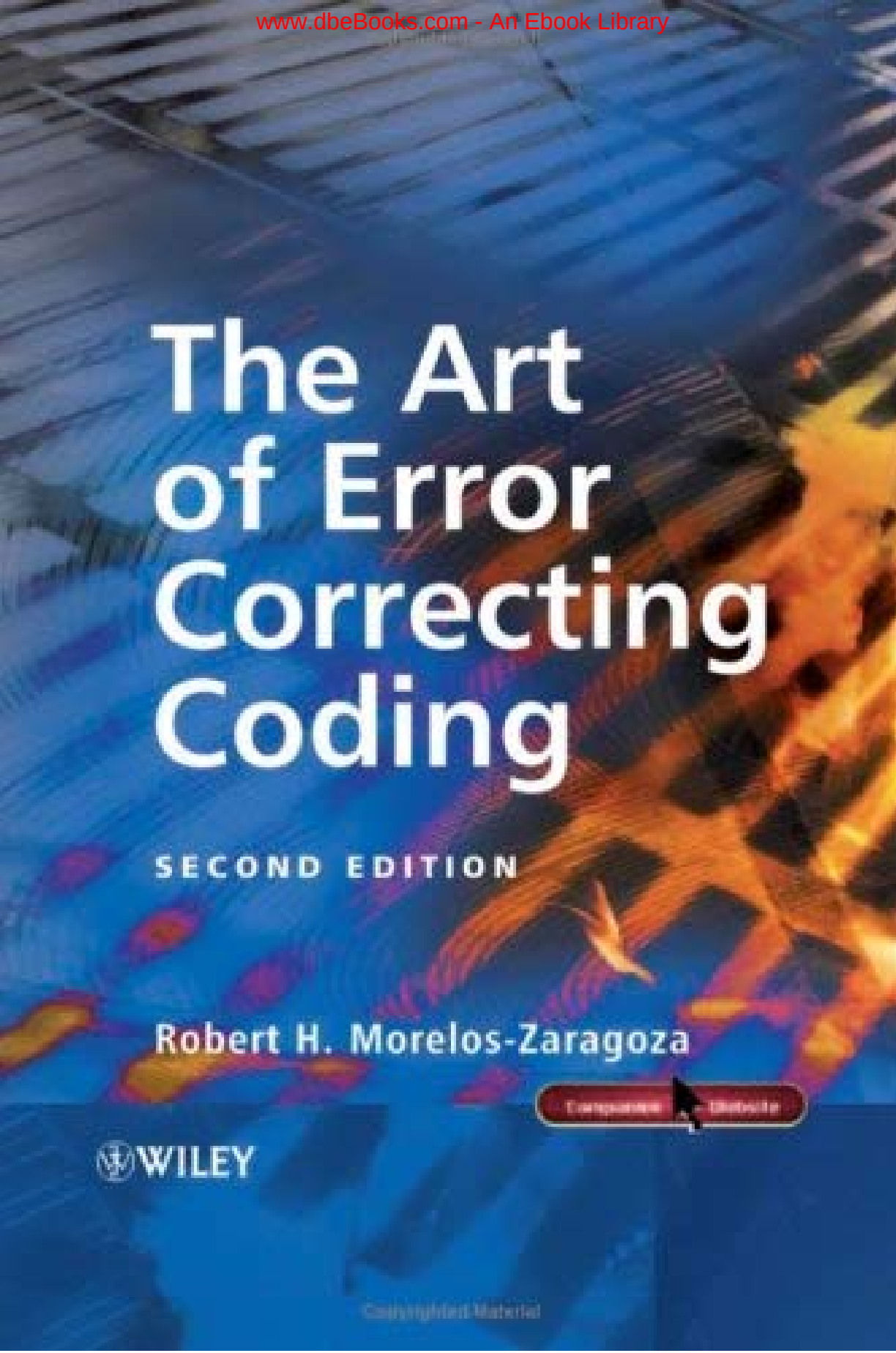


The Art of Error Correcting Coding

SECOND EDITION

Robert H. Morelos-Zaragoza

WILEY

The Art of Error Correcting Coding

The Art of Error Correcting Coding, Second Edition Robert H. Morelos-Zaragoza
 ISBN: 0-470-01558-6

The Art of Error Correcting Coding

Second Edition

Robert H. Morelos-Zaragoza
San Jose State University, USA

John Wiley & Sons, Ltd

John Wiley & Sons Ltd, The Atrium, Southern Gate, Chichester,
West Sussex PO19 8SQ, England

Telephone (+44) 1243 779777

Email (for orders and customer service enquiries): cs-books@wiley.co.uk
Visit our Home Page on www.wiley.com

Other Wiley Editorial Offices

John Wiley & Sons Inc., 111 River Street, Hoboken, NJ 07030, USA

Jossey-Bass, 989 Market Street, San Francisco, CA 94103-1741, USA

Wiley-VCH Verlag GmbH, Boschstr. 12, D-69469 Weinheim, Germany

John Wiley & Sons Australia Ltd, 42 McDougall Street, Milton, Queensland 4064, Australia

John Wiley & Sons (Asia) Pte Ltd, 2 Clementi Loop #02-01, Jin Xing Distripark, Singapore 129809

John Wiley & Sons Canada Ltd, 6045 Freemont Blvd, Mississauga, ONT, L5R 4J3, Canada

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

British Library Cataloguing in Publication Data

A catalogue record for this book is available from the British Library

ISBN-13: 978-0-470-01558-2 (HB)
ISBN-10: 0-470-01558-6 (HB)

Typeset in 10/12pt Times by Laserwords Private Limited, Chennai, India.
Printed and bound in Great Britain by Antony Rowe Ltd, Chippenham, England.
This book is printed on acid-free paper responsibly manufactured from sustainable forestry in which at least two trees are planted for each one used for paper production.

Contents

Preface

The first edition of this book was the result of hundreds of emails from all over the world with questions on the theory and applications of error correcting coding (ECC), from colleagues from both academia and industry. Most of the questions have been from engineers and computer scientists needing to select, implement or simulate a particular coding scheme. The questions were sparked by a popular web site[1] initially set up at Imai Laboratory at the Institute of Industrial Science, University of Tokyo, in early 1995. An important aspect of this text is the absence of theorems and proofs. The approach is to teach basic concepts using simple examples. References to theoretical developments are made when needed. This book is intended to be a reference guide to error correcting coding techniques for graduate students and professionals interested in learning the basic techniques and applications of ECC. Computer programs that implement the basic encoding and decoding algorithms of practical coding schemes are available on a companion web site. This site is referred to as the "ECC web site" throughout the text and is located at:

`http://the-art-of-ecc.com`

This book is unique in that it introduces the basic concepts of error correcting codes with simple illustrative examples. Computer programs written in C language and new Matlab[2] scripts are available on the ECC web site and help illustrate the implementation of basic encoding and decoding algorithms of important coding schemes, such as convolutional codes, Hamming codes, BCH codes, Reed–Solomon codes and turbo codes, and their application in digital communication systems. There is a rich theory of ECC that will be touched upon, by referring to the appropriate material. There are many good books dealing with the theory of ECC, for example, references (Lin and Costello 2005), (MacWilliams and Sloane 1977), (Peterson and Weldon 1972), (Blahut 1984), (Bossert 1999), (Wicker 1995), just to cite a few. Readers may wish to consult them before, during or after going through the material in this book. Each chapter describes, using simple and easy-to-follow numerical examples, the basic concepts of a particular coding or decoding scheme, rather than going into the detail of the theory behind it. Basic analysis tools are given to help in the assessment of the error performance of a particular ECC scheme.

The book deals with the *art* of error correcting coding, in the sense that it addresses the need for selecting, implementing and simulating algorithms for encoding and decoding of codes for error correction and detection. New features of the second edition include additional in-text examples as well as new problems at the end of each chapter, intended for use in a course on ECC. A comprehensive bibliography is included, for readers who wish

[1] `http://www.eccpage.com`

[2] Matlab is a registered trademark of The Mathworks, Inc.

to learn more about the beautiful theory that makes it all work. The book is organized as follows. In Chapter 1, the basic concepts of error correction and coding and decoding techniques are introduced. Chapter 2 deals with important and simple-to-understand families of codes, such as the Hamming, Golay and Reed–Muller codes. In Chapter 3, cyclic codes and the important family of BCH codes are described. Finite-field arithmetic is introduced and basic decoding algorithms, such as Berlekamp–Massey, Euclidean and PGZ, are described, and easy to follow examples are given to understand their operation. Chapter 4 deals with Reed–Solomon codes and errors-and-erasures decoding. A comprehensive treatment of the available algorithms is given, along with examples of their operation. In Chapter 5, binary convolutional codes are introduced. Focus in this chapter is on the understanding of the basic structure of these codes, along with a basic explanation of the Viterbi algorithm with Hamming metrics. Important implementation issues are discussed. In Chapter 6, several techniques for modifying a single code or combining several codes are given and illustrated by simple examples. Chapter 7 deals with soft-decision decoding algorithms, some of which have not yet received attention in the literature, such as a soft-output ordered-statistics decoding algorithm. Moreover, Chapter 8 presents a unique treatment of turbo codes, both parallel concatenated and serial concatenated, and block product codes, from a coding theoretical perspective. In the same chapter, low-density parity-check codes are examined. For all these classes of codes, basic decoding algorithms are described and simple examples are given. Finally, Chapter 9 deals with powerful techniques that combine error correcting coding with digital modulation, and several clever decoding techniques are described.

I would like to express my gratitude to the following persons for inspiring this work. Professor Francisco Garcia Ugalde, Universidad Nacional Autónoma de México, for introducing me to the exciting world of error correcting codes. Parts of this book are based on my Bachelor's thesis under his direction. Professor Edward Bertram, University of Hawaii, for teaching me the basics of abstract algebra. Professor David Muñoz, Instituto Technológico y de Estudios Superiores de Monterrey, México, for his kindness and support. Professors Tadao Kasami, Hiroshima City University, Toru Fujiwara, University of Osaka, and Hideki Imai, University of Tokyo, for supporting my stay as a visiting academic researcher in Japan. Dan Luthi and Advait Mogre, LSI Logic Corporation, for many stimulating discussions and the opportunity to experience the process of putting ideas into silicon. Marc P. C. Fossorier of University of Hawaii for his kind help. My former colleague Dr. Misa Mihaljević of Sony Computer Science Laboratories, for pointing out connections between decoding and cryptoanalysis. I would also like to thank wholeheartedly Dr. Mario Tokoro, President of Sony Computer Science Laboratories, and Professor Ryuji Kohno, Yokohama National University, for making it possible for me to have a fine environment in which to write the first edition of this book. In particular, I want to express my eternal gratitude to Professor Shu Lin of University of California at Davis. I am also grateful to the graduate students of San Jose State University who took my course and helped in designing and testing some of the problems in the second edition.

I dedicate this book to Richard W. Hamming, Claude Shannon and Gustave Solomon, three extraordinary gentlemen who greatly impacted the way people live and work today.

Robert H. Morelos-Zaragoza
San Jose, California, USA

Foreword

In modern digital communication and storage systems design, information theory is becoming increasingly important. The best example of this is the appearance and quick adoption of turbo and block product codes in many practical satellite and wireless communication systems. I am pleased to recommend this new book, authored by Dr. Robert Morelos-Zaragoza, to those who are interested in error correcting codes or have to apply them. The book introduces key concepts of error correcting coding (ECC) in a manner that is easy to understand. The material is logically well structured and presented using simple illustrative examples. This, together with the computer programs available on the web site, is a novel approach to teaching the basic techniques used in the design and application of error correcting codes.

One of the best features of the book is that it provides a natural introduction to the principles and decoding techniques of turbo codes, LDPC codes, and product codes, from an algebraic channel coding perspective. In this context, turbo codes are viewed as punctured product codes. With simple examples, the underlying ideas and structures used in the construction and iterative decoding of product codes are presented in an unparalleled manner. The detailed treatment of various algebraic decoding techniques for the correction of errors and erasures using Reed–Solomon codes is also worth a mention. On the applications of ECC in combined channel coding and digital modulation, or coded modulation, the author does a good job in introducing the basic principles that are used in the construction of several important classes of coded modulation systems.

I believe that practitioner engineers and computer scientists will find this book to be both a good learning tool and a valuable reference. The companion ECC web site is a unique feature that is not found anywhere else. Incidentally, this web site was born in my laboratory at the University of Tokyo in 1995, where Dr. Morelos-Zaragoza worked until June of 1997 and did a very good job as my associate researcher, writing many high-quality papers. Robert is polite, modest and hard-working, and is always friendly. In summary, I strongly recommend *The Art of Error Correcting Coding* as an excellent introductory and reference book on the principles and applications of error correcting codes.

Professor Hideki Imai
The University of Tokyo
Tokyo, Japan

The ECC web site

A companion web site for the book *The Art of Error Correcting Coding* has been set up and is located permanently at the following URL address:

`http://the-art-of-ecc.com`

The **ECC web site** contains computer programs written in both C and Matlab[3] to implement algorithms for encoding and decoding of important families of error correcting codes. New scripts to analyze the performance of error correcting coding schemes have been added. Also, an instructor's solutions manual is now available containing the answers to the problems at the end of each chapter. The web site is maintained by the author, to ensure that the domain name remains unchanged. An important advantage of having a companion web site is that it allows the author to post update notes, new computer programs and simulation results relevant to the contents of the book.

The computer programs in the ECC web site are organized in two ways: by topic and by function. In the topical organization of the programs, the logical structure of the book is closely followed, going from simple syndrome-based decoding of linear block codes to more elaborate algebraic decoding over finite fields of BCH and Reed-Solomon codes, passing through Viterbi decoding of convolutional codes and decoding of combinations and constructions of codes, to iterative decoding of turbo and product codes, belief-propagation decoding of low-density parity-check codes and applications in coded modulation techniques. The functional organization of the programs in the ECC web site is intended for readers who already know exactly what they are looking for. In particular, this classification of the programs is followed with respect to the decoding algorithms.

[3]Matlab is a registered trademark of The Mathworks, Inc.

1

Introduction

The history of error correcting coding (ECC) started with the introduction of the Hamming codes (Hamming 1974), at or about the same time as the seminal work of Shannon (1948). Shortly after, Golay codes were invented (Golay 1974). These two first classes of codes are optimal, and will be defined in a subsequent section.

Figure 1.1 shows the block diagram of a canonical digital communications/storage system. This is the famous *Figure 1* in most books on the theory of ECC and digital communications (Benedetto and Biglieri 1999). The information source and destination will include any source coding scheme matched to the nature of the information. The ECC encoder takes as input the information symbols from the source and adds redundant symbols to it, so that most of the errors – introduced in the process of modulating a signal, transmitting it over a noisy medium and demodulating it – can be corrected (Massey 1984; McEliece 1977; Moon 2005).

Usually, the channel is assumed to be such that samples of an additive noise process are added to the modulated symbols (in their equivalent complex baseband representation). The noise samples are assumed to be independent from the source symbols. This model is relatively easy to track mathematically and includes additive white Gaussian noise (AWGN) channels, flat Rayleigh fading channels, and binary symmetric channels (BSC). The case of frequency-selective channels can also be included, as techniques such as spread-spectrum and multicarrier modulation (MCM) effectively transform them into either AWGN channels or flat Rayleigh fading channels.

At the receiver end, the ECC decoder utilizes the redundant symbols and their relationship with the information symbols in order to correct channel errors. In the case of error detection, the ECC decoder can be best thought of as a reencoder of the received information, followed by a check that the redundant symbols generated are the same as those received.

In classical ECC theory, the combination of modulation, noisy medium and demodulation was modeled as a *discrete memoryless channel* with input $\bar{v}$ and output $\bar{r}$. An example of this is binary transmission over an AWGN channel, which is modeled as a *BSC*. This is illustrated in Figure 1.2. The BSC has a probability of channel error p – or transition probability – equal to the probability of a bit error for binary

The Art of Error Correcting Coding, Second Edition Robert H. Morelos-Zaragoza
 ISBN: 0-470-01558-6

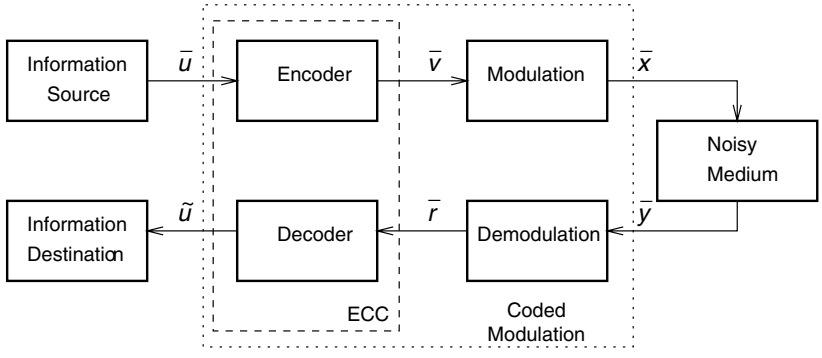


Figure 1.1 A canonical digital communications system.

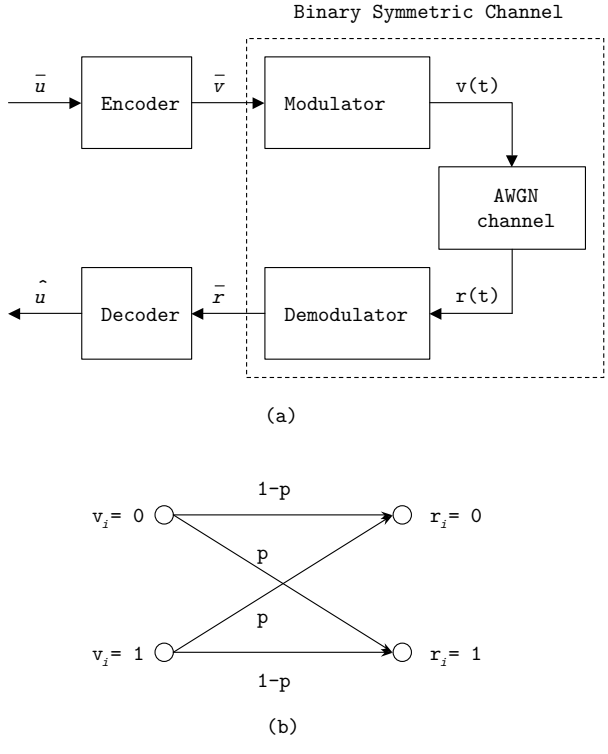


Figure 1.2 A binary communication system over an AWGN channel and corresponding BSC.

signaling over an AWGN channel,

$$p = Q\left(\sqrt{\frac{2E_b}{N_0}}\right), \tag{1.1}$$

where E_b/N_0 is the energy-per-bit-to-noise ratio – also referred to as the bit signal-to-noise ratio (SNR) or SNR per bit – and

$$Q(x) = \frac{1}{\sqrt{2\pi}} \int_x^\infty e^{-z^2/2}\, dz, \tag{1.2}$$

is the Gaussian Q-function. In terms of the *complementary error function*, the Q-function can be written as

$$Q(x) = \frac{1}{2}\ \text{erfc}\left(\frac{x}{\sqrt{2}}\right). \tag{1.3}$$

Equation (1.2) is useful in analytical derivations and Equation (1.3) is used in the computation with C programs or Matlab scripts of performance bounds and approximations.

Massey (1974) suggested considering ECC and modulation as a single entity, known in modern literature as *coded modulation*. This approach provides a higher efficiency and coding gain[1] rather than the serial concatenation of ECC and modulation, by joint design of codes and signal constellations. Several methods of combining coding and modulation are covered in this book, including the following: trellis-coded modulation (TCM) (Ungerboeck 1982) and multilevel coded modulation (MCM) (Imai and Hirakawa 1977). In a coded modulation system, the (soft-decision) channel outputs are directly processed by the decoder. In contrast, in a classical ECC system, the hard-decision bits from the demodulator are fed to a binary decoder.

Codes can be combined in several ways. An example of *serial concatenation* (that is, concatenation in the classical sense) is the following. For years, the most popular concatenated ECC scheme has been the combination of an outer Reed–Solomon (RS) code, through intermediate interleaving, and an inner binary convolutional code. This scheme has been used in numerous applications, ranging from space communications to digital broadcasting of high definition television. The basic idea is that the soft-decision decoder of the convolutional code produces bursts of errors that can be broken into smaller pieces by the deinterleaving process and handled effectively by the RS decoder. RS codes are nonbinary codes that work with symbols composed of several bits, and can deal with multiple bursts of errors. Serial concatenation has the advantage that it requires two separate decoders, one for the inner code and one for the outer code, instead of a single but very complex decoder for the overall code.

This book examines these types of ECC systems. First, basic code constructions and their decoding algorithms, in the Hamming space (that is, dealing with bits), are presented. In the second part of the book, important soft-decision decoding (SDD) algorithms for binary transmission are introduced. These algorithms work over the Euclidean space and achieve a reduction in the required transmitted power per bit of at least 2 dB, compared with Hamming-space (hard-decision) decoders. Several kinds of soft-decision decoders are

[1]Coding gain is defined as the difference in SNR between the coded system and an uncoded system with the same bit rate.

considered, with attention given to their algorithmic aspects (the "how" they work), rather than to their theoretical aspects (the 'why' they work). Finally, combinations of codes and interleaving for iterative decoding and of coding and modulation for bandwidth-efficient transmission are the topic of the last part of the book.

1.1 Error correcting coding: Basic concepts

All error correcting codes are based on the same basic principle: *redundancy* is added to information in order to correct any errors that may occur in the process of transmission or storage. In a basic (and practical) form, redundant symbols are appended to information symbols to obtain a coded sequence or *code word*. For the purpose of illustration, a code word obtained by encoding with a *block code* is shown in Figure 1.3. Such an encoding is said to be *systematic*. Systematic encoding means that the information symbols always appear in the first (leftmost) k positions of a code word. The remaining (rightmost) $n - k$ symbols in a code word are a function of the information symbols, and provide redundancy that can be used for error correction and/or detection purposes[2]. The set of all code sequences is called an *error correcting code*, and will henceforth be denoted by C.

1.1.1 Block codes and convolutional codes

According to the manner in which redundancy is added to messages, ECC can be divided into two classes: block and convolutional. Both types of coding schemes have found practical applications. Historically, convolutional codes have been preferred, apparently because of the availability of the soft-decision Viterbi decoding algorithm and the belief over many years that block codes could not be efficiently decoded with soft-decisions. However, recent developments in the theory and design of SDD algorithms for linear block codes have helped to dispel this belief. Moreover, the best ECC known to date remain block codes (long irregular low-density parity-check (LDPC) codes).

Block codes process the information on a block-by-block basis, treating each block of information bits independently from others. In other words, block coding is a memoryless operation, in the sense that code words are independent from each other. In contrast, the output of a convolutional encoder depends not only on the current input information, but also on previous inputs or outputs, either on a block-by-block or a bit-by-bit basis. For simplicity of exposition, we begin with a study of the structural properties of block codes. Many of these properties are common to both types of codes.

It should be noted that block codes have, in fact, memory, when encoding is thought of as a bit-by-bit process and within a code word. Most recently, the difference between block

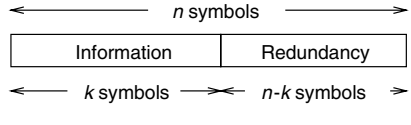


Figure 1.3 A systematic block encoding for error correction.

[2]Note: Without loss of generality, the order of the information and redundancy can be reversed (i.e., the first $n - k$ positions in a code word for redundant symbols and the remaining k positions for information symbols).

and convolutional codes has become less and less well defined, especially after recent advances in the understanding of the trellis structure of block codes, and the tail-biting structure of some convolutional codes. Indeed, colleagues working on convolutional codes sometimes refer to block codes as "codes with time-varying trellis structure." Similarly, researchers working with block codes may consider convolutional codes as "codes with a regular trellis structure."

1.1.2 Hamming distance, Hamming spheres and error correcting capability

Consider an error correcting code C with binary elements. As mentioned above, block codes are considered for simplicity of exposition. In order to achieve error correcting capabilities, not all the 2^n possible binary vectors of length n are allowed to be transmitted. Instead, C is a subset of the n-dimensional binary vector space $V_2 = \{0, 1\}^n$, such that its elements are as far apart as possible.

Consider two vectors $\bar{x}_1 = (x_{1,0}, x_{1,1}, \ldots, x_{1,n-1})$ and $\bar{x}_2 = (x_{2,0}, x_{2,1}, \ldots, x_{2,n-1})$ in V_2. Then the *Hamming distance* between $\bar{x}_1$ and $\bar{x}_2$, denoted $d_H(\bar{x}_1, \bar{x}_2)$, is defined as the number of elements in which the vectors differ,

$$d_H(\bar{x}_1, \bar{x}_2) \stackrel{\Delta}{=} \left| \left\{ i : x_{1,i} \neq x_{2,i}, \quad 0 \leq i < n \right\} \right| = \sum_{i=0}^{n-1} x_{1,i} \oplus x_{2,i}, \tag{1.4}$$

where $|A|$ denotes the number of elements in (or the cardinality of) a set A and $\oplus$ denotes addition modulo-2 (exclusive-OR).

Given a code C, its *minimum Hamming distance*, $d_{\min}$, is defined as the minimum Hamming distance among all possible distinct pairs of code words in C,

$$d_{\min} = \min_{\bar{v}_1, \bar{v}_2 \in C} \{ d_H(\bar{v}_1, \bar{v}_2) | \bar{v}_1 \neq \bar{v}_2 \}. \tag{1.5}$$

Throughout the book, the array $(n, k, d_{\min})$ is used to denote the parameters of a block code of length n, that encodes messages of length k bits and has a minimum Hamming distance $d_{\min}$. The assumption is made that the size of the code is $|C| = 2^k$.

Example 1.1.1 *The simplest error correcting code is a binary repetition code of length 3. It repeats each bit three times, so that a "0" is encoded onto the vector (000) and a "1" onto the vector (111). Since the two code words differ in all three positions, the Hamming distance between them is equal to three. Figure 1.4 is a pictorial representation of this code. The three-dimensional binary space corresponds to the set of* $2^3 = 8$ *vertices of the three-dimensional unit-volume cube. The Hamming distance between code words (000) and (111) equals the number of edges in a path between them. This is equivalent to the number of coordinates that one needs to change to convert (000) into (111), or vice versa. Thus* $d_H((000), (111)) = 3$. *There are only two code words in this case, as a result,* $d_{\min} = 3$.

The binary vector space V_2 is also known as a *Hamming space*. Let $\bar{v}$ denote a code word of an error correcting code C. A *Hamming sphere* $S_t(\bar{v})$, of *radius* t and centered around $\bar{v}$, is the set of vectors in V_2 at a distance less than or equal to t from the center $\bar{v}$,

$$S_t(\bar{v}) = \{ \bar{x} \in V_2 | d_H(\bar{x}, \bar{v}) \leq t \}. \tag{1.6}$$

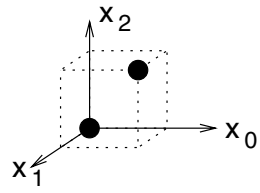


Figure 1.4 A (3,1,3) repetition code in a three-dimensional binary vector space.

Note that the size of (or the number of code words in) $S_t(\bar{v})$ is given by the following expression

$$\left|S_t(\bar{v})\right| = \sum_{i=0}^{t} \binom{n}{i}. \tag{1.7}$$

Example 1.1.2 *Figure 1.5 shows the Hamming spheres of radius* $t = 1$ *around the code words of the* $(3, 1, 3)$ *binary repetition code.*

Note that the Hamming spheres for this code are disjoint, that is, there is no vector in V_2 *(or vertex in the unit-volume three-dimensional cube) that belongs to both* $S_1(000)$ *and* $S_1(111)$. *As a result, if there is a change in any one position of a code word* $\bar{v}$, *then the resulting vector will still lie inside a Hamming sphere centered at* $\bar{v}$. *This concept is the basis of understanding and defining the error correcting capability of a code* C.

The *error correcting capability*, t, of a code C is the largest radius of Hamming spheres $S_t(\bar{v})$ around all the code words $\bar{v} \in C$, such that for all different pairs $\bar{v}_i, \bar{v}_j \in C$, the corresponding Hamming spheres are disjoint, that is,

$$t = \max_{\bar{v}_i, \bar{v}_j \in C} \left\{ \ell | S_\ell(\bar{v}_i) \cap S_\ell(\bar{v}_j) = \emptyset, \quad \bar{v}_i \neq \bar{v}_j \right\}. \tag{1.8}$$

In terms of the minimum distance of C, $d_{\min}$, an equivalent and more common definition is

$$t = \lfloor (d_{\min} - 1)/2 \rfloor, \tag{1.9}$$

where $\lfloor x \rfloor$ denotes the largest integer less than or equal to x.

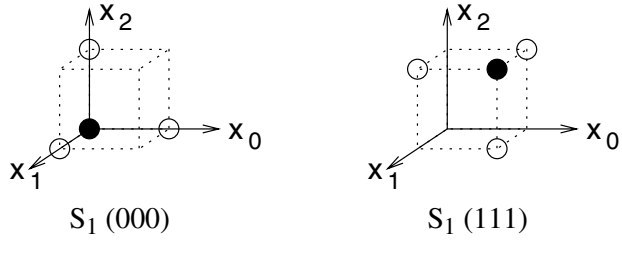


Figure 1.5 Hamming spheres of radius $t = 1$ around the code words of the (3,1,3) binary repetition code.

Note that in order to compute the minimum distance $d_{\min}$ of a block code C, in accordance with Equation (1.5), a total of $2^{k-1}(2^k - 1)$ distances between distinct pairs of code words are needed. This is practically impossible even for codes of relatively modest size, say, $k = 50$ (for which approximately 2^{99} code word pairs need to be examined). One of the advantages of *linear* block codes is that the computation of $d_{\min}$ requires to know the *Hamming weight* of all $2^k - 1$ nonzero code words.

1.2 Linear block codes

As mentioned above, finding a good code means finding a subset of V_2 with elements as far apart as possible. This is very difficult. In addition, even when such a set is found, there is still the problem of *how to assign code words to information messages*.

Linear codes are *vector subspaces* of V_2. This means that encoding can be accomplished by matrix multiplications. In terms of digital circuitry, simple encoders can be built using exclusive-OR gates, AND gates and D flip-flops. In this chapter, the binary vector space operations of sum and multiplication are meant to be the output of exclusive-OR (or modulo 2 addition) and AND gates, respectively. The tables of addition and multiplication for the binary elements in $\{0, 1\}$ are as follows:

a	b	$a + b$	$a \cdot b$
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

which are seen to correspond to the outputs of a binary exclusive-OR gate and an AND gate, respectively.

1.2.1 Generator and parity-check matrices

Let C denote a binary linear $(n, k, d_{\min})$ code. Now, C is a k-dimensional vector subspace, and therefore it has a *basis*, say $\{\bar{v}_0, \bar{v}_1, \ldots, \bar{v}_{k-1}\}$, such that any code word $\bar{v} \in C$ can be represented as a linear combination of the elements on the basis of:

$$\bar{v} = u_0\bar{v}_0 + u_1\bar{v}_1 + \cdots + u_{k-1}\bar{v}_{k-1}, \tag{1.10}$$

where $u_i \in \{0, 1\}$, $1 \leq i < k$. Equation (1.10) can be written in terms of a *generator matrix* G and a *message vector*, $\bar{u} = (u_0, u_1, \ldots, u_{k-1})$, as follows:

$$\bar{v} = \bar{u}G, \tag{1.11}$$

where

$$G = \begin{pmatrix} \bar{v}_0 \\ \bar{v}_1 \\ \vdots \\ \bar{v}_{k-1} \end{pmatrix} = \begin{pmatrix} v_{0,0} & v_{0,1} & \cdots & v_{0,n-1} \\ v_{1,0} & v_{1,1} & \cdots & v_{1,n-1} \\ \vdots & \vdots & \ddots & \vdots \\ v_{k-1,0} & v_{k-1,1} & \cdots & v_{k-1,n-1} \end{pmatrix}. \tag{1.12}$$

Due to the fact that C is a k-dimensional vector space in V_2, there is an $(n-k)$-dimensional *dual space* $C^\top$, generated by the rows of a matrix H, called the *parity-check matrix*, such that $GH^\top = 0$, where $H^\top$ denotes the transpose of H. In particular, note that for any code word $\bar{v} \in C$,

$$\bar{v}H^\top = \bar{0}. \tag{1.13}$$

Equation (1.13) is of fundamental importance in *decoding of linear codes*, as will be shown in section 1.3.2.

A linear code $C^\perp$ that is generated by H is a binary linear $(n, n-k, d_{\min}^\perp)$ code, called the *dual code* of C.

1.2.2 The weight is the distance

As mentioned in section 1.1.2, a nice feature of linear codes is that computing the minimum distance of the code amounts to computing the minimum Hamming weight of its nonzero code words. In this section, this fact is shown. The *Hamming weight*, $\mathrm{wt}_H(\bar{x})$, of a vector $\bar{x} = (x_0, x_1, \ldots, x_{n-1}) \in V_2$ is defined as the number of nonzero elements in $\bar{x}$, which can be expressed as the sum

$$\mathrm{wt}_H(\bar{x}) = \sum_{i=0}^{n-1} x_i. \tag{1.14}$$

From the definition of the Hamming distance, it is easy to see that $\mathrm{wt}_H(\bar{x}) = d_H(\bar{x}, \bar{0})$. For a binary linear code C, note that the distance

$$d_H(\bar{v}_1, \bar{v}_2) = d_H(\bar{v}_1 + \bar{v}_2, \bar{0}) = \mathrm{wt}_H(\bar{v}_1 + \bar{v}_2) = \mathrm{wt}_H(\bar{v}_3), \tag{1.15}$$

where, by linearity, $\bar{v}_1 + \bar{v}_2 = \bar{v}_3 \in C$. As a consequence, the minimum distance of C can be computed by finding the minimum Hamming weight among the $2^k - 1$ nonzero code words. This is simpler than the brute force search among all the pairs of code words, although still a considerable task even for codes of modest size (or dimension k).

1.3 Encoding and decoding of linear block codes

1.3.1 Encoding with G and H

Equation (1.11) gives an encoding rule for linear block codes that can be implemented in a straightforward way. If encoding is to be systematic, then the generator matrix G of a linear block $(n, k, d_{\min})$ code C can be brought to a *systematic form*, G_{sys}, by elementary row operations and/or column permutations. G_{sys} is composed of two submatrices: The k-by-k identity matrix, denoted I_k, and a k-by-$(n-k)$ *parity submatrix* P, such that

$$G_{\mathrm{sys}} = \left(I_k | P\right), \tag{1.16}$$

where

$$P = \begin{pmatrix} p_{0,0} & p_{0,1} & \cdots & p_{0,n-k-1} \\ p_{1,0} & p_{1,1} & \cdots & p_{1,n-k-1} \\ \vdots & \vdots & \ddots & \vdots \\ p_{k-1,0} & p_{k-1,1} & \cdots & p_{k-1,n-k-1} \end{pmatrix}. \tag{1.17}$$

Since $GH^\top = 0_{k,n-k}$, where $0_{k,n-k}$ denotes the k-by-$(n-k)$ all-zero matrix, it follows that the systematic form, H_{sys}, of the parity-check matrix is

$$H_{\text{sys}} = \left(P^\top | I_{n-k}\right). \tag{1.18}$$

Example 1.3.1 *Consider a binary linear* $(4, 2, 2)$ *code with generator matrix*

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

To bring G into systematic form, permute (exchange) the second and fourth columns and obtain

$$G_{\text{sys}} = \begin{pmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix}.$$

Thus, the parity-check submatrix is given by

$$P = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}.$$

It is interesting to note that in this case, the relation $P = P^\top$ *holds*[3]*. From (1.18) it follows that the systematic form of the parity-check matrix is*

$$H_{\text{sys}} = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix}.$$

In the following, let $\bar{u} = (u_0, u_1, \ldots, u_{k-1})$ denote an information message to be encoded and $\bar{v} = (v_0, v_1, \ldots, v_{n-1})$ the corresponding code word in C.

If the parameters of C are such that $k < (n-k)$, or equivalently the *code rate* $k/n < 1/2$, then encoding with the generator matrix is the most economical. The cost considered here is in terms of binary operations. In such a case

$$\bar{v} = \bar{u} G_{\text{sys}} = (\bar{u}, \bar{v}_p), \tag{1.19}$$

where $\bar{v}_p = \bar{u} P = (v_k, v_{k+1}, \ldots, v_{n-1})$ represents the parity-check (redundant) part of the code word.

However, if $k > (n-k)$, or $k/n > 1/2$, then alternative encoding with the parity-check matrix H requires less number of computations. In this case, we have encoding based on Equation (1.13), $(\bar{u}, \bar{v}_p)H^\top = 0$, such that the $(n-k)$ parity-check positions $v_k, v_{k+1}, \ldots, v_{n-1}$ are obtained as follows:

$$v_j = u_0 p_{0,j} + u_1 p_{1,j} + \cdots + u_{k-1} p_{k-1,j}, \qquad k \le j < n. \tag{1.20}$$

Stated in other terms, the systematic form of a parity-check matrix of a linear code has as entries of its rows the coefficients of the parity-check equations, from which the values of the redundant positions are obtained. This fact will be used when LDPC codes are presented, in Section 8.3.

