

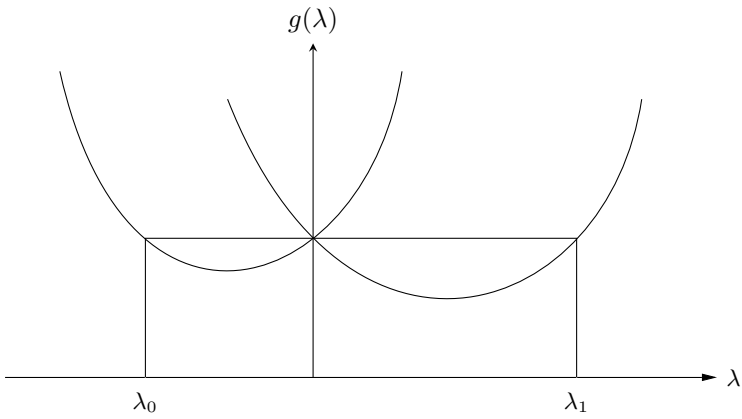


Figure B.1 Typical $g(\lambda) = E[2^{\lambda Z_i}]$ when $E[Z_i] > 0$ (left) and $E[Z_i] < 0$ (right).

and

$$g'(\lambda) \mid_{\lambda=0} = E[Z_i] \ln 2 \tag{B.6}$$

In Fig. B.1 we show the typical shapes of the moment-generating function $g(\lambda)$. The curve to the left corresponds to $E[Z_i] > 0$ and the one to the right to $E[Z_i] < 0$.

We shall now consider the real roots of the equation

$$g(\lambda) = 1 \tag{B.7}$$

From (B.5) it follows that $\lambda = 0$ is always a root. From (B.6) we conclude that for $E[Z_i] > 0$ we have at least one negative root $\lambda_0 < 0$ and for $E[Z_i] < 0$ at least one positive root $\lambda_1 > 0$.

In general we have

$$g(\lambda) = g_0 \tag{B.8}$$

where $0 < g_0 < 1$. This equation has at least two roots if

$$\min_{\lambda}\{g(\lambda)\} < g_0 \tag{B.9}$$

but only one root if

$$\min_{\lambda}\{g(\lambda)\} = g_0 \tag{B.10}$$

■ **EXAMPLE B.1**

Let

$$Z_i = \sum_{s=1}^{c} Y_{is} \tag{B.11}$$

where the Y_{is}'s are independent, identically distributed random variables,

$$Y_{is} = \begin{cases} \alpha > 0 & \text{with probability} \quad 1-\epsilon \\ \beta < 0 & \text{with probability} \quad \epsilon \end{cases} \tag{B.12}$$

and

$$P(Z_i = k\alpha + (c-k)\beta) = \binom{c}{k}(1-\epsilon)^k \epsilon^{c-k} \tag{B.13}$$

From (B.13) we have

$$\begin{aligned} g(\lambda) = E[2^{\lambda Z_i}] &= \prod_{l=1}^{c} E[2^{\lambda Y_{il}}] \\ &= \left((1-\epsilon)2^{\lambda\alpha} + \epsilon 2^{\lambda\beta}\right)^c \end{aligned} \tag{B.14}$$

The real root of the equation $g(\lambda) = 1$ is equal to the real root of

$$f(\lambda) = (1-\epsilon)2^{\lambda\alpha} + \epsilon 2^{\lambda\beta} = 1 \tag{B.15}$$

Let

$$\begin{aligned} \alpha &= (1-\epsilon)/(1-a) \\ \beta &= \epsilon/a \end{aligned} \tag{B.16}$$

where $0 < a < 1$. Then it is easily verified that $\lambda_0 = -1$.

Next we let

$$\begin{aligned} \alpha &= \log(1-\epsilon) + 1 - B \\ \beta &= \log\epsilon + 1 - B \end{aligned} \tag{B.17}$$

where we assume that

$$f'(0) = (1-\epsilon)\alpha + \epsilon\beta = 1 - h(\epsilon) - B > 0 \tag{B.18}$$

or, equivalently, that

$$1 - h(\epsilon) > B \tag{B.19}$$

where

$$h(x) = -x\log x - (1-x)\log(1-x) \tag{B.20}$$

is the binary entropy function. Then there exists a negative root $\lambda_0 < 0$ of the equation $g(\lambda) = 1$.

Inserting (B.17) into (B.15) yields

$$(1-\epsilon)^{1+\lambda_0} + \epsilon^{1+\lambda_0} = 2^{\lambda_0(B-1)} \tag{B.21}$$

or, equivalently,

$$B = 1 + \frac{1}{\lambda_0}\log\left((1-\epsilon)^{1+\lambda_0} + \epsilon^{1+\lambda_0}\right) \tag{B.22}$$

Let

$$\lambda_0 = -\frac{s}{1+s} \tag{B.23}$$

where s is the solution of

$$R = G(s)/s \tag{B.24}$$

and $G(s)$ is the Gallager function for the BSC (5.98). Then we obtain from (B.22) the important equality

$$B = R \tag{B.25}$$

■ **EXAMPLE B.2**

Consider Z_i given by (B.11), but in this example we let

$$Y_{is} = \begin{cases} \alpha > 0 & \text{with probability} \quad 1/2 \\ \beta < 0 & \text{with probability} \quad 1/2 \end{cases} \tag{B.26}$$

that is,

$$P(Z_i = k\alpha + (c-k)\beta) = \binom{c}{k} \left(\frac{1}{2}\right)^c \tag{B.27}$$

and

$$\begin{aligned} g(\lambda) &= \sum_{k=0}^{c} \binom{c}{k} \left(\frac{1}{2}\right)^c 2^{\lambda(k\alpha+(c-k)\beta)} \\ &= \left(\frac{1}{2}2^{\lambda\alpha} + \frac{1}{2}2^{\lambda\beta}\right)^c \end{aligned} \tag{B.28}$$

The real root of the equation

$$g(\lambda) = g_0 \tag{B.29}$$

is equal to the root of

$$f(\lambda) = \frac{1}{2}2^{\lambda\alpha} + \frac{1}{2}2^{\lambda\beta} = g_0^{1/c} \tag{B.30}$$

Let

$$g_0 = 2^{-b} \tag{B.31}$$

or, equivalently,

$$g_0^{1/c} = 2^{-R} \tag{B.32}$$

and let

$$\begin{aligned} \alpha &= 1-\rho \\ \beta &= -\rho \end{aligned} \tag{B.33}$$

where ρ is the Gilbert-Varshamov parameter, that is,

$$\rho = h^{-1}(1-R) \tag{B.34}$$

Then the function

$$f(\lambda) = \frac{1}{2}2^{\lambda\alpha} + \frac{1}{2}2^{\lambda\beta} \tag{B.35}$$

has its minimum for $\lambda_{\min} < 0$ such that

$$f'(\lambda_{\min}) = \left(\frac{1}{2}\alpha 2^{\lambda_{\min}\alpha} + \frac{1}{2}\beta 2^{\lambda_{\min}\beta}\right)\ln 2 = 0 \tag{B.36}$$

or, equivalently, for

$$\lambda_{\min} = \log\frac{\rho}{1-\rho} \tag{B.37}$$

Inserting (B.37) into (B.30) yields

$$\begin{aligned} f(\lambda_{\min}) &= \frac{1}{2}2^{(1-\rho)\log\frac{\rho}{1-\rho}} + \frac{1}{2}2^{-\rho\log\frac{\rho}{1-\rho}} \\ &= \frac{1}{2}\left(\left(\frac{\rho}{1-\rho}\right)^{1-\rho} + \left(\frac{\rho}{1-\rho}\right)^{-\rho}\right) = 2^{h(\rho)-1} = 2^{-R} \end{aligned} \tag{B.38}$$

that is, equation (B.30) has only one root, viz., $\lambda_{\min}$.