[3]In this case, the code in question is referred to as a *self-dual code*. See also Section 2.2.3.

Example 1.3.2 *Consider the binary linear* $(4, 2, 2)$ *code from Example 1.3.1. Let messages and code words be denoted by* $\bar{u} = (u_0, u_1)$ *and* $\bar{v} = (v_0, v_1, v_2, v_3)$*, respectively. From (1.20) it follows that*

$$\begin{aligned} v_2 &= u_0 + u_1 \\ v_3 &= u_0 \end{aligned}$$

The correspondence between the $2^2 = 4$ *two-bit messages and code words is as follows:*

$$\begin{aligned} (00) &\mapsto (0000) \\ (01) &\mapsto (0110) \\ (10) &\mapsto (1011) \\ (11) &\mapsto (1101) \end{aligned} \tag{1.21}$$

1.3.2 Standard array decoding

In this section, a decoding procedure is presented that finds the closest code word $\bar{v}$ to a received noisy word $\bar{r} = \bar{v} + \bar{e}$. The *error vector* $\bar{e} \in \{0, 1\}^n$ is produced by a BSC, as depicted in Figure 1.6. It is assumed that the crossover probability (or BSC parameter) p is such that $p < 1/2$.

As shown in Table 1.1 a *standard array* (Slepian 1956) for a binary linear $(n, k, d_{\min})$ code C is a table of all possible received vectors $\bar{r}$ arranged in such a way that the closest code word $\bar{v}$ to $\bar{r}$ can be read out. The standard array contains 2^{n-k} rows and $2^k + 1$ columns. The entries of the rightmost 2^k columns of the array contain all the vectors in $V_2 = \{0, 1\}^n$.

In order to describe the decoding procedure, the concept of *syndrome* is needed. The syndrome of a word in V_2 is defined from Equation (1.13) as

$$\bar{s} = \bar{r} H^{\top}, \tag{1.22}$$

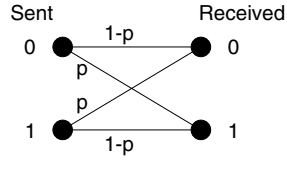


Figure 1.6 A binary symmetric channel model.

Table 1.1 The standard array of a binary linear block code.

$\bar{s}$	$\bar{u}_0 = \bar{0}$	$\bar{u}_2$	$\cdots$	$\bar{u}_{k-1}$
$\bar{0}$	$\bar{v}_0 = \bar{0}$	$\bar{v}_1$	$\cdots$	$\bar{v}_{2^k-1}$
$\bar{s}_1$	$\bar{e}_1$	$\bar{e}_1 + \bar{v}_1$	$\cdots$	$\bar{e}_1 + \bar{v}_{2^k-1}$
$\bar{s}_2$	$\bar{e}_2$	$\bar{e}_2 + \bar{v}_1$	$\cdots$	$\bar{e}_2 + \bar{v}_{2^k-1}$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
$\bar{s}_{2^{n-k}-1}$	$\bar{e}_{2^{n-k}-1}$	$\bar{e}_{2^{n-k}-1} + \bar{v}_1$	$\cdots$	$\bar{e}_{2^{n-k}-1} + \bar{v}_{2^k-1}$

where H is the parity-check matrix of C. That $\bar{s}$ is indeed a set of symptoms that indicate errors is shown as follows. Suppose that a code word $\bar{v} \in C$ is transmitted over a BSC and received as $\bar{r} = \bar{v} + \bar{e}$. The syndrome of $\bar{r}$ is

$$\bar{s} = \bar{r}H^{\top} = (\bar{v} + \bar{e})H^{\top} = \bar{e}H^{\top}, \tag{1.23}$$

where to obtain the last equality Equation (1.13) has been used. Therefore, the computation of the syndrome can be thought of as a *linear transformation* of an error vector.

Standard array construction procedure

1. As the first row, in the positions corresponding to the 2^k rightmost columns, enter all the code words of C, beginning with the all-zero code word in the leftmost position. In the position corresponding to the first column, enter the all-zero syndrome. Let $j = 0$.

2. Let $j = j + 1$. Find the smallest Hamming weight word $\bar{e}_j$ in V_2, not in C, and not included in previous rows. The corresponding syndrome $\bar{s}_j = \bar{e}_j H^{\top}$ is the first (rightmost) entry of the row. The 2^k remaining entries in that row are obtained by adding $\bar{e}_j$ to all the entries in the first row (the code words of C).

3. Repeat the previous step until all vectors in V_2 are included in the array. Equivalently, let $j = j + 1$. If $j < 2^{n-k}$, then repeat previous step, otherwise stop.

Example 1.3.3 *The standard array of the binary linear* $(4, 2, 2)$ *code is the following:*

$\bar{s}$	00	01	10	11
00	0000	0110	1011	1101
11	1000	1110	0011	0101
10	0100	0010	1111	1001
01	0001	0111	1010	1100

Decoding with the standard array proceeds as follows. Let $\bar{r} = \bar{v} + \bar{e}$ be the received word. Find the word in the array and output as decoded message $\bar{u}$ the header of the column in which $\bar{r}$ lies. Conceptually, this process requires storing the entire array and matching the received word to an entry in the array.

However, a simplified decoding procedure can be found by noticing that every row in the array has the same syndrome. Each row of the array, denoted Row_i, with $0 \leq i < 2^{n-k}$, a *coset* of C, is such that $\text{Row}_i = \left\{\bar{e}_i + \bar{v} \middle| \bar{v} \in C\right\}$. The vector $\bar{e}_i$ is known as the *coset leader*.

The syndrome of the elements in the i-th row is given by

$$\bar{s}_i = (\bar{e}_i + \bar{v})H^{\top} = \bar{e}_i H^{\top}, \tag{1.24}$$

which is *independent* of the particular choice of $\bar{v} \in C$. The simplified decoding procedure is: compute the syndrome of the received word $\bar{r} = \bar{e}_{i'} + \bar{v}$,

$$\bar{s}_{i'} = (\bar{e}_{i'} + \bar{v})\, H^{\top} = \bar{e}_{i'} H^{\top},$$

and find $\bar{s}_{i'}$ in the leftmost column of the standard array. Then read out the value of $\bar{e}_{i'}$, from the second column, and add it to the received word to obtain the closest code word $\bar{v}' \in C$ to $\bar{r}$. Therefore instead of $n \times 2^n$ bits, standard array decoding can be implemented with an array of $n \times 2^{n-k}$ bits.

Example 1.3.4 *Consider again the binary linear* $(4, 2, 2)$ *code from Example 1.3.1. Suppose that the code word* $\bar{v} = (0110)$ *is transmitted and that* $\bar{r} = (0010)$ *is received. Then the syndrome is*

$$\bar{s} = \bar{r}H^\top = (0010)\begin{pmatrix} 1 & 1 \\ 1 & 0 \\ 1 & 0 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 \end{pmatrix}.$$

From the standard array of the code, the corresponding coset leader $\bar{e}' = (0100)$ *is found, and therefore, the estimated code word is* $\bar{v}' = \bar{r} + \bar{e}' = (0010) + (0100) = (0110)$. *One error has been corrected! This may sound strange, since the minimum distance of the code is only two, and thus according to (1.9) single error, correction is impossible. However, this can be explained by looking again at the standard array of this code (Example 1.3.3 above). Note that the third row of the array contains two distinct binary vectors of weight one. This means that only three out of a total of four single-error patterns can be corrected. The error above is one of those correctable single-error patterns.*

It turns out that this $(4, 2, 2)$ *code is the simplest instance of a linear* Linear unequal error protection (LUEP) *code (van Gils 1983; Wolf and Viterbi 1996). This LUEP code has a separation vector* $\bar{s} = (3, 2)$*, which means that the minimum distance between any two code words for which the* first *message bit differs is at least three, and that for the* second *message bit is at least two.*

If encoding is *systematic*, then the above procedure gives the estimated message $\bar{u}'$ in the first k positions of $\bar{v}'$. This is a plausible reason for having a systematic encoding.

1.3.3 Hamming spheres, decoding regions and the standard array

The standard array is also a convenient way of understanding the concept of Hamming sphere and error correcting capability of a linear code C, introduced in Section 1.1.2.

By construction, note that the 2^k rightmost columns of the standard array, denoted Col_j, for $1 \leq j \leq 2^k$, contain a code word $\bar{v}_j \in C$ and a set of $2^{n-k} - 1$ words at the smallest Hamming distance from $\bar{v}_j$, that is,

$$\mathrm{Col}_j = \left\{ \bar{v}_j + \bar{e}_i \middle| \bar{e}_i \in \mathrm{Row}_i, \quad 0 \leq i < 2^{n-k} \right\}. \tag{1.25}$$

The sets Col_j are known as the *decoding regions*, in the Hamming space, around each code word $\bar{v}_j \in C$, for $0 \leq j \leq 2^k - 1$. This is to say that if code word $\bar{v}_j \in C$ is transmitted over a BSC and the received word $\bar{r}$ lies in the set Col_j, then it will be successfully decoded into $\bar{v}_j$.

Hamming bound

The set Col_j and the error correcting capability t of code C are related by the Hamming sphere $S_t(\bar{v}_j)$: A binary linear $(n, k, d_{\min})$ code C has decoding regions Col_j that properly contain Hamming spheres $S_t(\bar{v}_j)$, that is, $S_t(\bar{v}_j) \subseteq \mathrm{Col}_j$.

By noticing that the size of Col_j is 2^{n-k}, and using Equation (1.7), we obtain the celebrated *Hamming bound*

$$\sum_{i=0}^{t} \binom{n}{i} \leq 2^{n-k}. \tag{1.26}$$

The Hamming bound has several combinatorial interpretations. One of them is:

> *The number of syndromes, 2^{n-k}, must be greater than or equal to the number of correctable error patterns, $\sum_{i=0}^{t} \binom{n}{i}$.*

Example 1.3.5 *The binary repetition $(3, 1, 3)$ code has generator matrix $G = \begin{pmatrix} 1 & 1 & 1 \end{pmatrix}$ and parity-check matrix*

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

Accordingly, its standard array is the following:

$\bar{s}$	0	1
00	000	111
11	100	011
10	010	101
01	001	110

The four vectors in the second column of the array (i.e., the coset leaders) are the elements of the Hamming sphere $S_1(000)$ in Figure 1.5, which consists of all binary vectors of length three, with Hamming weight less than or equal to one. Similarly, the entries of the third (rightmost) column of the array are the elements of $S_1(111)$. For this code, the Hamming bound (1.26) holds with equality.

Block codes satisfying the bound (1.26) are said to be *perfect codes*. The only perfect nontrivial codes are the binary Hamming $(2^m - 1, 2^m - m - 1, 3)$ codes, the nonbinary Hamming $\left(\frac{q^m-1}{(q-1)}, \frac{q^m-1}{(q-1)} - m - 1, 3\right)$ codes, $q > 2$, the repetition $(n, 1, n)$ codes, the parity-check $(n, n-1, 2)$ codes, the binary Golay $(23, 12, 7)$ code and the ternary Golay $(11, 6, 5)$ code. The extended codes[4] of the Hamming and Golay codes are also perfect.

For nonbinary linear codes, defined over a field of q elements, with $q = p^m$ and $p > 2$ a prime number, the Hamming bound becomes

$$\sum_{i=0}^{n} \binom{n}{i} (q-1)^i \leq q^{n-k}. \tag{1.27}$$

1.4 Weight distribution and error performance

When selecting a particular coding scheme, it is important to assess its error performance. There are several measures of the performance of an ECC scheme. In this section,

[4]These codes are obtained by appending an additional overall parity-check bit, $v_n = v_0 \oplus \cdots \oplus v_{n-1}$, to each code word.

expressions for linear codes are introduced, for three basic channel models: The BSC model, the AWGN channel model and the flat Rayleigh fading channel model.

1.4.1 Weight distribution and undetected error probability over a BSC

The *weight distribution* $\mathcal{W}(C)$ of an error correcting code C, is defined as the set of $n+1$ integers $\mathcal{W}(C) = \{A_i,\ 0 \le i \le n\}$, such that there are A_i code words of Hamming weight i in C, for $i = 0, 1, \ldots, n$.

An expression for the probability of undetected error of a linear code over a BSC is derived next. First, note that the Hamming weight of a word $\bar{v}$, $\mathrm{wt}_H(\bar{v})$ equals the Hamming distance to the all-zero word, $\mathrm{wt}_H(\bar{v}) = d_H(\bar{v}, \bar{0})$. Also, as noted before, the Hamming distance between any two code words $\bar{v}_1$, $\bar{v}_2$ in a linear code C equals the Hamming weight of their difference,

$$d_H(\bar{v}_1, \bar{v}_2) = d_H(\bar{v}_1 + \bar{v}_2, \bar{0}) = \mathrm{wt}_H(\bar{v}_1 + \bar{v}_2) = \mathrm{wt}_H(\bar{v}_3),$$

where, by linearity of C, $\bar{v}_3 \in C$.

The *probability of an undetected error*, denoted $P_u(C)$, is the probability that the received word differs from the transmitted code word but the syndrome equals zero. That is,

$$\bar{s} = (\bar{v} + \bar{e})H^\top = \bar{e}H^\top = 0 \quad \Longleftrightarrow \quad \bar{e} \in C.$$

Therefore, the probability that the syndrome of the received word is zero equals the probability that an error vector is a nonzero code word in C.

With transmission over a BSC, the probability that the error vector $\bar{e}$ has weight i equals the probability that i bits are in error and that the remaining $n-i$ bits are correct. Let $P(\bar{e}, i)$ denote this probability. Then

$$P(\bar{e}, i) = p^i (1-p)^{n-i}.$$

For an undetected error to occur, the error vector $\bar{e}$ must be a nonzero code word. There are A_i vectors of weight i in C. It follows that

$$P_u(C) = \sum_{i=d_{\min}}^{n} A_i\ P(\bar{e}, i) = \sum_{i=d_{\min}}^{n} A_i\ p^i (1-p)^{n-i}. \tag{1.28}$$

Equation (1.28) gives the exact value of $P_u(C)$. Unfortunately, for most codes of practical interest, the weight distribution $\mathcal{W}(C)$ is unknown. In these cases, using the fact that the number of code words of weight i is less than or equal to the total number of words of weight i in the binary space V_2, the following upper bound is obtained:

$$P_u(C) \le \sum_{i=d_{\min}}^{n} \binom{n}{i} p^i (1-p)^{n-i}. \tag{1.29}$$

Expressions (1.28) and (1.29) are useful when an ECC scheme is applied for error detection only, such as in communication systems with feedback and automatic repeat request (ARQ). When a code is employed for error correction purposes, the expressions derived in the next sections are useful.

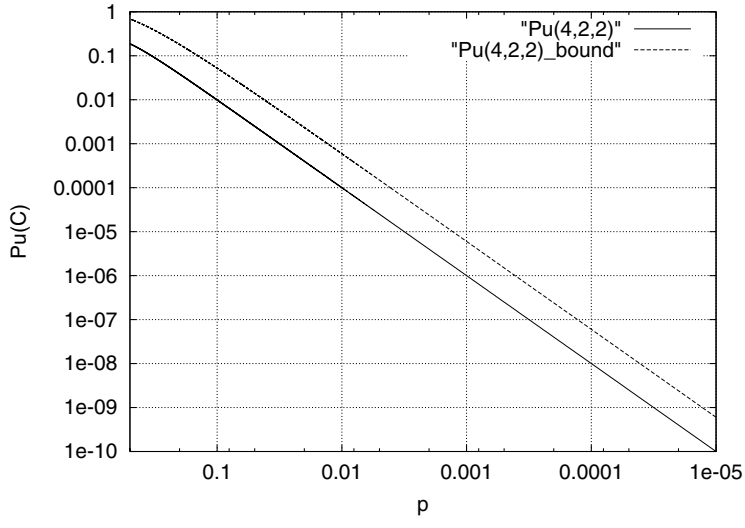


Figure 1.7 Exact value and upper bound on the probability of undetected error for a binary linear (4,2,2) code over a BSC.

Example 1.4.1 *For the binary linear* $(4, 2, 2)$ *code of Example 1.3.2,* $\mathcal{W}(C) = (1, 0, 1, 2, 0)$. *Therefore, Equation (1.28) gives*

$$P_u(C) = p^2(1-p)^2 + 2\ p^3(1-p).$$

Figure 1.7 shows a plot of $P_u(C)$ *compared with the upper bound in the right-hand side (RHS) of (1.29).*

1.4.2 Performance bounds over BSC, AWGN and fading channels

The purpose of this section is to introduce basic channel models that will be considered in the book, as well as the corresponding expressions on the error correction performance of linear codes. Error correction for the BSC is considered first.

The BSC model

For a binary linear code C, as explained in the previous section, a decoding procedure using the standard array will decode a received vector into the closest code word. A decoding error will be made whenever the received vectors falls outside the correct decoding region.

Let L_i denote the number of coset leaders of weight i in the standard array of a linear code C. The *probability of a correct decoding* equals the probability that an error vector is a coset leader and given by

$$P_c(C) = \sum_{i=0}^{\ell} L_i\ p^i(1-p)^{n-i}, \tag{1.30}$$

where ℓ is the largest Hamming weight of a coset leader $\bar{e}$ in the standard array. For perfect codes, $\ell = t$, and

$$L_i = \binom{n}{i}, \quad 0 \leq i \leq t,$$

such that, from the Hamming bound (1.26),

$$\sum_{i=0}^{\ell} L_i = \sum_{i=0}^{t} \binom{n}{i} = 2^{n-k}.$$

For binary codes in general, Equation (1.30) becomes a lower bound on $P_c(C)$, since there exist coset leaders of weight greater than t.

The *probability of incorrect decoding (PID)*, denoted by $P_e(C)$, also known as the *probability of a decoding error*, is equal to the probability of the complement set of the event of correct decoding, that is, $P_e(C) = 1 - P_c(C)$. From Equation (1.30), it follows that

$$P_e(C) = 1 - \sum_{i=0}^{\ell} L_i \; p^i (1-p)^{n-i}. \tag{1.31}$$

Finally, based on the above discussion for $P_c(C)$, the following upper bound is obtained,

$$P_e(C) \leq 1 - \sum_{i=0}^{t} \binom{n}{i} p^i (1-p)^{n-i}, \tag{1.32}$$

which can also be expressed as

$$P_e(C) \leq \sum_{i=t+1}^{n} \binom{n}{i} p^i (1-p)^{n-i}, \tag{1.33}$$

with equality if, and only if, code C is perfect (satisfies the Hamming bound with equality).

Example 1.4.2 *Figure 1.8 shows the values of $P_e(C)$ from (1.33), as a function of the cross-over probability p of a BSC, for the binary repetition (3,1,3) code.*

The AWGN channel model

Perhaps the most important channel model in digital communications is the AWGN channel. In this section, expressions are given for the probability of a decoding error and of a bit error for linear block codes over AWGN channels. Although similar expressions hold for convolutional codes, for clarity of exposition, they are introduced along with the discussion on SDD with the Viterbi algorithm and coded modulation in subsequent chapters. The following results constitute valuable analysis tools for the error performance evaluation of binary coded schemes over AWGN channels.

Consider a binary transmission system, with coded bits in the set $\{0, 1\}$ mapped onto real values $\{+1, -1\}$, respectively, as illustrated in Figure 1.9. In the following, vectors are n-dimensional and the following notation is used to denote a vector: $\bar{x} = (x_0, x_1, \ldots, x_{n-1})$.

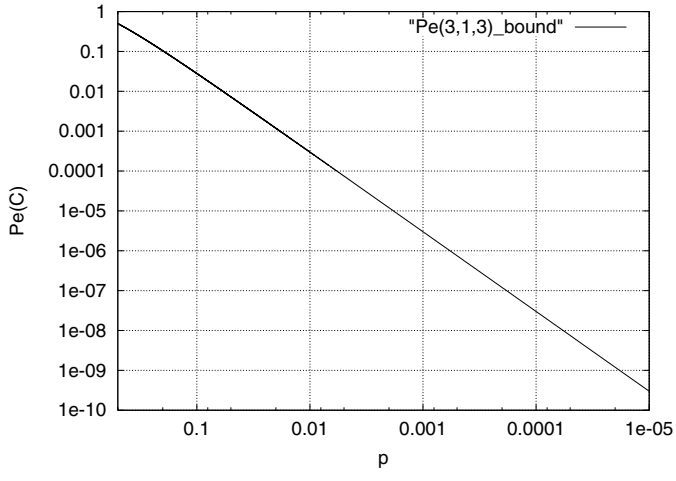


Figure 1.8 Probability of a decoding error for the binary repetition (3,1,3) code.

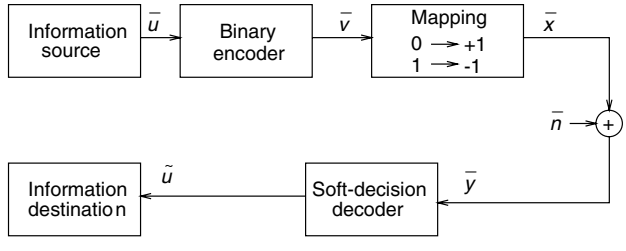


Figure 1.9 Binary coded transmission system over an AWGN channel.

The conditional probability density function (pdf) of the channel output sequence $\bar{y}$, given the input sequence $\bar{x}$ is given by

$$p(\bar{y}|\bar{x}) = p_{\bar{n}}(\bar{y} - \bar{x}) = \prod_{i=0}^{n-1} \frac{1}{\sqrt{\pi N_0}} e^{-\frac{(y_i - x_i)^2}{N_0}}, \tag{1.34}$$

where $p_{\bar{n}}(\bar{n})$ is the pdf of n statistically independent and identically distributed (i.i.d.) noise samples, each of which is Gaussian distributed with mean $\mu_n = 0$ and variance $\sigma_n = N_0/2$, and N_0 is the one-sided power spectral density of the noise. It is easy to show that *maximum-likelihood decoding (MLD)* of a linear code C over this channel selects a sequence $\hat{x}$ that minimizes the *squared Euclidean distance* between the received sequence $\bar{y}$ and $\hat{x}$,

$$D^2(\hat{x}, \bar{y}) \stackrel{\Delta}{=} \sum_{i=0}^{n-1} (x_i - y_i)^2. \tag{1.35}$$

See, for example, Wozencraft and Jacobs (1965), Wilson (1996), Benedetto and Montorsi (1996). It should be noted that a decoder using Equation (1.35) as a *metric* is referred to as

a *soft-decision decoder*, independently of whether or not MLD is performed. In Chapter 7, SDD methods are considered.

The probability of a decoding error with MLD, denoted $P_e(C)$, is equal to the probability that a coded sequence $\bar{x}$ is transmitted and the noise vector $\bar{n}$ is such that the received sequence $\bar{y} = \bar{x} + \bar{n}$ is closer to a different coded sequence $\hat{x} \in C$, $\hat{x} \neq \bar{x}$. For a linear code C, it can be assumed that the all-zero code word is transmitted. Then $P_e(C)$ can be upper bounded, based on the *union bound* (Clark and Cain 1981) and the *weight distribution* $\mathcal{W}(C)$, as follows:

$$P_e(C) \leq \sum_{w=d_{\min}}^{n} A_w \; Q\left(\sqrt{2wR\frac{E_b}{N_0}}\right), \tag{1.36}$$

where $R = k/n$ is the code rate, E_b/N_0 is the *energy-per-bit to noise ratio* (or SNR per bit) and $Q(x)$ is given by (1.2).

Figure 1.10 shows the evaluation of expressions for hard-decision decoding (HDD) (1.32) and SDD (1.36) for the binary (3,1,3) code. *HDD* means using a decoder for the BSC which is fed by the outputs from a binary demodulator. The equivalent BSC has a crossover probability equal to (Proakis 2001; Wozencraft and Jacobs 1965)

$$p = Q\left(\sqrt{2R\frac{E_b}{N_0}}\right).$$

Note that in this particular case, since the code is perfect and contains only two code words, both expressions are exact, not upper bounds. Figure 1.10 also serves to illustrate the fact that soft-decision decoding performs better than HDD, in the sense of requiring less transmitted power to achieve the same $P_e(C)$. The difference (in dB) between the corresponding SNR per bit is commonly referred to as *coding gain*.

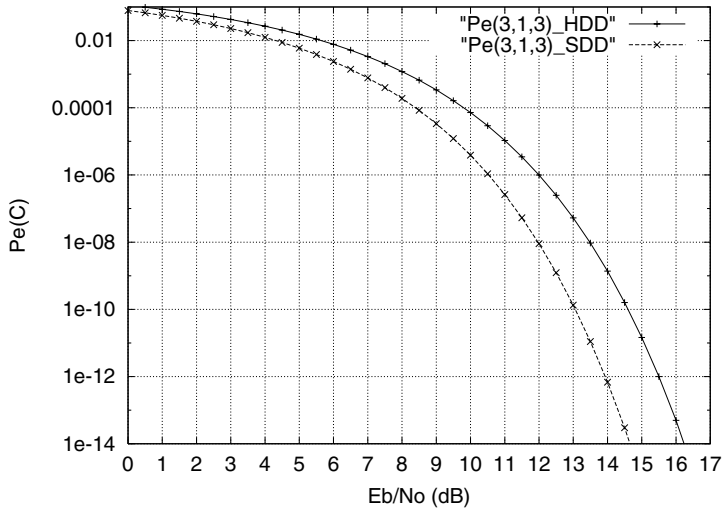


Figure 1.10 Probability of a decoding error for HDD and SDD of a binary (3,1,3) code with binary transmission over an AWGN channel.

In Fossorier *et al.* (1998), the authors show that for *systematic* binary linear codes with binary transmission over an AWGN channel, the *probability of a bit error*, denoted $P_b(C)$, has the following upper bound:

$$P_b(C) \leq \sum_{w=d_{\min}}^{n} \frac{wA_w}{n} Q\left(\sqrt{2wR\frac{E_b}{N_0}}\right). \tag{1.37}$$

Interestingly, besides the fact that the above bound holds only for systematic encoding, the results in Fossorier *et al.* (1998) show that *systematic encoding minimizes the probability of a bit error*. This means that systematic encoding is not only desirable, but actually optimal in the above sense.

Example 1.4.3 *Let C be a linear binary (6,3,3) code with generator and parity-check matrices*

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

respectively. The weight distribution of this code is $\mathcal{W}(C) = \{1, 0, 0, 4, 3, 0, 0\}$, which can be verified by direct computation of all the code words $\bar{v} = (\bar{u}, \bar{v}_p)$:

$\bar{u}$	$\bar{v}_p$
000	000
001	101
010	011
011	110
100	110
101	011
110	101
111	000

In this particular case, MLD can be performed by simply computing the squared Euclidean distance, Equation (1.35), between the received sequence and all of the eight candidate code words. The decoded code word is selected as that with the smallest distance. In Chapter 7, efficient methods for MLD and near-MLD of linear block codes are presented. Figure 1.11 shows simulations and union bounds with hard-decision and soft-decision MLD decoding with binary transmission over an AWGN channel.

The flat Rayleigh fading channel model

Another important channel model is that of flat Rayleigh fading. Fading occurs in wireless communication systems in the form of a time-varying distortion of the transmitted signal. In this book, we consider the case of flat Rayleigh fading. The term "flat" refers to the fact that the channel is not frequency selective, so that its transfer function in the frequency domain is constant (Benedetto and Montorsi 1996; Proakis 2001; Wozencraft and Jacobs 1965).

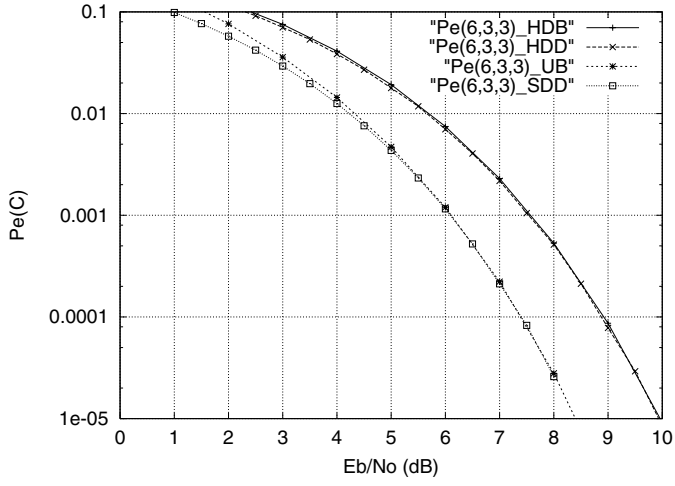


Figure 1.11 Simulations and union bounds for the binary (6,3,3) code. Binary transmission over an AWGN channel.

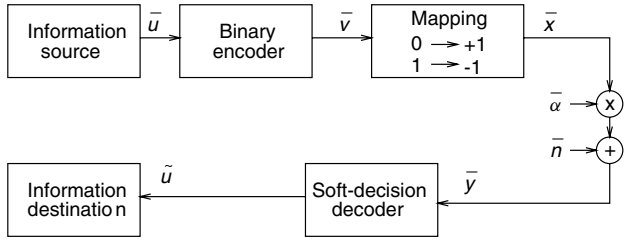


Figure 1.12 Binary coded transmission system over a flat Rayleigh fading channel.

As a result, a (component-wise) multiplicative distortion is present in the channel, as shown in the model depicted in Figure 1.12, where $\bar{\alpha}$ is a vector with n component i.i.d. random variables α_i, $0 \leq i < n$, each having a Rayleigh pdf,

$$p_{\alpha_i}(\alpha_i) = \alpha_i \, e^{-\alpha_i^2/2}, \qquad \alpha_i \geq 0. \tag{1.38}$$

With this pdf, the average SNR per bit equals E_b/N_0 (i.e., that of the AWGN), since the second moment of the fading amplitudes is $E\{\alpha_i^2\} = 1$.

In evaluating the performance of a binary linear code over a flat Rayleigh fading channel, a conditional probability of a decoding error, $P_e(C|\bar{\alpha})$, or of a bit error, $P_b(C|\bar{\alpha})$, is computed. The unconditional error probabilities are then obtained by integration over a product of α_i, with pdf given by Equation (1.38).

The conditional probabilities of error are identical to those obtained for binary transmission over an AWGN channel. The main difference is that the arguments in the $Q(x)$ function, which correspond to the pairwise probability of a decoding error, are now weighted by the fading amplitudes α_i. Considering a coded binary transmission system without *channel*

state information (CSI), we have that

$$P_e(C|\bar{\alpha}) \leq \sum_{w=d_{\min}}^{n} A_w \; Q\left(\sqrt{\frac{1}{w}\Delta_w^2 2R\frac{E_b}{N_0}}\right), \tag{1.39}$$

with

$$\Delta_w = \sum_{i=1}^{w} \alpha_i. \tag{1.40}$$

Finally, the probability of a decoding error with binary transmission over a flat Rayleigh fading channel is obtained by taking the expectation with respect to Δ_w,

$$P_e(C) \leq \sum_{w=d_{\min}}^{n} A_w \int_0^{\infty} Q\left(\sqrt{\frac{1}{w}\Delta_w^2 2R\frac{E_b}{N_0}}\right) p_{\Delta_w}(\Delta_w) d\Delta_w. \tag{1.41}$$

There are several methods to evaluate or further upper bound expression (1.41). One is to evaluate numerically (1.41) by *Monte Carlo (MC) integration*, with the approximation

$$P_e(C) \lesssim \frac{1}{N}\sum_{\ell=1}^{N}\sum_{w=d_{\min}}^{n} A_w \; Q\left(\sqrt{2\Delta_w(\ell)R\frac{E_b}{N_0}}\right), \tag{1.42}$$

where $\Delta_w(\ell)$ denotes the sum of the squares of w i.i.d. random variables with Rayleigh distribution, given by (1.40), generated in the ℓ-th outcome of a computer program, and N is a sufficiently large number that depends on the range of values of $P_e(C)$. A good rule of thumb is that N should be at least 100 times larger than the inverse of $P_e(C)$. (See Jeruchim *et al.* (1992), pp. 500–503.)

Another method is to bound the Q-function by an exponential function (see, e.g., (Wozencraft and Jacobs 1965), pp. 82–84) and to perform the integration, or to find a *Chernoff bound*. This approach results in a loose bound that, however, yields a closed expression (see also Wilson (1996), p. 526, and Benedetto and Montorsi (1996), p. 718):

$$P_e(C) \leq \sum_{w=d_{\min}}^{n} A_w \; \frac{1}{\left(1+\frac{RE_b}{N_0}\right)^w}. \tag{1.43}$$

The bound (1.43) is useful in cases where a first-order estimation of the code performance is desired.

Example 1.4.4 *Figure 1.13 shows computer simulation results SIM of the binary (3,1,3) code over a flat Rayleigh fading channel. Note that the MC integration of the union bound gives exactly the actual error performance of the code, since there is only one term in the bound. Also note that the Chernoff bound is about 2 dB away from the simulations, at an SNR per bit $E_n/N_0 > 18$ dB.*

Example 1.4.5 *Figure 1.14 shows the results of computer simulations of decoding the binary (6,3,3) code of Example 1.4.3, over a flat Rayleigh fading channel. In this case, the union bound is relatively loose at low values of E_b/N_0 due to the presence of more terms in the bound. Again, it can be seen that the Chernoff bound is loose by about 2 dB, at an SNR per bit $E_n/N_0 > 18$ dB.*

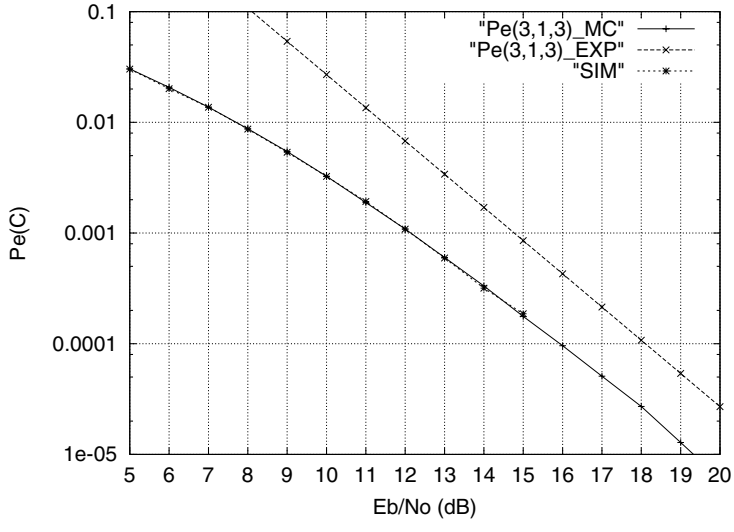


Figure 1.13 Simulation results (SIM), compared with the Monte Carlo union bound (Pe(3,1,3)_MC) and the Chernoff bound (Pe(3,1,3)_EXP), for the binary (3,1,3) code. Binary transmission over a flat Rayleigh fading channel.

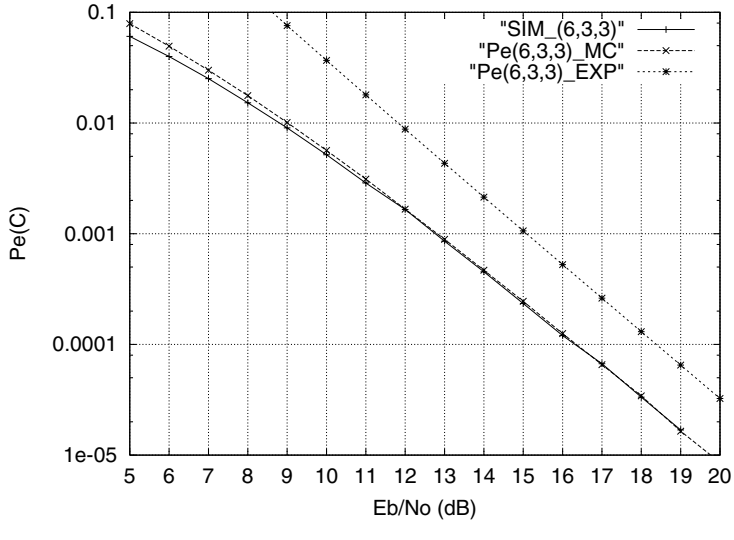


Figure 1.14 Simulation results (SIM_(6,3,3)), compared with the Monte Carlo union bound (Pe(6,3,3)_MC) and the Chernoff bound (Pe(6,3,3)_EXP), for the binary (6,3,3) code. Binary transmission over a flat Rayleigh fading channel.

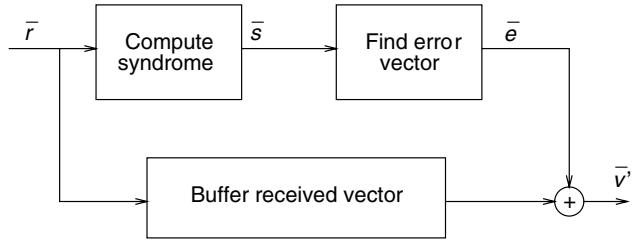


Figure 1.15 General structure of a hard-decision decoder of a linear block code for the BSC model.

1.5 General structure of a hard-decision decoder of linear codes

In this section, the general structure of a hard-decision decoder for linear codes is summarized. Figure 1.15 shows a simple block diagram of the decoding process. Note that since hard decisions are assumed, bits at the output of a demodulator are fed into a decoder designed for a BSC.

Let $\bar{v} \in C$ denote a transmitted code word. The decoder has at its input a noisy received vector $\bar{r} = \bar{v} + \bar{e}$. A two-step decoding procedure of a linear code is:

- Compute the syndrome $\bar{s} = \bar{r}H^{\top}$. On the basis of the code properties, the syndrome is a linear transformation of the error vector introduced in the channel,

$$\bar{s} = \bar{e}H^{\top}, \tag{1.44}$$

- On the basis of the syndrome $\bar{s}$, estimate the most likely error vector $\bar{e}$ and subtract it (modulo 2 addition in the binary case) from the received vector.

Although most practical decoders will not perform the above procedure as stated, it is nonetheless instructive to think of a hard-decision decoder as implementing a method of solving equation (1.44). Note that any method of solving this equation constitutes a decoding method. For example, one could attempt to solve the key equation by finding a pseudoinverse of $H^{\top}$, denoted $(H^{\top})^{+}$, such that $H^{\top}(H^{\top})^{+} = I_n$, and the solution

$$\bar{e} = \bar{s}(H^{\top})^{+}, \tag{1.45}$$

has the *smallest Hamming weight possible*. As easy as this may sound, it is a formidable task. This issue will be visited again when discussing decoding methods for Bose-Chauduri-Hocquenghem (BCH) and Reed-Solomon (RS) codes.

Problems

1. Write a computer program to simulate a binary communication system with binary phase shift keying (BPSK) modulation over an AWGN channel. (Hint: Generate random values $S \in \{-1, +1\}$ so that $E_b = 1$. To generate a sample W of AWGN, first

generate a Gaussian random variable N of zero mean and unit variance. The transformation $W = \sigma_W N$ then results in a Gaussian random variable W of zero mean and variance $\sigma_W^2 = N_0/2$. Given a value of E_b/N_0 *in dB*, compute $N_0 = \sqrt{10^{(-(E_b/N_o)/10)}}$. The receiver estimates (RE) the value $\hat{S}$ based on the sign of the received value $R = S + W$. An error event occurs whenever $\hat{S} \neq S$.)

2. Prove that in general a total of $2^{k-1}(2^k - 1)$ code word pairs need to be examined in a block code to determine its minimum distance.

3. Prove that $G_{\text{sys}} H_{\text{sys}}^T = 0_{k,n-k}$.

4. Prove the nonbinary version of the Hamming bound, that is, inequality (1.27).

5. A binary linear block code C has a generator matrix

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

   (a) Find the weight distribution $\mathcal{W}(C)$.

   (b) Using elementary matrix operations, bring G into a systematic form G' that generates a binary linear block code C'.

   (c) Using the weight distribution $\mathcal{W}(C')$, argue that G and G' generate the same linear code, up to a permutation in bit positions.

6. A binary linear block code C has a generator matrix

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

   (a) What are the values of length n and dimension k of this code?

   (b) Find the weight distribution $\mathcal{W}(C)$ and use it to determine the probability of an undetected error.

   (c) Determine the error correcting capability t of C.

   (d) Find a parity-check matrix H of C.

   (e) Find the standard array of C, based on H of part (c).

   (f) Use standard array decoding to find the closest code word to the received word $\bar{r} = (11011)$.

7. Carefully sketch circuits used in encoding and decoding of a binary linear (5,3,2) code with generator matrix

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

8. Prove that the decoding region Col_j centered around a code word $\bar{v}_j \in C$ of a linear code C contains the Hamming sphere $\mathcal{S}_t(\bar{v}_j)$, that is,

$$\mathcal{S}_t(\bar{v}_j) \subseteq \mathrm{Col}_j$$

9. Use the Hamming bound to disprove the existence of a binary linear $(32, 24, 5)$ code.

10. A network designer needs to correct up to 3 random errors in 64 information bits. What is the minimum amount of redundant bits required?

11. A memory system uses a bus with 64 information bits and 16 redundant bits. With parallel processing (80 bits at a time), what is that maximum number of errors that can be corrected?

12. Determine the smallest length of a binary linear code capable of correcting any two random bit errors in 21 information bits.

13. Let C be a binary linear $(5, 2, d_{\min})$ code.

    (a) With the aid of the Hamming bound, determine the minimum Hamming distance $d_{\min}$ of C.

    (b) Find a generator matrix and a parity-check matrix of C.

    (c) Determine the exact probability of a decoding error $P_e(C)$ over a BSC with crossover probability p.

14. Using the union bound on the error performance of a linear block code with binary transmission over an AWGN channel, show that the binary $(3, 1, 3)$ repetition code offers no advantage over uncoded BPSK modulation. Generalize the result to all binary $(n, 1, n)$ repetition codes and comment on the basic trade-off involved.

15. Show that the binary $(3, 1, 3)$ repetition code C outperforms BPSK modulation with transmission over a flat Rayleigh fading channel. Specifically, show that the slope of the bit-error-rate (BER) curve of C is three times more negative than that of uncoded binary phase shift keying (BPSK) modulation. (Thus, C has a diversity order equal to three.)

16. Compare the probability of a bit error, with binary modulation over an AWGN channel, of a binary $(3, 1, 3)$ repetition code and the binary linear $(4, 2, 2)$ code introduced in Example 1.3.1. What is the difference in performance measured in dB?

17. Compare the performance of the two codes in the previous problem with binary transmission over a Rayleigh fading channel.

18. (Peterson and Weldon (1972)) Given a binary linear (n_1, k, d_1) code and a binary linear (n_2, k, d_2) code, construct a binary linear $(n_1 + n_2, k, d)$ code with $d \geq d_1 + d_2$.

2

Hamming, Golay and Reed–Muller codes

In this chapter, important cases of linear binary codes are introduced. They serve to introduce more error correcting coding (ECC) concepts, as well as clever decoding algorithms. Hamming codes are perhaps the most widely known class of block codes, with the possible exception of Reed–Solomon codes. As mentioned in Chapter 1, Hamming codes are optimal in the sense that they require the smallest amount of redundancy, for a given block length, to correct any single error. The binary Golay code is the only other nontrivial instance of an optimal triple-error correcting code. (The only other binary optimal codes are repetition and single parity-check (SPC) codes.) Reed–Muller (RM) codes can be defined as codes with an elegant combinatorial definition that are easy to decode.

2.1 Hamming codes

From Equation (1.13), any code word $\bar{v}$ in a linear $(n, k, d_{\min})$ block code C satisfies

$$\bar{v}H^{\top} = \bar{0}. \tag{2.1}$$

A useful interpretation of this equation is that *the maximum number of linearly independent columns of the parity-check matrix H of C is equal to $d_{\min} - 1$.*

In the binary case, for $d_{\min} = 3$, the above equation translates into the sum of any two columns of H not equal to the all-zero vector. Suppose that the columns of H are binary vectors of length m. There are up to $2^m - 1$ possible nonzero distinct columns. Therefore, the length of a binary single-error correcting code is given by

$$n \leq 2^m - 1.$$

This inequality is precisely the Hamming bound (1.26) for an error correcting code of length n, with $n - k = m$ and $t = 1$. Consequently, a code achieving this bound with equality is known as a *Hamming code*.

The Art of Error Correcting Coding, Second Edition Robert H. Morelos-Zaragoza
 ISBN: 0-470-01558-6

Example 2.1.1 *With $m = 3$, we obtain the Hamming $(7, 4, 3)$ code, with parity-check matrix*

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

In the rest of this section, an encoding and decoding algorithm is given that is suitable for software implementation and for simulation purposes. Later in the book, effective *soft-decision* decoding algorithms are given, which can be used over real-valued channels such as the additive white Gaussian noise (AWGN) channel and the Rayleigh fading channel.

As noted before, Hamming codes contain the property of their parity-check matrix H having different columns, for each and every one. If a single error occurs, in position j, $1 \leq j \leq n$, then the syndrome of the received vector equals the column of H in the position in which the error occurred. Let $\bar{e}$ denote the error vector added in the transmission of a code word over a binary symmetric channel (BSC) channel, and assume that all of its components are equal to zero except for the j-th component, $e_j = 1$. Then, the syndrome of the received word equals

$$\bar{s} = \bar{r}H^\top = \bar{e}H^\top = \bar{h}_j, \tag{2.2}$$

where $\bar{h}_j$ denotes the j-th column of H, and $1 \leq j \leq n$.

2.1.1 Encoding and decoding procedures

From Equation (2.2) above, it follows that if it is possible to express the columns of H as the binary representation of integers, then the value of the syndrome directly gives the position of the error. This is the idea in the algorithms for encoding and decoding presented below. The columns of the parity-check matrix H are expressed as binary representations of integer numbers i in the range $[1, n]$ and in increasing order. Let the resulting matrix be denoted by H'. Clearly, the code associated with H' is equivalent to the original Hamming code with the parity-check matrix H, up to a permutation (or exchange) of positions.

The parity-check matrix in its systematic form contains the $(n-k) \times (n-k)$ identity matrix, I_{n-k}, as in Equation (1.18). Clearly, when expressing H' with columns equal to the binary representation of the (integer) column number, the identity matrix is contained in those columns of H' that correspond to even powers of 2, that is, of the form 2^ℓ, $0 \leq \ell < m$. This is illustrated in the example below.

Example 2.1.2 *Let $m = 3$. Then a systematic parity-check matrix is*

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

Matrix H' is given by the binary representation of integers 1 to 7 (in the following, the topmost part corresponds to the least-significant bit (LSB) in the binary representation of an integer):

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

and matrix I_{n-k} is contained in columns 1, 2 and 4.

In general, for a $(2^m - 1, 2^m - 1 - m, 3)$ Hamming code, the identity matrix is contained in column numbers $1, 2, 4, \ldots, 2^{m-1}$ of H'.

Encoding

Encoding proceeds as dictated by Equation (1.20) of Chapter 1. When computing the parity check bit p_j, for $1 \leq j \leq m$, the column position numbers are examined and those which are not powers of two correspond to message positions; furthermore, the corresponding message bits are included in the computation. This encoding procedure is somewhat more complicated than that of a systematic Hamming code. In return, however, decoding is extremely simple. Depending on the application, this approach may prove the most appropriate, since it is usually decoding that is required to be very fast.

Decoding

Once the code words have been computed in accordance to matrix H', the decoding is easy. The syndrome (2.2) equals the position number in which the error occurred! In a decoding procedure, after the computation of syndrome s, when regarded as an integer, the erroneous position is then corrected as

$$v_s = v_s \oplus 1,$$

where $\oplus$ denotes exclusive-or (i.e., $0 \oplus 0 = 0$, $0 \oplus 1 = 1$, $1 \oplus 0 = 1$ and $1 \oplus 1 = 0$).

The program `hamming.c` in the ECC web site implements the above encoding and decoding procedures for binary Hamming codes.

2.2 The binary Golay code

Golay (Golay 1974) noticed that

$$\sum_{i=0}^{3} \binom{23}{i} = 2^{11}.$$

This equality shows the possible existence of a perfect binary (23, 12, 7) code, with $t = 3$, that is, capable of correcting all possible patterns of three errors in 23 bit positions, at the most. In his paper, Golay gave a generator matrix of such a triple-error correcting binary code.

Because of its relatively small length (23), dimension (12) and number of redundant bits (11), the binary (23, 12, 7) Golay code can be encoded and decoded simply by using look-up tables (LUTs). The program `golay23.c` in the ECC web site uses a 16 K $\times$ 23 bits encoding table and an 8 K $\times$ 23 bits decoding table.

2.2.1 Encoding

Encoding is based on an LUT that contains a list of all the $2^{12} = 4096$ code words, which are directly indexed by the data. Let $\bar{u}$ denote a 12-bit binary vector representing the data to be encoded, and let $\bar{v}$ denote the corresponding 23-bit code word. The encoder LUT is

constructed by generating all the 4096 12-bit vectors and by computing the syndrome of a pattern for which the 12 most-significant bit (MSB) equal to the information bits and the 11 LSB equal to zero. The 11-bit syndrome then becomes the LSB part of the code word.

The LUT is a one-to-one mapping from $\bar{u}$ onto $\bar{v}$, which can be expressed as

$$\bar{v} = \text{LUT}(\bar{u}) = (\bar{u}, \text{get\_syndrome}(\bar{u}, \bar{0})). \tag{2.3}$$

In the construction of the encoder LUT, advantage is taken from the cyclic nature of the Golay code. Its *generator polynomial*[1] is

$$g(x) = x^{11} + x^{10} + x^6 + x^5 + x^4 + x^2 + 1,$$

which in *hexadecimal notation* equals `C75`. This polynomial is used to generate the syndrome in the procedure "get_syndrome" indicated in Equation (2.3) above.

2.2.2 Decoding

Recall from Chapter 1, Figure 1.15, that the decoder's task is to estimate the most-likely (i.e., least Hamming weight) error vector $\bar{e}$ from the received vector $\bar{r}$. The decoder for the Golay code is based on an LUT that accepts as input the syndrome $\bar{s}$ of the received vector $\bar{r}$, and outputs the error vector $\bar{e}$.

The procedure to construct the decoder LUT is as follows:

1. Generate all possible error patterns $\bar{e}$ of Hamming weight less than or equal to three;
2. For each error pattern, compute the corresponding syndrome $\bar{s} = \text{get\_syndrome}(\bar{e})$;
3. Store at location $\bar{s}$ of the LUT, the error vector $\bar{e}$,

$$\text{LUT}(\bar{s}) = \bar{e}.$$

With the decoder LUT, upon reception of a corrupted received word $\bar{r}$, the correction of up to three bit errors is accomplished by the following:

$$\hat{v} = \bar{r} \oplus \text{LUT}\left(\text{get\_syndrome}(\bar{r})\right),$$

where $\hat{v}$ denotes the corrected word.

2.2.3 Arithmetic decoding of the extended (24, 12, 8) Golay code

In this section, a decoding procedure for the extended $(24, 12, 8)$ Golay code, C_{24}, is presented based on an arithmetic decoding algorithm (Vanstone and van Oorschot 1989; Wicker 1995). The algorithm utilizes the rows and columns of the parity submatrix B in the parity-check matrix $H = \left(B \mid I_{12\times 12}\right)$. Note that an extended $(24, 12, 8)$ Golay code, C'_{24}, which is equivalent to C_{24} up to a permutation in the bit positions, can be obtained by adding an overall parity-check bit at the end of each code word in the $(23, 12, 7)$ Golay code.

In hexadecimal notation, the 12 rows of B, denoted row_i, $1 \leq i \leq 12$, are as follows:

```
0x7ff, 0xee2, 0xdc5, 0xb8b, 0xf16, 0xe2d,
0xc5b, 0x8b7, 0x96e, 0xadc, 0xdb8, 0xb71.
```

[1]The definition of a generator polynomial is given in Chapter 3.

It is interesting to note that submatrix B of the parity-check matrix of C_{24} satisfies $B = B^\top$. This means that code C_{24} is a *self-dual code*. Details of self-dual codes are not covered in this book. Interested readers are referred to (MacWilliams and Sloane 1977; Wicker 1995).

In program `golay24.c`, the encoding is performed by recurrence with H, as indicated in Equation (1.20). As before, let $\mathrm{wt}_H(\bar{x})$ denote the Hamming weight of a vector $\bar{x}$.

The decoding steps in arithmetic decoding of the extended (24, 12, 8) Golay code are as follows (Vanstone and van Oorschot 1989; Wicker 1995):

1. Compute the syndrome $\bar{s} = \bar{r}H^\top$.
2. If $\mathrm{wt}_H(\bar{s}) \leq 3$, then set $\bar{e} = (\bar{s}, \bar{0})$ and go to step 8.
3. If $\mathrm{wt}_H(\bar{s} + \mathrm{row}_i) \leq 2$, then set $\bar{e} = (\bar{s} + \mathrm{row}_i, \bar{x}_i)$, where $\bar{x}_i$ is a 12-bit vector with only the i-th coordinate nonzero.
4. Compute $\bar{s}B$.
5. If $\mathrm{wt}_H(\bar{s}B) \leq 3$, then set $\bar{e} = (\bar{0}, \bar{s}B)$ and go to step 8.
6. If $\mathrm{wt}_H(\bar{s}B + \mathrm{row}_i) \leq 2$, then set $\bar{e} = (\bar{x}_i, \bar{s}B + \mathrm{row}_i)$, with $\bar{x}_i$ defined as above, and go to step 8.
7. $\bar{r}$ is corrupted by an uncorrectable error pattern, set error failure flag. End of decoding.
8. Set $\hat{c} = \bar{r} + \bar{e}$. End of decoding.

2.3 Binary Reed–Muller codes

Binary RM codes constitute a family of error correcting codes that are easy to decode using *majority-logic (ML) circuits*. In addition, codes in this family are known to have relatively simple and highly structured trellises (Lin *et al.* 1998). Details about trellises of linear block codes can be found in Chapter 7.

An elegant definition of binary RM code is obtained with the use of binary polynomials (or Boolean functions). With this definition, RM codes become close relatives of Bose–Chauduri–Hocquenghem (BCH) codes and Reed–Solomon codes, all members of the class of *polynomial codes*.[2]

2.3.1 Boolean polynomials and RM codes

This section closely follows the development of (MacWilliams and Sloane 1977). Let

$$f(x_1, x_2, \ldots, x_m)$$

denote a *Boolean function* on m binary-valued variables $x_1, x_2, \ldots, x_m$. It is well known that such a function can be specified by a *truth table*. The truth table lists the value of f for all 2^m combinations of values of its arguments. All the usual Boolean operations such as AND (conjunction), OR (disjunction) and NOT (negation) can be defined in a Boolean function.

[2]Polynomial codes are presented in Section 3.4.

Example 2.3.1 *Consider the function* $f(x_1, x_2)$ *with the following truth table:*

x_2	0	0	1	1
x_1	0	1	0	1
$f(x_1, x_2)$	0	1	1	0

Then,

$$f(x_1, x_2) = (x_1 \; AND \; NOT(x_2)) \;\; OR \;\; (NOT(x_1) \; AND \; x_2) \, .$$

Associated with each Boolean function f, let $\bar{f}$ denote the binary vector of length 2^m which is obtained from evaluating f at all possible 2^m values of the m variables $x_1, x_2, \ldots, x_m$. In Example 2.3.1, $\bar{f} = (0110)$, where the convention taken for ordering the bit positions of $\bar{f}$ is in accordance with a binary representation of integers, with x_1 being the LSB and x_m the MSB.

Also note that a Boolean function can be written directly from its truth table to get the *disjunctive normal form* (DNF). Using the DNF, any Boolean function can be expressed as the sum[3] of 2^m elementary functions: $1, x_1, x_2, \ldots, x_m, x_1x_2, \ldots, x_1x_2 \ldots x_m$, such that

$$\bar{f} = \bar{1} + a_1\bar{x}_1 + a_2\bar{x}_2 + \cdots + a_m\bar{x}_m + a_{12}\bar{x}_1\bar{x}_2 + \cdots + a_{12\cdots m}\bar{x}_1\bar{x}_2 \ldots \bar{x}_m, \tag{2.4}$$

where $\bar{1}$ is added to account for independent terms (degree 0). For Example 2.3.1 above, $\bar{f} = \bar{x}_1 + \bar{x}_2$.

A binary RM $(2^m, k, 2^{m-r})$ code, denoted RM(r, m), is defined as the set of vectors associated with all Boolean functions of degree up to r in m variables. RM(r, m) is also known as the r-th order RM code of length 2^m. The dimension of RM(r, m) can easily be shown to be equal to

$$k = \sum_{i=0}^{r} \binom{m}{i},$$

which corresponds to the number of ways polynomials of degree up to r can be constructed with m variables.

In view of Equation (2.4), a generator matrix for RM(r, m) is formed by taking as rows the vectors associated with the k Boolean functions which can be expressed as polynomials of degree up to r in m variables.

Example 2.3.2 *The first order RM code of length 8,* RM$(1, 3)$*, is an* $(8, 4, 4)$ *binary code, and can be constructed from Boolean functions of degree up to one in three variables:* $\{1, x_1, x_2, x_3\}$*, so that*

$$\begin{array}{lcccccccc}
\bar{1} = & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
\bar{x}_1 = & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\
\bar{x}_2 = & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\
\bar{x}_3 = & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1
\end{array}$$

[3]"sum" means logical "XOR" and "multiplication" means logical "AND" in this context.

A generator matrix for RM$(1, 3)$ *is thus given by*

$$G(1,3) = \begin{pmatrix} \bar{1} \\ \bar{x}_1 \\ \bar{x}_2 \\ \bar{x}_3 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix}. \tag{2.5}$$

Note that code RM$(1, 3)$ *can be also obtained from a Hamming* $(7, 4, 3)$ *code by appending at the end of each code word an overall parity-check bit. The only difference between the extended Hamming code and RM*$(1, 3)$ *will be a possible permutation of bit (column) positions.*

Dual codes of RM codes are also RM codes

It can be shown that RM$(m - r - 1, m)$ is the dual code of RM(r, m). In other words, the generator matrix of RM$(m - r - 1, m)$ can be used as a parity-check matrix of RM(r, m).

2.3.2 Finite geometries and majority-logic decoding

RM codes can be formulated in terms of a *finite geometry*. An *Euclidean geometry* $EG(m, 2)$ of dimension m over $GF(2)$ consists of 2^m *points*, which are all the binary vectors of length m. Note that the columns of the matrix formed by the last three rows of the generator matrix of RM$(1, 3)$, see Example 2.3.2 above, are the 8 points of $EG(3, 2)$. If the zero point is deleted, then a *projective geometry* $PG(m - 1, 2)$ is obtained. The reader is referred to (Lin and Costello 2005) for an excellent treatment of finite geometries and RM codes. Finite geometry codes are in effect a generalization of RM codes.[4]

The connection between codes and finite geometries can be introduced as follows: consider $EG(m, 2)$. The columns of the matrix $(\bar{x}_1^\top \bar{x}_2^\top \ldots \bar{x}_m^\top)^\top$ are taken as the coordinates of points of the geometry $EG(m, 2)$. Then, there is a one-to-one correspondence between the components of binary vectors of length 2^m and the points of $EG(m, 2)$. A given binary vector of length 2^m is associated with a subset of points of $EG(m, 2)$. In particular, a subset of $EG(m, 2)$ can be associated with each binary vector $\bar{w} = (w_1, w_2, \ldots, w_{2^m})$ of length 2^m, by interpreting it as selecting points whenever $w_i = 1$. Stated otherwise, $\bar{w}$ is an *incidence vector*.

Binary RM codes can then be defined as follows: the code words of RM(r, m) are the incidence vectors of all subspaces (i.e., linear combinations of points) of dimension $m - r$ in $EG(m, 2)$ (Theorem 8 of (MacWilliams and Sloane 1977)). From this it follows that the number of minimum-weight code words of RM(r, m) is

$$A_{2^{m-r}} = 2^r \prod_{i=0}^{m-r-1} \frac{2^{m-i} - 1}{2^{m-r-i} - 1}. \tag{2.6}$$

The code obtained by deleting (puncturing) the coordinate corresponding to

$$x_1 = x_2 = \cdots = x_m = 0$$

[4] See Section 3.4.

from all the code words of RM(r, m) is the binary *cyclic* RM$^\star(r, m)$ code, which has

$$A^\star_{2^{m-r}-1} = \prod_{i=0}^{m-r-1} \frac{2^{m-i}-1}{2^{m-r-i}-1} \tag{2.7}$$

as minimum-weight code words.

In terms of decoding, it turns out that RM codes can be decoded with *ML decoding*. The idea is the following: the parity-check matrix induces 2^{n-k} parity-check equations. Designing an ML decoder is selecting a subset of the parity-check equations in such a way that a majority vote is taken on the value of a specific code position. As an illustration, consider again the RM(1, 3) code of Example 2.3.2 above.

Example 2.3.3 *Let $\bar{v} = \bar{u}G = (v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8)$ be a code word in RM*(1, 3). *As mentioned before, in this case equation (2.5) is also a parity-check matrix, since $r = 1$ and $m - r - 1 = 1$, and the code is* self-dual. *All possible 15 nonzero combinations of parity-check equations (rows of H) are the following:*

$$\begin{aligned}
v_1 + v_2 + v_3 + v_4 + v_5 + v_6 + v_7 + v_8 &= 0 \\
v_5 + v_6 + v_7 + v_8 &= 0 \\
v_3 + v_4 + v_7 + v_8 &= 0 \\
v_2 + v_4 + v_6 + v_8 &= 0 \\
v_1 + v_2 + v_3 + v_4 &= 0 \\
v_1 + v_2 + v_5 + v_6 &= 0 \\
v_1 + v_3 + v_5 + v_7 &= 0 \\
v_3 + v_4 + v_5 + v_6 &= 0 \\
v_2 + v_4 + v_5 + v_7 &= 0 \\
v_2 + v_3 + v_6 + v_7 &= 0 \\
v_1 + v_2 + v_7 + v_8 &= 0 \\
v_1 + v_3 + v_6 + v_8 &= 0 \\
v_1 + v_4 + v_5 + v_8 &= 0 \\
v_2 + v_3 + v_5 + v_8 &= 0 \\
v_1 + v_4 + v_6 + v_7 &= 0
\end{aligned} \tag{2.8}$$

The reader is invited to verify that the sum $v_i + v_j$ of every pair of code bits v_i, v_j, with $i \neq j$, appears in exactly four equations. Whenever a set of equations includes a term $v_i + v_j$, but no other sum of pairs appears more than once, the parity checks involved are said to be orthogonal *on positions v_i and v_j.*

It is now shown how a single error can be corrected. Let

$$\bar{r} = \bar{v} + \bar{e} = (r_1, r_2, r_3, r_4, r_5, r_6, r_7, r_8)$$

denote a received vector after code word $\bar{v}$ is transmitted over a BSC. Suppose that a single error is to be corrected in the fifth position, v_5. A procedure to design an ML decoder for this case is as follows:

Two equations involving the term $v_i + v_5$ are selected, with $i \neq 5$, and another set of two equations with the term $v_j + v_5$, $j \neq 5$, with $i \neq j$. Select (arbitrarily, as long as $i \neq j$ and both different than 5), say, $i = 3$, $j = 4$. There are four parity checks orthogonal to the term $v_3 + v_5$. Select any two of them. Do the same for the term $v_4 + v_5$.

The syndromes *associated with these equations are denoted* S_1 *and* S_2 *for* $v_3 + v_5$ *and* S_3 *and* S_4 *for* $v_4 + v_5$,

$$\begin{aligned} S_1 &\triangleq r_1 + \mathbf{r_3} + \mathbf{r_5} + r_7 \\ S_2 &\triangleq \mathbf{r_3} + r_4 + \mathbf{r_5} + r_6 \\ S_3 &\triangleq r_2 + \mathbf{r_4} + \mathbf{r_5} + r_7 \\ S_4 &\triangleq r_1 + \mathbf{r_4} + \mathbf{r_5} + r_8 \end{aligned} \tag{2.9}$$

Because $\bar{v}$ *is a code word in* $RM(1, 3)$, *the set of equations (2.9) is equivalent to*

$$\begin{aligned} S_1 &\triangleq e_1 + \mathbf{e_3} + \mathbf{e_5} + e_7 \\ S_2 &\triangleq \mathbf{e_3} + e_4 + \mathbf{e_5} + e_6 \\ S_3 &\triangleq e_2 + \mathbf{e_4} + \mathbf{e_5} + e_7 \\ S_4 &\triangleq e_1 + \mathbf{e_4} + \mathbf{e_5} + e_8 \end{aligned} \tag{2.10}$$

Since S_1, S_2, S_3 *and* S_4 *are orthogonal on* $e_3 + e_5$ *and* $e_4 + e_5$, *a new pair of equations orthogonal on* e_5 *can be formed as:*

$$\begin{aligned} S_1' &\triangleq e_3' + e_5' \\ S_2' &\triangleq e_4' + e_5' \end{aligned} \tag{2.11}$$

where e_j', $j = 3, 4, 5$, *represents the ML estimate from (2.9). Equations (2.11) are orthogonal on* e_5' *and consequently, the value of* e_5' *can be obtained by a majority vote, for example, setting it to the output of a logic AND gate with inputs* S_1' *and* S_2'.

Suppose that code word $\bar{v} = (11110000)$ *is transmitted and* $\bar{r} = (11111000)$ *is received. Then (2.10) gives:*

$$\begin{aligned} S_1 &= r_1 + \mathbf{r_3} + \mathbf{r_5} + r_7 = 1 \\ S_2 &= \mathbf{r_3} + r_4 + \mathbf{r_5} + r_6 = 1 \\ S_3 &= r_2 + \mathbf{r_4} + \mathbf{r_5} + r_7 = 1 \\ S_4 &= r_1 + \mathbf{r_4} + \mathbf{r_5} + r_8 = 1 \end{aligned} \tag{2.12}$$

Thus both $e_3 + e_5$ *and* $e_4 + e_5$ *are estimated as having value equal to '1'. From (2.11), it is concluded that* $e_5 = 1$, *and the estimated error vector is* $\bar{e} = (00001000)$, *from which the estimated code word is* $\bar{v} = \bar{r} + \bar{e} = (11110000)$. *This shows how an error in the fifth position can be corrected with a* two-step *ML decoder.*

In the previous example, it was shown how an error in a specific position of an RM(1, 3) code can be corrected. A similar procedure can be applied to every position in the code. Therefore, a total of eight ML estimates will be obtained.

In general, an $RM(r, m)$ code can be decoded with an $(r + 1)$-step ML decoder capable of correcting any combination of up to $\lfloor (2^{m-2} - 1)/2 \rfloor$ random errors (Lin and Costello 2005; MacWilliams and Sloane 1977).

In addition, it is next shown that a cyclic $RM^{\star}(r, m)$ code is simpler to decode. The argument is as follows. It is shown in Chapter 3 that in a cyclic code C, if $(v_1, v_2, \ldots, v_n)$ is a code word of C, then its *cyclic shift* $(v_n, v_1, \ldots, v_{n-1})$ is also a code word of C. Therefore, once a particular position can be corrected with ML decoding, all the other positions can also be corrected with the same algorithm (or hardware circuit) by cyclically shifting received code words, until all n positions have been tried.

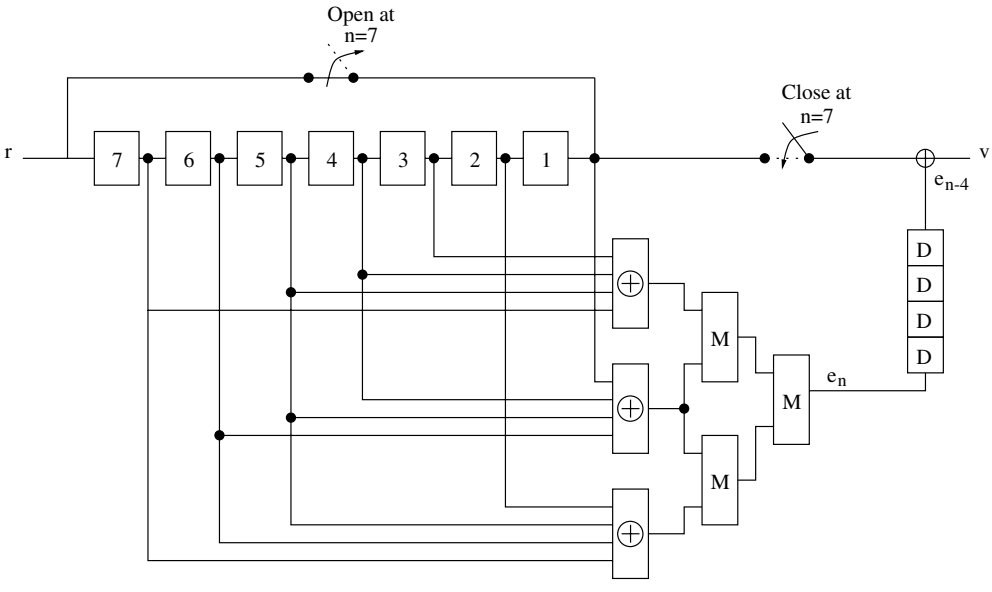


Figure 2.1 A majority-logic decoder for a cyclic RM$^\star(1, 3)$ code.

Example 2.3.4 *In this example, a decoder for the cyclic RM$^\star(1, 3)$ code, a binary Hamming $(7, 4, 3)$ code, is derived. To obtain the parity-check equations from those of the RM$(1, 3)$ code, remove the coordinate v_1 for which $x_1 = x_2 = \cdots = x_m$ from all equations. Let the code words of RM$^\star(1, 3)$ be indexed by relabeling the code word elements:*

$$(v_2, v_3, v_4, v_5, v_6, v_7, v_8) \rightarrow (v_1, v_2, v_3, v_4, v_5, v_6, v_7).$$

As before, an ML decoder for correcting an error in an arbitrary position (say, the fifth *position again) can be derived. This can be shown to result in the following seven nonzero (linearly independent) parity-check equations:*

$$\begin{aligned} v_1 + v_2 + v_3 + v_5 &= 0 \\ v_2 + v_3 + v_4 + v_6 &= 0 \\ v_3 + v_4 + v_5 + v_7 &= 0 \\ v_1 + v_4 + v_5 + v_6 &= 0 \\ v_2 + v_5 + v_6 + v_7 &= 0 \\ v_1 + v_3 + v_6 + v_7 &= 0 \\ v_1 + v_2 + v_4 + v_7 &= 0 \end{aligned} \tag{2.13}$$

In a manner similar to the previous example, the syndromes S_1 and S_2 below are orthogonal on v_4 and v_5, and S_2 and S_3 are orthogonal on v_5 and v_6:

$$\begin{aligned} S_1 &= e_3 + e_4 + e_5 + e_7 \\ S_2 &= e_1 + e_4 + e_5 + e_6 \\ S_3 &= e_2 + e_5 + e_6 + e_7 \end{aligned} \tag{2.14}$$

Based on the estimates $e'_4 + e'_5$ and $e'_5 + e'_6$ two additional orthogonal equations on e_5 can be formed to give the final estimate,

$$\begin{aligned} S'_1 &= e'_4 + e'_5 \\ S'_2 &= e'_5 + e'_6 \end{aligned} \tag{2.15}$$

where e'_j, $j = 4, 5, 6$, represents the ML estimate from the previous step. This results in the circuit shown in the Figure 2.1. The circuit operates as follows. Initially, the contents of the seven-bit register are set to zero. Suppose that a single error is contained in the received word in position i, for $1 \leq i \leq 7$. At each clock cycle, the contents of the register are cyclically shifted to the right by one position. Time, in clock cycles, is denoted by the subindex n in the following.

Consider first the case $i = 1$. That is, there is an error in the first code word position. After three cycles, the error is contained in register 5 (v_5). The output of the ML circuit is set to $e_n = 1$. Four cycles later (a total of seven cycles), the first received bit is output and the error is corrected. Consider now the case $i = 7$. After nine cycles, the error is detected and $e_n = 1$. Again, four cycles later (total 13 cycles), the bit in the last position is output and the error corrected. This decoder has a latency of 13 cycles. Every 13 cycles, the contents of the shift register are cleared and a new code word can be processed.

Problems

1. Using the union bound, estimate the probability of a bit error of a binary Hamming (7,4,3) code, denoted C_{H1}, with binary transmission over an AWGN channel.

2. Find the weight distribution of the (23, 12, 7) Golay code.

3. Repeat problem 1 for the binary (23, 12, 7) Golay code, denoted C_G. Compare the performances of C_{H1} and C_G. Comment on the underlying trade-off between the code rate and the coding gain.

4. Repeat problem 1 with binary transmission over a flat Rayleigh fading channel. (Use Monte Carlo integration for the union bound.)

5. Consider a binary Hamming (15,11) code, denoted C_{H2}.

   (a) Give the generator and parity-check matrices in systematic from, denoted G_{sys} and H_{sys}, for this code.

   (b) Suppose that the parity-check matrix is rearranged in such a way that the syndrome vector is the binary representation of an integer denoting the position of the error. Let H_{int} denote this matrix. Find the permutation of columns π_c needed to be applied to H_{sys} in order to obtain H_{int}.

   (c) A code word $\bar{v} = \bar{u}G_{\text{sys}}$ is sent through a BSC and received with a single error as $\bar{r}$. Show that the syndrome of the vector $\pi_c(\bar{r})$ is the binary representation of the position of the error, where π_c is the permutation in part (b).

   (d) Sketch block diagrams of the circuits used in encoding and decoding C_{H2}.

6. Repeat problem 1 for the binary Hamming (15,11,3) code. Compare the performance with that of the binary Hamming (7,4,3) code. In a binary communication system over an AWGN channel (no fading) which one of these two codes is expected to perform better?

7. Show that the extended (24,12,8) Golay code is self-dual. (Hint: Show that $B = B^T$.)

8. Prove that the dimension of RM(r, m) is

$$k = \sum_{i=0}^{r} \binom{m}{i}.$$

9. Obtain a generator matrix for the Reed–Muller code RM$(2, 4)$.

10. Show that the generator matrix $G(r, m)$ of the r-th order Reed–Muller code RM(r, m) can be written in terms of the generator matrices of shorter Reed–Muller codes as[5]

$$G(r, m) = \begin{pmatrix} G(r, m-1) & G(r, m-1) \\ 0 & G(r-1, m-1) \end{pmatrix}.$$

11. Prove that Reed–Muller codes RM(r, m) and RM$(m - r - 1, m)$ are duals.

12. Prove that the Reed–Muller codes satisfy the following subcode property

$$\text{RM}(r_1, m) \subset \text{RM}(r_2, m),$$

where $0 \leq r_1 < r_2 \leq m$.