Consider next the case when α and β are given by (B.17). Then, for $B = R$ we have

$$\begin{aligned} f'(\lambda)\mid_{\lambda=0} &= \frac{1}{2}(\alpha+\beta)\ln 2 \\ &= \left(-R + \log\left(2\sqrt{\epsilon(1-\epsilon)}\right)\right)\ln 2 < 0 \end{aligned} \tag{B.39}$$

and there exists a $\lambda_1 > 0$ such that

$$f(\lambda_1) = 2^{-R} \tag{B.40}$$

In fact, from (B.30) it follows that

$$\left(\frac{1}{2}(1-\epsilon)^{\lambda_1} + \frac{1}{2}\epsilon^{\lambda_1}\right)2^{\lambda_1(1-R)} = 2^{-R} \tag{B.41}$$

or, equivalently, that

$$R = 1 - \frac{1}{1-\lambda_1}\log\left((1-\epsilon)^{\lambda_1} + \epsilon^{\lambda_1}\right) \tag{B.42}$$

If we write λ_1 as

$$\lambda_1 = \frac{1}{1+s} \tag{B.43}$$

then it follows from (B.42) that s satisfies (B.24).

Let the random variable N denote the time at which the random walk first crosses the threshold $u < 0$, that is,

$$P(N = n) = P(S_j \geq u,\ \ 0 \leq j < n,\ \ \&\ \ S_n < u) \tag{B.44}$$

Then we have the following:

Lemma B.1 If $E[Z_i] < 0$, then the random walk will eventually cross the threshold $u < 0$, that is,

$$\lim_{n\to\infty} P(N \geq n) = 0 \tag{B.45}$$

Proof: Clearly,

$$P(N \geq n) \leq P(S_n \geq u) \tag{B.46}$$

From (B.1) it follows that

$$E[2^{\lambda S_n}] = \prod_{j=0}^{n-1} E[2^{\lambda Z_j}] = g(\lambda)^n \tag{B.47}$$

and for $\lambda > 0$ we have

$$\begin{aligned} E[2^{\lambda S_n}] &= \sum_{\text{all } s} 2^{\lambda s} P(S_n = s) \\ &\geq \sum_{s \geq u} 2^{\lambda s} P(S_n = s) \geq 2^{\lambda u} P(S_n \geq u) \end{aligned} \tag{B.48}$$

Then, by combining (B.46), (B.47), and (B.48) we obtain

$$P(N \geq n) \leq 2^{-\lambda u} E[2^{\lambda S_n}] \leq 2^{-\lambda u} g(\lambda)^n \tag{B.49}$$

From the assumption $E[Z_i] < 0$ it follows that there exists a $\lambda > 0$ such that $g(\lambda) < 1$. Then, from (B.49) we conclude that (B.45) holds. ∎

Now we consider the general case when $E[Z_i]$ can be not only negative but also positive.

When $E[Z_i] > 0$ we change the probability assignment of the random walk in such a way that the new random walk will have a negative drift and hence will be absorbed with probability 1.

We introduce the "tilted" probability assignment

$$q_{Z,\lambda}(z) = \frac{f_Z(z) 2^{\lambda z}}{g(\lambda)} \tag{B.50}$$

where $f_Z(z)$ is the probability assignment for the original random walk. We notice that

$$q_{Z,\lambda}(z) \geq 0, \quad \text{all } z \tag{B.51}$$

and

$$\begin{aligned} \sum_z q_{Z,\lambda}(z) &= \frac{1}{g(\lambda)} \sum_z f_Z(z) 2^{\lambda z} \\ &= \frac{1}{g(\lambda)} E[2^{\lambda Z}] = 1 \end{aligned} \tag{B.52}$$

Let us introduce the corresponding random walk $S_{0,\lambda} = 0, S_{1,\lambda}, S_{2,\lambda} \ldots$, where

$$S_{j,\lambda} = \sum_{i=0}^{j-1} Z_{i,\lambda} \tag{B.53}$$

for $j \geq 1$, where the $Z_{i,\lambda}$'s are independent, identically distributed random variables.

We find that

$$\begin{aligned} E[Z_{i,\lambda}] &= \sum_z q_{Z,\lambda}(z) z \\ &= \frac{1}{g(\lambda)} \sum_z f_Z(z) z 2^{\lambda z} \\ &= \frac{1}{g(\lambda) \ln 2} \frac{d}{d\lambda} \sum_z f_Z(z) 2^{\lambda z} \\ &= \frac{1}{g(\lambda) \ln 2} \frac{dg(\lambda)}{d\lambda} \end{aligned} \tag{B.54}$$

Thus by choosing λ such that $g'(\lambda) < 0$ we see that $E[Z_{i,\lambda}] < 0$ and our tilted random walk has a negative drift.

Let $f_{\lambda,n}(u, v)$ denote the probability that the tilted random walk is not absorbed by the barrier at $u < 0$ at time $j < n$ and assumes the value v at time n, that is,

$$f_{\lambda,n}(u, v) \stackrel{\text{def}}{=} P(S_{j,\lambda} \geq u, \;\; 0 \leq j < n, \;\; S_{n,\lambda} = v) \tag{B.55}$$

For the tilted random walk, (B.44) can be rewritten as

$$P(N = n) = \sum_{v<u} f_{\lambda,n}(u, v) \tag{B.56}$$

If we choose λ such that the drift is negative, then it follows from Lemma B.1 that the tilted random walk will eventually achieve a value less than u with probability 1, that is,

$$\sum_{n=1}^{\infty} \sum_{v<u} f_{\lambda,n}(u, v) = 1 \tag{B.57}$$

From (B.50) it follows that

$$\prod_{i=0}^{n-1} P(Z_{i,\lambda} = a_i) = \prod_{i=0}^{n-1} P(Z_i = a_i) 2^{\lambda a_i} g(\lambda)^{-1} \tag{B.58}$$

Hence we have

$$f_{\lambda,n}(u, v) = f_{0,n}(u, v) 2^{\lambda v} g(\lambda)^{-n} \tag{B.59}$$

where $\sum_{i=0}^{n-1} a_i = v$. Combining (B.57) and (B.59) we obtain

$$\sum_{n=1}^{\infty} \sum_{v<u} f_{0,n}(u, v) 2^{\lambda v} g(\lambda)^{-n} = 1 \tag{B.60}$$

which is known as *Wald's identity* for a random walk with one absorbing barrier at $u < 0$ [Wal47].

Wald's identity can also be written in a more compact form:

$$E[2^{\lambda S_N} g(\lambda)^{-N}] = 1 \tag{B.61}$$

and we have the following:

Theorem B.2 Let S_j be the random walk given by (B.1) and let $g(\lambda)$ be the moment-generating function of the random variable Z_i which satisfies (B.2) and (B.3). Let N be the smallest n for which $S_n < u$, where the barrier $u < 0$. Then, for all λ such that $g'(\lambda) < 0$ Wald's identity (B.61) holds.

Corollary B.3 Let S_j be the random walk given by (B.1) with $E[Z_i] > 0$ and let $g(\lambda)$ be the moment-generating function of the random variable Z_i which satisfies (B.2) and (B.3). Let $\lambda_0 < 0$ be a root of equation (B.7) such that $g'(\lambda_0) < 0$. Then,

$$P(S_{\min} < u) \leq 2^{-\lambda_0 u} \tag{B.62}$$

where

$$S_{\min} = \min_j \{S_j\} \tag{B.63}$$

Proof: From (B.60) we have

$$\begin{aligned} 1 &= \sum_{n=1}^{\infty} \sum_{v<u} f_{0,n}(u,v) 2^{\lambda_0 v} g(\lambda_0)^{-n} \\ &\geq 2^{\lambda_0 u} \sum_{n=1}^{\infty} \sum_{v<u} f_{0,n}(u,v) = 2^{\lambda_0 u} P(S_{\min} < u) \end{aligned} \tag{B.64}$$

■

In a more general case we would like to upper-bound

$$\sum_{n=1}^{\infty} \sum_{v<u} f_{0,n}(u,v) 2^{bn} \tag{B.65}$$

Then we choose $\lambda = \lambda_1 > 0$ such that

$$g(\lambda_1) = 2^{-b} \tag{B.66}$$

Assume that $g'(\lambda_1) < 0$. Hence we can use Wald's identity (B.60) to obtain

$$\begin{aligned} 1 &= \sum_{n=1}^{\infty} \sum_{v<u} f_{0,n}(u,v) 2^{\lambda_1 v} 2^{bn} \\ &\leq 2^{\lambda_1 u} \sum_{n=1}^{\infty} \sum_{v<u} f_{0,n}(u,v) 2^{bn} \end{aligned} \tag{B.67}$$

or, equivalently,

$$\sum_{n=1}^{\infty}\sum_{v<u} f_{0,n}(u,v)2^{bn} \geq 2^{-\lambda_1 u} \tag{B.68}$$

Suppose that

$$\min\{Z_i\} = z_{\min} < 0 \tag{B.69}$$

Then,

$$S_N > u + z_{\min} \tag{B.70}$$

where $z_{\min}$ is called the maximal overshot, and we obtain from (B.60) that

$$1 \geq 2^{\lambda_1(u+z_{\min})}\sum_{n=1}^{\infty}\sum_{v<u} f_{0,n}(u,v)2^{bn} \tag{B.71}$$

or, equivalently,

$$2^{-\lambda_1(u+z_{\min})} \geq \sum_{n=1}^{\infty}\sum_{v<u} f_{0,n}(u,v)2^{bn} \geq 2^{-\lambda_1 u} \tag{B.72}$$

where the last inequality follows from (B.68).