[5]This is an example of the $|u|u + v|$-construction, discussed in Section 6.2.

3

Binary cyclic codes and BCH codes

The aim of this chapter is to introduce a minimum set of concepts necessary for the understanding of binary cyclic codes and for the efficient implementation of their encoding and decoding procedures. In this chapter, the important family of binary BCH codes is also introduced. BCH codes are a family of *cyclic codes*, with an algebraic structure that is useful in simplifying their encoding and decoding procedures. Binary BCH codes with minimum distance 3, better known as *Hamming codes*, have been a very popular choice in computer networks and in memory devices, because of their simple and fast encoding and decoding. Also, a shortened $(48, 36, 5)$ BCH code was adopted in the U.S. cellular time division multiple access (TDMA) system specification, standard IS-54.

3.1 Binary cyclic codes

Cyclic codes are a class of error correcting codes (ECC) that can be efficiently encoded and decoded using simple shift registers and combinatorial logic elements, on the basis of their representation using polynomials. In this section, the fundamental concepts of cyclic codes are discussed.

3.1.1 Generator and parity-check polynomials

Let C denote a linear (n, k) block code. Let $\bar{u}$ and $\bar{v}$ denote a *message* and the corresponding *code word* in C, respectively. Cyclic codes are linear codes with properties that make them suitable for hardware implementation. To every code word $\bar{v}$ *a polynomial* $\bar{v}(x)$ is associated,

$$\bar{v} = (v_0, v_1, \ldots, v_{n-1}) \longrightarrow \bar{v}(x) = v_0 + v_1 x + \cdots v_{n-1} x^{n-1}.$$

The indeterminant x serves to indicate the relative position of an element v_i of $\bar{v}$ as a term $v_i x^i$ of $\bar{v}(x)$, $0 \leq i < n$.

The Art of Error Correcting Coding, Second Edition Robert H. Morelos-Zaragoza
 ISBN: 0-470-01558-6

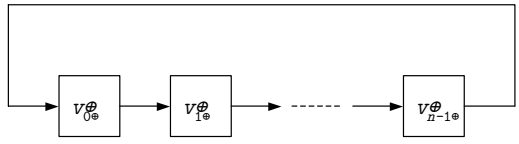


Figure 3.1 A cyclic shift register.

A linear block code C is cyclic if and only if *every cyclic shift of a code word is another code word*,

$$\bar{v} = (v_0, v_1, \ldots, v_{n-1}) \in C \iff \bar{v}^{(1)} = (v_{n-1}, v_0, \ldots, v_{n-2}) \in C.$$

In the language of polynomials, a cyclic shift by one position, denoted by $\bar{v}^{(1)}(x)$, is accomplished by a multiplication by x modulo $(x^n - 1)$,

$$\bar{v}(x) \in C \iff \bar{v}^{(1)}(x) = x\bar{v}(x) \bmod(x^n - 1) \in C.$$

Shift registers can be used for this purpose, as illustrated in Figure 3.1.

Example 3.1.1 *Consider the case of $n = 7$. Then, a cyclic shift by one position of the vector $\bar{v} = (0101011)$ equals $\bar{v}^{(1)} = (1010101)$. That is, $\bar{v}(x) = x^6 + x^5 + x^3 + x$ and*

$$\begin{aligned}\bar{v}^{(1)}(x) = x\ \bar{v}(x) &= x^7 + x^6 + x^4 + x^2 \bmod(x^7 + 1)\\ &= x^7 + x^6 + x^4 + x^2 + (x^7 + 1)\\ &= x^6 + x^4 + x^2 + 1,\end{aligned}$$

where, in the second equality, the fact that $(x^7 + 1) \bmod(x^7 + 1) = 0$ is used.

3.1.2 The generator polynomial

An important property of cyclic codes is that all code polynomials $\bar{v}(x)$ are multiples of a unique polynomial, $\bar{g}(x)$, called the *generator polynomial* of the code. This polynomial is *specified by its roots*, which are called the *zeros of the code*. It can be shown that the generator polynomial $\bar{g}(x)$ divides $(x^n - 1)$. (As with integers, "$a(x)$ divides $b(x)$" whenever $b(x) = q(x)a(x)$.) Therefore, to find a generator polynomial, the polynomial $(x^n - 1)$ must be factored into its *irreducible factors*, $\phi_j(x)$, $j = 1, 2, \ldots, \ell$,

$$(x^n - 1) = \phi_1(x)\phi_2(x)\ldots\phi_\ell(x). \tag{3.1}$$

Also, note that over the field of binary numbers, $a - b$ and $a + b$ (modulo 2) give the same result. In the remainder of the text, as all the codes are defined over the binary field or its extensions, no distinction is made between these operations.

As a consequence of the above, the polynomial $\bar{g}(x)$ is given by

$$\bar{g}(x) = \prod_{j \in J \subset \{1,2,\ldots,\ell\}} \phi_j(x). \tag{3.2}$$

Example 3.1.2 *With coefficients over* $Z_2 = \{0, 1\}$*, the polynomial* $x^7 + 1$ *factors as*

$$x^7 + 1 = (x + 1)(x^3 + x + 1)(x^3 + x^2 + 1).$$

Some examples of cyclic codes of length 7 are:

- *A binary cyclic Hamming* $(7, 4, 3)$ *code is generated by the polynomial*

$$\bar{g}(x) = x^3 + x + 1.$$

- *A binary cyclic parity-check* $(7, 6, 2)$ *code is generated by*

$$\bar{g}(x) = (x^3 + x + 1)(x^3 + x^2 + 1).$$

- *A binary* maximum-length-sequence $(7, 3, 4)$ *code is generated by*

$$\bar{g}(x) = (x + 1)(x^3 + x + 1).$$

3.1.3 Encoding and decoding of binary cyclic codes

The dimension of a binary cyclic (n, k) code is given by

$$k = n - \deg[\bar{g}(x)],$$

where $\deg[\cdot]$ denotes the degree of the argument. Since a cyclic code C is also linear, any set of k linearly independent (LI) vectors can be selected as a generator matrix. In particular, the binary vectors associated with $\bar{g}(x)$, $x\bar{g}(x), \ldots, x^{k-1}\bar{g}(x)$ are LI. These vectors can be used as rows of a generator matrix of C. In this case, a *nonsystematic* encoding rule is achieved. That is, the message bits do not appear explicitly in any position of the code words.

Example 3.1.3 *Consider the cyclic Hamming* $(7, 4, 3)$ *code, with generator polynomial*

$$\bar{g}(x) = x^3 + x + 1 \iff \bar{g} = (1101).$$

A generator matrix for this code is

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

Alternatively, the parity submatrix of a generator matrix of a cyclic code can be constructed with the vectors associated with the following polynomials:

$$\begin{gathered} x^{n-1} \bmod \bar{g}(x), \\ \vdots \\ x^{n-k+1} \bmod \bar{g}(x), \\ x^{n-k} \bmod \bar{g}(x), \end{gathered}$$

and a *systematic* encoding is obtained, as illustrated in the example below.

Example 3.1.4 *Let C be the cyclic Hamming $(7, 4, 3)$ code. Then $\bar{g}(x) = x^3 + x + 1$, and*

$$\begin{aligned} x^6 \bmod(x^3 + x + 1) &= x^2 + 1, \\ x^5 \bmod(x^3 + x + 1) &= x^2 + x + 1, \\ x^4 \bmod(x^3 + x + 1) &= x^2 + x, \\ x^3 \bmod(x^3 + x + 1) &= x + 1. \end{aligned}$$

It follows that a systematic generator matrix of C is

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

Let $\bar{u}(x)$ be associated with a message to be encoded. Encoding of code words of a binary cyclic code can be either nonsystematic or systematic, depending on how the message is processed:

- Nonsystematic encoding

$$\bar{v}(x) = \bar{u}(x)\ \bar{g}(x). \tag{3.3}$$

- Systematic encoding

$$\bar{v}(x) = x^{n-k}\ \bar{u}(x) + [x^{n-k}\bar{u}(x) \bmod \bar{g}(x)]. \tag{3.4}$$

3.1.4 The parity-check polynomial

Another polynomial, $\bar{h}(x)$, called the *parity-check* polynomial, can be associated with the parity-check matrix. The generator polynomial and the parity-check polynomial are related by

$$\bar{g}(x)\ \bar{h}(x) = x^n + 1. \tag{3.5}$$

The parity-check polynomial can be computed from the generator polynomial as

$$\bar{h}(x) = (x^n + 1)/\bar{g}(x) = h_0 + h_1 x + \cdots + h_k x^k.$$

Then, a parity-check matrix for C is given by using as rows the binary vectors associated with the first $n - k - 1$ nonzero cyclic shifts

$$\bar{h}^{(j)}(x) = x^j \bar{h}(x) \bmod(x^n - 1),\ j = 0, 1, \ldots, n - k - 1.$$

$$H = \begin{pmatrix} h_0 & h_1 & \cdots & & h_k & 0 & 0 & \cdots & 0 \\ 0 & h_0 & h_1 & \cdots & \cdots & h_k & 0 & \cdots & 0 \\ 0 & 0 & h_0 & h_1 & \cdots & \cdots & h_k & \cdots & 0 \\ 0 & 0 & 0 & \ddots & 0 & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & 0 & \cdots & \cdots & \cdots & h_k \end{pmatrix}. \tag{3.6}$$

Example 3.1.5 *The parity-check polynomial for the cyclic Hamming (7,4,3) code with the generator polynomial* $\bar{g}(x) = x^3 + x + 1$ *is*

$$\bar{h}(x) = (x^7 + 1)/(x^3 + x + 1) = x^4 + x^2 + x + 1.$$

A parity-check matrix for this code is

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

In the same manner as with linear codes, systematic encoding can be performed by a recursion with $h(x)$ using

$$\bar{v}(x)\ \bar{h}(x) = 0 \bmod (x^n + 1).$$

Systematic encoding can be performed as follows (Lin and Costello 2005; Peterson and Weldon 1972). Suppose first that the code rate $k/n \leq 0.5$. Let $\bar{u}(x) = u_0 + u_1 x + \cdots + u_{k-1}$ denote a polynomial of degree $k-1$, whose coefficients u_ℓ, $\ell = 0, 1, \ldots, k-1$, are the message bits to be encoded. Let $\bar{v}(x)$ denote the code polynomial in C corresponding to the information polynomial $\bar{u}(x)$. In the first step, $v_\ell = u_\ell$, for $\ell = 0, 1, \ldots, k-1$. From the cyclic structure of the code, it follows that the redundant bits can be obtained recursively via the parity-check relations

$$v_\ell = \sum_{j=0}^{\ell-1} v_j h_{(\ell-k),j}, \quad \ell = k, k+1, \ldots, n-1 \tag{3.7}$$

where $h_{(\ell-k),j}$ denotes the j-th entry in the $(\ell-k)$-th row of matrix (3.6).

In the case of high-rate cyclic (n, k) codes, say $k/n > 0.5$, encoding by the division of $x^{n-k}\bar{u}(x)$ by $\bar{g}(x)$ is more efficient. Either by recursion with $\bar{h}(x)$ or by division by $\bar{g}(x)$, the coefficients of the code polynomials are in the systematic form, so that the first k coefficients are the message bits, and the remaining $n-k$ coefficients constitute the redundant bits.

Figure 3.2 shows the block diagram of an encoder for a binary cyclic code with generator polynomial $\bar{g}(x)$. Initially, the switch (lower right portion of the figure) is in position 1, and the message bits are both transmitted through the channel and simultaneously fed to a sequential circuit that multiplies by x^{n-k} and divides by $\bar{g}(x)$. After k cycles, the shift register contains the remainder of the division, the switch moves to position 2 and the contents of the register are sent through the channel.

Duals of cyclic codes and maximum-length-sequence codes

By analogy with linear codes, the *dual code* of a cyclic code C with generator polynomial $\bar{g}(x)$ is the cyclic code $C^\perp$ generated by the polynomial $\bar{h}(x)$. The important class of *maximum-length-sequence* (MLS) cyclic codes (Peterson and Weldon 1972) has as members the duals of the cyclic Hamming codes. An MLS cyclic $(2^m - 1, m, 2^{m-1})$ code is generated by the polynomial $\bar{g}(x) = (x^n + 1)/p(x)$, where $p(x)$ is a *primitive polynomial.*[1]

[1]For a definition of primitive polynomial, see page 49.

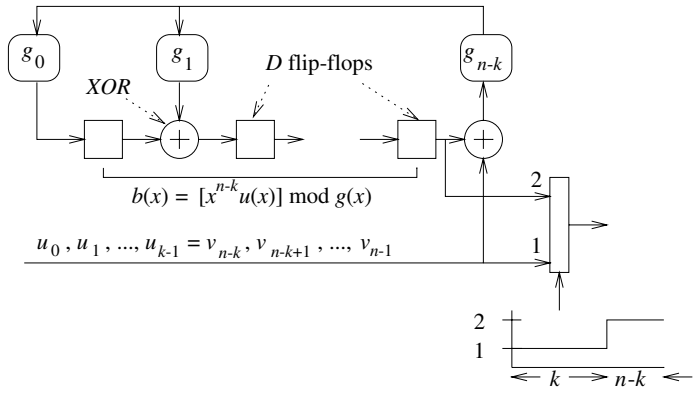


Figure 3.2 Circuit for systematic encoding: division by $\bar{g}(x)$.

3.1.5 Shortened cyclic codes and CRC codes

There are many practical applications in which an error correcting code with simple encoding and decoding procedures is desired, but existing constructions do not give the desired length, dimension and minimum distance.

The following text is from an email sent to the author:

> We plan to use a simple FEC/ECC scheme to detect/correct single-bit errors in a 64-bit data block. The objective is to find or choose an ECC scheme to correct single-bit errors with up to 8 bits of overhead, giving a maximum of 72 bits (64 data bits plus 8 redundant bits) in total.

Naturally, since 72 is not of the form $2^m - 1$, none of the cyclic codes studied so far can be applied directly. One possible solution is to use a cyclic Hamming $(127, 120, 3)$ code and to *shorten* it until a dimension $k = 64$ is reached. This yields a *shortened* Hamming $(71, 64, 3)$ code.[2]

Shortening is accomplished by not using all the information bits of a code. Let s denote the number of information bits not used, referred to as the *shortening depth*. Let C denote a cyclic (n, k, d) code. A shortened message is obtained by fixing s (arbitrary) message positions to zero. This leaves $k - s$ positions available for the message bits. Without loss of generality, let the highest positions in a message be set to zero. Then

$$\bar{u}(x) = u_0 + u_1 x + \cdots + u_{k-1-s} x^{k-1-s}.$$

The output of a *systematic encoder*, when the input is the message polynomial $\bar{u}(x)$, produces the code polynomial $\bar{v}(x) = x^{n-k}\bar{u}(x) + [x^{n-k}\bar{u}(x) \bmod \bar{g}(x)]$, of degree up to $n - 1 - s$. This shows that the resulting shortened code C_s is a linear $(n - s, k - s, d_s)$ code with $d_s \geq d$. In general, the shortened code C_s is no longer cyclic.

[2]This is an example to introduce the concept of shortening. A (72,64,4) single-error-correcting/double-error-detecting (SEC/DED) code, based on a shortened Hamming code, and adding an overall parity-check bit, was proposed for the IBM model 360 mainframe (Hsiao 1970). This code has also been used in byte-oriented memory devices.

Example 3.1.6 *Let C denote the cyclic Hamming* $(7, 4, 3)$ *code with generator polynomial* $\bar{g}(x) = 1 + x + x^3$. *A new code is derived from C by sending 2 leading zeros followed by two information bits and the same three redundant bits computed by an encoder in C. This process gives a set of code words that forms a* shortened *linear* $(5, 2, 3)$ *code.*

The fundamental property of a shortened code C_s obtained from a cyclic code is that, although the code is generally no longer cyclic, the same encoder and decoder can be used, after the leading zeros are properly taken into account. In computer simulations, it is easy to simply pad the code words with zeros followed by the code word in C_s and use the same encoding and decoding algorithms discussed in the book. This method is widely used in hardware implementations of Reed–Solomon decoders. Alternatively, the leading zeros in a message do not need to be included in the code word. Instead, the decoder circuit is modified to multiply the incoming received polynomial $\bar{r}(x)$ by x^{n-k+s} modulo $\bar{g}(x)$, instead of x^{n-k} modulo $\bar{g}(x)$ in the conventional decoder. More details on the modified encoder and decoder structures for a shortened cyclic code can be found in Lin and Costello (2005), Peterson and Weldon (1972), Wicker (1995), among other references.

Another possible solution is to try to construct other classes of cyclic codes with the desired parameters. Good families of cyclic codes not covered in the text are Euclidean geometry (EG) and projective geometry (PG) codes (Lin and Costello 2005).[3]

Yet another possibility is to use a *nonbinary* cyclic code, such as the Reed–Solomon code discussed in Chapter 4, and to express it in terms of bits. This binary image of a Reed–Solomon (RS) code will have the additional feature of being able to correct many bursts of errors. Binary images of shortened RS codes have found numerous applications, including memory devices.

CRC codes

One of the most popular forms of ECC are the *cyclic redundancy check* codes, or CRC codes. These cyclic codes are used to detect errors in blocks of data. CRC codes are cyclic codes of length $n \leq 2^m - 1$. Typically, CRC codes have generator polynomials of the form $(1 + x)\bar{g}(x)$, where $\bar{g}(x)$ is the generator polynomial of a cyclic Hamming code. Common values of m are 12, 16 and 32. The choice of the generator polynomials is dictated by the *undetected error probability*, which depends on the weight distribution of the code. The computation of the undetected error probability of a cyclic code is tantamount to determining its weight distribution. This has remained an elusive task, even after 50 years of coding theory, with some progress reported in Fujiwara *et al.* (1985), Kazakov (2001) and references therein. Table 3.1 shows a list of the most popular generator polynomials of CRC codes, or *CRC polynomials*.

3.1.6 Fire codes

A natural extension of CRC codes are Fire codes. These binary cyclic (n, k, d) codes are capable of correcting single error bursts of length b or less and have a generator polynomial of the form

$$\bar{g}(x) = (x^{2b-1} + 1)\bar{p}(x),$$

[3]These codes can be used to construct powerful low-density parity-check (LDPC) codes that have excellent performance with iterative decoding techniques. See Section 8.3 in Chapter 8.

Table 3.1 Generator polynomials of some CRC codes.

Code	m	$\bar{g}(x)$
CRC-12	12	$x^{12}+x^{11}+x^3+x^2+x+1$
CRC-16	16	$x^{16}+x^{15}+x^2+1$
CRC-CCITT	16	$x^{16}+x^{12}+x^5+1$
CRC-32	32	$x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}$ $+x^{10}+x^8+x^7++x^5+x^4+x^2+x+1$

where $\bar{p}(x)$ is an irreducible polynomial of degree $c \geq b$ and relatively prime to $(x^{2b-1}+1)$. The length of a Fire code is $n = \text{LCM}(e, 2b-1)$, where e is the smallest integer such that $\bar{p}(x)$ divides (x^e+1). (This is called *exponent* in Peterson and Weldon (1972) and *period* in Lin and Costello (2005).) The dimension is $k = \text{LCM}(e, 2b-10) - (2b-1) - c$. Note that the parity-check bits associated with the polynomial $(x^{2b-1}+1)$ are interleaved and evenly spaced at every $2b-1$ positions. Therefore, at most $b-1$ successive parity-check bits are affected by bursts of length up to b. This fact, together with the polynomial $\bar{p}(x)$, is sufficient to determine the location of the error burst. Decoding is accomplished by an error-trapping decoder (Kasami 1964) that is similar in structure to the Meggit decoder presented in the next section. For more details, see Section 11.3 of Peterson and Weldon (1972) and Section 20.3.1 of Lin and Costello (2005).

Example 3.1.7 *Let $b = 4$ and $c = 4$. Choose $\bar{p}(x) = x^4 + x + 1$ with $e = 15$. The length and dimension of this Fire code are $n = \text{LCM}(15, 7) = 105$ and $k = 105 - 7 - 4 = 94$, respectively. In other words, a Fire* $(105, 94)$ *code capable of correcting any single burst of four or less errors has generator polynomial*

$$\bar{g}(x) = (x^7+1)(x^4+x+1) = x^{11}+x^8+x^7+x^4+x+1.$$

3.2 General decoding of cyclic codes

Let $\bar{r}(x) = \bar{v}(x) + \bar{e}(x)$, where $\bar{e}(x)$ is the *error polynomial* associated with an error vector produced after transmission over a BSC channel. Then the *syndrome polynomial* is defined as

$$\bar{s}(x) \stackrel{\Delta}{=} \bar{r}(x) \bmod \bar{g}(x) = \bar{e}(x) \bmod \bar{g}(x). \tag{3.8}$$

Figure 3.3 shows the general architecture of a decoder for cyclic codes. The syndrome polynomial $\bar{s}(x)$ is used to determine the error polynomial $\bar{e}(x)$. Since a cyclic code is first of all a linear code, this architecture can be thought of as a "standard array approach" to the decoding of cyclic codes.

The decoding problem amounts to finding the (unknown) error polynomial $\bar{e}(x)$ from the (known) syndrome polynomial $\bar{s}(x)$. These two polynomials are related by equation (3.8), which is the basis of a *syndrome decoder* (also referred to as a Meggit decoder (Meggit 1960)) for cyclic codes. A related decoder is the *error-trapping decoder* (Kasami 1964), which checks if the error polynomial $\bar{e}(x)$ is contained ("trapped") in the syndrome polynomial $\bar{s}(x)$. Only a limited number of classes of codes have relatively simple

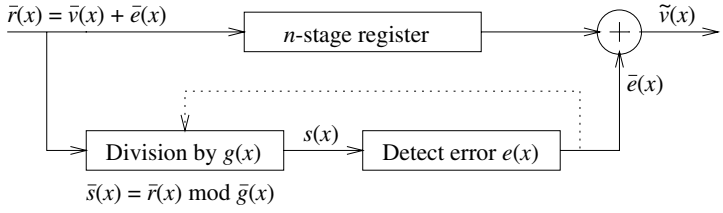


Figure 3.3 General architecture of a decoder for cyclic codes.

decoders, for example, cyclic Hamming and Golay codes. As the error-correcting capability $t = [(d_{\min} - 1)/2]$ increases, however, the complexity of an architecture based only on the detection of errors with combinatorial logic becomes too large.

Suppose that an error in the position corresponding to x^{n-1} (the first received bit) occurs. In other words,

$$\bar{e}(x) = x^{n-1}.$$

The corresponding syndrome polynomial is

$$\bar{s}(x) = x^{n-1} \bmod \bar{g}(x).$$

The code is cyclic, and thus if an error pattern affecting a given position is detected, any other error can be detected as well, by cyclically shifting the contents of the syndrome polynomial and the error polynomial. The syndrome decoder checks the syndrome for each received position and, if the pattern $x^{n-1} \bmod \bar{g}(x)$ is detected, that position is corrected.

Example 3.2.1 *In this example, the decoding of a cyclic* $(7, 4, 3)$ *Hamming code is illustrated. For this code,*

$$\bar{g}(x) = x^3 + x + 1.$$

The syndrome decoding circuit is shown in Figure 3.4. The received bits are stored in a shift register and at the same time fed to a divide-by-$\bar{g}(x)$ *circuit. After all the seven bits have been received, the shift register contents are shifted one at a time, and a combinatorial gate checks if the syndrome polynomial* $x^6 \bmod (1 + x + x^3) = 1 + x^2$*, or (101) in binary vector notation is present in the shift register when the output of the gate is equal to one, and the error is at the position* x^6 *and is corrected. At the same time, the error is fed back to the divide-by-*$\bar{g}(x)$ *circuit to bring all the contents of the register equal to zeros, upon successful completion of decoding. This also allows detection of any anomalies at the end of the decoding process, by checking that the contents of the shift register are not all equal to zero.*

Attention is now focused on cyclic codes with large error-correcting capabilities, for which the decoding problem can be treated as that of solving sets of equations. Because of this, the notion of a *field*, a set in which one can multiply, add and find inverses, is required. Cyclic codes have a rich algebraic structure. It will be shown later that powerful decoding algorithms can be implemented efficiently when the roots of the generator polynomial are invoked and arithmetic over a *finite field* is used.

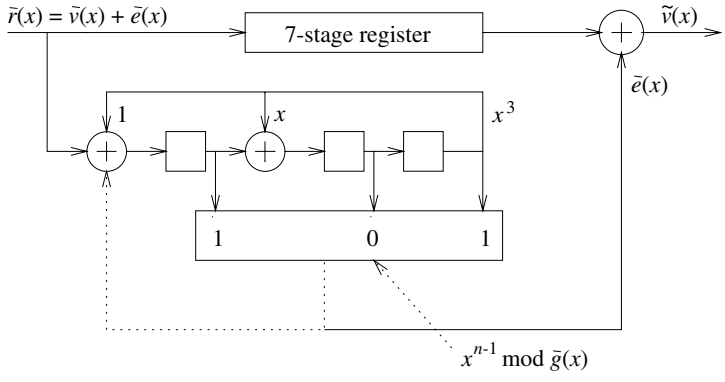


Figure 3.4 Syndrome decoder for a binary cyclic Hamming (7,4) code.

Recall that the generator polynomial is the product of binary irreducible polynomials:

$$\bar{g}(x) = \prod_{j \in J \subset \{1,2,\ldots,\ell\}} \phi_j(x).$$

The algebraic structure of cyclic codes enables one to find the factors (roots) of each $\phi_j(x)$ in a *splitting field* (also known as extension field). In the case of interest, that is, when the underlying symbols are bits, the splitting field becomes a *Galois field*.[4] Some authors refer to Galois fields as *finite fields*. The standard notation that will be used in the text is $GF(q)$, where $q = 2^m$. (Although, in general, q can be the power of any prime number.)

Example 3.2.2 *In this example, the reader is reminded that the concept of splitting field is very familiar. Consider the field of real numbers. Over this field, it is well known that the polynomial* $x^2 + 1$ *is irreducible. However, over the* complex field*, it splits into* $(x + i)(x - i)$*, where* $i = \sqrt{-1}$*. Thus the complex field is the splitting field of the real field!*

3.2.1 $GF(2^m)$ arithmetic

It can be shown, with basic abstract algebra concepts (Lin and Costello 2005; Peterson and Weldon 1972), that in the field of binary numbers any polynomial of degree m can be split over $GF(2^m)$. For the purposes of this book, it is sufficient to learn basic computational aspects of finite fields. Serious readers are urged to study a good textbook on abstract algebra.[5]

Decoding with $GF(2^m)$ arithmetic allows replacement of complex combinatorial circuits with practical processor architectures that can solve Equation (3.8) as a set of linear equations. In the following text, the necessary tools to solve equations involved in decoding of cyclic codes are introduced.

Important properties of $GF(2^m)$

The field $GF(2^m)$ is isomorphic (with respect to "+") to the linear space $\{0, 1\}^m$. In other words, for every element $\beta \in GF(2^m)$, there exists a unique m-dimensional binary vector $\bar{v}_\beta \in \{0, 1\}^m$.

[4] After the famous French mathematician Evariste Galois (1811-1832).

[5] The author likes Herstein (1975).

There is a *primitive element* $\alpha \in GF(2^m)$, such that every nonzero element β in $GF(2^m)$ can be expressed as $\beta = \alpha^j$, $0 \leq j \leq 2^m - 2$. This element α is the root of an irreducible polynomial, called a *primitive polynomial*, $p(x)$ over $\{0, 1\}$, that is, $p(\alpha) = 0$. A primitive element α of the field $GF(2^m)$ satisfies the equation $\alpha^{2^m-1} = 1$, and $n = 2^m - 1$ is the smallest positive integer such that $\alpha^n = 1$.

Example 3.2.3 *Let α be a primitive element of $GF(2^3)$ such that $p(\alpha) = \alpha^3 + \alpha + 1 = 0$ and $\alpha^7 = 1$. The table below shows three different ways to express, or represent, elements in $GF(2^3)$.*

Power	*Polynomial*	*Vector*
–	0	000
1	1	001
α	α	010
α^2	α^2	100
α^3	$1 + \alpha$	011
α^4	$\alpha + \alpha^2$	110
α^5	$1 + \alpha + \alpha^2$	111
α^6	$1 + \alpha^2$	101

When adding elements in $GF(2^m)$, the vector representation is the most useful, because a simple exclusive -or operation is needed. However, when elements are to be multiplied, the power representation is the most efficient. Using the power representation, a multiplication becomes simply an addition modulo $2^m - 1$. The polynomial representation may be appropriate when making operations modulo a polynomial. An example of the need of this polynomial representation was seen in the discussion of shortened cyclic codes, where the value of $x^{n-1} \bmod \bar{g}(x)$ was required.

In the power representation, because $\alpha^{2^m-1} = 1$ holds, note that $\alpha^{2^m} = \alpha\alpha^{2^m-1} = \alpha$, $\alpha^{2^m+1} = \alpha^2\alpha^{2^m-1} = \alpha^2$, and so on. This is to say that the powers of α are to be computed modulo $2^m - 1$. Applying the same argument shows that $\alpha^{-1} = \alpha^{-1+2^m-1} = \alpha^{2^m-2}$. In Example 3.2.3 above, $\alpha^{-1} = \alpha^{2^3-2} = \alpha^6$. In general, the inverse $\beta^{-1} = \alpha^k$ of an element $\beta = \alpha^\ell$ is found by determining the integer k, $0 \leq k < 2^m - 1$ such that $\alpha^{\ell+k} = 1$, which can be expressed as $\ell + k = 0 \mod (2^m - 1)$. Therefore, $\ell = 2^m - 1 - k$. Also, in the polynomial representation, the equation $p(\alpha) = 0$ is used to reduce the expressions. In Example 3.2.3, $\alpha^3 = \alpha^3 + 0 = \alpha^3 + (\alpha^3 + \alpha + 1) = \alpha + 1$.

Log and antilog tables

A convenient way to perform *both* multiplications and additions in $GF(2^m)$ is to use two look-up tables, with different interpretations of the address. This allows one to change between polynomial (vector) representation and power representation of an element of $GF(2^m)$.

The antilog table $A(i)$ is useful when performing additions. The table gives the value of a binary vector, represented as an integer in natural representation, $A(i)$, that corresponds to the element α^i. The log table $L(i)$ is used when performing multiplications. This table

gives the value of a power of alpha, $\alpha^{L(i)}$, that corresponds to the binary vector represented by the integer i. The following equality holds:

$$\alpha^{L(i)} = A(i).$$

The best way to understand how to use the tables in the computation of arithmetic operations in $GF(2^m)$ is through an example.

Example 3.2.4 *Consider $GF(2^3)$ with $p(\alpha) = \alpha^3 + \alpha + 1$, and $\alpha^7 = 1$. The log and antilog tables are the following:*

Address i	$GF(2^m)$*-to-vector Antilog table,* $A(i)$	*Vector-to-*$GF(2^m)$ *Log table,* $L(i)$
0	1	-1
1	2	0
2	4	1
3	3	3
4	6	2
5	7	6
6	5	4
7	0	5

Consider the computation of an element $\gamma = \alpha(\alpha^3 + \alpha^5)^3$ in vector form. Using the properties of $GF(2^3)$, γ can be computed as follows:

$$\alpha^3 + \alpha^5 = 110 \oplus 111 = 001 = \alpha^2.$$

Thus, $\gamma = \alpha(\alpha^2)^3 = \alpha^{1+6} = \alpha^7 (= 1)$.

On the other hand, using the log and antilog tables, the computation of γ proceeds as follows:

$$\begin{aligned}\gamma &= A(L(A(3) \oplus A(5)) * 3 + 1) \\ &= A(L(3 \oplus 7) * 3 + 1) \\ &= A(L(4) * 3 + 1) = A(2 * 3 + 1) = A(7) = (A(0) = 1).\end{aligned}$$

In the last step, use was made of the fact that $\alpha^7 = 1$.

Antilog and log tables are used in performing addition and multiplication over $GF(2^m)$. Computer programs are available on the ECC web site for simulating encoding and decoding algorithms of BCH and Reed–Solomon codes, with arithmetic in $GF(2^m)$. These algorithms are described in the subsequent sections.

More properties of $GF(2^m)$

The *minimal polynomial* $\phi_i(x)$ of an element α^i is the smallest degree polynomial that has α^i as a root. The following properties regarding minimal polynomials can be shown. The minimal polynomial $\phi_i(x)$ has binary coefficients and is irreducible over $GF(2) = \{0, 1\}$.

Moreover, $\phi_i(x)$ has roots α^i, $\alpha^{2i}, \ldots, \alpha^{2^{\kappa-1}i}$, where κ divides m. These elements are known as the *conjugate elements* of α^i in $GF(2^m)$. The powers of the conjugate elements form a *cyclotomic coset* (see MacWilliams and Sloane (1977), p. 104, and (von zur Gathen and Gerhard 1999), p. 391):

$$\mathcal{C}_i \triangleq \{i, 2i, 4i, \ldots, 2^{\kappa-1}i\}.$$

Cyclotomic cosets (also called cycle sets in Peterson and Weldon (1972), p. 209) have the property that they partition the set $\mathcal{I}_{2^m-1}$ of integers modulo $2^m - 1$. In other words, cyclotomic cosets are disjoint, that is, their intersection is the empty set $C_i \cap C_j = \emptyset$, $i \neq j$, and the union of all cyclotomic cosets $\bigcup_i C_i = \mathcal{I}_{2^m-1}$.

Example 3.2.5 *The cyclotomic sets modulo 7 are:*

$$C_0 = \{0\}$$
$$C_1 = \{1, 2, 4\}$$
$$C_3 = \{3, 6, 5\}$$

The primitive element α of $GF(2^m)$ satisfies the equation $\alpha^{2^m-1} = 1$, and all elements can be expressed as powers of α. From this it follows that the polynomial $(x^{2^m-1} + 1)$ factors over the binary field as

$$(x^{2^m-1} + 1) = \prod_{j=0}^{M} \phi_{\ell_j}(x),$$

and splits completely over $GF(2^m)$ as

$$(x^{2^m-1} + 1) = \prod_{j=0}^{2^m-2} (x + \alpha^j). \tag{3.9}$$

The *order* n_i of an element $\beta = \alpha^i$ of $GF(2^m)$ is the smallest positive integer such that $\beta^{n_i} = 1$. The order n_i of every element in $GF(2^m)$ divides $2^m - 1$.

Importantly, the degree of a minimal polynomial $\phi_i(x)$ is equal to the cardinality (number of elements) of the cyclotomic coset $\mathcal{C}_i$,

$$\deg[\phi_i(x)] = |\mathcal{C}_i|.$$

This suggests the following method for finding all factors of $(x^{2^m-1} + 1)$:

1. Generate the cyclotomic cosets modulo $2^m - 1$.
2. For each cyclotomic coset $\mathcal{C}_s$, compute the minimal polynomial $\phi_s(x)$ as the product of linear factors $(x - \alpha^{i_s})$, where $i_s \in \mathcal{C}_s$,

$$\phi_s(x) = \prod_{i_s \in \mathcal{C}_s} (x + \alpha^{i_s}). \tag{3.10}$$

This method can be used in computing the generator polynomial of any cyclic code of length $n = 2^m - 1$. It is used in the computer simulation programs for BCH codes available on the ECC web site, to compute the generator polynomial given the zeros of the code.

Example 3.2.6 *Consider GF*(2^3) *with* $p(x) = x^3 + x + 1$. *The roots of each of the factors of the polynomial* $x^7 + 1$ *are shown in the following table. The reader is invited to verify that in fact the products of the linear factors in (3.10) give the resulting binary polynomials.*

$\mathcal{C}_s$	*Conjugate elements*	*Minimal polynomial,* $\phi_s(x)$
$\mathcal{C}_0 = \{0\}$	1	$\phi_0(x) = x + 1$
$\mathcal{C}_1 = \{1, 2, 4\}$	α,α^2,α^4	$\phi_1(x) = x^3 + x + 1$
$\mathcal{C}_3 = \{3, 6, 5\}$	$\alpha^3,\alpha^6,\alpha^5$	$\phi_3(x) = x^3 + x^2 + 1$

3.3 Binary BCH codes

BCH codes are cyclic codes that are constructed by specifying their zeros, that is, the roots of their generator polynomials:

> *A BCH code of* $d_{min} \geq 2t_d + 1$ *is a cyclic code whose generator polynomial* $\bar{g}(x)$ *has* $2t_d$ *consecutive roots* α^b, α^{b+1}, α^{b+2t_d-1}.

Therefore, a binary BCH $(n, k, d_{\min})$ code has a generator polynomial

$$\bar{g}(x) = LCM\{\phi_b(x), \phi_{b+1}(x), \ldots, \phi_{b+2t_d-1}(x)\},$$

length $n = LCM\{n_b, n_{b+1}, \ldots, n_{b+2t_d-1}\}$, and dimension $k = n - \deg[\bar{g}(x)]$. A binary BCH code has a *designed minimum distance* equal to $2t_d + 1$. However, it should be noted that its true minimum distance may be larger.