Notice that (B.68) is still valid if $g'(\lambda_1) = 0$ for $\lambda_1 > 0$. Then we have to replace b in (B.66) by $b - \varepsilon$, where $\varepsilon > 0$. Let $\lambda_1(\varepsilon)$ be the root of

$$g(\lambda) = 2^{-b+\varepsilon} \tag{B.73}$$

then $\lambda_1(\varepsilon) < \lambda_1$ and $g'(\lambda_1(\varepsilon)) < 0$. Hence we use Wald's identity and obtain

$$2^{-\lambda_1(u+z_{\min})} \geq \sum_{n=1}^{\infty}\sum_{v<u} f_{0,n}(u,v)2^{(b-\varepsilon)n} \geq 2^{-\lambda_1(\varepsilon)u} \tag{B.74}$$

When $\varepsilon \to 0$ (B.74) will approach (B.68).

Corollary B.4 Let S_j be the random walk given by (B.1) with the absorbing rule $S_j < u$, where the barrier $u < 0$, and let $g(\lambda)$ be the moment-generating function of the random variable Z_i which satisfies (B.2) and (B.3) and whose minimum value $z_{\min} = \min\{Z_i\} < 0$. Let $\lambda_1 > 0$ be a root of equation (B.63) such that $g'(\lambda_1) \leq 0$. Then, inequality (B.72) holds.

Corollary B.5 (Wald's equality) Let S_j be the random walk given by (B.1) and assume that the random variable Z_i satisfies (B.2) and (B.3) and has $E[Z_i] < 0$. Then $E[N]$ exists and

$$E[N] = \sum_{n=1}^{\infty} n \sum_{v<u} f_{0,n}(u,v) = E[S_N]/E[Z_i] \tag{B.75}$$

Furthermore, if a maximal overshot $z_{\min}$ exists, then

$$\frac{u + z_{\min}}{E[Z_i]} \geq E[N] \geq \frac{u}{E[Z_i]} = \frac{u \ln 2}{g'(0)} \tag{B.76}$$

where $u < 0$ is the barrier and $g(\lambda)$ is the moment-generating function satisfying (B.5).

Proof: Taking the derivative of Wald's identity (B.60) yields

$$\sum_{n=1}^{\infty} \sum_{v<n} f_{0,n}(u, v) v 2^{\lambda v} g(\lambda)^{-n} \ln 2 - \sum_{n=1}^{\infty} \sum_{v<n} f_{0,n}(u, v) 2^{\lambda v} n g(\lambda)^{-n-1} g'(\lambda) = 0 \tag{B.77}$$

Since

$$g'(\lambda) = E[Z_i] \ln 2 \tag{B.78}$$

we have for $\lambda = 0$

$$\sum_{n=1}^{\infty} \sum_{v<n} v f_{0,n}(u, v) = E[Z_i] \sum_{n=1}^{\infty} \sum_{v<u} n f_{0,n}(u, v) \tag{B.79}$$

which is equivalent to (B.75). Since

$$u + z_{\min} \leq S_N \leq u \tag{B.80}$$

(B.76) follows. ∎

Finally, we remark that Wald's identity is valid not only in the situations when the random walk crosses the barrier $u < 0$ but also when it hits it, that is, as soon as $S_j \leq u$. The corresponding reformulations of Lemma B.1, Theorem B.2, and Corollary B.5 are left as exercises. Since the corresponding reformulations of Corollary B.3 and Corollary B.4 are used in Chapters 3–7 we give them here.

As a counterpart to Corollary B.3 we have the following:

Corollary B.6 Let S_j be the random walk given by (B.1) with $E[Z_i] > 0$ and let $g(\lambda)$ be the moment-generating function of the random variable Z_i which satisfies (B.2) and (B.3). Let $\lambda_0 < 0$ be a root of equation (B.7) such that $g'(\lambda_0) < 0$. Then,

$$P(S_{\min} \leq u) \leq 2^{-\lambda_0 u} \tag{B.81}$$

where

$$S_{\min} = \min_j \{S_j\} \tag{B.82}$$

Finally, as a counterpart to Corollary B.4 we have the following:

Corollary B.7 Let S_j be the random walk given by (B.1) with the absorbing rule $S_j \leq u$, where the barrier $u < 0$, and let $g(\lambda)$ be the moment-generating function of the random variable Z_i which satisfies (B.2) and (B.3) and minimum value $z_{\min} = \min\{Z_i\} < 0$. Let $\lambda_1 > 0$ be a root of equation (B.63) such that $g'(\lambda_1) \leq 0$. Then inequality (B.72) holds.

REFERENCES

Anc77. Ancheta, T. C., Jr. (1977), Bounds and techniques for linear source coding. Ph.D. Thesis, Dept. Elec. Eng., Univ. of Notre Dame, Notre Dame, Ind.

And69. Anderson, J. B. (1969), Instrumentable tree encoding of information sources. M. Sc. Thesis, School of Elec. Eng., Cornell University, Ithaca, N.Y.

And89. Anderson, J. B. (1989), Limited search trellis decoding of convolutional codes. *IEEE Trans. Inform. Theory*, IT-35:944–956.

And92. Anderson, J. B. (1992), Sequential decoding based on an error criterion. *IEEE Trans. Inform. Theory*, IT-38:987–1001.

AnM91. Anderson, J. B. and Mohan, S. (1991), *Source and Channel Coding—An Algorithmic Approach*. Kluwer Academic, Boston.

BaT97. Baggen, C. P. M. J. and Tolhuizen, L. M. G. M. (1997), On diamond codes. *IEEE Trans. Inform. Theory*, IT-43:1400–1411.

BBL95. Best, M. R., Burnashev, M. V., Levy, Y., Rabinovich, A., Fishburn, P. C., Calderbank, A. R., and Costello, D. J., Jr. (1995), On a technique to calculate the exact performance of a convolutional code. *IEEE Trans. Inform. Theory*, IT-41:441–447.

BCJ74. Bahl, L., Cocke, J., Jelinek, F., and Raviv, J. (1974), Optimal decoding of linear codes for minimizing symbol error rate. *IEEE Trans. Inform. Theory*, IT-20:284–287.

BeB93. Berger, Y. and Beéry, Y. (1993), Bounds on the trellis size of linear codes. *IEEE Trans. Inform. Theory*, IT-39:203–209.

BeG96. Berrou, C. and Glavieux, A. (1996), Near-optimum error-correcting coding and decoding: Turbo codes. *IEEE Trans. Commun.*, COM-44:1261–1271.

Ber82. Berlekamp, E. R. (1982), Bit-serial Reed-Solomon encoders. *IEEE Trans. Inform. Theory*, IT-28:869–874.

BGT93. Berrou, C., Glavieux, A., and Thitimajshima, P. (1993), Near Shannon limit error-correcting coding and decoding: Turbo codes (1). *Proc. ICC'93*, 1064–1070, Geneva, Switzerland.

Bha43. Bhattacharyya, A. (1943), On a measure of divergence between two statistical populations defined by their probability distributions. *Bull. Calcutta Math. Soc.*, 35:99–110.

BHJ01. Bocharova, I. E., Handlery, M., Johannesson, R., and Kudryashov, B. D. (2001), A BEAST for prowling in trees. *Proc. 39th Annual Allerton Conf. Commun., Control, and Computing*, Monticello, Ill., October.

BHJ02. Bocharova, I. E., Handlery, M., Johannesson, R., and Kudryashov, B. D. (2002), Tailbiting codes obtained via convolutional codes with large active distance-slopes. *IEEE Trans. Inform. Theory*, IT-48:2577–2587.

BHJ04. Bocharova, I. E., Handlery, M., Johannesson, R., and Kudryashov, B. D. (2004), A BEAST for prowling in trees. *IEEE Trans. Inform. Theory*, IT-50:1295–1302.

BHJ05. Bocharova, I. E., Handlery, M., Johannesson, R., and Kudryashov, B. D. (2005), BEAST decoding of block codes obtained via convolutional codes. *IEEE Trans. Inform. Theory*, IT-51:1880–1891.

BHJ10. Bocharova, I. E., Hug, F., Johannesson, R., and Kudryashov, B. D. (2010), On weight enumerators and MacWilliams identity for convolutional codes. *Information Theory and Applications Workshop*, San Diego, Calif., Jan. 31–Feb. 05.

BHJ12. Bocharova, I. E., Hug, F., Johannesson, R., and Kudryashov, B. D. (2012), A closed-form expression for the exact bit error probability for Viterbi decoding of convolutional codes. *IEEE Trans. Inform. Theory*, IT-58:4635–4644.

BJK02. Bocharova, I. E., Johannesson, R., Kudryashov, B. D., and Ståhl, P. (2002), Tailbiting codes: Bounds and search results. *IEEE Trans. Inform. Theory*, IT-48:137–148.

BJK04a. Bocharova, I. E., Johannesson, R., Kudryashov, B. D., and Lončar, M. (2004), BEAST for decoding block codes. *European Trans. Telecomm.*, 15(4):297–305.

BJK04b. Bocharova, I. E., Johannesson, R., and Kudryashov, B. D. (2004), Low state complexity block codes via convolutional codes. *IEEE Trans. Inform. Theory*, IT-50:2022–2030.

BJK07. Bocharova, I. E., Johannesson, R., and Kudryashov, B. D. (2007), Trellis complexity for short linear codes. *IEEE Trans. Inform. Theory*, IT-53:361–368.

Bla92. Blahut, R. E. (1992), Presentation at the Workshop on Information Theory, Mathematisches Forschungsinstitut, Oberwolfach, Germany, April 5–11.

BMc74. Butman, S. A. and McEliece, R. J. (1974), The ultimate limits of binary coding for a wideband Gaussian channel. *DSN Progress Report 42–22*, vol. May–June 1974, Jet Propulsion Laboratory, Pasadena, Calif., 78–80.

Bre04. Breiling, M. (2004), A logarithmic upper bound on the minimum distance of turbo codes. *IEEE Trans. Inform. Theory*, IT-50:1692–1710.

BrV93. Brouwer, A. G. and Verhoeff, T. (1993), An updated table of minimum distance bounds for linear codes. *IEEE Trans. Inform. Theory*, IT-39:662–677.

Bus65. Bussgang, J. J. (1965), Some properties of binary convolutional code generators. *IEEE Trans. Inform. Theory*, IT-11:90–100.

CCG79. Cain, J. B., Clark, G. C., Jr., and Geist, J. M. (1979), Punctured codes of rate $(n-1)/n$ and simplified maximum-likelihood decoding. *IEEE Trans. Inform. Theory*, IT-25:97–100.

CeJ89. Cedervall, M. and Johannesson, R. (1989), A fast algorithm for computing distance spectrum of convolutional codes. *IEEE Trans. Inform. Theory*, IT-35:1146–1159.

CFV99. Calderbank, A. R., Forney, G. D., Jr., and Vardy, A. (1999), Minimal tail-biting trellises: The Golay code and more. *IEEE Trans. Inform. Theory*, IT-45: 1435–1455.

ChC76. Chevillat, P. R. and Costello, D. J., Jr. (1976), Distance and computing in sequential decoding. *IEEE Trans. Commun.*, COM-24:440–447.

CJZ84a. Cedervall, M., Johannesson, R., and Zigangirov, K. Sh. (1984), A new upper bound on the first-event error probability for maximum-likelihood decoding of fixed binary convolutional codes. *IEEE Trans. Inform. Theory*, IT-30:762–766.

CJZ84b. Cedervall, M., Johannesson, R., and Zigangirov, K. Sh. (1984), Creeper—an easily implementable algorithm for sequential decoding. *Proc. Sixth Int. Symp. Inform. Theory*, Tashkent, USSR.

Cos69. Costello, D. J., Jr. (1969), A construction technique for random-error-correcting convolutional codes. *IEEE Trans. Inform. Theory*, IT-19:631–636.

Cos74. Costello, D. J., Jr. (1974), Free distance bounds for convolutional codes. *IEEE Trans. Inform. Theory*, IT-20:356–365.

CSZ92. Chepyzhov, V. V., Smeets, B. J. M., and Zigangirov, K. Sh. (1992), The free distance of fixed convolutional rate $2/4$ codes meets the Costello bound. *IEEE Trans. Inform. Theory*, IT-38:1360–1366.

DiP95. Divsalar, D. and Pollara, F. (1995), Multiple turbo codes for deep-space communication, *JPL TDA Progr. Rep.*, 66–77.

Eli54. Elias, P. (1954), Error-free coding. *IRE Trans. Inform. Theory*, PGIT-4:29–37. Also in *Key Papers in the Development of Coding Theory*, E. R. Berlekamp, Ed., (1974), IEEE Press, New York.

Eli55. Elias, P. (1955), Coding for noisy channels. *IRE Conv. Rec.*, pt. 4, 37–46. Also in *Key Papers in the Development of Coding Theory*, E. R. Berlekamp, Ed., (1974), IEEE Press, New York.

ELT00. Engdahl, K., Lentmaier, M., Truhachev, D., and Zigangirov, K. Sh. (2000), Analytical approach to low-density convolutional codes. *Proc. IEEE Int. Symp. Inform. Theory*, Sorrento, Italy.

ELZ99. Engdahl, K., Lentmaier, M., and Zigangirov, K. Sh. (1999), On the theory of low-density convolutional code. *Proceedings of Symposium on Applied Algebra, Algebraic Algorithms, and Error Correcting Codes*, Hawaii.

EnZ99. Engdahl, K. and Zigangirov, K. Sh. (1999), On the theory of low-density convolutional codes I. *Problemy Peredachi Informazii*, 35(4):12–28.

Fal66. Falconer, D. D. (1966), A hybrid sequential and algebraic decoding scheme. Ph.D. Thesis, Dept. of Elec. Eng, MIT, Cambridge, Mass.

Fan61. Fano, R. M. (1961), *Transmission of Information*. MIT Press, Cambridge, Mass., and Wiley, New York.

Fan63. Fano, R. M. (1963), A heuristic discussion of probabilistic decoding. *IEEE Trans. Inform. Theory*, IT-9:64–74.

Fel68. Feller, W. (1968), *An Introduction to Probability Theory and Its Applications*, 3rd ed.. Wiley, New York.

FHB89. Fuja, T., Heegard, C., and Blaum, M. (1991), Cross parity check convolutional codes. *IEEE Trans. Inform. Theory*, IT-35:1265–1276.

FJW96. Forney, G. D., Jr., Johannesson, R., and Wan, Z.-X. (1996), Minimal and canonical rational generator matrices for convolutional codes. *IEEE Trans. Inform. Theory*, IT-42:1865–1880.

For67. Forney, G. D., Jr. (1967), *Review of Random Tree Codes*. NASA Ames Res. Cen., Contract NAS2-3637, NASA CR 73176, Final Rep.; Appx A. See also Forney, G. D., Jr. (1974), and Convolutional codes II: Maximum-likelihood decoding and Convolutional codes III: Sequential decoding. *Inform Contr.*, 25:222–297.

For70. Forney, G. D., Jr. (1970), Convolutional codes I: Algebraic structure. *IEEE Trans. Inform. Theory*, IT-16:720–738.