Example 3.3.1 *With GF*(2^3), $p(x) = x^3 + x + 1$, $t_d = 1$ *and* $b = 1$, *the polynomial*

$$\bar{g}(x) = LCM\{\phi_1(x), \phi_2(x)\} = x^3 + x + 1,$$

generates a binary BCH (7,4,3) code. (This is actually a binary cyclic Hamming code!) Note that the Hamming weight of $\bar{g}(x)$ *is equal to 3, so that – in this case, but not always – the designed distance is equal to the true minimum distance of the code.*

Example 3.3.2 *Consider GF*(2^4), $p(x) = x^4 + x + 1$, *with* $t_d = 2$ *and* $b = 1$. *Then*

$$\begin{aligned}\bar{g}(x) &= LCM\{\phi_1(x), \phi_3(x)\} \\ &= (x^4 + x + 1)(x^4 + x^3 + x^2 + x + 1) \\ &= x^8 + x^7 + x^6 + x^4 + 1\end{aligned}$$

generates a double-error-correcting binary BCH (15,7,5) code.

Example 3.3.3 *With GF*(2^4), $p(x) = x^4 + x + 1$, $t_d = 3$ *and* $b = 1$, *the polynomial*

$$\begin{aligned}\bar{g}(x) &= LCM\{\phi_1(x), \phi_3(x)\phi_5(x)\} \\ &= (x^4 + x + 1)(x^4 + x^3 + x^2 + x + 1) \\ &\quad (x^2 + x + 1) \\ &= x^{10} + x^8 + x^5 + x^4 + x^2 + x + 1\end{aligned}$$

generates a triple-error-correcting binary BCH (15,5,7) code.

3.3.1 BCH bound

A lower bound on the minimum distance of a BCH code, known as the *BCH bound*, is derived next. This is useful not only to estimate the error-correcting capabilities of cyclic codes in general, but also to point out particular features of BCH codes. Note that the elements $\alpha^b, \alpha^{b+1}, \ldots, \alpha^{b+2t_d-1}$ are roots of the generator polynomial $\bar{g}(x)$, and that every code word $\bar{v}$ in the BCH code is associated with a polynomial $\bar{v}(x)$, which is a multiple of $\bar{g}(x)$. It follows that

$$\bar{v}(x) \in C \iff \bar{v}(\alpha^i) = 0, \quad b \le i < b + 2t_d. \tag{3.11}$$

Code word $\bar{v}$ then satisfies the following set of $2t_d$ equations, expressed in matrix form, based on (3.11):

$$\begin{pmatrix} 1 & 1 & \cdots & 1 \\ \alpha^b & \alpha^{b+1} & \cdots & \alpha^{b+2t_d-1} \\ (\alpha^b)^2 & (\alpha^{b+1})^2 & \cdots & (\alpha^{b+2t_d-1})^2 \\ \vdots & \vdots & \ddots & \vdots \\ (\alpha^b)^{n-1} & (\alpha^{b+1})^{n-1} & \cdots & (\alpha^{b+2t_d-1})^{n-1} \end{pmatrix} \begin{pmatrix} v_0 & v_1 & v_2 & \cdots & v_{n-1} \end{pmatrix} = \bar{0}. \tag{3.12}$$

Consequently, the parity-check matrix of a binary cyclic BCH code is given by

$$H = \begin{pmatrix} 1 & \alpha^b & (\alpha^b)^2 & \cdots & (\alpha^b)^{n-1} \\ 1 & \alpha^{b+1} & (\alpha^{b+1})^2 & \cdots & (\alpha^{b+1})^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha^{b+2t_d-1} & (\alpha^{b+2t_d-1})^2 & \cdots & (\alpha^{b+2t_d-1})^{n-1} \end{pmatrix} \tag{3.13}$$

This matrix H has the characteristic that every $2t_d \times 2t_d$ submatrix (formed by an arbitrary set of $2t_d$ columns of H) is a *Vandermonde matrix* (see, e.g., von zur Gathen and Gerhard (1999), p. 95). Therefore (see Section 2.1), any $2t_d$ columns of H are linearly independent, from which it follows that the minimum distance of the code is $d \ge 2t_d + 1$. (Peterson and Weldon (1972) p. 270, MacWilliams and Sloane (1977) p. 201, and Lin and Costello (2005) p. 149.) Another interpretation of the results above is the following:

BCH bound If the generator polynomial $\bar{g}(x)$ of a cyclic (n, k, d) code has ℓ consecutive roots, say $\alpha^b, \alpha^{b+1}, \ldots, \alpha^{b+\ell-1}$, then $d \ge 2\ell + 1$.

3.4 Polynomial codes

The important class of cyclic polynomial codes includes cyclic RM codes, BCH and Reed–Solomon codes, and finite-geometry codes (Kasami *et al.* 1968; Lin and Costello 2005; Peterson and Weldon 1972). Polynomial codes are also specified by setting conditions on their zeros:

Let α be a primitive element of GF(2^{ms}). Let s be a positive integer, and b a divisor of $2^s - 1$. Then α^h is a root of the generator polynomial $g(x)$ of a μ-th order polynomial code if and only if b divides h and

$$\min_{0 \le \ell < s} W_{2^s}(h2^\ell) = jb, \quad \text{with } 0 < j < \left[\frac{m}{b}\right] - \mu,$$

where for an integer i, $i = \sum_{\ell=0}^{m-1} i_\ell 2^{s\ell}$, $W_{2^s}(i)$ *is defined as the* 2^s*-ary weight of integer* i,

$$W_{2^s}(i) = \sum_{i=0}^{m-1} i_\ell.$$

According to this definition, both BCH and Reed–Solomon codes are polynomial codes with $b = m = 1$. Reed–Muller codes are subcodes of polynomial codes with $s = 1$. Finite-geometry codes (Lin and Costello (2005), Chapter 8) occur as *dual codes of polynomial codes* Peterson and Weldon (1972). Below, following Lin and Costello (2005), the specifications of the zeros of finite-geometry codes are presented.

Euclidean geometry (EG) codes

Let α *be a primitive element of* $GF(2^{ms})$. *Let* h *be a nonnegative integer less than* $2^{ms} - 1$. *Then* α^h *is a root of the generator polynomial* $g(x)$ *of a* (μ, s)*-order EG code of length* $2^{ms} - 1$ *if and only if*

$$0 < \max_{0 \le \ell < s} W_{2^s}(h^{(\ell)}) \le (m - \mu - 1)(2^s - 1),$$

where $h^{(\ell)} = 2^\ell h$ modulo $2^{ms} - 1$.

For $s = 1$, EG codes become cyclic $\mathrm{RM}^\star_{m,\mu}$ codes, and therefore EG codes are regarded as generalized RM codes.

Projective geometry (PG) codes

Let α *be a primitive element of* $GF(2^{(m+1)s})$. *Let* h *be a nonnegative integer less than* $2^{(m+1)s} - 1$. *Then* α^h *is a root of the generator polynomial* $g(x)$ *of a* (μ, s)*-order PG code of length* $n = (2^m s - 1)/(2^s - 1)$ *if and only if* h *is divisible by* $2^s - 1$ *and*

$$0 < \max_{0 \le \ell < s} W_{2^s}(h^{(\ell)}) \le j(2^s - 1),$$

where $h^{(\ell)} = 2^\ell h$ modulo $2^{ms} - 1$, *and* $0 \le j \le m - u$.

3.5 Decoding of binary BCH codes

The main idea in decoding binary BCH codes is to use the elements $\beta \in GF(2^m)$ to *number the positions* of a code word (or, equivalently, the order of the coefficients of the associated polynomial). This numbering is illustrated in Figure 3.5 for a vector $\bar{r} = \begin{pmatrix} r_0 & r_1 & \dots & r_{n-1} \end{pmatrix}$ with corresponding $\bar{r}(x)$.

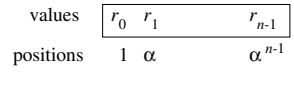


Figure 3.5 Code word position numbering using elements of $GF(2^m)$.

Using $GF(2^m)$ arithmetic, the positions of the errors can be found by solving a set of equations. These equations can be obtained from the error polynomial $\bar{e}(x)$ and the zeros of the code, α^j, for $b \le j \le b + 2t_d - 1$, as shown in the following.

Let $\bar{r}(x) = \bar{v}(x) + \bar{e}(x)$ represent the polynomial associated with a received code word, where the *error polynomial* is defined as

$$\bar{e}(x) = e_{j_1} x^{j_1} + e_{j_2} x^{j_2} + \cdots + e_{j_\nu} x^{j_\nu}, \tag{3.14}$$

and $\nu \le t_d$ is the number of errors. The sets $\{e_{j_1}, e_{j_2}, \ldots, e_{j_\nu}\}$, and $\{\alpha^{j_1}, \alpha^{j_2}, \ldots, \alpha^{j_\nu}\}$ are known as the *error values* and *error positions*, respectively, where $e_j \in \{0, 1\}$ for binary BCH codes[6] and $\alpha \in GF(2^m)$.

The *syndromes* are the evaluation of $\bar{r}(x)$ at each of the zeros of the code:

$$\begin{aligned}
S_1 &= r(\alpha^b) = e_{j_1}\alpha^{bj_1} + \cdots + e_{j_\nu}\alpha^{bj_\nu} \\
S_2 &= r(\alpha^{b+1}) = e_{j_1}\alpha^{(b+1)j_1} + \cdots + e_{j_\nu}\alpha^{(b+1)j_\nu} \\
&\vdots \qquad\qquad \vdots \\
S_{2t_d} &= r(\alpha^{b+2t_d-1}) = e_{j_1}\alpha^{(b+2t_d-1)j_1} + \cdots + e_{j_\nu}\alpha^{(b+2t_d-1)j_\nu}
\end{aligned}$$

and are equivalent to $\bar{s} = H\bar{r}^\top$, with H given by (3.13).

Let the *error-locator polynomial* be defined as

$$\sigma(x) \triangleq \prod_{\ell=1}^{\nu} (1 + \alpha^{j_\ell} x) = 1 + \sigma_1 x + \sigma_2 x^2 + \cdots + \sigma_\nu x^\nu, \tag{3.15}$$

with roots equal to the *inverses of the error locations*. Then the following relation between the coefficients of $\sigma(x)$ and the syndromes holds (see, e.g., Peterson (1974), Lin and Costello (2005) p. 154, Peterson and Weldon (1972) p. 284):

$$\begin{pmatrix} S_{\nu+1} \\ S_{\nu+2} \\ \vdots \\ S_{2\nu} \end{pmatrix} = \begin{pmatrix} S_1 & S_2 & \cdots & S_\nu \\ S_2 & S_3 & \cdots & S_{\nu+1} \\ \vdots & \vdots & \ddots & \vdots \\ S_\nu & S_{\nu+1} & \cdots & S_{2\nu-1} \end{pmatrix} \begin{pmatrix} \sigma_\nu \\ \sigma_{\nu-1} \\ \vdots \\ \sigma_1 \end{pmatrix}. \tag{3.16}$$

Solving the *key equation* given by (3.16) constitutes the most computationally intensive operation in decoding BCH codes. Common methods to solve the key equation are:

1. *Berlekamp–Massey algorithm (BMA)*

   The BMA was invented by Berlekamp (1984), Massey (1969). This is a computationally efficient method to solve the key equation, in terms of the number of operations in $GF(2^m)$. The BMA is a popular choice to simulate or implement BCH and RS decoders in software.

2. *Euclidean algorithm (EA)*

   This is a method to solve the key equation in polynomial form, introduced in Sugiyama *et al.* (1975) and further studied in Mandelbaum (1977). Owing to its regular structure, the EA is widely used in hardware implementations of BCH and RS decoders.

[6] It will be seen later that for Reed–Solomon codes $e_j \in GF(2^m)$.

3. *Direct solution*

   Proposed first by Peterson (1974), this method directly finds the coefficients of $\sigma(x)$, by solving (3.16) as a set of linear equations. The term *PGZ (Peterson–Gorenstein–Zierler) decoder* is often used in the literature because in Gorenstein and Zierler (1961), Peterson's method is applied in decoding nonbinary BCH codes (Reed–Solomon codes). Naturally, as the complexity of inverting a matrix grows with the cube of the error-correcting capability, the direct solution method works only for small values of t_d. Solutions to (3.16) up to $t_d \leq 6$ are given in Michelson and Levesque (1985), Section 5.3.

3.5.1 General decoding algorithm for BCH codes

Figure 3.6 shows the block diagram of a decoder for BCH codes (both binary and nonbinary). The decoder consists of digital circuits and processing elements to accomplish the following tasks:

- Compute the syndromes,[7] by evaluating the received polynomial at the zeros of the code

$$S_i \triangleq \bar{r}(\alpha^i), \quad i = b, b+1, \ldots, b+2t_d-1. \tag{3.17}$$

- Find the coefficients of the *error-locator* polynomial $\sigma(x)$.
- Find the *inverses of the roots of* $\sigma(x)$, that is, the locations of the errors, $\alpha^{j_1}, \ldots, \alpha^{j_v}$.
- Find the *values of the errors* $e_{j_1}, \ldots, e_{j_v}$. (Not needed for binary codes.)
- *Correct the received word* with the error locations and values found.

One of the advantages gained in introducing $GF(2^m)$ arithmetic is that the decoding operations can be implemented with relatively simple circuitry and processing elements. As an example, Figure 3.7 shows how the computation of a syndrome S_j can be implemented in hardware. The multiplier is over $GF(2^m)$, and can be constructed with relatively simple combinatorial logic. Some details on the hardware design of processing elements over $GF(2^m)$ can be found, for example, in Peterson and Weldon (1972), and Chapters 5 and 10 of Wicker and Bhargava (1994).

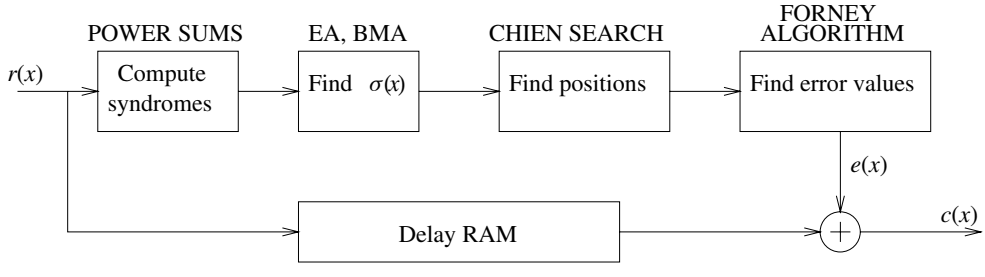


Figure 3.6 Architecture of a BCH decoder with $GF(2^m)$ arithmetic.

[7]For binary BCH codes, it holds that $S_{2i} = S_i^2$, which is useful in reducing the number of computations.

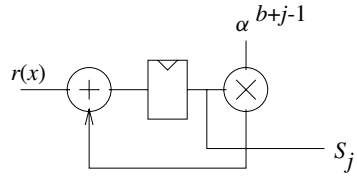


Figure 3.7 Example of a circuit for computing a syndrome.

3.5.2 The Berlekamp–Massey algorithm (BMA)

The BMA is best understood as an iterative procedure to construct a minimum linear feedback shift-register (LFSR) structure, like the one shown in Figure 3.8, that reproduces the known *syndrome sequence* $S_1, S_2, \ldots, S_{2t_d}$.

The goal of the BMA is to find a (connection) polynomial $\sigma^{(i+1)}(x)$ of minimal degree that satisfies the following equations, derived from (3.16):

$$\sum_{j=0}^{\ell_{i+1}} S_{k-j}\sigma_j^{(i+1)} = 0, \qquad \ell_i < k < i+1. \tag{3.18}$$

This is equivalent to requiring that $\sigma^{(i+1)}(x) = 1 + \sigma_1^{(i+1)}x + \cdots + \sigma_{\ell_{i+1}}^{(i+1)}x^{\ell_{i+1}}$ be the LFSR connection polynomial that produces the partial sequence of syndromes.

The *discrepancy* at iteration i defined as

$$d_i = S_{i+1} + S_i\sigma_1^{(i)} + \cdots + S_{i-\ell_i+1}\sigma_{\ell_i}^{(i)},$$

measures how well the LFSR reproduces the syndrome sequence, and constitutes a correction term used in computing the value of $\sigma^{(i+1)}$ in the next iteration. There are two cases to consider in the algorithm (Peterson 1974):[8]

- If $d_i = 0$, then the equations (3.18) are satisfied for

$$\sigma^{(i+1)}(x) = \sigma^i(x), \qquad \ell_{i+1} = \ell_i. \tag{3.19}$$

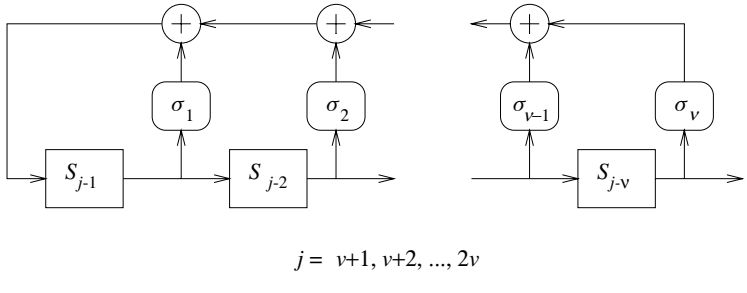


Figure 3.8 LFSR with taps $\sigma_1, \sigma_2, \ldots \sigma_v$ and output $S_1, S_2, \ldots, S_{2v}$.

[8]There is a variation of the BMA, inspired by Massey (1969), that is used in some references. This variation will be described in the next chapter. Of course, both versions of the BMA will give the same result!

- If $d_i \neq 0$: Let $\sigma^{(m)}(x)$ be the solution at iteration m, such that $-1 \leq m < i$, $d_m \neq 0$, and $(m - \ell_m)$ is maximal. Then

$$\begin{aligned} \sigma^{(i+1)}(x) &= \sigma^{i}(x) + d_i d_m^{-1} x^{i-m} \sigma^{(m)}(x) \\ \ell_{i+1} &= \max\{\ell_i, \ell_m + i - m\}. \end{aligned} \tag{3.20}$$

With an initial value of $i = 0$, the computation of $\sigma^{(i+1)}(x)$ continues until the conditions $i \geq \ell_{i+1} + t_d - 1$ or $i = 2t_d - 1$ or both are satisfied.

The initial conditions of the algorithm are:

$$\begin{aligned} \sigma^{(-1)}(x) = 1, \quad \ell_{-1} = 0, \quad d_{-1} = 1, \\ \sigma^{(0)}(x) = 1, \quad \ell_0 = 0, \quad d_0 = S_1. \end{aligned} \tag{3.21}$$

Also note that the BMA has control flow instructions (if-else). For this reason, its use is not favored in hardware implementations. However, in terms of the number of $GF(2^m)$ operations, it is very efficient. This version of the algorithm is implemented in most of the C programs to simulate BCH codes that can be found on the ECC web site.

Example 3.5.1 *Let C be the triple-error-correcting BCH (15,5,7) code of Example 3.3.3. As a reference, to check the numerical computations, the power and vector representations of $GF(2^4)$, with primitive polynomial $p(x) = 1 + x + x^4$, are listed in Table 3.2.*

A generator polynomial for C is

$$\bar{g}(x) = x^{10} + x^8 + x^5 + x^4 + x^2 + x + 1.$$

Table 3.2 Table of elements of $GF(2^4)$, $p(x) = x^4 + x + 1$.

Power	Vector
0	0000
1	0001
α	0010
α^2	0100
α^3	1000
α^4	0011
α^5	0110
α^6	1100
α^7	1011
α^8	0101
α^9	1010
α^{10}	0111
α^{11}	1110
α^{12}	1111
α^{13}	1101
α^{14}	1001

Suppose that the information polynomial is

$$\bar{u}(x) = x + x^2 + x^4.$$

Then the corresponding code polynomial is given by

$$\bar{v}(x) = x + x^2 + x^3 + x^4 + x^8 + x^{11} + x^{12} + x^{14}.$$

Let

$$\bar{r}(x) = 1 + x + x^2 + x^3 + x^4 + x^6 + x^8 + x^{11} + x^{14}$$

be the polynomial associated with a vector $\bar{r} = \bar{v} + \bar{e}$ received after transmission of code word $\bar{v}$ over a BSC channel.

We note that vector $\bar{e}$ corresponds to the error polynomial $\bar{e}(x) = 1 + x^6 + x^{12}$. Obviously, the decoder does not know this. However, in computing the syndromes below, this knowledge is used to simplify expressions.

The syndromes are:

$$\begin{aligned}
S_1 &= \bar{r}(\alpha) = 1 + \alpha^6 + \alpha^{12} = \alpha \\
S_2 &= S_1^2 = \alpha^2 \\
S_3 &= \bar{r}(\alpha^3) = 1 + \alpha^3 + \alpha^6 = \alpha^8 \\
S_4 &= S_2^2 = \alpha^4 \\
S_5 &= \bar{r}(\alpha^5) = 1 + 1 + 1 = 1 \\
S_6 &= S_3^2 = \alpha
\end{aligned}$$

Berlekamp–Massey algorithm:

- **Iteration 0:** *Initialize.* $\sigma^{(-1)}(x) = 1,\ \ell_{-1} = 0,\ d_{-1} = 1,\ \sigma^{(0)}(x) = 1,\ \ell_0 = 0,$ *and* $d_0 = S_1 = \alpha$.

- **Iteration 1:** $i = 0$, $d_0 = \alpha \neq 0$, $m = -1$ *maximizes* $(-1 + 0) = -1$ *for* $d_{-1} \neq 0$.

$$\begin{aligned}
\sigma^{(1)}(x) &= \sigma^{(0)}(x) + d_0 d_{-1}^{-1} x^{(0-(-1))} \sigma^{(-1)}(x) = 1 + \alpha x, \\
\ell_1 &= \max\{\ell_0, \ell_{-1} + 0 - (-1)\} = 1, \\
\ell_1 + 3 - 1 &\leq 0 \ \text{? No:} \\
d_1 &= S_2 + S_1 \sigma_1^{(1)} = \alpha^2 + \alpha(\alpha) = 0.
\end{aligned}$$

- **Iteration 2:** $i = 1$, $d_1 = 0$,

$$\begin{aligned}
\sigma^{(2)}(x) &= \sigma^{(1)}(x) = 1 + \alpha x, \\
\ell_2 &= \ell_1, \\
\ell_2 + 3 - 1 &\leq 1 \ \text{? No:} \\
d_2 &= S_3 + s_2 \sigma_1^{(1)} = \alpha^8 + \alpha^2(\alpha) = \alpha^{13}.
\end{aligned}$$

- **Iteration 3:** $i = 2$, $d_2 = \alpha^{13} \neq 0$, $m = 0$ *maximizes* $(0 - 0) = 0$ *for* $d_0 \neq 0$.

$$\begin{aligned}
\sigma^{(3)}(x) &= \sigma^{(2)}(x) + d_2 d_0^{-1} x^{(2-0)} \sigma^{(0)}(x) \\
&= (1 + \alpha x) + \alpha^{13}(\alpha^{-1}) x^2(1) = 1 + \alpha x + \alpha^{12} x^2, \\
\ell_3 &= \max\{\ell_2, \ell_0 + 2 - 0\} = 2, \\
\ell_3 + 3 - 1 &\leq 2 \ \text{? No:} \\
d_3 &= S_4 + S_3 \sigma_1^{(3)} + S_2 \sigma_2^{(3)} = \alpha^4 + \alpha^8(\alpha) + \alpha^2(\alpha^{12}) = 0.
\end{aligned}$$

- **Iteration 4:** $i = 3$, $d_3 = 0$,

$$\begin{aligned}\sigma^{(4)}(x) &= \sigma^{(3)}(x) = 1 + \alpha x + \alpha^{12}x^2,\\ \ell_4 &= \ell_3,\\ \ell_4 + 3 - 1 &\leq 3 \ \text{? No:}\\ d_4 &= S_5 + S_4\sigma_1^{(4)} + S_3\sigma_2^{(4)} = 1 + \alpha^4(\alpha) + \alpha^8\alpha^{12} = 1.\end{aligned}$$

- **Iteration 5:** $i = 4$, $d_4 = 1 \neq 0$, $m = 2$ *maximizes* $(2 - 1) = 1$ *for* $d_2 \neq 0$.

$$\begin{aligned}\sigma^{(5)}(x) &= \sigma^{(4)}(x) + d_4 d_2^{-1} x^{(4-2)}\sigma^{(2)}(x)\\ &= (1 + \alpha x + \alpha^{12}x^2) + (1)(\alpha^{13})^{-1}x^2(1 + \alpha x)\\ &= 1 + \alpha x + \alpha^7 x^2 + \alpha^3 x^3,\\ \ell_5 &= \max\{\ell_4, \ell_2 + 4 - 2\} = 3,\\ \ell_5 + 3 - 1 &\leq 4 \ \text{? No:}\\ d_5 &= S_6 + S_5\sigma_1^{(5)} + S_4\sigma_2^{(5)} + S_3\sigma_3^{(5)} = \alpha + 1 \cdot \alpha + \alpha^4(\alpha^7) + \alpha^8(\alpha^3) = 0.\end{aligned}$$

- **Iteration 6:** $i = 5$, $d_5 = 0$,

$$\begin{aligned}\sigma^{(6)}(x) &= \sigma^{(5)}(x) = 1 + \alpha x + \alpha^7 x^2 + \alpha^3 x^3,\\ \ell_6 &= \ell_5 = 3,\\ \ell_6 + 3 - 1 &\leq 3 \ \text{? Yes: End.}\end{aligned}$$

Therefore $\sigma(x) = 1 + \alpha x + \alpha^7 x^2 + \alpha^3 x^3$.

That the odd numbered iterations always gave $d_i = 0$ in the previous example is no accident. For *binary* BCH codes this is always the case. The iterations can proceed with only even values of i, with the same results. The only difference is that the stopping criterion needs to be changed to

$$i \geq \ell_{i+2} + t_d - 2.$$

As a result, decoding complexity is reduced. The reader is invited to solve $\sigma(x)$ for the previous example, using only three iterations.

3.5.3 PGZ decoder

This decoding algorithm was first considered by Peterson (1974). A solution to the key equation (3.16) is to be found using standard techniques for solving a set of linear equations. This solution gives the coefficients of $\sigma(x)$. The decoding problem is that the number of actual errors is unknown. Therefore, a guess has to be made as to the actual number of errors, ν, in the received word. Assume that not all the syndromes, S_i, $1 \leq i \leq 2t$, are equal to zero. (If all syndromes are zero, then the received word is a code word and decoding finishes!)

The decoder assumes that the maximum number of errors has occurred, $\nu_{\max} = t_d$, and checks if the determinant Δ_i, for $i = \nu_{\max} = t_d$,

$$\Delta_i = \det \begin{pmatrix} S_1 & S_2 & \cdots & S_i \\ S_2 & S_3 & \cdots & S_{i+1} \\ \vdots & \vdots & \ddots & \vdots \\ S_i & S_{i+1} & \cdots & S_{2i-1} \end{pmatrix} \tag{3.22}$$

is equal to zero. If it is, then a smaller number of errors must have occurred. The value of i is decreased by one, and Δ_i tested again, repeating the procedure if necessary, until $i = 1$. Otherwise, if $\Delta_i \neq 0$, the inverse of the syndrome matrix is computed and the values of $\sigma_1, \sigma_2, \ldots, \sigma_\nu$ are found, where $\nu = i$. In the event that $\Delta_i = 0, i = 1, 2, \ldots, t_d$, decoding is unsuccessful and an uncorrectable error pattern is detected.

Example 3.5.2 *In this example, the error-locator polynomial for the BCH* $(15, 5, 7)$ *code in Example 3.5.1 is found by the PGZ decoding algorithm. First, assume that* $i = t_d = 3$ *errors occurred. Then the determinant* Δ_3 *is computed (using cofactors) as follows:*

$$\begin{aligned}\Delta_3 = \det \begin{pmatrix} \alpha & \alpha^2 & \alpha^8 \\ \alpha^2 & \alpha^8 & \alpha^4 \\ \alpha^8 & \alpha^4 & 1 \end{pmatrix} &= \alpha(\alpha^8 + \alpha^8) + \alpha^2(\alpha^2 + \alpha^{12}) + \alpha^8(\alpha^6 + \alpha) \\ &= \alpha^{2+7} + \alpha^{8+11} = \alpha^{14}.\end{aligned}$$

Therefore $\Delta_3 \neq 0$ *and three errors are assumed to have occurred. Substituting the syndromes found in Example 3.5.1 into the key equation (3.16),*

$$\begin{pmatrix} \alpha & \alpha^2 & \alpha^8 \\ \alpha^2 & \alpha^8 & \alpha^4 \\ \alpha^8 & \alpha^4 & 1 \end{pmatrix} \begin{pmatrix} \sigma_3 \\ \sigma_2 \\ \sigma_1 \end{pmatrix} = \begin{pmatrix} \alpha^4 \\ 1 \\ \alpha \end{pmatrix}. \tag{3.23}$$

Note that $\Delta_3^{-1} = \alpha^{-14} = \alpha^{-14}(1) = \alpha^{-14}\alpha^{15} = \alpha$. *The solution to (3.23) is found to be:*

$$\begin{aligned}\sigma_3 &= \alpha \det \begin{pmatrix} \alpha^4 & \alpha^2 & \alpha^8 \\ 1 & \alpha^8 & \alpha^4 \\ \alpha & \alpha^4 & 1 \end{pmatrix} = \alpha(\alpha^7 + \alpha^{12}) = \alpha^3, \\ \sigma_2 &= \alpha \det \begin{pmatrix} \alpha & \alpha^4 & \alpha^8 \\ \alpha^2 & 1 & \alpha^4 \\ \alpha^8 & \alpha & 1 \end{pmatrix} = \alpha(\alpha^{11} + \alpha) = \alpha^7, \\ \sigma_1 &= \alpha \det \begin{pmatrix} \alpha & \alpha^2 & \alpha^4 \\ \alpha^2 & \alpha^8 & 1 \\ \alpha^8 & \alpha^4 & \alpha \end{pmatrix} = \alpha(\alpha^{15} + \alpha^{15} + \alpha^{15}) = \alpha.\end{aligned}$$

It follows that $\sigma(x) = 1 + \alpha x + \alpha^7 x^2 + \alpha^3 x^3$, *which is the same as the result obtained by the BMA in example 3.5.1.*

3.5.4 Euclidean algorithm

This algorithm is a well-known recursive procedure to find the greatest common divisor (GCD) between two polynomials (or integers). Its application to BCH decoding is described next. Let the *error evaluator polynomial* be defined as $\Lambda(x) = \sigma(x)S(x)$, with a *syndrome polynomial*

$$S(x) = 1 + S_1 x + \cdots + S_{2t_d} x^{2t_d}. \tag{3.24}$$

From Equation (3.16), it follows that

$$\Lambda(x) = \sigma(x)S(x) \bmod x^{2t_d+1}. \tag{3.25}$$

The decoding problem can be translated into finding the polynomial $\Lambda(x)$ satisfying (3.25). This can be achieved by applying the extended EA to the polynomials $r_0(x) = x^{2t_d+1}$ and $r_1(x) = S(x)$, such that if at the j-th step

$$r_j(x) = a_j(x)x^{2t_d+1} + b_j(x)S(x),$$

with $\deg[r_j(x)] \leq t_d$, then $\Lambda(x) = r_j(x)$ and $\sigma_i(x) = b_j(x)$. We note that there is no interest, from the decoding viewpoint, in polynomial $a_i(x)$. The extended EA for computing the GCD of two polynomials is provided below.

Euclidean algorithm: Compute GCD($r_0(x)$,$r_1(x)$)

- Inputs: $r_0(x), r_1(x), \deg[r_0(x)] \geq \deg[r_1(x)]$
- Initial conditions: $a_0(x) = 1, b_0(x) = 0, a_1(x) = 0, b_1(x) = 1$.
- At step j ($j \geq 2$), apply *long division* to $r_{j-2}(x)$ and $r_{j-1}(x)$,

  $$r_{j-2}(x) = q_j(x)r_{j-1}(x) + r_j(x), \qquad 0 \leq \deg[r_j(x)] < \deg[r_{j-1}(x)]$$

  and compute

  $$a_j(x) = a_{j-2}(x) - q_j(x)a_{j-1}(x), \quad b_j(x) = b_{j-2}(x) - q_j(x)b_{j-1}(x).$$

- Stop at iteration j_{last} when $\deg[r_{j_{\text{last}}}(x)] = 0$.

Then $GCD(r_0(x), r_1(x)) = r_k(x)$, k being the largest nonzero integer such that $r_k(x) \neq 0$ and $k < j_{\text{last}}$.

Example 3.5.3 *In this example, the error-locator polynomial $\sigma(x)$ for the BCH $(15, 5, 7)$ code in Example 3.5.1 is computed using the EA.*

- **Initial conditions**

$$\begin{aligned} r_0(x) &= x^7, \\ r_1(x) &= S(x) = 1 + \alpha x + \alpha^2 x^2 + \alpha^8 x^3 + \alpha^4 x^4 + x^5 + \alpha x^6, \\ b_0(x) &= 0, \\ b_1(x) &= 1. \end{aligned}$$

- $j = 2$:

$$\begin{aligned} x^7 &= (1 + \alpha x + \alpha^2 x^2 + \alpha^8 x^3 + \alpha^4 x^4 + x^5 + \alpha x^6)(\alpha^{14} x + \alpha^{13}) \\ &\quad + \alpha^8 x^5 + \alpha^{12} x^4 + \alpha^{11} x^3 + \alpha^{13}. \\ r_2(x) &= \alpha^8 x^5 + \alpha^{12} x^4 + \alpha^{11} x^3 + \alpha^{13}, \\ q_2(x) &= \alpha^{14} x + \alpha^{13}, \text{ and} \\ b_2(x) &= b_0(x) + q_2(x)b_1(x) = \alpha^{14} x + \alpha^{13}. \end{aligned}$$

- $j = 3$:

$$\begin{aligned} S(x) &= (\alpha^8 x^5 + \alpha^{12} x^4 + \alpha^{11} x^3 + \alpha^{13})(\alpha^8 x + \alpha^2) \ + \alpha^{14} x^4 + \alpha^3 x^3 + \alpha^2 x^2 + \alpha^{11} x. \\ r_3(x) &= \alpha^{14} x^4 + \alpha^3 x^3 + \alpha^2 x^2 + \alpha^{11} x, \\ q_3(x) &= \alpha^8 x + \alpha^2, \text{ and} \\ b_3(x) &= b_1(x) + q_3(x)b_2(x) = \alpha^7 x^2 + \alpha^{11} x. \end{aligned}$$

- $j = 4$:

$$\alpha^8 x^5 + \alpha^{12} x^4 + \alpha^{11} x^3 + \alpha^{13} = (\alpha^{14} x^4 + \alpha^3 x^3 + \alpha^2 x^2 + \alpha^{11} x)(\alpha^9 x) + \alpha^5 x + \alpha^{13}.$$
$$r_4(x) = \alpha^5 x + \alpha^{13},$$
$$q_4(x) = \alpha^9 x, \text{ and}$$
$$b_4(x) = b_2(x) + q_4(x) b_3(x) = \alpha x^3 + \alpha^5 x^2 + \alpha^{14} x + \alpha^{13}.$$

Because $\deg[r_4(x)] = 1 \leq 3$*, the algorithm stops.*

It follows that $\sigma(x) = b_4(x) = \alpha^{13}(1 + \alpha x + \alpha^7 x^2 + \alpha^3 x^3)$*, which has the same roots as the polynomial obtained by the BMA and PGZ decoder (see Example 3.5.4 below), and differs only by a constant factor.*[9]

As the example above illustrates, generally the error-locator polynomial obtained with the EA will differ from that obtained by BMA or PGZ decoding by a constant factor. In decoding, however, the roots of these polynomials are of interest and not their coefficients, and thus any of the three methods discussed so far can be used to determine $\sigma(x)$.

3.5.5 Chien search and error correction

To find the roots of $\sigma(x)$, a simple trial-and-error procedure – called *Chien search* – is performed. All nonzero elements β of $GF(2^m)$ are generated in sequence $1, \alpha, \alpha^2, \ldots$ and the condition $\sigma(\beta^{-1}) = 0$ tested. This process is easy to implement in hardware. Moreover, finding roots (factoring) of polynomials over $GF(2^m)$ is a challenging mathematical problem that remains to be solved.

For binary BCH codes, once the error locations $j_1, \ldots, j_\nu$ are known, the corresponding bits in the received word are complemented

$$\hat{v}_{j_\ell} = v_{j_\ell} + 1, \qquad 1 \leq \ell \leq \nu$$

and the estimated code word $\hat{v}(x)$ generated.

Example 3.5.4 *In Example 3.5.1, the roots of* $\sigma(x)$ *are* 1, $\alpha^9 = \alpha^{-6}$ *and* $\alpha^3 = \alpha^{-12}$*. Stated in a different way,* $\sigma(x)$ factors *as*

$$\sigma(x) = (1 + x)(1 + \alpha^6 x)(1 + \alpha^{12} x).$$

Consequently, the estimated error polynomial is $\bar{e}(x) = 1 + x^6 + x^{12}$*, and*

$$\bar{r}(x) = x + x^2 + x^3 + x^4 + x^6 + x^8 + x^{11} + x^{14}.$$

Three errors have been corrected.