For73. Forney, G. D., Jr. (1973), Structural analyses of convolutional codes via dual codes. *IEEE Trans. Inform. Theory*, IT-19:512–518.

For74. Forney, G. D., Jr. (1974), Convolutional codes II: Maximum-likelihood decoding and convolutional codes. *Inform. Contr.*, 25:222–266.

For75. Forney, G. D., Jr. (1975), Minimal bases of rational vector spaces, with applications to multivariable systems. *SIAM J. Control*, 13:493–520.

For88. Forney, G. D., Jr. (1988), Coset codes—Part II: Binary lattices and related codes. *IEEE Trans. Inform. Theory*, IT-34:1152–1187.

For91. Forney, G. D., Jr. (1991), Algebraic structure of convolutional codes, and algebraic system theory. In *Mathematical System Theory*, A. C. Antoulas, Ed., Springer, Berlin, 527–558.

For94. Forney, G. D., Jr. (1994), Trellises old and new, 115–128. In *Communications and Cryptography—Two Sides of the One Tapestry*, R. E. Blahut, *et al.*, Eds., Published in honor of James L. Massey on the occasion of his 60th birthday. Kluwer Academic, Boston.

For97. Forney, G. D., Jr. (1997), On iterative decoding and the two-way algorithm. *Proc. Int. Symp. on Turbo Codes & Related Topics*, Brest, France.

FoT93. Forney, G. D., Jr. and Trott, M. D. (1993), The dynamics of group codes: State spaces, trellis diagrams, and canonical encoders. *IEEE Trans. Inform. Theory*, IT-39:1491–1513.

FTL09. Feltström, A. J., Truhachev, D., Lentmaier, M., and Zigangirov, K. Sh. (2009), Braided block codes. *IEEE Trans. Inform. Theory*, IT-55:2640–2658.

Gal62. Gallager, R. G. (1962), Low-density parity-check codes. *IRE Trans. Inform. Theory*, IT-8:21–28.

Gal63. Gallager, R. G. (1963), *Low-Density Parity-Check Codes*. MIT Press, Cambridge, Mass.

Gal65. Gallager, R. G. (1965), A simple derivation of the coding theorem and some applications. *IEEE Trans. Inform. Theory*, IT-11:3–18.

Gal68. Gallager, R. G. (1968), *Information Theory and Reliable Communication*. Wiley, New York.

Gil52. Gilbert, E. N. (1952), A comparison of signalling alphabets. *Bell System Tech. J.* 31:504–522.

Gil66. Gill, A. (1966), *Linear Sequential Circuits: Analysis, Synthesis, and Applications*. McGraw-Hill, New York.

Gol49. Golay, M. J. E. (1949), Notes on digital coding. *Proc. I. R. E.*, 37:657.

Gol67. Golomb, S. W. (1967), *Shift Register Sequences*, Holden-Day, San Francisco, 1967. Revised ed., Aegean Park Press, Laguna Hills, Calif., 1982.

Gri60. Griesmer, J. H. (1960), A bound for error-correcting codes. *IBM J. Res. Develop.*, 4:532–542.

Hac66. Haccoun, D. (1966), Simulated communication with sequential decoding and phase estimation. S.M. Thesis, Dept. of Elec. Eng., MIT, Cambridge, Mass.

HaH70. Hartley, B. and Hawkes, T. O. (1970), *Rings, Modules and Linear Algebra*. Chapman and Hall, London.

Ham50. Hamming, R. W. (1950), Error-detecting and error-correcting codes. *Bell Sys. Techn. J.*, 29:147–160.

Han02. Handlery, M. (2002), Tales of tailbiting codes. Ph.D. Thesis, Dept. of Information Technology, Lund University, Lund, Sweden.

HBJ10. Hug, F., Bocharova, I. E., Johannesson, R., and Kudryashov, B. D. (2010), A rate $R = 5/20$ hypergraph-based woven convolutional code with free distance 120. *IEEE Trans. Inform. Theory*, IT-56:1618–1623.

Hel67. Heller, J. A. (1967), Sequential decoding for channels with time varying phase. Sc.D. Thesis, Dept. of Elec. Eng., MIT, Cambridge, Mass.

Hel68. Heller, J. A. (1968), Short constraint length convolutional codes. Jet Propulsion Lab., California Inst. Technol., Pasadena, Space Programs Summary 37–54, 3:171–177.

HJM01. Handlery, M., Johannesson, R., Massey, J. L., and Ståhl, P., An upper bound on decoding bit-error probability with linear coding on extremely noisy channels. *Proc. IEEE Int. Symp. Inform. Theory*, Washington, D.C., 131.

HJS98. Höst, S., Johannesson, R., Sidorenko, V. R., Zigangirov, K. Sh., and Zyablov, V. V. (1998), Cascaded convolutional codes. In *Communications and Coding*, 10–29. M. Darnell and B. Honary, Eds., Published in honor of Paddy G. Farrell on the occasion of his 60th birthday. Research Studies Press Ltd. and Wiley.

HJZ95. Höst, S., Johannesson, R., and Zyablov, V. V. (1995), On the construction of concatenated codes based on binary conventional convolutional codes. *Proc. Seventh Joint Swedish-Russian Int. Workshop on Inform. Theory*, St. Petersburg, Russia, 114–118.

HJZ99. Höst, S., Johannesson, R., Zigangirov, K. Sh., and Zyablov, V. V. (1999), Active distances for convolutional codes. *IEEE Trans. Inform. Theory*, IT-45:658–669.

HLC06. He, C., Lentmaier, M., Costello, D. J., Jr., and Zigangirov, K. Sh. (2006), Joint permutor analysis and design for multiple turbo codes. *IEEE Trans. Inform. Theory*, IT-52:4068–4083.

HoJ90. Horn, R. A., and Johnson C. R. (1990), *Matrix Analysis*, Cambridge University Press, Cambridge, UK.

Hug12. Hug, F. (2012), Codes on graphs and more. Ph.D. Thesis, Dept. of Elec. and Information Technology, Lund University, Lund, Sweden.

HZC08. Huebner, A., Zigangirov, K. Sh., and Costello, D. J., Jr. (2008), Laminated turbo codes: A new class of block-convolutional codes. *IEEE Trans. Inform. Theory*, IT-54:3024–3034.

JaB67. Jacobs, I. M. and Berlekamp, E. R. (1967), A lower bound to the distribution of computation for sequential decoding. *IEEE Trans. Inform. Theory*, IT-13:167–174.

Jac85. Jacobson, N. (1985), *Basic Algebra I*, 2nd ed. Freeman, New York.

Jac89. Jacobson, N. (1989), *Basic Algebra II*, 2nd ed. Freeman, New York.

Jel69. Jelinek, F. (1969), A fast sequential decoding algorithm using a stack. *IBM J. Res. Dev.*, 13:675–685.

Jim06. Jiménez-Feltström, A. (2006), Iteratively decodable convolutional codes: Analysis and implementation aspects. Ph.D. Thesis, Dept. of Inform. Theory, Lund University, Lund, Sweden.

JiZ99. Jiménez, A. and Zigangirov, K. Sh. (1999), Periodically time-varying convolutional codes with low-density parity-check matrices. *IEEE Trans. Inform. Theory*, IT-45:2181–2190.

JJB04. Jordan, R., Johannesson, R., and Bossert, M. (2004), On nested convolutional codes and their applications to woven codes. *IEEE Trans. Inform. Theory*, IT-50:380–384.

JMS00. Johannesson, R., Massey, J. L., and Ståhl, P. (2000), On the decoding bit error probability for binary convolutional codes. *Proc. IEEE Int. Symp. Inform. Theory*, Sorrento, Italy, 328.

JMS02. Johannesson, R., Massey, J. L., and Ståhl, P. (2002), Systematic bits are better and no buts about it. In *Codes, Graphs, and Systems*, R. E. Blahut, and R. Koetter, Eds., 77–89. Published in honor of G. D. Forney, Jr., on the occasion of his 60th birthday. Kluwer Academic, Boston.

Joh75. Johannesson, R. (1975), Robustly optimal rate one-half binary convolutional codes. *IEEE Trans. Inform. Theory*, IT-21:464–468.

Joh76. Johannesson, R. (1976), Some long rate one-half binary convolutional codes with an optimum distance profile. *IEEE Trans. Inform. Theory*, IT-22:629–631.

Joh77a. Johannesson, R. (1977), Some rate $1/3$ and $1/4$ binary convolutional codes with an optimum distance profile. *IEEE Trans. Inform. Theory*, IT-23:281–283.

Joh77b. Johannesson, R. (1977), On the error probability of general trellis codes with applications to sequential decoding. *IEEE Trans. Inform. Theory*, IT-23:609–611.

JoP78. Johannesson, R. and Paaske, E. (1978), Further results on binary convolutional codes with an optimum distance profile. *IEEE Trans. Inform. Theory*, IT-24:264–268.

Jor02. Jordan, R. (2002), Design aspects of woven convolutional codes. Ph.D. Thesis, Univ. of Ulm, Fortschr.-Ber. VDI Reihe 10 Nr. 714, VDI Verlag, Dusseldorf.

JoS99. Johannesson, R. and Ståhl, P. (1999), New rate $1/2$, $1/3$, and $1/4$ binary convolutional encoders with an optimum distance profile. *IEEE Trans. Inform. Theory*, IT-45:1653–1658.

JoW93. Johannesson, R. and Wan, Z.-X. (1993), A linear algebra approach to minimal convolutional encoders. *IEEE Trans. Inform. Theory*, IT-39:1219–1233.

JoW94. Johannesson, R. and Wan, Z.-X. (1994), On canonical encoding matrices and the generalized constraint lengths of convolutional codes. In *Communications and Cryptography—Two Sides of the One Tapestry*, R. E. Blahut, *et al.*, Eds., 187–200. Published in honor of James L. Massey on the occasion of his 60th birthday. Kluwer Academic, Boston.

JoW98. Johannesson, R. and Wittenmark, E. (1998), Two 16-state, rate $R = 2/4$ trellis codes whose free distances meet the Heller bound. *IEEE Trans. Inform. Theory*, IT-44:1602–1604.

JoZ89. Johannesson, R. and Zigangirov, K. Sh. (1989), Distances and distance bounds for convolutional codes. In *Topics in Coding Theory—In Honour of Lars H. Zetterberg*, G. Einarsson, *et al.*, Eds., Springer, Berlin, 109–136.

JoZ96. Johannesson, R. and Zigangirov, K. Sh. (1996), Towards a theory for list decoding of convolutional codes. *Probl. Peredachi Inform.*, 1.

JoZ98. Johansson, T. and Zigangirov, K. Sh. (1998), A simple one-sweep algorithm for optimal APP symbol decoding of linear block codes. *IEEE Trans. Inform. Theory*, IT-44:3124–3129.

JSW00. Johannesson, R., Ståhl, P., and Wittenmark, E. (2000), A note on Type II convolutional codes. *IEEE Trans. Inform. Theory*, IT-46:1510–1514.

JTZ88. Justesen, J., Thommesen, C., and Zyablov, V. V. (1988), Concatenated codes with convolutional inner codes. *IEEE Trans. Inform. Theory*, IT-34:1217–1225.

JZZ95. Johannesson, R., Zigangirov, K. Sh., and Zyablov, V. V. (1995), Lower bounds on the free distance for random concatenated convolutional codes. *Proc. Seventh Joint Swedish-Russian Int. Workshop on Inform. Theory*, St. Petersburg, Russia, 133–136.

Kab91. Kabatyanskii, G. A. (1991), About metrics and decoding domains of Forney algorithm. *Proc. 5th Joint Swedish-Russian Int. Workshop on Inform. Theory*, Moscow, Russia, 81–85.

KaU98. Kahale, N. and Urbanke R. (1998), On the minimum distance of parallel and serially concatenated codes. *Proc. IEEE Int. Symp. Information Theory*, Cambridge, Mass.

KFA69. Kalman, R. E., Falb, P. L., and Arbib, M. A. (1969), *Topics in Mathematical System Theory*. McGraw-Hill, New York.

KFL01. Kschischang, F. R., Frey, B., and Loeliger, H.-A. (2001), Factor graphs and the sum-product algorithm. *IEEE Trans. Inform. Theory*, IT-47:498–519.

Knu73. Knuth, D. E. (1973), *The Art of Computer Programming, Vol. 3, Searching and Sorting*. Addison-Wesley, Reading, Mass.

KoV98. Kötter, R. and Vardy, A. (1998), Construction of minimal tail-biting trellises. *Proc. IEEE Inform. Theory Workshop*, Killarney, Ireland, 72–74.

KRU11. Kudekar, S., Richardson T. J., and Urbanke R. L. (2011), Threshold saturation via spatial coupling: Why convolutional LDPC ensembles perform so well over the BEC. *IEEE Trans. Inform. Theory*, IT-57:803–834.

KsS95. Kschischang, F. R. and Sorokine, V. (1995), On the trellis structure of block codes. *IEEE Trans. Inform. Theory*, IT-41:1924–1937.

LaM70. Layland, J. and McEliece, R. (1970), An upper bound on the free distance of a tree code. Jet Propulsion Lab., California Inst. Technol., Pasadena, Space Programs Summary 37–62, 3:63–64.

Lee76. Lee, L.-N. (1976), On optimal soft-decision demodulation. *IEEE Trans. Inform. Theory*, IT-22:437–444.

Lee78. Lee, S. C. (1978), *Modern Switching Theory and Digital Design*. Prentice-Hall, Englewood Cliffs, N.J.

Len03. Lentmaier, M. (2003), Towards a theory of codes for iterative decoding. Ph.D. Thesis, Dept. of Inform. Theory, Lund University, Lund, Sweden.

LeZ98. Lentmaier, M. and Zigangirov, K. Sh. (1998), Iterative decoding of generalized low-density parity-check codes. *Proc. IEEE Int. Symp. Inform. Theory*, Boston.

LeZ99. Lentmaier, M. and Zigangirov, K. Sh. (1999), On generalized low-density parity-check codes based on Hamming component codes. *IEEE Commun. Lett.*, 3(8):248–250.

LFM94. Loeliger, H.-A., Forney, G. D., Jr., Mittelholzer, T., and Trott, M. D. (1994), Minimality and observability of group systems. *Linear Algebra and Its Appli.*, 205–206:937–963.

Lin86. Lin, C.-F. (1986), A truncated Viterbi algorithm approach to trellis codes. Ph.D. Thesis, ECSE Dept., Rensselaer Poly. Inst., Troy, N.Y.

LoM96. Loeliger, H.-A. and Mittelholzer, T. (1996), Convolutional codes over groups. *IEEE Trans. Inform. Theory*, IT-42:1660–1686.

Lon07. Lončar, M. (2007), Taming of the BEAST. Ph.D. Thesis, Dept. of Information Technology, Lund University, Lund, Sweden.

LSC10. Lentmaier, M., Sridharan, A., Costello, D. J., Jr., and Zigangirov, K. Sh. (2010), Iterative decoding thresholds analysis for LDPC convolutional codes, *IEEE Trans. Inform. Theory*, IT-56:5274–5289.

LTZ01. Lentmaier, M., Truhachev, D. V., and Zigangirov, K. Sh. (2001), Analysis of the asymptotic iterative decoding performance of turbo codes, *Proc. IEEE Int. Symp. Inform. Theory*, Washington, D.C., 190.

LTZ02. Lentmaier, M., Truhachev, D. V., and Zigangirov, K. Sh. (2002), Iteratively decodable codes on graphs, *Proc. ACCT-VIII*, St. Peterburg, Russia, Sept., 190–193.

LTZ04. Lentmaier, M., Truhachev, D. V., and Zigangirov, K. Sh. (2004), Analytic expression for the bit error probability of rate-1/2 memory 2 convolutional encoders, *IEEE Trans. Inform. Theory*, IT-50:1303–1311.

LTZ05. Lentmaier, M., Truhachev, D. V., Zigangirov, K. Sh., and Costello, D. J., Jr. (2005), An analysis of the block error probability performance of iterative decoding, *IEEE Trans. Inform. Theory*, IT-51:3834–3855.

MaC71. Massey, J. L. and Costello, D. J., Jr. (1971), Nonsystematic convolutional codes for sequential decoding in space applications. *IEEE Trans. Commun. Technol.*, COM-19:806–813.