3.5.6 Errors-and-erasures decoding

There are many situations in which a decision on a received symbol is not considered reliable. An example is binary transmission over an AWGN channel, with bits mapped to

[9] The normalized polynomial $\sigma_{\text{norm}}(x) = \sigma_0^{-1}\sigma(x)$ is identical to the one obtained in the BMA and PGZ procedures.

real amplitudes, for example, binary phase shift keying (BPSK) with $0 \mapsto +1$ and $1 \mapsto -1$. If the received values are too close to zero, then it may be more advantageous, from the viewpoint of minimizing the probability of a decoding error, to declare a "no-decision". In such a case, the received symbol is "erased" and it is called an *erasure*.[10] Declaring erasures is the simplest form of *soft-decision*, which will be the focus of attention in Chapter 7.

Introduction of erasures has the advantage, with respect to errors-only decoding, that the *positions are known to the decoder*. Let d be the minimum distance of a code, ν be the number of errors and μ be the number of erasures contained in a received word. Then, the minimum Hamming distance between code words is reduced to at least $d - \mu$ in the nonerased positions. It follows that the error-correcting capability is $\lfloor (d - \mu - 1)/2 \rfloor$ and the following relation holds:

$$d > 2\nu + \mu. \tag{3.26}$$

The above inequality is intuitively satisfying: for a fixed minimum distance, it is twice as difficult to correct an error as it is to correct an erasure, because the erased positions are already known.

For binary linear codes, including binary BCH codes, erasures can be corrected with the following method:

1. Place zeros in all erased positions and decode to a code word $\hat{v}_0(x)$.
2. Place ones in all erased positions and decode to a code word $\hat{v}_1(x)$.
3. Choose as decoded word the closest $\hat{v}_j(x)$ to the received word $\bar{r}(x)$ in the nonerased positions, $j = 0, 1$. (Alternatively, choose the code word that requires the smallest number of error corrections, (Michelson and Levesque 1985).)

As a result, erasures can be corrected with two rounds of errors-only decoding.

Example 3.5.5 *Consider a binary cyclic Hamming* $(7, 4, 3)$ *code with*

$$\bar{g}(x) = 1 + x + x^3,$$

and the Galois field $GF(2^3)$ *with a primitive element* α *such that* $p(\alpha) = 1 + \alpha + \alpha^3 = 0$. *Suppose that* $\bar{v}(x) = x + x^2 + x^4$ *is transmitted and that* $\mu = 2$ *erasures are introduced at the receiver. Since* $d = 3 > \mu$, *these erasures can be corrected. Let*

$$\bar{r}(x) = f + x + x^2 + fx^3 + x^4$$

be the polynomial associated with the received vector, where f *denotes an erasure.*

First decoding ($f = 0$)*:*

$\bar{r}_0(x) = x + x^2 + x^4$. *The syndromes are* $S_1 = \bar{r}_0(\alpha) = 0$, *and* $S_2 = S_1^2 = 0$. *Therefore,* $\hat{v}_0(x) = \bar{r}_0(x) = x + x^2 + x^4$.

Second decoding ($f = 1$)*:*

$\bar{r}_1(x) = 1 + x + x^2 + x^3 + x^4$. *The syndromes are* $S_1 = \alpha$ *and* $S_2 = \alpha^2$. *Equation (3.16) in this case is:* $\alpha\sigma_1 = \alpha^2$. *Therefore* $\sigma(x) = 1 + \alpha x$, $\bar{e}(x) = x$ *and*

$$\hat{v}_1(x) = \bar{r}_1(x) + \bar{e}(x) = 1 + x^2 + x^3 + x^4,$$

[10] In information theoretical terms, the BSC channel becomes a two-input three-output *binary erasure channel* (BEC) (Cover and Thomas 1991).

which differs from $\bar{r}(x)$ in one of the nonerased positions. As a result, $\hat{v}_0(x)$ is selected as the decoded word. Two erasures have been corrected.

3.6 Weight distribution and performance bounds

In general, the weight distribution of a binary linear (n, k) code C can be obtained by enumerating all 2^k code vectors $\bar{v}$ and computing their Hamming weight. Obviously, for large values of k, this is a formidable task. Let A_w denote the number of code words of Hamming weight w. The *MacWilliams identity* relates the *weight distribution sequence* (WDS) of a linear code, $A(x) \triangleq A_0 + A_1 x + x_2 x^2 + \cdots + A_n x^n$, with the WDS of its *dual*[11] $(n, n-k)$ *code* $C^{\perp}$, $B(x) \triangleq B_0 + B_1 x + x_2 x^2 + \cdots + B_n x^n$, by

$$A(x) = 2^{-n+k}(1+x)^n B\left[\frac{1-x}{1+x}\right], \tag{3.27}$$

which can also be expressed as

$$B(x) = 2^{-k}(1+x)^n A\left[\frac{1-x}{1+x}\right]. \tag{3.28}$$

Therefore, for high-rate codes, it is simpler to compute the WDS $B(x)$ of the dual code and then use (3.27) to compute $A(x)$. Alternatively, the WDS of a low-rate code is easy to compute and the WDS of the dual code can be obtained using (3.28).

For some classes of codes, the trellis structure can be used.[12] In Desaki *et al.* (1997), a trellis-based method for computing the weight distribution of *extended BCH codes* of length 128 is given. For shorter lengths, the weight distributions are not difficult to compute using MacWilliams identity.

The weight distribution of binary cyclic codes can be obtained directly from that of binary *extended cyclic codes* with good trellis structure, as in the following text. A binary extended cyclic code is obtained from a binary cyclic code by appending, at the start of each code vector, an *overall parity-check bit*. The extended code of a cyclic code is no longer cyclic. Let H denote the parity-check of the cyclic code, then the parity-check matrix of the extended code, denoted as H_{ext}, is given by

$$H_{\text{ext}} = \begin{pmatrix} 1 & 1 & \cdots & 1 \\ 0 & & & \\ 0 & & H & \\ 0 & & & \end{pmatrix}. \tag{3.29}$$

Appendix A lists the weight distributions of all extended binary BCH codes of length up to 128. These data files are available on the ECC web site. The appendix lists only those terms of Hamming weight up to $(n+1)/2$, for $n = 2^m - 1$. In the case of extended BCH codes, it holds that $A^{(\text{ext})}_{n+1-w} = A^{(\text{ext})}_w$. The data is useful in finding the weight distribution of the binary cyclic BCH codes of length up to 127. This can be done by the application

[11] Recall that the dual code $C^{\perp}$ has generator matrix H, where H is the parity-check matrix of C.

[12] More on trellises can be found in Chapter 7.

of the following result (which is obtained as a special case of Theorems 8.14 and 8.15 in Peterson and Weldon (1972)):

Let C be a binary cyclic BCH (n, k) code with WDS $A(x)$, obtained by eliminating the overall parity-check bit in a binary extended BCH $(n+1, k)$ code C_{ext}, with WDS $A^{(\text{ext})}(x)$. Then, for w even,

$$\begin{aligned}(n+1)A_{w-1} &= wA_w^{(\text{ext})},\\ wA_w &= (n+1-w)A_{w-1}\end{aligned} \tag{3.30}$$

Example 3.6.1 *Consider the binary extended Hamming $(8, 4, 4)$ code. This code has parity-check matrix (see also Example 3.1.5),*

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

It can be easily verified that $A^{(\text{ext})}(x) = 1 + 14x^4 + x^8$. To compute the WDS of the binary cyclic Hamming $(7, 4, 3)$, use (3.30) to obtain

$$\begin{aligned} 8A_3 = 4A_4^{(\text{ext})} &\longrightarrow A_3 = 7,\\ 4A_4 = (8-4)A_3 &\longrightarrow A_4 = 7,\\ 8A_7 = 8A_8^{(\text{ext})} &\longrightarrow A_7 = 1.\end{aligned}$$

3.6.1 Error performance evaluation

With knowledge of the WDS of a code C, the error performance can be estimated, as discussed in Chapter 1. The WDS and the union bound (1.36) give good estimates of the error performance of code C with binary transmission over an AWGN channel.

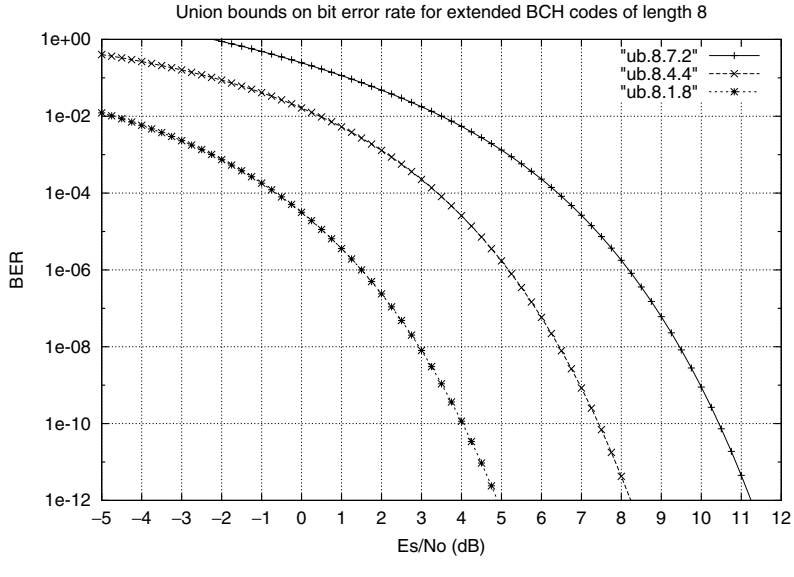


Figure 3.9 Union bounds on the BER for extended BCH code of length 8.

As an example, the union bound was evaluated using Appendix A for extended BCH codes of length 8 to 64. The results are shown in Figures 3.9 to 3.12. Using the WDS from the appendix and the union bound for flat Rayleigh fading channels (1.42), with Monte Carlo integration and the term A_w replaced by wA_w/n to account for bit errors, Figure 3.13 shows union bounds on the bit error rate (BER) for extended BCH codes of length 8. Bound for other codes can be computed in the same way.

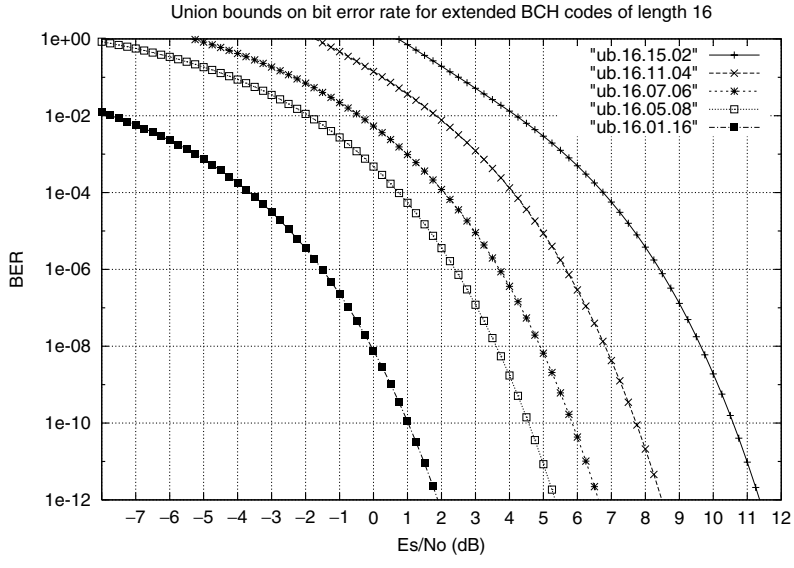


Figure 3.10 Union bounds on the BER for extended BCH code of length 16.

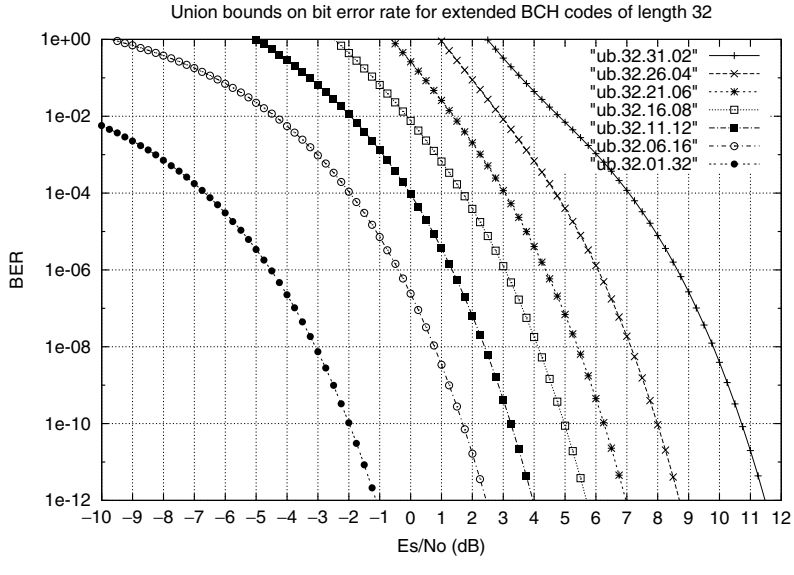


Figure 3.11 Union bounds on the BER for extended BCH code of length 32.

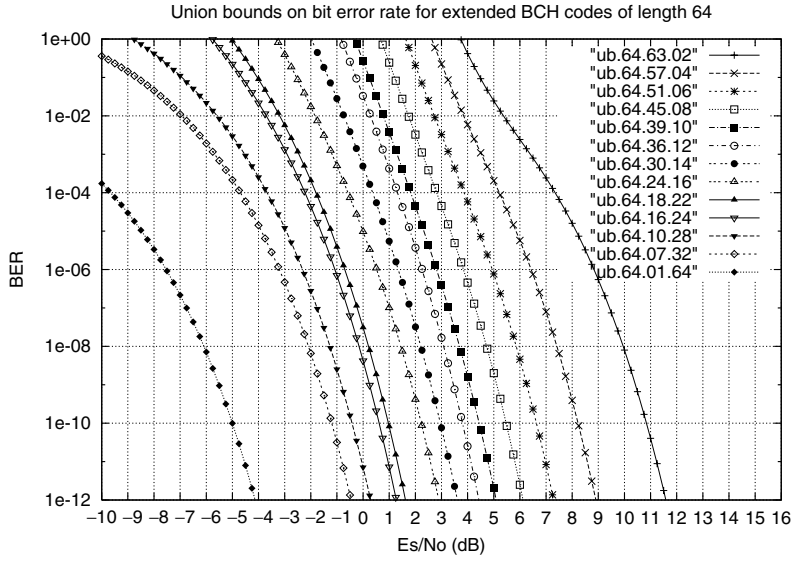


Figure 3.12 Union bounds on the BER for extended BCH code of length 64.

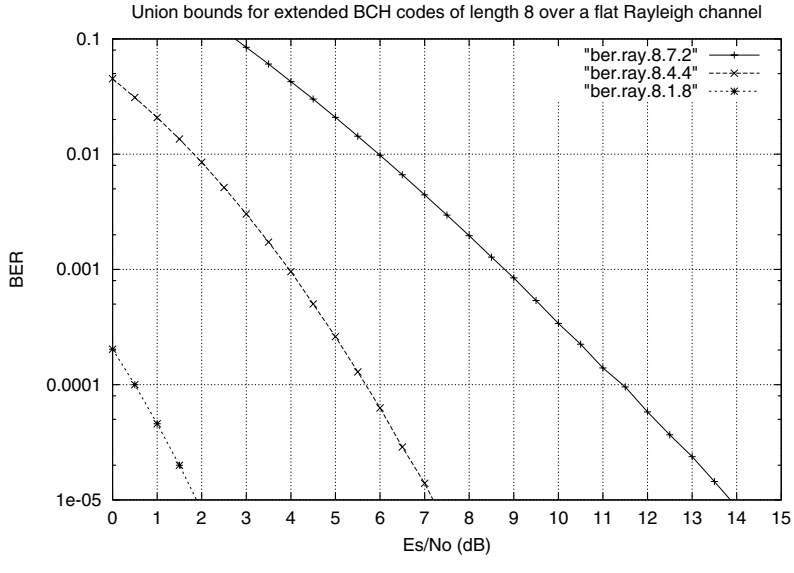


Figure 3.13 Union bounds on the BER for extended BCH code of length 8. Binary transmission over a flat Rayleigh fading channel.

The main point of this section is that, before choosing a particular soft-decision decoding algorithm (see Chapter 7), the weight distribution and the union bound can give an estimate of the performance that is achievable over a certain channel. For the AWGN channel and the flat Rayleigh fading channel, the union bounds are tight at BERs below 10^{-4}.

Problems

1. Prove that if C is a binary cyclic code with $\bar{g}(x) = x + 1$, then C is the $(n, n-1, 2)$ single-parity-check (SPC) code. (Hint: If $\bar{g}(x)|\bar{v}(x)$ and $\bar{g}(a) = 0$ then $\bar{v}(a) = 0$.)

2. (Peterson and Weldon (1972)). Show that the all-one vector is a code word of a binary cyclic code if the polynomial $x + 1$ is a factor of the generator polynomial.

3. Consider the binary cyclic Hamming $(7, 4, 3)$ code.

   (a) Prove that there are two different versions of this code. (Hint: Over $GF(2)$, the polynomial $x^7 + 1 = (x + 1)(x^3 + x + 1)(x^3 + x^2 + 1)$.)

   (b) The set $O(\bar{v}(x)) \triangleq \left\{\bar{v}^{(i)}(x),\ 0 \le i < \ell_i\right\}$ is called the orbit of $\bar{v}(x)$. Show that the code words of C are contained in two orbits of size 7 and one of size 1. (Hint: Note that $\mathcal{W}(C) = \{1, 0, 0, 7, 7, 0, 0, 1\}$ and $\mathrm{wt}_H(\bar{v}(x)) = \mathrm{wt}_H(\bar{v}^{(i)}(x))$.)

4. Let C be a MLS $(7, 3, 4)$ code with generator polynomial

$$\bar{g}(x) = x^4 + x^3 + x^2 + 1.$$

   (a) Show that $\mathcal{W}(C) = \{1, 0, 0, 0, 7, 0, 0, 0\}$.

   (b) Suppose that $\bar{r}(x) = x^5 + x + 1$. Find the most likely code polynomial $\bar{v}(x)$ if transmission took place over a BSC with $p < 1/2$.

5. Let C be a binary cyclic Hamming $(7, 4, 3)$ code with generator polynomial

$$\bar{g}(x) = x^3 + x^2 + 1.$$

   (a) Sketch carefully the encoding and decoding circuits of C. Use a syndrome or Meggit decoder architecture.

   (b) The received polynomial is $\bar{r}(x) = x^5 + x^2$. With the decoding circuit in part (a), show cycle-by-cycle the contents of the syndrome register and determine the most likely code polynomial $\bar{r}(x)$.

6. A Galois field $F = GF(2^4)$ is generated by a primitive element α with

$$p(\alpha) = \alpha^4 + \alpha^3 + 1 = 0.$$

   (a) Build the power-polynomial-vector table of F.

(b) Find the value of X in F in the following expressions:

(i) $\alpha^5 X = 1$

(ii) $(\alpha^2 + \alpha^9)^3 X = \alpha^{10}$

(i) $\alpha^{11} + X = \alpha$

(c) Determine the cyclotomic sets modulo 15 and use them to find the minimal polynomials $\phi_3(x)$ and $\phi_5(x)$.

7. Find the generator polynomial of a (1,2)-order EG code of length 63.

8. Show that the (0,2)-order EG code of length 15 is a binary BCH $(15, 5, 7)$ code.

9. Show that the (1,1)-order EG code of length 15 is a binary BCH $(15, 7, 5)$ code.

10. Find the generator polynomial of a (1,2)-order PG code of length 21. This code is a binary $(21, 11)$ difference-set code. See Lin and Costello (2005).

11. (MacWilliams and Sloane (1977)). Use the BCH bound to show that the dual of a binary cyclic Hamming code, a binary $(2^m - 1, m, d)$ simplex code, has the minimum distance $d = 2^{m-1}$.

12. (MacWilliams and Sloane (1977)). A code C is *reversible* if $(c_0, c_1, \ldots, c_{n-1}) \in C$ implies $(c_{n-1}, c_{n-2}, \ldots, c_0) \in C$. For example, the code $C = \{000, 110, 101, 011\}$ is reversible. Show that the BCH code with $b = -t$ and designed distance $d = 2t + 2$ is reversible.

13. Let C be a binary cyclic triple-error-correcting BCH $(15, 5, 7)$ code.

(a) Using the field F in problem 2, determine the generator polynomial of C.

(b) The information message is $\bar{u} = (10101)$. Determine the corresponding code polynomial $\bar{v}(x)$ in C.

(c) Suppose that the error polynomial is $\bar{e}(x) = x^{12} + x^5 + x$. Compute the received polynomial $\bar{r}(x) = \bar{v}(x) + \bar{e}(x)$. Using $\bar{r}(x)$, do the following:

(i) Determine the syndromes $S_i = \bar{r}(\alpha^i)$, $1 \leq i \leq 6$.

(ii) Determine the error-locator polynomial $\sigma(x)$.

(iii) Determine the error locations α^{j_i}, $i = 1, 2, 3$.

(iv) Correct the errors in $\bar{r}(x)$ by constructing the estimated error polynomial $\tilde{e}(x)$ and the estimated code polynomial $\tilde{c}(x) = \bar{r}(x) + \tilde{e}(x)$. Verify that errors have been corrected successfully.

14. Let $C_i \triangleq \{i, 2i, 4i, \ldots, 2^{m_i - 1} i\} \bmod 2^m - 1$ be a cyclotomic set, where i is the smallest integer in C_i. Prove that i cannot be an even number.

15. Factor the polynomials $(x^4 + 1)$ and $(x^5 + 1)$ over $GF(2)$.

16. Prove that for a binary BCH code, the following relation between syndromes holds: $S_{2i} = S_i^2$.

17. Let C be a double-error-correcting $(15, 7, 5)$ BCH code. Suppose that the all-zero code word is transmitted and that there are erasures in the last four consecutive positions so that
$$\bar{r}(x) = f + fx + fx^2 + fx^3,$$
where f denotes an erasure. Find the most likely code polynomial that was transmitted.

18. Prove the existence of a binary cyclic $(63, 58)$ code. (Hint: How many binary cyclic codes of length 63 are there?)

19. Let C_1 be a double-error-correcting BCH (15,7,5) code and C_2 be a triple-error-correcting BCH (15,5,7) code.

    (a) Find generator matrices G_1 and G_2 of C_1 and C_2 respectively.

    (b) Compute the weight distributions $\mathcal{W}(C_1)$ and $\mathcal{W}(C_2)$. (Hint: You may use a computer program for this task.)

    (c) Estimate the performance of codes C_1 and C_2 (BER curves) with soft-decision decoding over an AWGN channel. Compare the curves with respect to uncoded BPSK modulation. What are the respective coding gains?

20. Let A_ℓ be an $\ell \times \ell$ Vandermonde matrix over a field.

    (a) Show that $\det(A_\ell) \neq 0$, that is, the rank of A_ℓ is ℓ.

    (b) Show that any $\nu \times \nu$ submatrix A_ν of A_ℓ has rank ν.

21. Let $2^m - 1 = n_1 n_2$, where $n_1 > 1$ and $n_2 > 1$.

    (a) Show that the element $\beta = \alpha^{n_1}$, where α is a primitive element in $GF(2^m)$, has order n_2. That is, n_2 is the smallest positive integer such that $\beta^{n_2} = 1$.

    (b) Using β, a *binary nonprimitive BCH code* can be constructed. With $n = 63$, determine the generator polynomial of a binary nonprimitive BCH $(21, 12, 5)$ code.

4

Nonbinary BCH codes: Reed–Solomon codes

In this chapter, one of the most celebrated class of ECC schemes is introduced and their encoding and decoding algorithms explained. Reed–Solomon (RS) codes have found numerous applications in digital storage and communication systems. Examples include the famous RS $(255, 223, 33)$ code for NASA space communications, shortened RS codes over $GF(2^8)$ for CD-ROM, DVD and Terrestrial Digital HDTV transmission applications, an extended RS $(128, 122, 7)$ code over $GF(2^7)$ for cable modems, among many others.

4.1 RS codes as polynomial codes

Similar to Reed–Muller (RM) codes, RS codes can be defined as code words with components equal to the evaluation of a certain polynomial. As a matter of fact, this was the way that RS codes were originally defined by Reed and Solomon in Reed and Solomon (1960). RM codes, finite-geometry codes (Lin and Costello 2005) and RS codes are all members of a large class of codes: *Polynomial codes* (Peterson and Weldon 1972), which are closely related to *algebraic-geometry (AG) codes* (Pretzel 1998). Let

$$\bar{u}(x) = u_0 + u_1 x + \cdots + u_{k-1} x^{k-1} \tag{4.1}$$

be an information polynomial, with $u_i \in GF(2^m)$, $1 \leq i < k$. Clearly, there are 2^{mk} such polynomials. By evaluating (4.1) over the nonzero elements of $GF(2^m)$, a code word in an RS $(2^m - 1, k, d)$ code of length 2^m is obtained,

$$\bar{v} = \left(u(1) \quad u(\alpha) \quad u(\alpha^2) \quad \cdots \quad u(\alpha^{2^m-2})\right). \tag{4.2}$$

4.2 From binary BCH to RS codes

RS codes can also be interpreted as *nonbinary* BCH codes. That is, RS codes are BCH codes in which the values of the code coefficient are taken from $GF(2^m)$. In particular,

The Art of Error Correcting Coding, Second Edition Robert H. Morelos-Zaragoza
 ISBN: 0-470-01558-6

for a t_d-error correcting RS code, the zeros of the code are $2t_d$ consecutive powers of α. Moreover, because over $GF(2^m)$ minimal polynomials are of the form $\phi_i(x) = (x - \alpha^i)$, $0 \leq i < 2^m - 1$, see Equation (3.9), the factors of the generator polynomial are now *linear*, and

$$\bar{g}(x) = \prod_{j=b}^{b+2t_d-1} (x + \alpha^j), \tag{4.3}$$

where b is an integer, usually $b = 0$ or $b = 1$.

It follows from (4.3), and the BCH bound on page 53, that the minimum distance of a RS (n, k, d) code C over $GF(2^m)$ is $d \geq n - k + 1$. On the other hand, the Singleton bound (Singleton 1964) $d \leq n - k + 1$ implies that $d = n - k + 1$. A code that satisfies this equality is known as a *maximum distance separable* (MDS) code (Singleton 1964). Therefore, RS codes are MDS codes. This gives RS codes useful properties. Among them, shortened RS codes are also MDS codes.

Using the isomorphism between $GF(2^m)$ and $\{0, 1\}^m$, for every m-bit vector $\bar{x}_\beta$ there is a corresponding element $\beta \in GF(2^m)$,

$$\bar{x}_{\beta_j} \in \{0, 1\}^m \iff \beta_j \in GF(2^m), \quad 0 \leq j < 2^m - 1.$$

In other words, m information bits can be grouped to form symbols in $GF(2^m)$. Conversely, if the elements of $GF(2^m)$ are expressed as vectors of m bits, then a binary linear code of length and dimension $n = m(2^m - 1)$ and $k = m(2^m - 1 - 2t_d)$, respectively, is obtained. The minimum distance of this code is at least $2t_d + 1$.

This *binary image* code can correct, in addition to up to t_d random errors, many *random bursts of errors*. For example, any single burst of up to $m(t_d - 1) + 1$ bits can be corrected. This follows from the fact that a burst of errors of length up to $m(q - 1) + 1$ bits is contained in at most q symbols of $GF(2^m)$. Therefore, there are many combinations of random errors and bursts of errors that an RS code can correct. To a great extent, this is the reason why RS codes are so popular in practical systems.

Example 4.2.1 *Let $m = 3$, and $GF(2^3)$ be generated by a primitive element α with*

$$p(\alpha) = \alpha^3 + \alpha + 1 = 0.$$

Let $b = 0$ and $t_d = 2$. Then there is an RS(7, 3, 5) code C with generator polynomial

$$\begin{aligned} \bar{g}(x) &= (x + 1)(x + \alpha)(x + \alpha^2)(x + \alpha^3) \\ &= x^4 + \alpha^2 x^3 + \alpha^5 x^2 + \alpha^5 x + \alpha^6. \end{aligned}$$

By mapping the symbols of $GF(2^3)$ into binary vectors of length 3, code C becomes a binary $(21, 9, 5)$ code that is capable of correcting up to 2 random errors, as well as any single burst of up to 4 bits.

4.3 Decoding RS codes

The core of the decoding algorithms of RS codes is similar to that of binary BCH codes. The only difference is that the *error values*, e_{j_ℓ}, $1 \leq \ell \leq \nu$, for $\nu \leq t_d$, have to be computed.

In general, this is done using the *Forney algorithm* (Forney 1974). The expression below holds for RS codes with an arbitrary set of $2t_d$ consecutive zeros $\{\alpha^b, \alpha^{b+1}, \ldots \alpha^{b+2t_d-1}\}$,

$$e_{j_\ell} = \frac{(\alpha^{j_\ell})^{2-b}\Lambda(\alpha^{-j_\ell})}{\sigma'(\alpha^{-j_\ell})}, \tag{4.4}$$

where $\sigma'(x)$ represents the formal derivative of $\sigma(x)$ with respect to x. (A similar expression can be found in Reed and Chen (1999), p. 276.) The polynomial $\Lambda(x)$ in (4.4) is an *error evaluator polynomial*, which is defined as

$$\Lambda(x) = \sigma(x)S(x) \bmod x^{2t_d+1}. \tag{4.5}$$

Before introducing the first example of RS decoding, an alternative version of the Berlekamp–Massey algorithm (BMA) is presented, referred to as the *Massey algorithm* (or MA). The algorithm was invented by Massey (1969), and is also described in Michelson and Levesque (1985), Wicker (1995).

Massey algorithm to synthesize a linear feedback shift-register (LFSR)

1. Initialize the algorithm with $\sigma(x) = 1$ (the LFSR connection polynomial), $\rho(x) = x$ (the correction term), $i = 1$ (syndrome sequence counter), $\ell = 0$ (register length).

2. Get a new syndrome and compute discrepancy:

$$d = S_i + \sum_{j=1}^{\ell} \sigma_j S_{i-j}$$

3. Test discrepancy: $d = 0$? Yes: Go to 8.

4. Modify connection polynomial:

$$\sigma_{\text{new}}(x) = \sigma(x) - d\rho(x)$$

5. Test register length: $2\ell \geq i$? Yes: Go to 7.

6. Change register length and update correction term: Let $\ell = i - \ell$ and $\rho(x) = \sigma(x)/d$

7. Update connection polynomial: $\sigma(x) = \sigma_{\text{new}}(x)$.

8. Update correction term: $\rho(x) = x\rho(x)$.

9. Update syndrome sequence counter: $i = i + 1$.

10. Stopping condition: If $i < d$ go to 2. Else, stop.

Example 4.3.1 *Let C be the same RS*$(7, 3, 5)$ *code as in Example 4.2.1. Suppose that*

$$\bar{r}(x) = \alpha x^2 + \alpha^5 x^4$$

is the received polynomial. Then $S_1 = \bar{r}(1) = \alpha + \alpha^5 = \alpha^6$, $S_2 = \bar{r}(\alpha) = \alpha^3 + \alpha^2 = \alpha^5$, $S_3 = \bar{r}(\alpha^2) = \alpha^5 + \alpha^6 = \alpha$ *and* $S_4 = \bar{r}(\alpha^3) = 1 + \alpha^3 = \alpha$. *Equation (3.16) gives:*

$$\begin{pmatrix} \alpha^6 & \alpha^5 \\ \alpha^5 & \alpha \end{pmatrix} \begin{pmatrix} \sigma_2 \\ \sigma_1 \end{pmatrix} = \begin{pmatrix} \alpha \\ \alpha \end{pmatrix}.$$

Three methods of finding $\sigma(x)$ *are shown below.*

Direct solution *(Peterson–Gorenstein–Zierler (PGZ) algorithm)*

Assume two errors. Then $\Delta_2 = \alpha^7 + \alpha^{10} = 1 + \alpha^3 = \alpha \neq 0$. *Therefore, two errors must have occurred and*

$$\sigma_2 = \alpha^6 \det \begin{pmatrix} \alpha & \alpha^5 \\ \alpha & \alpha \end{pmatrix} = \alpha^6,$$
$$\sigma_1 = \alpha^6 \det \begin{pmatrix} \alpha^6 & \alpha \\ \alpha^5 & \alpha \end{pmatrix} = \alpha,$$

from which it follows that

$$\sigma(x) = 1 + \alpha x + \alpha^6 x^2 = (1 + \alpha^2 x)(1 + \alpha^4 x).$$

Massey algorithm

$$S_1 = \alpha^6,\ S_2 = \alpha^5,\ S_3 = \alpha,\ S_4 = \alpha.$$

- $i = 0$: $\sigma(x) = 1$, $\ell = 0$, $\rho(x) = x$.
- $i = 1$: $d = S_1 = \alpha^6$,

$$\begin{aligned} \sigma_{\text{new}}(x) &= \sigma(x) + d\rho(x) = 1 + \alpha^6 x, \\ 2\ell = 0 < i,\ \ell &= i - \ell = 1, \\ \rho(x) &= \sigma(x)/d = \alpha^{-6} = \alpha, \\ \rho(x) &= x\rho(x) = \alpha x, \qquad \sigma(x) = \sigma_{\text{new}}(x). \end{aligned}$$

- $i = 2$: $d = S_2 + \sum_{j=1}^{1} \sigma_j S_{2-j} = \alpha^5 + \alpha^6\alpha^6 = 0$.

$$\rho(x) = x\rho(x) = \alpha x^2.$$

- $i = 3$: $d = S_3 + \sum_{j=1}^{1} \sigma_j S_{3-j} = \alpha + \alpha^6\alpha^5 = \alpha^2$.

$$\begin{aligned} \sigma_{\text{new}}(x) &= \sigma(x) + d\rho(x) = 1 + \alpha^6 x + \alpha^3 x^2, \\ 2\ell = 2 < i,\ \ell &= i - \ell = 2, \\ \rho(x) &= \sigma(x)/d = \alpha^5 + \alpha^4 x, \\ \rho(x) &= x\rho(x) = \alpha^5 x + \alpha^4 x^2, \qquad \sigma(x) = \sigma_{\text{new}}(x). \end{aligned}$$

- $i = 4$: $d = S_4 + \sum_{j=1}^{2} \sigma_j S_{4-j} = \alpha + \alpha^6\alpha + \alpha^3\alpha^5 = 1$.

$$\begin{aligned} \sigma_{\text{new}}(x) &= \sigma(x) + d\rho(x) = 1 + \alpha^6 x + \alpha^3 x^2 + (1)(\alpha^5 x + \alpha^4 x^2) \\ &= 1 + \alpha x + \alpha^6 x^2, \\ 2\ell &\geq 4 \\ \rho(x) &= x\rho(x) = \alpha^5 x^2 + \alpha^4 x^3, \qquad \sigma(x) = \sigma_{\text{new}}(x). \end{aligned}$$

- $i = 5 > d$. *Stop.*

Euclidean algorithm

- *Initial conditions:*

$$\begin{aligned} r_0(x) &= x^5, \\ r_1(x) &= S(x) = 1 + \alpha^6 x + \alpha^5 x^2 + \alpha x^3 + \alpha x^4, \\ b_0(x) &= 0, \\ b_1(x) &= 1. \end{aligned}$$

- $j = 2$*:*

$$\begin{aligned} x^5 &= (1 + \alpha^6 x + \alpha^5 x^2 + \alpha x^3 + \alpha x^4)(\alpha^6 x + \alpha^6) + \alpha^5 x^3 + x^2 + \alpha x + \alpha^6. \\ r_2(x) &= \alpha^5 x^3 + x^2 + \alpha x + \alpha^6, \\ q_2(x) &= \alpha^6 x + \alpha^6, \\ b_2(x) &= 0 + (\alpha^6 x + \alpha^6)(1) = \sigma^6 x + \sigma^6. \end{aligned}$$

- $j = 3$*:*

$$\begin{aligned} 1 + \alpha^6 x + \alpha^5 x^2 + \alpha x^3 + \alpha x^4 &= (\alpha^5 x^3 + x^2 + \alpha x + \alpha^6)(\alpha^3 x + \alpha^2) \\ &\quad + \alpha^6 x^2 + \alpha x + \alpha^3. \\ r_3(x) &= \alpha^6 x^2 + \alpha x + \alpha^3, \\ q_3(x) &= \alpha^3 x + \alpha^2, \\ b_3(x) &= 1 + (\alpha^3 x + \alpha^2)(\alpha^6 x + \alpha^6) = \alpha^3 + \alpha^4 x + \alpha^2 x^2. \end{aligned}$$

Algorithm stops, as $\deg[r_3(x)] = 2 = t_d$.

It follows that $\sigma(x) = \alpha^3 + \alpha^4 x + \alpha^2 x^2 = \alpha^3(1 + \alpha x + \alpha^6 x^2)$.