MaN96. MacKay, D. J. C. and Neal, R. M. (1996), Near Shannon limit performance of low density parity check codes. *Electron. Lett.*, 32(18):1645–1646.

Mar82. Margulis, G. A. (1982), Explicit construction of graphs without short cycles and low density codes. *Combinatorica*, 2:71–78.

Mas63. Massey, J. L. (1963), *Threshold Decoding*. MIT Press, Cambridge, Mass.

Mas74. Massey, J. L. (1974), Coding and modulation in digital communications. *Proc. Int. Zurich Seminar on Digital Commun.*, E2(1)–E2(4).

Mas75. Massey, J. L. (1975), Error bounds for tree codes, trellis codes, and convolutional codes with encoding and decoding procedures. In *Coding and Complexity—CISM Courses and Lectures No. 216*, G. Longo, Ed., Springer, Wien.

Mas78. Massey, J. L. (1978), Foundations and methods of channel coding. *Proc. Int. Conf. Inform. Theory and Systems, NTG Fachberichte*, Vol. 65, Berlin, Sept.

Mas82. Massey, J. L. (1982), What is a bit of information? *Scienta Electric*, 28(1):1–11.

Mas84. Massey, J. L. (1984), The how and why of channel coding. *Proc. 1984 Int. Zurich Seminar on Digital Communications*, Zurich, 67–73.

MaS67. Massey, J. L. and Sain, M. K. (1967), Codes, automata, and continuous systems: Explicit interconnections. *IEEE Trans. Automatic Control*, AC-12:644–650.

MaS68. Massey, J. L. and Sain, M. K. (1968), Inverses of linear sequential circuits. *IEEE Trans. Comput.*, C-17:330–337.

MaS77. MacWilliams, F. J. and Sloane, N. J. A. (1977), *The Theory of Error-Correcting Codes*. North-Holland, Amsterdam.

MaW86. Ma, J. H. and Wolf, J. K. (1986), On tail-biting convolutional codes. *IEEE Trans. Commun.*, 34:104–111.

MaZ60. Mason, S. and Zimmermann, H. (1960), *Electronic Circuits, Signals, and Systems*. Wiley, New York.

McE77. McEliece, R. J. (1977), *The Theory of Information and Coding*. Addison-Wesley, Reading, Mass.

McE96. McEliece, R. J. (1996), On the BCJR trellis for linear block codes. *IEEE Trans. Inform. Theory*, IT-42:1072–1092.

McE98. McEliece, R. J. (1998), The algebraic theory of convolutional codes. In *Handbook of Coding Theory*, Vol. I, V. S. Pless, and W. C. Huffman, Eds., Elsevier Science, Amsterdam, 1065–1138.

Mee74. Van de Meeberg, L. (1974), A tightened upper bound on the error probability of binary convolutional codes with Viterbi decoding. *IEEE Trans. Inform. Theory*, IT-20:389–391.

MoA84. Mohan, S. and Anderson, J. B. (1984), Sequential coding algorithms: A survey and cost analysis. *IEEE Trans. Commun. Technol.*, COM-32:169–176.

Mon70. Monna, A. F. (1970), *Analyse Non-Archimedienne*. Springer, Berlin.

Mud88. Muder, D. J. (1988), Minimal trellises for block codes. *IEEE Trans. Inform. Theory*, IT-34:1049–1053.

NJZ97. Nyström, J., Johannesson, R., and Zigangirov, K. Sh. (1997), Creeper—an algorithm for sequential decoding. Submitted to *IEEE Trans. Inform. Theory*.

Nys93. Nyström, J. (1993), Creeper—an algorithm for sequential decoding. Ph.D. Thesis, Dept. of Inform. Theory, Lund University, Lund, Sweden.

OAJ98. Osthoff, H., Anderson, J. B., Johannesson, R., and Lin, C.-F. (1998), Systematic feed-forward convolutional encoders are better than other encoders with an M-algorithm decoder. *IEEE Trans. Inform. Theory*, 44:831–838.

Ols70. Olson, R. R. (1970), Note on feedforward inverses of linear sequential circuits. *IEEE Trans. Comput.*, C-19:1216–1221.

Omu69. Omura, J. K. (1969), On the Viterbi decoding algorithm. *IEEE Trans. Inform. Theory*, IT-15:177–179.

Ost93. Osthoff, H. (1993), Reduced complexity decoding with systematic encoders. Ph.D. Thesis, Dept. Inform. Theory, Lund University, Lund, Sweden.

Pin67. Pinsker, M. S. (1967), Bounds for error probability and for number of correctable errors for nonblock codes. *Probl. Peredachi Inform.*, 3:58–71.

Pir88. Piret, P. (1988), *Convolutional Codes: An Algebraic Approach*. MIT Press, Cambridge, Mass.

PJS08. Pusane, A. E., Jimenes, A. J. F., Sridharan, A. J., Lentmaier, M., Zigangirov, K. Sh., and Costello, D. J., Jr. (2008), Implementation aspects of LDPC convolutional codes. *IEEE Trans. Commun.*, COM-50:1060–1069.

RoC89. Rouanne, M. and Costello, D. J., Jr. (1989), An algorithm for computing the distance spectrum of trellic codes. *IEEE J. Select. Areas Commun.*, 7:929–940.

Roo79. Roos, C. (1979), On the structure of convolutional and cyclic convolutional codes. *IEEE Trans. Inform. Theory*, IT-25:676–683.

RUr08. Richardson, T., and Urbanke, R. (2008), *Modern Coding Theory*. Cambridge University Press, Cambridge, UK.

SAJ99. Ståhl, P., Anderson, J. B., and Johannesson, R. (1999), Optimal and near-optimal encoders for short and moderate-length tailbiting trellises. *IEEE Trans. Inform. Theory*, IT-45:2562–2571.

SAJ02. Ståhl, P., Anderson, J. B., and Johannesson, R. (2002), A note on tailbiting codes and their feedback encoders. *IEEE Trans. Inform. Theory*, 48:529–534.

SaM69. Sain, M. K. and Massey, J. L. (1969), Invertability of linear time-invariant dynamical systems. *IEEE Trans. Automatic Contr.*, AC-14:141–149.

Sav66. Savage, J. E. (1966), Sequential decoding—the computation problem. *Bell Syst. Tech. J.*, 45:149–176.

SGB67. Shannon, C. E., Gallager, R. G., and Berlekamp, E. R. (1967), Lower bounds to error probability for coding on discrete memoryless channels. *Inform. Contr.*, 10, Part I: 65–103 and Part II: 522–552.

Sha48. Shannon, C. E. (1948), A mathematical theory of communications. *Bell Sys. Tech. J.* 27:379–423 (Part I), 623–656 (Part II). Also reprinted in *Key Papers in the Development of Information Theory*, D. Slepian, Ed., (1974), IEEE Press, New York, 5–29.

Sov79. Solomon, G. and van Tilborg, H. C. A. (1979), A connection between block and convolutional codes. *SIAM J. Appl. Math.*, 37:358–369.

Sta01. Ståhl, P. (2001), On tailbiting codes from convolutional codes. Ph.D. Thesis, Dept. of Information Technology, Lund University, Lund, Sweden.

Sti63. Stiglitz, I. G. (1963), Sequential decoding with feedback. Sc.D. Thesis, Dept. of Elec. Eng., MIT, Cambridge, Mass.

STL07. Shridharan, A, Truhachev, D., Lentmaier, M., Costello, D. J., Jr., and Zigangirov, K. Sh. (2007), Distance bounds for an ensemble of LDPC convolutional codes. *IEEE Trans. Inform. Theory*, IT-53:4537–4555.

SVZ98. Shamai, S., Verdu, S., and Zamir, R. (1998), Systematic lossy source/channel coding, *IEEE Trans. Inform. Theory*, IT-44:564–579.

Tan81. Tanner, R. M. (1981), A recursive approach to low-complexity codes. *IEEE Trans. Inform. Theory*, IT-27:533–547.

ThJ83. Thommesen, C. and Justesen, J. (1983), Bounds on distances and error exponents of unit-memory codes. *IEEE Trans. Inform. Theory*, IT-29:637–649.