In all the above algorithms, the following error locator polynomial is found, up to a constant term:

$$\sigma(x) = 1 + \alpha x + \alpha^6 x^2 = (1 + \alpha^2 x)(1 + \alpha^4 x).$$

Therefore, the error positions are $j_1 = 2$ *and* $j_2 = 4$. *In a computer program or a hardware implementation, Chien search yields these two values as the (inverse) roots of* $\sigma(x)$. *Also, note that* $\sigma'(x) = \alpha$. *(Because, in* $GF(2^m)$, $2a = a + a = 0$.*)*

To compute the error values, using either the Berlekamp–Massey algorithm (BMA or MA versions) or the PGZ algorithm, the error evaluator polynomial (4.5) is needed,

$$\begin{aligned} \Lambda &= (1 + \alpha x + \alpha^6 x^2)(1 + \alpha^6 x + \alpha^5 x^2 + \alpha x^3 + \alpha x^4) \bmod x^5 \\ &= (1 + \alpha^5 x + \alpha^3 x^2) \bmod x^5, \end{aligned}$$

It is important to note that the Euclidean algorithm (EA) computes simultaneously $\sigma(x)$ *and* $\Lambda(x)$, *as* $\sigma(x) = b_{j_{\text{last}}}(x)$ *and* $\Lambda(x) = r_{j_{\text{last}}}(x)$. *To verify this note that*

$$r_3(x) = \alpha^3 + \alpha x + \alpha^6 x^2 = \alpha^3(1 + \alpha^5 x + \alpha^3 x^2) = \alpha^3 \Lambda(x).$$

With the error locations determined, the error values from Equation (4.4) are

$$\begin{aligned} e_2 &= (\alpha^2)^2(1 + \alpha^5 \alpha^{-2} + \alpha^3 \alpha^{-4})\alpha^{-1} = \alpha, \\ e_4 &= (\alpha^4)^2(1 + \alpha^5 \alpha^{-4} + \alpha^3 \alpha^{-8})\alpha^{-1} = \alpha^5. \end{aligned}$$

Therefore, $\bar{e}(x) = \alpha x^2 + \alpha^5 x^4$ *and the decoded word is*

$$\hat{c}(x) = \bar{r}(x) + \bar{e}(x) = 0.$$

The two errors have been corrected.

Note that the constant β is the same for both polynomials found by application of the extended EA. The EA finds $\beta \cdot \sigma(x)$ and $\beta \cdot \Lambda(x)$, for some nonzero constant $\beta \in GF(2^m)$. Nevertheless, both error locator and error evaluator polynomials have *the same roots* as those obtained by the PGZ or BMA algorithms, and thus the error values obtained are the same.

In most of the computer programs to simulate encoding and decoding procedures of RS codes on the ECC web site, the following equivalent method of finding the error values is used (Lin and Costello 2005). Let

$$z(x) = 1 + (S_1 + \sigma_1)x + (S_2 + \sigma_1 S_1 + \sigma_2)x^2 + \cdots + (s_\nu + \sigma_1 S_{\nu-1} + \cdots + \sigma_\nu)x^\nu. \quad (4.6)$$

Then the error value is computed as (Berlekamp 1984)

$$e_{j_\ell} = \frac{(\alpha^{j_\ell})^{1-b} z(\alpha^{j_\ell})}{\prod_{\substack{i=1 \\ i \neq \ell}}^{\nu} (1 + \alpha^{j_i - j_\ell})}, \quad (4.7)$$

where $1 \leq \ell \leq \nu$.

Yet another alternative to Forney algorithm, for small values of t_d, is to determine the error values directly as follows. For $1 \leq \ell \leq \nu$, the error values e_{j_ℓ} are related to the syndromes S_i by a set of linear equations. Let $\beta_\ell = \alpha^{j_\ell}$ denote the *error enumerator*, $1 \leq \ell \leq \nu$. Then

$$S_i = \bar{e}(\alpha^{b+i-1}) = \sum_{\ell=1}^{\nu} e_{j_\ell} \alpha^{(b+i-1)j_\ell} = \sum_{\ell=1}^{\nu} e_{j_\ell} \beta_\ell^{(b+i-1)}. \quad (4.8)$$

where $1 \leq i \leq 2t_d$.

Each $\nu \times \nu$ submatrix formed by the already known terms $\beta_\ell^{(b+i-1)}$ is a Vandermonde matrix. After all the ν error locations j_ℓ are known, any set of ν equations of the form (4.8) can be used to find the error values. In particular, choosing the first ν syndromes:

$$\begin{pmatrix} S_1 \\ S_2 \\ \vdots \\ S_\nu \end{pmatrix} = \begin{pmatrix} \beta_1^b & \beta_2^b & \cdots & \beta_\nu^b \\ \beta_1^{b+1} & \beta_2^{b+1} & \cdots & \beta_\nu^{b+1} \\ \vdots & \vdots & \ddots & \vdots \\ \beta_1^{b+\nu-1} & \beta_2^{b+\nu-1} & \cdots & \beta_\nu^{b+\nu-1} \end{pmatrix} \begin{pmatrix} e_{j_1} \\ e_{j_2} \\ \vdots \\ e_{j_\nu} \end{pmatrix}, \quad (4.9)$$

is a system of linear equations that can be solved using $GF(2^m)$ arithmetic.

Example 4.3.2 *Consider the same RS code and received polynomial in Example 4.3.1. Then (4.9) gives:*

$$\begin{pmatrix} \alpha^6 \\ \alpha^5 \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ \alpha^2 & \alpha^4 \end{pmatrix} \begin{pmatrix} e_2 \\ e_4 \end{pmatrix}.$$

The determinant of the 2×2 *matrix is* $\Delta = \alpha^4 + \alpha^2 = \alpha$. *From this it follows that*

$$e_2 = \alpha^{-1} \det \begin{pmatrix} \alpha^6 & 1 \\ \alpha^5 & \alpha^4 \end{pmatrix} = \alpha^6(\alpha^3 + \alpha^5) = \alpha^6 \cdot \alpha^2 = \alpha,$$

and

$$e_4 = \alpha^{-1} \det \begin{pmatrix} 1 & \alpha^6 \\ \alpha^2 & \alpha^5 \end{pmatrix} = \alpha^6(\alpha^5 + \alpha) = \alpha^6 \cdot \alpha^6 = \alpha^5,$$

which are the same error values as those obtained with the Forney algorithm. Again, it is emphasized that this can only be done efficiently (and practically) for relatively small values of the error correcting capability, t_d, *of the RS code.*

4.3.1 Remarks on decoding algorithms

Unlike the BMA, all the syndromes in the EA are used in the first computation step. However, in terms of the number of $GF(2^m)$ operations, the BMA is generally more efficient than the EA. On the other hand, all the steps in the EA are identical, which translates into a more efficient hardware implementation. Also, the three decoding methods discussed here for (binary and nonbinary) BCH codes are examples of incomplete – or bounded distance – decoding. That is, they are able to detect situations in which the number of errors exceeds the capability of the code.

There are other approaches to decoding BCH codes, the most notable being the use of a discrete Fourier transform over $GF(2^m)$. This is covered extensively in Blahut (1984), where the reader is referred to for details. Sudan (1997) introduced an algorithm that allows the correction of errors beyond the minimum distance of the code. It applies to RS codes and more generally to AG codes. This algorithm produces a list of code words (it is a list-decoding algorithm) and is based on interpolation and factorization of polynomials over $GF(2^m)$ and its extensions. Sudan algorithm was improved in Guruswami and Sudan (1999).

4.3.2 Errors-and-erasures decoding

For the correction of erasures, the main change to the RS decoding procedures described above is that an *erasure locator polynomial* $\tau(x)$ needs to be introduced, defined as

$$\tau(x) = \prod_{\ell=1}^{\mu} (1 + y_{i_\ell} x),$$

where $y_{i_\ell} = \alpha^{i_\ell}$, for $1 \leq \ell \leq \mu$, denotes the position of an erasure.

By definition, the positions of the erasures are known. Therefore, *only the erasure values* need to be found. This can be done, as before, in the Forney algorithm step. In computing

the syndromes of the received polynomial, it can be shown that any values of the erasures can be replaced, without any difference in the decoded word.

The decoding procedure is similar to the errors-only RS decoder, with the following exceptions. A modified syndrome polynomial, or *modified Forney syndrome*, is formed,

$$T(x) = S(x)\tau(x) + 1 \bmod x^{2t_d+1}. \tag{4.10}$$

The BMA algorithm can be applied to find $\sigma(x)$ with the following modifications:

1. The discrepancy is now defined as

$$d_i = T_{i+\mu+1} + \sum_{j=1}^{\ell_i} \sigma_j^{(i)} T_{i+\mu+1-j}, \tag{4.11}$$

with $d_0 = T_{\mu+1}$.

2. The algorithm finishes when the following stopping condition is met:

$$i \geq l_{i+1} + t_d - 1 - \mu/2.$$

After $\sigma(x)$ is obtained, a *modified errors-and-erasure evaluator*, or errata evaluator, is computed as $\omega(x)$,

$$\omega(x) = [1 + T(x)]\,\sigma(x) \bmod x^{2t_d+1}. \tag{4.12}$$

In addition, the following *errata locator polynomial* is computed,

$$\phi(x) = \tau(x)\sigma(x). \tag{4.13}$$

The resulting errata evaluation, or *modified Forney algorithm*, is given by

$$e_{j_\ell} = \frac{(\alpha^{j_\ell})^{2-b}\omega(\alpha^{-j_\ell})}{\phi'(\alpha^{-j_\ell})}, \tag{4.14}$$

$1 \leq \ell \leq \nu$, for the error values, and

$$f_{i_\ell} = \frac{(y_{i_\ell})^{2-b}\omega(y_{i_\ell}^{-1})}{\phi'(y_{i_\ell}^{-1})}, \tag{4.15}$$

$1 \leq \ell \leq \mu$, for the erasure values.

For errors-and-erasures decoding, the EA can also be applied to the modified syndrome polynomial $T(x)$, using $1 + T(x)$ instead of $S(x)$ as in errors-only decoding. That is, the initial conditions are $r_0(x) = x^{2t_d+1}$ and $r_1(x) = 1 + T(x)$. The algorithm stops when $\deg[r_j(x)] \leq \lfloor (d - 1 + \mu)/2 \rfloor$, with $\omega(x) = r_j(x)$ and $\sigma(x) = b_j(x)$.

Example 4.3.3 *Let C be an RS $(15, 9, 7)$ code over $GF(2^4)$ with zeros $\{\alpha, \alpha^2, \ldots, \alpha^6\}$, where α is a primitive element satisfying $p(\alpha) = \alpha^4 + \alpha^3 + 1 = 0$. As a reference, a table of elements of $GF(2^4)$ as powers of a primitive element α, with $\alpha^4 + \alpha^3 + 1 = 0$, is shown below.*

Table of elements of $GF(2^4)$, $p(x) = x^4 + x^3 + 1$.

Power	*Vector*
0	*0000*
1	*0001*
α	*0010*
α^2	*0100*
α^3	*1000*
α^4	*1001*
α^5	*1011*
α^6	*1111*
α^7	*0111*
α^8	*1110*
α^9	*0101*
α^{10}	*1010*
α^{11}	*1101*
α^{12}	*0011*
α^{13}	*0110*
α^{14}	*1100*

The generator polynomial of C is

$$\bar{g}(x) = \prod_{i=1}^{6}(x + \alpha^i) = x^6 + \alpha^{12}x^5 + x^4 + \alpha^2 x^3 + \alpha^7 x^2 + \alpha^{11}x + \alpha^6.$$

Suppose that the polynomial associated with a code word $\bar{v}$ *is*

$$\bar{v}(x) = \alpha^5 + \alpha^3 x + \alpha^{13}x^2 + \alpha x^3 + \alpha^7 x^4 + \alpha^4 x^5 + \alpha x^6 + \alpha^4 x^7 + \alpha^6 x^8 + \alpha^3 x^{10} + \alpha^5 x^{11} + \alpha^6 x^{12} + \alpha^{13}x^{13} + \alpha^{10}x^{14}.$$

Let the received polynomial be

$$\bar{r}(x) = \alpha^7 + \alpha^3 x + \alpha^{13}x^2 + \alpha^{14}x^3 + \alpha^7 x^4 + \alpha x^5 + \alpha x^6 + \alpha^4 x^7 + \alpha^6 x^8 + \alpha^3 x^{10} + \alpha^5 x^{11} + \alpha^{11}x^{12} + \alpha^{13}x^{13} + \alpha^{10}x^{14}.$$

Assume that, aided by side information from the receiver, it is determined that the values in positions α^0 *and* α^5 *are unreliable, and thus declared as erasures.*

Note that $\bar{e}(x) = \alpha^{14} + \alpha^8 x^3 + \alpha^5 x^5 + \alpha x^{12}$ *is the polynomial associated with the errata.*[1]

After reception, besides $\bar{r}(x)$*, the decoder knows that* $\mu = 2$ *erasures have occurred in positions* α^0 *and* α^5*. Therefore, it computes the erasure locator polynomial*

$$\tau(x) = (1 + x)(1 + \alpha^5 x) = 1 + \alpha^{10}x + \alpha^5 x^2. \tag{4.16}$$

[1]The decoder obviously does not know this, except for the positions of the erasures. This polynomial is used as a reference against which the correctness of the decoding results can be verified.

The syndromes of $\bar{r}(x)$ are computed as:

$$S_1 = \bar{r}(\alpha) = \alpha^{11}, \quad S_2 = \bar{r}(\alpha^2) = \alpha^5, \quad S_3 = \bar{r}(\alpha^3) = \alpha^2,$$
$$S_4 = \bar{r}(\alpha^4) = \alpha^2, \quad S_5 = \bar{r}(\alpha^5) = 1 \quad \text{and} \quad S_6 = \bar{r}(\alpha^6) = \alpha^{14}.$$

Accordingly,

$$S(x) = 1 + \alpha^{11}x + \alpha^5 x^2 + \alpha^2 x^3 + \alpha^2 x^4 + x^5 + \alpha^{14} x^6. \tag{4.17}$$

The Forney syndrome polynomial (4.10) becomes

$$\begin{aligned} T(x) &= S(x)\tau(x) + 1 \bmod x^{2t_d+1} \\ &= (1 + \alpha^{11}x + \alpha^5 x^2 + \alpha^2 x^3 + \alpha^2 x^4)(1 + \alpha^{10}x + \alpha^5 x^2) + 1 \bmod x^7 \\ &= \alpha^7 x + \alpha^6 x^2 + \alpha^7 x^3 + \alpha^{11} x^4 + \alpha^9 x^5 + x^6. \end{aligned}$$

BMA for errors-and-erasures correction

- **Iteration 0:** *Initialize.* $\sigma^{(-1)}(x) = 1$, $\ell_{-1} = 0$, $d_{-1} = 1$, $\sigma^{(0)}(x) = 1$, $\ell_0 = 0$, *and* $d_0 = T_3 = \alpha^7$.

- **Iteration 1:** $i = 0$, $d_0 = \alpha^7 \neq 0$. $m = -1$ *maximizes* $(-1 + 0) = -1$ *for* $d_{-1} \neq 0$.

$$\begin{aligned} \sigma^{(1)}(x) &= \sigma^{(0)}(x) + d_0 d_{-1}^{-1} x^{(0-(-1))} \sigma^{(-1)}(x) = 1 + \alpha^7 x, \\ \ell_1 &= \max\{\ell_0, \ell_{-1} + 0 - (-1)\} = 1, \\ i &\geq \ell_1 + 1 \ \text{? No :} \\ d_1 &= T_4 + T_3 \sigma_1^{(1)} = \alpha^{11} + \alpha^7 \alpha^7 = 1. \end{aligned}$$

- **Iteration 2:** $i = 1$, $m = 0$ *maximizes* $(0 - 0) = 0$ *for* $d_0 \neq 0$.

$$\begin{aligned} \sigma^{(2)}(x) &= \sigma^{(1)}(x) + d_1 d_0^{-1} x^{(1-0)} \sigma^{(0)}(x) = 1 + \alpha^4 x. \\ \ell_2 &= \max\{1, 0 + 1 - 0\} = 1, \\ i &\geq 2 \ \text{? No:} \\ d_2 &= T_5 + T_4 \sigma_1^{(2)} = \alpha^9 + \alpha^{11} \alpha^4 = \alpha^2. \end{aligned}$$

- **Iteration 3:** $i = 2$, $m = 0$ *maximizes* $(0 - 0) = 0$ *for* $d_0 \neq 0$.

$$\begin{aligned} \sigma^{(3)}(x) &= \sigma^{(2)}(x) + d_2 d_0^{-1} x^{(2-0)} \sigma^{(0)}(x) = 1 + \alpha^4 x + \alpha^{10} x^2. \\ \ell_3 &= \max\{1, 0 + 2 - 0\} = 2, \\ i &\geq 3 \ \text{? No:} \\ d_3 &= T_6 + T_5 \sigma_1^{(3)} + T_4 \sigma_2^{(3)} = 1 + \alpha^9 \alpha^4 + \alpha^{11} \alpha^{10} = \alpha^3. \end{aligned}$$

- **Iteration 4:** $i = 3$, $m = 2$ *maximizes* $(2 - 1) = 1$ *for* $d_2 \neq 0$.

$$\begin{aligned} \sigma^{(4)}(x) &= \sigma^{(3)}(x) + d_3 d_2^{-1} x^{(3-2)} \sigma^{(2)}(x) = 1 + \alpha^5 x + x^2. \\ \ell_4 &= \max\{2, 1 + 3 - 2\} = 2, \\ i &\geq 2 \ \text{Yes : Algorithm ends.} \end{aligned}$$

Therefore, $\sigma(x) = 1 + \alpha^5 x + x^2 = (1 + \alpha^3 x)(1 + \alpha^{12} x)$. *For the last equality, recall that the inverses of the error positions are found, via Chien search, by evaluating* $\sigma(x)$ *at all the*

nonzero elements of GF(2^4)*. This gives* $\sigma(\alpha^{12}) = 0$ *and* $\sigma(\alpha^3) = 0$*. As a result, the error positions are* $\alpha^{-12} = \alpha^3$ *and* $\alpha^{-3} = \alpha^{12}$.

From Equation (4.12), the modified errors-and-erasures evaluator is

$$\begin{aligned}\omega(x) &= [1 + T(x)]\,\sigma(x) \bmod x^{2t_d+1} \\ &= (1 + \alpha^7 x + \alpha^6 x^2 + \alpha^7 x^3 + \alpha^{11} x^4 + \alpha^9 x^5 + x^6)(1 + \alpha^5 x + x^2) \bmod x^7 \\ &= 1 + \alpha^{14} x + \alpha^{11} x^2 + \alpha^{11} x^3 + x^4, \end{aligned} \tag{4.18}$$

and the errata locator polynomial

$$\phi(x) = \tau(x)\sigma(x) = (1 + \alpha^{10} x + \alpha^5 x^2)(1 + \alpha^5 x + x^2) = 1 + x + \alpha^5 x^2 + \alpha^5 x^4,$$

from which it follows that $\phi'(x) = 1$.

The values of the errors and erasures are found from (4.14) and (4.15), respectively,

$$\begin{aligned} e_3 &= \alpha^3(1 + \alpha^{11} + \alpha^5 + \alpha^2 + \alpha^3) = \alpha^3\alpha^5 = \alpha^8, \\ e_{12} &= \alpha^{12}(1 + \alpha^2 + \alpha^2 + \alpha^5 + \alpha^{12}) = \alpha^{12}\alpha^4 = \alpha, \\ f_0 &= (1 + \alpha^{14} + \alpha^{11} + \alpha^{11} + 1) = \alpha^{14}, \\ f_5 &= \alpha^5(1 + \alpha^9 + \alpha + \alpha^{11} + \alpha^{10}) = \alpha^5, \end{aligned}$$

from which it follows that the errata polynomial *is*

$$\bar{e}(x) = \alpha^{14} + \alpha^8 x^3 + \alpha^5 x^5 + \alpha x^{12}.$$

The decoded polynomial $\hat{v}(x) = \bar{r}(x) + \bar{e}(x)$ *is identical to* $\bar{v}(x)$*. Two errors and two erasures have been corrected.*

Direct solution of errata values

For small values of the minimum distance of an RS code, the erasure values may be obtained by solving a set of linear equations. Let $\bar{e}(x)$ be the error polynomial associated with an error pattern resulting from the presence of ν errors and μ erasures,

$$\bar{e}(x) = \sum_{\ell=1}^{\nu} e_{j_\ell} x^{j_\ell} + \sum_{\ell'=0}^{\mu} f_{j'_\ell} x^{j'_\ell}. \tag{4.19}$$

Then, the following set of linear equations, similar to (4.8), hold between the syndromes and the values of the errors and positions:

$$S_i = \bar{e}(\alpha^{b+i}) = \sum_{\ell=1}^{\nu} e_{j_\ell} \alpha^{(b+i)j_\ell} + \sum_{\ell'=1}^{\mu} f_{j'_\ell} \alpha^{(b+i)j'_\ell}, \tag{4.20}$$

where $1 \le i \le 2t_d$. As before, any set of $\nu + \mu \le t_d$ equations can be used to solve the values of the errors and erasures.

Example 4.3.4 *Direct solution of the errata values for the code in the previous example: After the BMA and Chien search, the decoder knows that the error polynomial is of the form*

$$\bar{e}(x) = f_0 + e_3 x^3 + f_5 x^5 + e_{12} x^{12}.$$

The errata values can be found by solving the set of linear equations given by (4.20), which can be put in the following matrix form,

$$\begin{pmatrix} 1 & \alpha^3 & \alpha^5 & \alpha^{12} \\ 1 & \alpha^6 & \alpha^{10} & \alpha^9 \\ 1 & \alpha^9 & 1 & \alpha^6 \\ 1 & \alpha^{12} & \alpha^5 & \alpha^5 \end{pmatrix} \begin{pmatrix} f_0 \\ e_3 \\ f_5 \\ e_{12} \end{pmatrix} = \begin{pmatrix} \alpha^{11} \\ \alpha^5 \\ \alpha^2 \\ \alpha^2 \end{pmatrix}. \tag{4.21}$$

It can be verified that $f_0 = \alpha^{14}$, $e_3 = \alpha^8$, $f_5 = \alpha^5$ and $e_{12} = \alpha$ are the solutions to (4.21). These are the same errata values as those computed before with the modified Forney algorithm.

4.4 Weight distribution

As mentioned before, RS codes are also MDS codes. The weight distribution of MDS codes can be computed exactly with a closed form expression (Lin and Costello 2005; MacWilliams and Sloane 1977):

The number of code words of weight i in an (n, k, d) MDS code over $GF(q)$ is

$$A_i = \binom{n}{i}(q-1)\sum_{j=0}^{i-d}(-1)^j\binom{i-1}{j}q^{i-j-d}. \tag{4.22}$$

For error performance evaluation of an RS (n, k, d) code over $GF(2^m)$, the following upper bound on the probability of a bit error $P_b(C)$ for an RS decoder[2] is simple to compute and relatively good,

$$P_b(C) \leq \frac{2^{m-1}}{2^m - 1}\sum_{i=t_d+1}^{n}\frac{i + t_d}{n}\binom{n}{i}P_s^i(1-P_s)^{n-i}, \tag{4.23}$$

where P_s denotes the probability of a *symbol error* at the input of the RS decoder,

$$P_S = 1 - (1-p)^m,$$

and p denotes the probability of a *bit error* at the input of the RS decoder. The probability of a word error can be upper bounded by (1.33),

$$P_e(C) < 1 - \sum_{i=0}^{t_d}\binom{n}{i}P_s^i(1-P_s)^{n-i}. \tag{4.24}$$

[2]It should be noted that the bound is tight only for *bounded distance* decoders, such as those based on BMA, EA or PGZ.

Problems

1. Let C be a Reed–Solomon $(7, 3, 5)$ code with zeros $\alpha^2, \alpha^3, \alpha^4, \alpha^5$, where α is a primitive element of $GF(2^3)$ satisfying $p(\alpha) = \alpha^3 + \alpha^2 + 1 = 0$.

   (a) Obtain the generator polynomial of C.

   (b) Suppose that $\bar{r}(x) = 1 + \alpha^3 x^6$ is the received polynomial. Without doing any computation, what is the closest code polynomial $\hat{v}(x)$ to $\bar{r}(x)$? (Hint: The minimum distance is 5.)

   (c) With the same received polynomial as in part (b), use the direct-solution method (PGZ decoder) to find the estimated error vector $\hat{e}(x)$ and the associated estimated code word $\hat{c}(x)$.

2. Let C be a RS $(7, 2, 6)$ code over $GF(2^3)$.

   (a) What are the parameters of the equivalent binary linear code C_2 obtained from C by expressing each element in $GF(2^3)$ as a binary vector of length 3?

   (b) Determine a generator polynomial of C.

   (c) Suppose that the received polynomial is

   $$\bar{r}(x) = \alpha^4 + \alpha^2 x + x^2 + \alpha^5 x^4,$$

   and is known to contain three erasures in the last three positions. Determine the most likely code polynomial.

3. Consider a RS $(15, 9, 7)$ code over $GF(2^4)$ with $b = 1$.

   (a) Obtain the generator polynomial of C.

   (b) Suppose that the received polynomial is

   $$\bar{r}(x) = \alpha^7 x^3 + \alpha^3 x^6 + \alpha^4 x^{12}.$$

   Determine the most likely code polynomial $\hat{v}(x)$ using the BMA.

   (c) With the same received polynomial as in part (b), and without any computations, what is the closest code polynomial $\hat{v}(x)$ to $\bar{r}(x)$?

4. Let C be an RS $(7, 3, 5)$ code with $b = 1$ and generator polynomial

   $$\bar{g}(x) = \alpha^3 + \alpha x + x^2 + \alpha^3 x^3 + x^4.$$

   The received vector is $\bar{r}(x) = \alpha^3 x^2 + x^4 + x^6$. The position α^3 has been erased. Find the most likely code polynomial using the BMA.

5. A RS $(31, 25)$ code with $b = 1$ is used in a system. The received polynomial is

   $$\bar{r}(x) = \alpha^3 x + \alpha^6 x^2 + \alpha^6 x^4 + x^5 + \alpha^{23} x^6.$$

   Using the BMA, determine the most likely code polynomial.

6. A communication system employs the binary image of an RS $(7, 5, 3)$ code over $GF(2^3)$ with binary modulation over an additive white Gaussian noise (AWGN) channel. Estimate the probability of a bit error of this system and compare it with uncoded binary phase shift keying (BPSK) modulation. In particular, estimate the coding gain.

7. Consider the binary image C_b of an RS $(7, 3, 5)$ code over $GF(2^3)$.

   (a) Determine the parameters (n, k, d) of C_b and its burst-error-correcting capability.

   (b) How does C_b compare with the Fire code in example 3.1.7?

8. Discuss the error correcting capabilities of the binary image C_b of an RS $(127, 121, 7)$ code over $GF(2^7)$.

9. Show that the binary image C_b of an RS $(7, 5, 3)$ code over $GF(2^3)$ is equivalent to a *Nonprimitive* binary Hamming code of length 21. (See also problem 21 in Chapter 3.)

10. (MacWilliams and Sloane (1977)). Show that the dual of an RS code is another RS code.

11. Let C denote an RS (n, k, d) code over $GF(2^m)$. A new code is constructed by setting $s < k$ information symbols always equal to zero. Show that the resulting code is equivalent to a *shortened* RS code[3] C_s of dimension $k - s$. Show that C_s is still an MDS code.

[3]Code shortening is discussed in section 6.1.1

5

Binary convolutional codes

First introduced by Elias (1955), binary convolutional codes are one of the most popular forms of binary error correcting codes that have found numerous applications: wireless communications (IMT-2000, GSM, IS-95), digital terrestrial and satellite communication and broadcasting systems and space communication systems, just to cite a few. They were referred to in Elias (1955) as *convolutional parity-check symbols codes*. Their most popular decoding method to date, the *Viterbi algorithm* (Viterbi 1967), also finds applications in combinatorially equivalent problems such as maximum likelihood sequence detection (e.g., equalization) and partial-response signaling (e.g., magnetic recording). Most recently, it has been shown that convolutional codes, when combined with interleaving in a concatenated scheme, can perform very close to the Shannon limit (Berrou *et al.* 1993). In this chapter, the basic properties and decoding procedures of binary convolutional codes are described.

5.1 Basic structure

A convolutional code is an error correcting code that processes information *serially* or, continuously, in short block lengths (Lee 1997). A convolutional encoder has *memory*, in the sense that the output symbols depend not only on the input symbols but also on previous inputs and/or outputs. In other words, the encoder is a *sequential circuit* or a finite-state machine. The *state* of the encoder is defined as the contents of the memory. In the computer programs that implement the Viterbi algorithm and other decoding procedures involving a trellis, found on the ECC web site, a state transition table, indicating the relation between the input, the previous and current state, and the current output, is employed.

A convolutional code consists of the set of all binary sequences produced by a convolutional encoder. In theory, these sequences have infinite duration. In practice, the state of the convolutional code is periodically forced to a known state and code sequences are produced in a block-wise manner.

Example 5.1.1 *Consider the convolutional encoder depicted in Figure 5.1. For analysis and decoding purposes, this encoder is described in Table 5.1.*

The Art of Error Correcting Coding, Second Edition Robert H. Morelos-Zaragoza
 ISBN: 0-470-01558-6

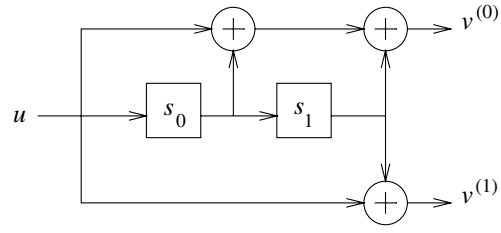


Figure 5.1 An encoder of a memory-2 rate-1/2 convolutional encoder.

Table 5.1 Input bits, state transitions and output bits.

Initial state $s_0[i]s_1[i]$	Information $u[i]$	Final state $s_0[i+1]s_1[i+1]$	Outputs $v^{(0)}[i]v^{(1)}[i]$
00	0	00	00
00	1	10	11
01	0	00	11
01	1	10	00
10	0	01	10
10	1	11	01
11	0	01	01
11	1	11	10

Note that the coding rate *of the convolutional encoder is equal to* $1/2$ *because two output coded bits are produced for every input information bit.*

In general, a rate-k/n convolutional encoder has k shift registers, one per input information bit, and n output coded bits that are given by linear combinations (over the binary field, i.e., with exclusive -OR gates) of the contents of the registers and the input information bits.

For simplicity of exposition, and for practical purposes, only rate-$1/n$ binary convolutional codes are considered in the remainder of the chapter. One reason for doing this is that these are the most widely used binary codes. A technique known as *puncturing* can be applied, which results in convolutional encoders of higher rate. This technique is discussed in Section 5.6.

The total length of the shift registers in the encoder is referred to as the *memory*. For discussion in this chapter, *state labels* are integers I that are associated with the binary representation of the memory contents as $I = \sum_{j=0}^{m-1} s_j[i]2^{m-1-j}$.

The *constraint length*, which equals $K = m + 1$ for rate-$1/2$ encoders, is defined as the number of inputs $(u[i], u[i-1], \ldots, u[i-m])$ that affect the outputs $(v^{(0)}[i], \ldots, v^{(n-1)}[i])$ at time i. For the encoder in Figure 5.1, $K = 3$. An encoder with m memory elements is referred to as a memory-m encoder. A *memory-m rate-$1/n$* convolutional encoder can be represented by a *state diagram*. There are 2^m states in the diagram. Because there is only one information bit, two branches enter and leave each state and are labeled as $u[i]/v^{(0)}[i] \ldots v^{(n-1)}[i]$.

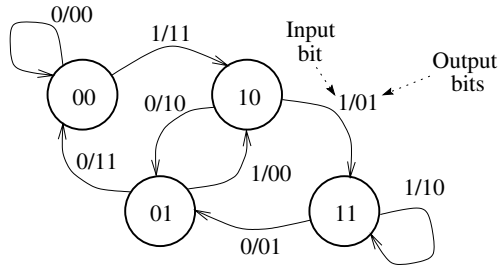


Figure 5.2 State diagram of a memory-2 rate-1/2 convolutional encoder.

Example 5.1.2 *A state diagram of the memory-2 rate-*$1/2$ *convolutional code of Example 5.1.1 is shown in Figure 5.2.*

The state diagram is also useful in determining whether a convolutional encoder is *catastrophic*. A convolutional encoder is said to be catastrophic if a finite number of channel errors produces an infinite number of errors after decoding. This type of encoder can be recognized from its state diagram whenever an all-zero weight cycle is associated with the self-loop around the all-zero state. The reader is referred to Massey and Sain (1967) for details.

The encoder of a memory-m rate-$1/n$ binary convolutional code can also be considered a *discrete linear time-invariant*[1] system. Therefore, the impulse response, that is, the output of the encoder when the input is an "impulse" $\bar{u} = (1000\ldots00\ldots)$, completely specifies the code.

Convention: In writing a sequence, the leftmost symbol is meant to be the first symbol that goes into the channel.

There are n impulse responses of a convolutional encoder of rate $1/n$, one for each output $\bar{v}^{(j)}$, for $j = 0, 1, \ldots, n-1$. As the impulse goes across the memory elements of the encoder, it "picks up" the connections between the memory elements and the outputs. Evidently, this is a finite-impulse-response (FIR) system.

Let $\bar{g}_0, \ldots \bar{g}_{n-1}$ denote the impulse responses of a rate-$1/n$ convolutional encoder. These are also called the *generator sequences* – or *generators* – of the code, on the basis of the observation made in the preceding text, and indicate the actual physical connections of the encoder.

Example 5.1.3 *Continuing with the memory-2 rate-*$1/2$ *convolutional code of Example 5.1.1, the encoder in Figure 5.1 produces* $\bar{g}_0 = (1, 1, 1)$ *and* $\bar{g}_1 = (1, 0, 1)$*, as shown in Figure 5.3.*

Note that the generator sequences are equal to zero after K bits. Therefore, it is sufficient to consider vectors of this length. To stress the dynamic nature of the encoder, the indeterminate D (for "delay") is employed, and the generators are written as polynomials in D, that is, $\bar{g}_0(D), \ldots \bar{g}_{n-1}(D)$.

[1]Note, however, that *time-varying* convolutional codes can be defined and are studied in Johannesson and Zigangirov (1999).

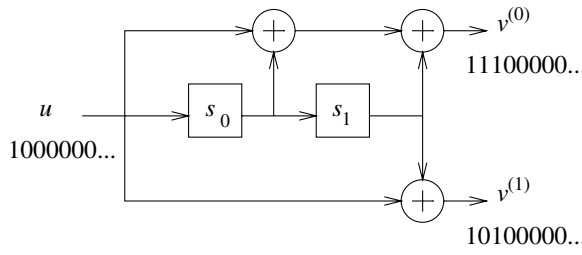


Figure 5.3 Generator sequences of a memory-2 rate-1/2 convolutional encoder.

Example 5.1.4 *For the encoder in Figure 5.1,*

$$\bar{g}_0(D) = 1 + D + D^2 \quad \text{and} \quad \bar{g}_1(D) = 1 + D^2.$$

In general, when referring to a convolutional code, the generators are expressed as

$$(\bar{g}_0, \ldots, \bar{g}_{n-1})$$

and written in octal form.

For the encoder of Example 5.1.4 the generators are $(7, 5)$. The most widely used convolutional encoder to date is a memory-6 rate-1/2 encoder with generators (171,133). This is the so-called NASA standard code, as it was first used in the Voyager space missions. It is also used in many digital communications standards, as well as in the Control, Communications and Display Subsystem for Satellite (CCSD) standard.

As a result of the dynamical structure of a convolutional encoder, the state diagram can be represented, as it evolves in time, with a *trellis diagram*, which is constructed by placing the state diagram of the code at each time interval, with branches connecting the states between time i and $i + 1$, in correspondence with the encoder table. The branches of the trellis are labeled in the same way as the state diagram.

Convention: When there is no dichotomy, the input information bit does not need to appear explicitly in the branch label. For FIR encoders, the information bit u can be inferred directly from the state transition[2]:

$$(s_0 s_1 \ldots s_{m-1}) \rightarrow (u s_0 \ldots s_{m-2}).$$

It should be emphasized that, in the case of a recursive systematic convolutional (RSC) (infinite-impulse-response (IIR)) encoder, this is not generally the case, as is shown in Section 5.1.1.

Example 5.1.5 *For example 5.1.1, the trellis structure is shown in Figure 5.4. Note that state transitions (branches) from a state $s[i]$ to a (possibly different) state $s'[i + 1]$, which are caused by the input bit $u[i] = 0$, are the upper branches.*

[2]This simple observation is the basis for using a *traceback memory* in a Viterbi decoder (VD) to recover the information bits from the decoded sequence.