Tho83. Thompson, T. M. (1983), *From Error-Correcting Codes through Sphere Packings to Simple Groups*, The Carus Mathematical Monographs No. 21, The Mathematical Association of America.

TLZ01. Truhachev, D. V., Lentmaier, M., and Zigangirov, K. Sh. (2001), Some results concerning design and decoding of turbo codes. *Probl. Inf. Transm.,* 37:190–205.

TLZ03. Truhachev, D. V., Lentmaier, M., and Zigangirov, K. Sh. (2003), On braided block codes. *Proc. IEEE Int. Symp. Inform. Theory*, Yokohama, Japan, June.

Tru04. Truhachev, D. V. (2004), On the construction and analysis of iteratively decodable codes. Ph.D. Thesis, Dept. of Inform. Theory, Lund University, Lund, Sweden.

TZC10. Truhachev, D. V., Zigangirov, K. Sh., and Costello, D. J., Jr. (2010), Distance bounds for periodically time-varying and tail-biting LDPC convolutional codes. *IEEE Trans. Inform. Theory*, IT-56:4301–4308.

Var57. Varshamov, R. R. (1957), Estimate of the number of signals in error correcting codes. *Doklady Akad. Nauk SSSR*, 117:739–741.

ViO79. Viterbi, A. J. and Omura, J. K. (1979), *Principles of Digital Communication and Coding*. McGraw-Hill, New York.

Vit67. Viterbi, A. J. (1967), Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Trans. Inform. Theory*, IT-13:260–269.

Vit71. Viterbi, A. J. (1971), Convolutional codes and their performance in communication systems. *IEEE Trans. Commun. Technol.*, COM-19:751–772.

Vit90. Viterbi, A. J. (1990), From proof to product. *1990 IEEE Commun. Theory Workshop*, Ojai, Calif.

Wal47. Wald, A. (1947), *Sequential Analysis*. Wiley, New York.

Wib96. Wiberg, N. (1996), Codes and decoding on general graphs. Ph.D. Thesis, Dept. of Elec. Eng., Linköping University, Linköping, Sweden.

Wil96. Wilson, S. G. (1996), *Digital Modulation and Coding*. Prentice-Hall, London.

WLK95. Wiberg, N., Loeliger, H.-A., and Kötter, R. (1995), Codes and iterative decoding on general graphs. *Euro. Trans. Telecommun.*, 6:513–526.

Wol78. Wolf, J. K. (1978), Efficient maximum likelihood decoding of linear block code using a trellis. *IEEE Trans. Inform. Theory*, IT-24:76–80.

Woz57. Wozencraft, J. M. (1957), Sequential decoding for reliable communication. *IRE Conv. Rec.*, 5(2):11–25. See also Wozencraft, J. M., and Reiffen, B. (1961), *Sequential Decoding*. MIT Press, Cambridge, Mass.

Yud64. Yudkin, H. L. (1964), Channel state testing in information decoding. Sc.D. Thesis, Dept. of Elec. Eng., MIT, Cambridge, Mass.

ZiC89. Zigangirov, K. Sh. and Chepyzhov, V. V. (1989), Study of tail biting convolutional codes. *Proc. 4th Joint Swedish-Soviet Int. Workshop Inform. Theory*, Gotland, Sweden, 52–55.

ZiC91. Zigangirov, K. Sh. and Chepyzhov, V. V. (1991), On the existence of time-invariant convolutional codes with transmission rates $2/c$, $c \geq 4$ which meets the Costello bound. *Probl. Peredachi Inform.*, 3.

Zig66. Zigangirov, K. Sh. (1966), Some sequential decoding procedures. *Probl. Peredachi Inform.*, 2:13–25.

Zig72. Zigangirov, K. Sh. (1972), On the error probability of the sequential decoding in the BSC. *IEEE Trans. Inform. Theory*, IT-18:199–202.

Zig74. Zigangirov, K. Sh. (1974), *Procedures of Sequential Decoding*. Svjaz, Moscow.

Zig75. Zigangirov, K. Sh. (1975), Procedures of sequential decoding. In *Coding and Complexity—CISM Courses and Lectures No. 216*, G. Longo, Ed., Springer, Wien.

Zig85. Zigangirov, K. Sh. (1985), New upper bounds for decoding error probability for convolutional codes. *Probl. Peredachi Inform.*, 21:20–31.

Zig86. Zigangirov, K. Sh. (1986), New asymptotic lower bound on the free distance for time-invariant convolutional codes. *Probl. Peredachi Inform.*, 2:34–42.

ZiK80. Zigangirov, K. Sh. and Kolesnik, V. D. (1980), List decoding of trellis codes. *Probl. Control Inform. Theory*, 9:347–364.

ZiO93. Zigangirov, K. Sh. and Osthoff, H. (1993), Analysis of global-list decoding for convolutional codes. *European Trans. Telecommun.*, 4:165–173.

ZLZ10. Zhang, W., Lentmaier, M., Zigangirov, K. Sh., and Costello, D. J., Jr. (2010), Braided convolutional codes: A new class of turbo-like codes. *IEEE Trans. Inform. Theory*, IT-56:316–331.

ZPZ08. Zigangirov, K. Sh., Pusane, A. E., Zigangirov, D. K., and Costello, D. J., Jr. (2008), On the error-correcting capability of LDPC codes. *Probl. Peredachi Inform.*, 44:214–225.

ZyP74. Zyablov, V. V. and Pinsker, M. S. (1974), Decoding complexity of low-density codes for transmission in a channel with erasures, *Probl. Peredachi Inform.*, 10:15–28.

ZyP75. Zyablov, V. V. and Pinsker, M. S. (1975), Estimation of the error-correction complexity of Gallager low-density codes, *Probl. Peredachi Inform.*, 11:23–36.

INDEX

IEEE PRESS SERIES ON DIGITAL AND MOBILE COMMUNICATION

John B. Anderson, *Series Editor*
University of Lund

---

1. *Wireless Video Communications: Second to Third Generation and Beyond*
   Lajos Hanzo, Peter J. Cherriman, and Jurgen Streit
2. *Wireless Communications in the 2lst Century*
   Mansoor Sharif, Shigeaki Ogose, and Takeshi Hattori
3. *Introduction to WLLs: Application and Deployment for Fixed and Broadband Services*
   Raj Pandya
4. *Trellis and Turbo Coding*
   Christian B. Schlegel and Lance C. Perez
5. *Theory of Code Division Multiple Access Communication*
   Kamil Sh. Zigangirov
6. *Digital Transmission Engineering,* Second Edition
   John B. Anderson
7. *Wireless Broadband: Conflict and Convergence*
   Vern Fotheringham and Shamla Chetan
8. *Wireless LAN Radios: System Definition to Transistor Design*
   Arya Behzad
9. *Millimeter Wave Communication Systems*
   Kao-Cheng Huang and Zhaocheng Wang
10. *Channel Equalization for Wireless Communications: From Concepts to Detailed Mathematics*
    Gregory E. Bottomley
11. *Handbook of Position Location: Theory, Practice, and Advances*
    Edited by Seyed (Reza) Zekavat and R. Michael Buehrer
12. *Digital Filters: Principle and Applications with MATLAB*
    Fred J. Taylor
13. *Resource Allocation in Uplink OFDMA Wireless Systems: Optimal Solutions and Practical Implementations*
    Elias E. Yaacoub and Zaher Dawy
14. *Non-Gaussian Statistical Communication Theory*
    David Middleton
15. *Frequency Stabilization: Introduction and Applications*
    Venceslav F. Kroupa
16. *Mobile Ad Hoc Networking: Cutting Edge Directions,* Second Edition
    Stefano Basagni, Marco Conti, Silvia Giordano, and Ivan Stojmenovic
17. *Techniques for Surviving the Mobile Data Explosion*
    Dinesh Chandra Verma and Paridhi Verma

18. *Cellular Communications: A Comprehensive and Practical Guide*
    Nishith D. Tripathi and Jeffrey H. Reed
19. *Fundamentals of Convolutional Coding,* Second Edition
    Rolf Johannesson and Kamil Sh. Zigangirov

WILEY END USER LICENSE AGREEMENT

Go to www.wiley.com/go/eula to access Wiley’s ebook EULA.