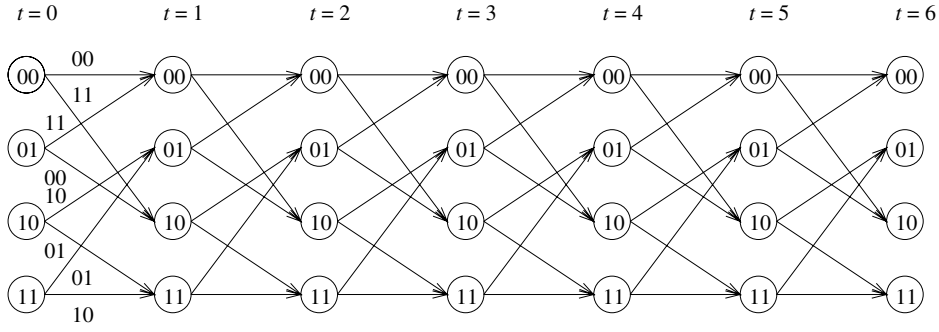


Figure 5.4 Six sections of the trellis of a memory-2 rate-1/2 convolutional encoder.

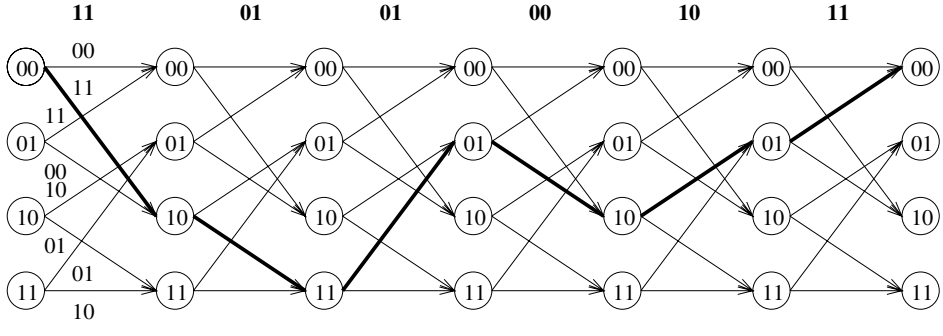


Figure 5.5 A path in the trellis of a memory-2 rate-1/2 convolutional encoder.

Example 5.1.6 *Consider the encoder in Figure 5.1 and the input sequence* $\bar{u} = (110100)$. *Then, the corresponding output sequence can be obtained either directly from the encoder table (Table 5.1) or as a path in the trellis, as shown in Figure 5.5. It follows that the corresponding coded sequence is* $\bar{v} = (11, 01, 01, 00, 10, 11)$.

A convolutional encoder is a linear time-invariant system, with impulse responses given by the code generators $\bar{g}_0(D)$, $\bar{g}_1(D), \ldots, \bar{g}_n(D)$, where

$$\bar{g}_j(D) = g_j[0] + g_j[1]D + g_j[2]D^2 + \cdots + g_j[m]D^m, \tag{5.1}$$

for $0 \leq j < n$. In terms of the generators of a code, the output sequences can be expressed as

$$v^{(j)}[i] = \sum_{\ell=0}^{m} u[i-\ell]g_j[\ell], \tag{5.2}$$

for $0 \leq j < n$. The output sequences $\bar{v}^{(j)}(D)$, $0 \leq j < n$, are as expected, equal to the *discrete convolution* between the input sequence $\bar{u}(D)$ and the code generators $\bar{g}_0(D)$, $\bar{g}_1(D), \ldots, \bar{g}_{n-1}(D)$. From this fact comes the name – "convolutional" – of the encoder (Elias 1955).

Now observe that Equation (5.2) can be written as a matrix multiplication

$$\bar{v} = \bar{u}G, \tag{5.3}$$

where G is a *generator matrix* of the convolutional code. In particular, for a memory-m rate-$1/2$ convolutional code,

$$G = \begin{pmatrix} g_0[0]g_1[0] & \cdots & g_0[m]g_1[m] & & \\ & g_0[0]g_1[0] & \cdots & g_0[m]g_1[m] & \\ & & g_0[0]g_1[0] & \cdots & g_0[m]g_1[m] \\ & & \ddots & & \ddots \end{pmatrix}, \tag{5.4}$$

where the blank entries represent zeros. Generalization to any rate-$1/n$ encoder is straightforward.

Example 5.1.7 *For the memory-2 rate-*$1/2$ *encoder in Figure 5.1,*

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

Let $\bar{u} = (110100)$. *Then it follows from Equation (5.3) that* $\bar{v} = \bar{u}G = (11, 01, 01, 00, 10, 11)$, *which is the same output as in Example 5.1.6.*

Let $\bar{v}(D) = v^{(0)}(D) + Dv^{(1)}(D) + \cdots + D^{n-1}v^{(n-1)}(D)$. The relationship between input and output sequences can be written as

$$\bar{v}(D) = \bar{u}(D)G(D), \tag{5.5}$$

where generators of a rate-$1/n$ convolutional code are arranged in a matrix referred to as a *polynomial generator matrix*,

$$G(D) = \begin{pmatrix} \bar{g}_0(D) & \bar{g}_1(D) & \cdots & \bar{g}_{n-1}(D) \end{pmatrix}. \tag{5.6}$$

5.1.1 Recursive systematic convolutional codes

Using Equation (5.6), a memory-m rate-$1/n$ RSC code has a generator matrix of the form

$$G'(D) = \begin{pmatrix} I & \dfrac{\bar{g}_1(D)}{\bar{g}_0(D)} & \cdots & \dfrac{\bar{g}_1(D)}{\bar{g}_0(D)} \end{pmatrix}. \tag{5.7}$$

As a short notation of this generator matrix, the *generators* $(1, \bar{g}_1/\bar{g}_0, \ldots, \bar{g}_{n-1}/\bar{g}_0)$ can be specified. Division and multiplication of the right-hand side of Equation (5.5) by $\bar{g}_0(D)$ give, together with Equation (5.6),

$$\bar{v}(D) = \bar{u}'(D)G'(D), \tag{5.8}$$

where $\bar{u}'(D) = \bar{g}_0(D)\bar{u}(D)$. This shows that both the nonsystematic encoder and the systematic encoder produce the same coded sequences but that the corresponding information sequences are scrambled by $\bar{g}_0(D)$ in the RSC code.

A systematic encoder is also an example of a discrete linear time-invariant system. Because the generator matrix contains rational functions, the encoder is an *IIR* linear system, as opposed to a nonsystematic encoder, which is an *FIR* linear system. The preferred form of expressing the encoder circuit is the *controller canonical form* (Forney 1970). An encoder for an RSC code consists of shift registers, a circuit[3] for dividing by $\bar{g}_0(D)$ and $n-1$ circuits for multiplying by $\bar{g}_1(D), \ldots, g_{n-1}(D)$.

Example 5.1.8 *A memory-2 rate-*$1/2$ *binary RSC encoder with generators* $(1, 5/7)$ *is shown in Figure 5.6. The trellis structure of this code is the same as the nonsystematic code with generators* $(7, 5)$. *However the input/output bits per transition are different. The state diagram of this code is shown in Figure 5.7. Compare it with Figure 5.2 on page 89. Among other things, this difference in input/output mapping means that the all-zero sequence may not necessarily bring the state back to* $s_0 s_1 = 00$. *This is evident from Figure 5.7. In particular, the impulse response of this encoder is* $(11, 01, 01, 00, 01, 01, 00, 01, 01, 00, \ldots)$.

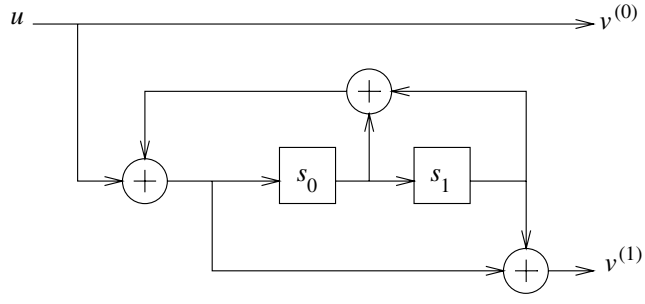


Figure 5.6 Encoder of a memory-2 rate-$1/2$ RSC encoder.

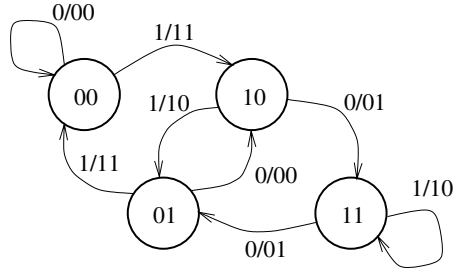


Figure 5.7 State diagram of a memory-2 rate-$1/2$ RSC encoder.

[3]The general structure of circuits that multiply and divide by two polynomials can be found, among others, in Peterson and Weldon (1972) (Figure 7.8) or (Lin and Costello 2005) (Section 4.7).

5.1.2 Free distance

The *free distance* d_f of a convolutional code is the smallest distance between any two distinct code sequences. The length of the sequences has to be sufficiently large, much larger than the constraint length of the code. The free distance of a convolutional code can be obtained from its weight enumerator polynomial, as described in Section 5.3. There are other distances associated with a convolutional code, when the length of the sequence is of the order of the constraint length, but these are not relevant for the discussion in this book. More details on the structure of a general rate-k/n convolutional encoder can be found in the references Lin and Costello (2005) and Johannesson and Zigangirov (1999).

5.2 Connections with block codes

There is a connection between convolutional codes and block codes, as is evident from the description of convolutional encoders in the earlier section. As indicated previously, it is customary for the information sequences input to a convolutional encoder to be broken into blocks of *finite length* (e.g., a few thousands of bits). Generally, a fixed sequence of length m is appended at the end of each information sequence. This sequence is typically a *unique word* that serves to synchronize the receiver and forces the convolutional encoder to return to a known state.

In the following text, for simplicity of exposition, let C be a nonsystematic FIR convolutional code C with free distance d_f, obtained from a memory-m rate-$1/n$ convolutional encoder. Similar arguments apply to RSC codes.

5.2.1 Zero-tail construction

By appending a "tail" of m zeros to an information vector $\bar{u}$ of length $(K-m)$, all paths in the trellis corresponding to 2^{K-m} code words merge at the all-zero state. The result is a linear block $(nK, (K-m), d_{\mathrm{ZT}})$ code, denoted by C_{ZT}. If K is large enough, the rate of C_{ZT} approaches the rate of C. As long as $K > m$ holds, with K being sufficiently large, even though the code sequences are restricted to end at the all-zero state, the minimum distance of C_{ZT} satisfies $d_{\mathrm{ZT}} = d_f$.

Example 5.2.1 *Let C be the convolutional code obtained from the memory-2 rate-$1/2$ encoder in Figure 5.1, of free distance*[4] $d_f = 5$. *Assume that message vectors of length $K = 3$ bits are encoded, each followed by $m = 2$ zeros. Then the zero-tail construction results in a binary linear block* $(10, 3, 5)$ *code C_{ZT} with generator matrix,*

$$G = \begin{pmatrix} 11 & 10 & 11 & 00 & 00 \\ 00 & 11 & 10 & 11 & 00 \\ 00 & 00 & 11 & 10 & 11 \end{pmatrix}.$$

The weight distribution sequence (WDS) of this code is $A(x) = 1 + 3x^5 + 3x^6 + x^7$, and $d_{\mathrm{ZT}} = 5$. The code rate of C_{ZT} is 0.3, *which is less than the rate $k/n = 1/2$ of C.*

[4]That this is the value of d_f will be shown in Section 5.3.

5.2.2 Direct-truncation construction

In this method, the code words of a linear block (nK, K, d_{DT}) code C_{DT} associated with a rate-k/n convolutional encoder are all code sequences associated with paths in the trellis that start in the all-zero state and, after K bits are fed into the encoder, can end at any state. The rate of C_{DT} is the same as that of the convolutional encoder. However, the minimum distance of the resulting block code is reduced and $d_{\mathrm{DT}} < d_f$.

Example 5.2.2 *Consider the memory-2 rate-*$1/2$ *encoder in Figure 5.1. Information vectors of length* $K = 3$ *information bits are encoded. The direct truncation construction then gives a binary linear block* $(6, 3, d_{\mathrm{DT}})$ *code with generator matrix,*

$$G = \begin{pmatrix} 11 & 10 & 11 \\ 00 & 11 & 10 \\ 00 & 00 & 11 \end{pmatrix}.$$

The WDS of this code is $A(x) = 1 + x^2 + 3x^3 + 2x^4 + x^5$ *and the minimum distance* $d_{\mathrm{DT}} = 2 < d_f$.

5.2.3 Tail-biting construction

The code words of the tail-biting block code C_{TB} are those code sequences associated with paths in the trellis that start from a state equal to the last m bits of an information vector of K data bits. This ensures that all code words in C_{TB} begin and end *at the same state*. The rate of the resulting block $(2K, K, d_{\mathrm{TB}})$ code is the same as that of the convolutional encoder. However, unless $K > m$ and K is sufficiently large, the minimum distance $d_{\mathrm{TB}} \leq d_f$.

Example 5.2.3 *Consider the memory-2 rate-*$1/2$ *encoder in Figure 5.1. Assume that a block of* $K = 5$ *information bits is encoded. Then the tail-biting construction gives a binary linear block* $(10, 5, d_{\mathrm{TB}})$ *code with generator matrix,*

$$G = \begin{pmatrix} 11 & 10 & 11 & 00 & 00 \\ 00 & 11 & 10 & 11 & 00 \\ 00 & 00 & 11 & 10 & 11 \\ 11 & 00 & 00 & 11 & 10 \\ 10 & 11 & 00 & 00 & 11 \end{pmatrix}.$$

The WDS[5] *is* $A(x) = 1 + 5x^3 + 5x^4 + 6x^5 + 10x^6 + 5x^7$ *and the minimum distance* $d_{\mathrm{TB}} = 3 < d_f$.

5.2.4 Weight distributions

In this section, a method is described to obtain the weight distribution of linear block codes derived from binary rate-$1/n$ convolutional codes by any of the constructions in the previous sections (Wolf and Viterbi 1996). Let $\Omega(x)$ be a $2^m \times 2^m$ *state transition matrix* with entries of the form

$$\Omega_{ij}(x) = \delta_{ij} x^{h_{ij}}, \tag{5.9}$$

[5]This WDS was obtained with a program from the ECC web site that computes the weight distribution of a binary linear code from its generator matrix.

where $\delta_{ij} = 1$, if and only if there is a transition from state i to state j, and $\delta_{ij} = 0$ otherwise, and h_{ij} is the Hamming weight of the corresponding output vector (of length n).

Example 5.2.4 *For the convolutional encoder with the state diagram shown in Figure 5.2, the state transition matrix $\Omega(x)$ is given by*

$$\Omega(x) = \begin{pmatrix} 1 & 0 & x^2 & 0 \\ x^2 & 0 & 1 & 0 \\ 0 & x & 0 & x \\ 0 & x & 0 & x \end{pmatrix}.$$

The WDS of a binary linear block (n, k) code constructed from any of the previous methods can be obtained by symbolically raising matrix $\Omega(x)$ to the ℓ-th power, denoted by $\Omega^{\ell}(x)$, and combining different terms.

The term $\Omega_{ij}^{\ell}(x)$ gives the weight distribution of trellis paths that start in state i and end in state j after ℓ time intervals. For the ZT construction, the value $\ell = k + m$ is used, while for both DT and TB constructions, $\ell = k$. The WDSs for each of the construction methods presented in the preceding text can be computed as follows.

- **ZT construction**:

$$A(x) = \Omega_{00}^{k+m}(x).$$

- **DT construction**:

$$A(x) = \sum_{j=0}^{2^m} \Omega_{0j}^{k}(x).$$

- **TB construction**:

$$A(x) = \sum_{j=0}^{2^m} \Omega_{jj}^{k}(x).$$

Example 5.2.5 *Consider again the memory-2 rate-$1/2$ convolutional encoder. Then*

$$\Omega^3(x) = \begin{pmatrix} 1+x^5 & x^3+x^4 & x^2+x^3 & x^3+x^4 \\ x^2+x^3 & x^5+x^2 & x^4+x & x^5+x^2 \\ x^3+x^4 & x^2+x^3 & x^5+x^2 & x^2+x^3 \\ x^3+x^4 & x^2+x^3 & x^5+x^2 & x^2+x^3 \end{pmatrix}.$$

The WDS of the code from the DT construction in Example 5.2.2 is obtained by adding the terms in the first row of $\Omega^3(x)$.

References on other methods of computing the WDS of a linear block code derived from a convolutional code are Fossorier *et al.* (1999), Chaudhari and Khandani (2001) and Desaki *et al.* (1994).

5.3 Weight enumeration

The performance of convolutional codes over memoryless channels can be estimated by using union bounds, such as those presented in Chapter 1 for block codes. It was Viterbi (1971) who first considered the performance of convolutional codes. See also Viterbi and Omura (1979). For a convolutional code, the *weight enumerating sequence* (WES) is defined as

$$T(x) = T_1 x + T_2 x^2 + \cdots + T_w x^w + \cdots, \tag{5.10}$$

where the coefficient T_w in $T(x)$ denotes the number of code sequences of weight w. Note that, in principle, the inputs to a convolutional code are infinite sequences. As a result, in general, the WES has an infinite number of terms. It is possible, however, to obtain closed-form expressions for the WES of convolutional codes, as shown in the following text.

Notation: A state $(s_0 s_1 \ldots s_{m-1})$ of the encoder will be denoted by $S^{(I)}$, where

$$I = \sum_{i=0}^{m-1} s_i 2^{m-1-i}.$$

A *modified state diagram* enables us to compute the WES. This diagram has the all-zero state split into an initial state $S^{(0)}_{\text{init}}$ and a final state $S^{(0)}_{\text{final}}$. The branches of the diagram are labeled by terms x^w, where w is the Hamming weight of the output sequence.

Example 5.3.1 *Let C be the memory-2 rate-$1/2$ convolutional code with generators $(7, 5)$ of Example 5.1.1. Figure 5.8 shows the modified state diagram of C.*

There are basically two methods to compute $T(x)$. In the first method, the modified state diagram is considered a signal flow graph. Mason's rule is then applied to find $T(x)$. This is the method covered in most textbooks on error correcting codes. For details, the reader is referred to Lin and Costello (2005).

Another method, originally developed by Viterbi (1971) consists of assigning intermediate variables $\mu_j(x)$ to each state $S^{(j)}$, $j = 1, 2, \ldots, 2^m - 1$. Each $\mu_j(x)$ can be expressed

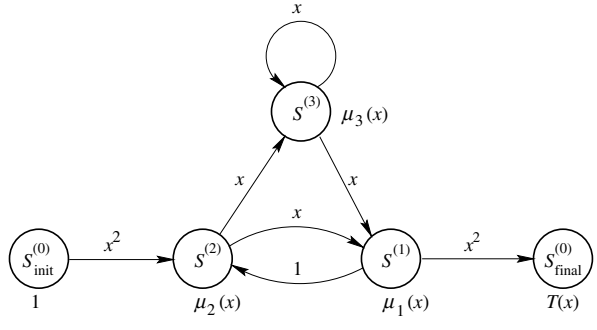


Figure 5.8 The modified state diagram of a memory-2 rate-$1/2$ convolutional code.

as a linear combination of weighted variables associated with states connected to $S(j)$,

$$\mu_j(x) = \sum_{\substack{k=1 \\ k \neq j}}^{2^m-1} \mu_k(x)\delta_{kj}x^{h_{kj}}, \tag{5.11}$$

for all $k \in \{1, 2, \ldots, 2^m - 1\}$, $k \neq j$, and where δ_{ij} and h_{kj} are defined as in Equation (5.9). The initial state $S_{\text{init}}^{(0)}$ is assigned a variable of value 1, while WES $T(x)$ is assigned as a variable to the final state $S_{\text{final}}^{(0)}$.

This yields a system of linear equations that can be solved to find $\mu_1(x)$, from which, using the adopted notation to designate states,

$$T(x) = \mu_1(x)x^{h_{10}}. \tag{5.12}$$

Example 5.3.2 *Consider the modified state diagram in Figure 5.8. The following set of equations is obtained*

$$\begin{aligned} \mu_1(x) &= \mu_2(x)x + \mu_3(x)x \\ \mu_2(x) &= x^2 + \mu_1(x) \\ \mu_3(x) &= \mu_2(x)x + \mu_3(x)x \end{aligned} \tag{5.13}$$

and $T(x) = \mu_1(x)x^2$. Equations (5.13) can be written in matrix form as

$$\begin{pmatrix} 1 & -x & -x \\ -1 & 1 & 0 \\ 0 & -x & 1-x \end{pmatrix} \begin{pmatrix} \mu_1(x) \\ \mu_2(x) \\ \mu_3(x) \end{pmatrix} = \begin{pmatrix} 0 \\ x^2 \\ 0 \end{pmatrix}, \tag{5.14}$$

and solving for $\mu_1(x)$ gives

$$\mu_1(x) = \frac{x^3}{1-2x}.$$

It follows that

$$T(x) = \frac{x^5}{1-2x} = x^5 + 2x^6 + 4x^7 + 8x^8 + \cdots \tag{5.15}$$

This shows that the minimum free distance is $d_f = 5$ for the binary 4-state rate-$1/2$ convolutional code with generators $(7, 5)$.

More detailed information about the structure of a convolutional code is obtained by labeling the modified state diagram with terms of the form $x^w y^\ell z^m$, where w is the Hamming weight of the coded outputs, ℓ is the Hamming weight of the input vector (of length equal to k, in general, for a rate-k/n convolutional code) and m is the number of branches differing from the all-zero sequence. (See Lin and Costello (2005), p. 493.) This results in a *complete weight enumerator sequence* (CWES), $T(x, y, z)$, which can be used to derive various bounds on the error performance of convolutional codes (Johannesson and Zigangirov 1999; Lin and Costello 2005; Viterbi 1971). A WES similar to the CWES was used to estimate the performance of turbo codes in Benedetto and Montorsi (1996).

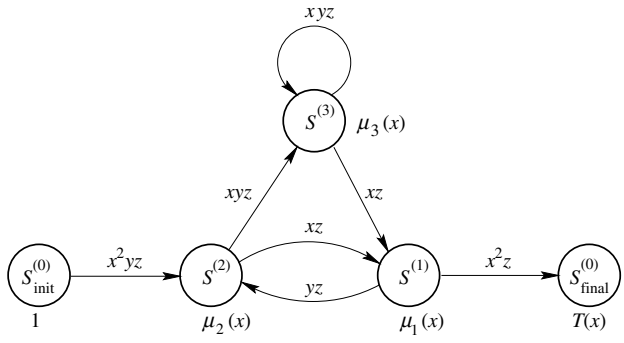


Figure 5.9 The modified state diagram of a memory-2 rate-1/2 convolutional code.

Example 5.3.3 *Continuing with Example 5.3.2 , a modified state diagram for the computation of the CWES is shown in Figure 5.9.*

The equations are

$$\begin{pmatrix} 1 & -xz & -xz \\ -yz & 1 & 0 \\ 0 & -xyz & 1-xyz \end{pmatrix} \begin{pmatrix} \mu_1(x) \\ \mu_2(x) \\ \mu_3(x) \end{pmatrix} = \begin{pmatrix} 0 \\ x^2yz \\ 0 \end{pmatrix},$$

with $T(x, y, z) = x^2z\mu_1(x)$. *The solution is:*

$$T(x, y, z) = \frac{x^5yz^3}{1 - xyz(1+z)} = x^5yz^3 + x^6y^2z^4(1+z) + x^7y^3z^5(1+z)^2 + \cdots$$

5.4 Performance bounds

Bounds on the bit error probability of a binary convolutional code of rate k/n can be obtained with the aid of the CWES described in the previous section. For transmission over a binary symmetric channel (BSC) and binary transmission over an additive white Gaussian noise (AWGN) channel (Viterbi and Omura 1979) and maximum-likelihood decoding (MLD)[6], the following upper bounds hold, respectively,

$$P_b < \frac{1}{k} \left. \frac{\partial T(x, y, z)}{\partial y} \right|_{x=\sqrt{4p(1-p)}, y=1, z=1}, \tag{5.16}$$

$$P_b < \frac{1}{k} \left. \frac{\partial T(x, y, z)}{\partial y} \right|_{x=e^{-RE_b/N_0}, y=1, z=1}, \tag{5.17}$$

where the energy-per-bit-to-noise ratio E_b/N_0 is related to the energy-per-symbol-to-noise ratio E_s/N_0 via the code rate $R = k/n$ is as follows:

$$\frac{E_b}{N_0} = \frac{1}{R} \frac{E_s}{N_0}. \tag{5.18}$$

[6]An example of an MLD algorithm is the Viterbi decoder, explained in the next section.

Union bounds may be used to estimate the bit-error rate (BER) performance of a convolutional code. The reader should be aware, however, that the bounds in Equations (5.16) and (5.17) are quite loose. Fortunately, tighter (close to the actual BER performance) bounds exist at relatively mild channel conditions, that is, low values of p for the BSC and high values of E_s/N_0 for the AWGN channel, and are given by the following (Johannesson and Zigangirov 1999):

$$P_b < \frac{1}{k}\left(\frac{(1+x)}{2}\frac{\partial T(x,y,z)}{\partial y} + \frac{(1-x)}{2}\frac{\partial T(-x,y,z)}{\partial y}\right)\Bigg|_{x=\sqrt{4p(1-p)},y=1,z=1}, \quad (5.19)$$

$$P_b < \frac{1}{k}\, Q\left(\sqrt{2R\, d_f \frac{E_b}{N_0}}\right) e^{2R\, d_f E_b/N_0}\, \frac{\partial T(x,y,z)}{\partial y}\Bigg|_{x=e^{-RE_b/N_0},y=1,z=1}. \quad (5.20)$$

The application of these bounds is given in a numerical example 5.5.4 after discussing the Viterbi algorithm in the next section.

Example 5.4.1 *The bounds in Equations (5.16) and (5.19) on the probability of a bit error P_b for the 4-state rate-1/2 convolutional code of Example 5.3.3, with transmission over a BSC with crossover probability p and MLD, are plotted in Figure 5.10. Evident from the figure is the fact that the bound in Equation (5.19) is tighter than the bound in Equation (5.16).*

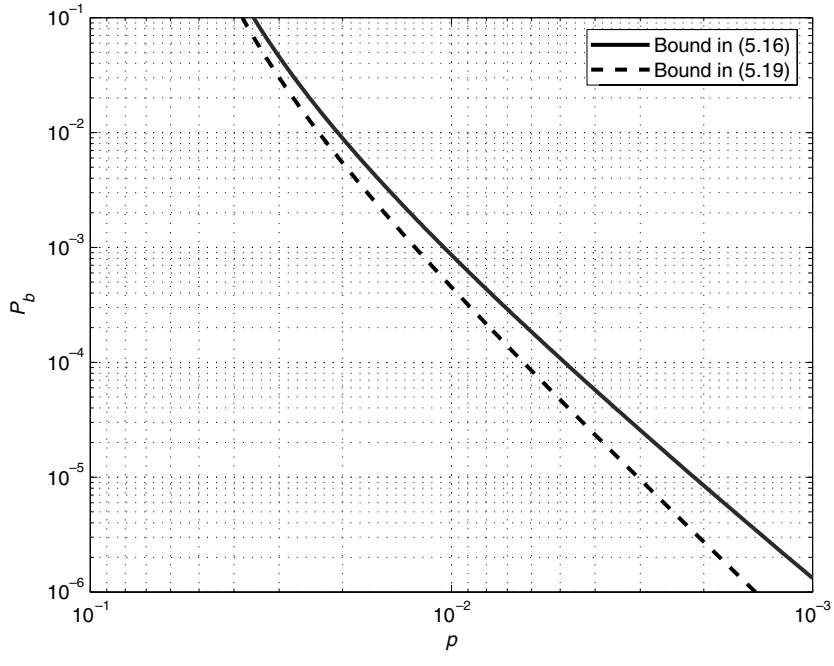


Figure 5.10 Bounds in Equations (5.16) and (5.19) on the BER of a memory-2 rate-1/2 convolutional code with $d_f = 5$. Transmission over BSC with crossover probability p and MLD.

5.5 Decoding: Viterbi algorithm with Hamming metrics

The trellis of convolutional codes has a regular structure. It is possible to take advantage of the repetitive pattern of the trellis in decoding. However, for linear block codes obtained from terminating convolutional codes and long information sequences, MLD is simply too complex and inefficient to implement.

An efficient solution to the decoding problem is a dynamic programming algorithm known as the *Viterbi algorithm*, also known as the *Viterbi decoder* (VD). This is a *maximum likelihood decoder* in the sense that it finds the closest coded sequence $\bar{v}$ to the received sequence $\bar{r}$ by processing the sequences on an information bit-by-bit (branches of the trellis) basis. In other words, instead of keeping a score of each possible coded sequence, *the VD tracks the states of the trellis*.

5.5.1 Maximum-likelihood decoding and metrics

The *likelihood* of a received sequence $\bar{R}$ after transmission over a noisy memoryless channel, given that a coded sequence $\bar{V}$ is sent, is defined as the conditional probability density function

$$p_{\bar{R}|\bar{V}}(\bar{r}|\bar{v}) = \prod_{i=0}^{n-1} P(r_i|v_i), \tag{5.21}$$

where $\bar{V}$ and $\bar{R}$ are the transmitted and received sequences, respectively.
It is easy to show that for a BSC with parameter p,

$$p_{\bar{R}|\bar{V}}(\bar{r}|\bar{v}) = \prod_{i=0}^{n-1}(1-p)\left(\frac{p}{1-p}\right)^{d_H(r_i,v_i)}, \tag{5.22}$$

with $d_H(r_i, v_i) = 1$, if $r_i \neq v_i$, and $d_H(r_i, v_i) = 0$, if $r_i = v_i$. That is, $d_H(r_i, v_i)$ is the *Hamming distance* between bits r_i and v_i. For an AWGN channel, the likelihood is given by

$$p_{\bar{R}|\bar{V}}(\bar{r}|\bar{v}) = \prod_{i=0}^{n-1} \frac{1}{\sqrt{\pi N_0}} e^{-\frac{1}{N_0}(r_i - m(v_i))^2}, \tag{5.23}$$

where $m(\cdot)$ denotes a binary modulated signal. Here, m is defined as a *one-to-one mapping* between bits $\{0, 1\}$ and real numbers $\{-\sqrt{E_s}, +\sqrt{E_s}\}$, where E_s is the energy per symbol. This mapping is also known as binary phase-shift keying (BPSK) modulation or polar mapping.

An *MLD* selects a coded sequence $\bar{v}'$ that *maximizes* Equation (5.21). By taking the logarithm of Equation (5.21), the following can be shown. For the BSC, an MLD is equivalent to choosing the code sequence that minimizes the *Hamming distance*

$$d_H(\bar{r}, \bar{v}) = \sum_{i=0}^{n-1} d_H(r_i, v_i). \tag{5.24}$$

Similarly, for the AWGN channel, it is the squared *Euclidean distance*

$$d_E(\bar{r}, \bar{v}) = \sum_{i=0}^{n-1} \left(\bar{r} - m(\bar{v})\right)^2 \tag{5.25}$$

that is minimized by the coded sequence selected by the MLD. In this section, a BSC with crossover error probability p is considered. The AWGN channel is covered in Chapter 7.

5.5.2 The Viterbi algorithm

Let $S_i^{(k)}$ denote a state in the trellis at stage i. Each state $S_i^{(k)}$ in the trellis is assigned a state metric, or simply a *metric*, $M(S_i^{(k)})$, and a *path* in the trellis, $\bar{y}^{(k)}$. A key observation in applying the Viterbi algorithm is:

> With i being the time, most likely paths per state $\bar{y}_i^{(k)}$ (the ones closest to the received sequence) will eventually coincide at some time $i - \ell$.

In his paper, Viterbi (1967) indicates that the value of ℓ for memory-m rate-1/2 binary convolutional codes should be $\ell > 5m$. The VD operates within a range of L received n-tuples (output bits per state transition) known as the *decoding depth*. The value of L must be such that $L > \ell$.

In the following text, the Viterbi algorithm applied to a memory-m rate-$1/n$ binary convolutional code is described and its operation illustrated via a simple example. Some additional notation is needed: Let

$$\bar{v}[i] = (v_0[i]v_1[i] \ldots v_{n-1}[i])$$

denote the coded bits in a branch (state transition), and let

$$\bar{r}[i] = (r_0[i]r_1[i] \ldots r_{n-1}[i])$$

denote the output of the channel.

Basic decoding steps

Initialization

Set $i = 0$. Set metrics and paths

$$M(S_0^{(k)}) = 0, \qquad \bar{y}_0^{(k)} = () \quad \text{(empty)}.$$

The specific way in which the initialization of the paths is performed is irrelevant, as shown later. For the sake of clarity of presentation of the algorithm, it is assumed that the paths are represented as *lists* that are initialized to the empty list.

1. **Branch metric computation**

   At stage i, compute the partial *branch metrics*

   $$BM_i^{(b)} = d_H(\bar{r}[i], \bar{v}[i]), \tag{5.26}$$

   $b \triangleq \sum_{\ell=0}^{n-1} v_\ell[i]2^{n-1-\ell}$, associated with the n outputs $\bar{v}[i]$ of every branch (or state transition) and the n received bits $\bar{r}[i]$.

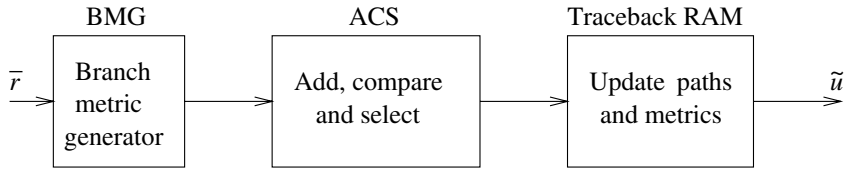


Figure 5.11 Block diagram of a Viterbi decoder.

2. **Add, compare and select (ACS)**

   For each state $S_i^{(k)}$, $k = 0, 1, \ldots, 2^m - 1$, and corresponding pair of incoming branches from two precursor states $S_{i-1}^{(k_1)}$ and $S_{i-1}^{(k_2)}$, the algorithm compares the extended branch metrics $M(S_{i-1}^{(k_1)}) + BM_i^{(b_1)}$ and $M(S_{i-1}^{(k_2)}) + BM_i^{(b_2)}$, where

$$b_j = \sum_{\ell=0}^{n-1} v_\ell[i] 2^{n-1-\ell} i = 1, 2,$$

   and selects a *winning branch*, giving the smallest path metric, and updates the metric,

$$M(S_i^{(k)}) = \min\{M(S_{i-1}^{(k_1)}) + BM_i^{(b_1)}, M(S_{i-1}^{(k_2)}) + BM_i^{(b_2)}\}. \tag{5.27}$$

3. **Path memory update**

   For each state $S_i^{(k)}$, $k = 0, 1, \ldots, 2^m - 1$, update the *survivor paths* $\bar{y}^{(k)}$ as follows, with the output of the winning branch $\bar{v}_{k_j}$, $j \in \{1, 2\}$,

$$\bar{y}_i^{(k)} = (\bar{y}_{i-1}^{(k_j)}, \bar{v}_{k_j}). \tag{5.28}$$

4. **Decode symbols**

   If $i > L$, then output as the estimated coded sequence $\bar{y}_{i-L}^{(k')}$, where k' is the index of the state $S^{(k')}$ with the smallest metric. Set $i = i + 1$ and go to decoding step 1.

It should be stressed that this is not the only way to implement the Viterbi algorithm. The procedure in the preceding text can be considered a *classical* algorithm. This is shown in Fig. 5.11. There are alternative implementations that, depending on the particular structure of the underlying convolutional encoder, may offer advantages (see, e.g., Fossorier and Lin (2000)). In addition, in the last step of the algorithm, symbol decoding can be applied to information bits directly. This is the form usually employed in the software implementations of VDs that are available on the ECC web site. In hardware implementations, a method based on a *traceback memory* is favored that estimates the original information sequence, indirectly, on the basis of state transitions. This technique is discussed later in the chapter.

Example 5.5.1 *Consider again the memory-2 rate-*$1/2$ *convolutional encoder with generators* $(7, 5)$. *Note that* $d_f = 5$ *for this code. This example shows how a single error can be corrected. Suppose that* $\bar{v} = (11, 01, 01, 00, 10, 11)$ *is transmitted over a BSC and that* $\bar{r} = (10, 01, 01, 00, 10, 11)$ *is received (one error in the second position). The operation of the VD is illustrated in Figures 5.12 to 5.17. The evolution of the metric values with respect to the decoding stages is shown in the following table:*

State/Stage	$i=0$	$i=1$	$i=2$	$i=3$	$i=4$	$i=5$	$i=6$
$S_i^{(0)}$	*0*	*1*	*1*	*1*	*1*	*2*	*1*
$S_i^{(1)}$	*0*	*0*	*0*	*1*	*2*	*1*	*3*
$S_i^{(2)}$	*0*	*1*	*1*	*1*	*1*	*2*	*2*
$S_i^{(3)}$	*0*	*0*	*1*	*1*	*2*	*2*	*2*

After processing six stages of the trellis ($i=6$), the state with the smallest metric is $S_6^{(0)}$ with associated (survivor) path $\bar{y}_6^{(0)} = \bar{v}$. One error has been corrected.

5.5.3 Implementation issues

In this section, some of the implementation issues related to VDs are discussed. The techniques provided here apply equally to any VD that operates over channels with additive metrics, such as the BSC, AWGN and flat Rayleigh fading channels.

Path metric initialization

The VD can operate *in the same mode* from the start ($i=0$). The survivor paths can have arbitrary values, without affecting the decoder's performance. The first L decoded bits are

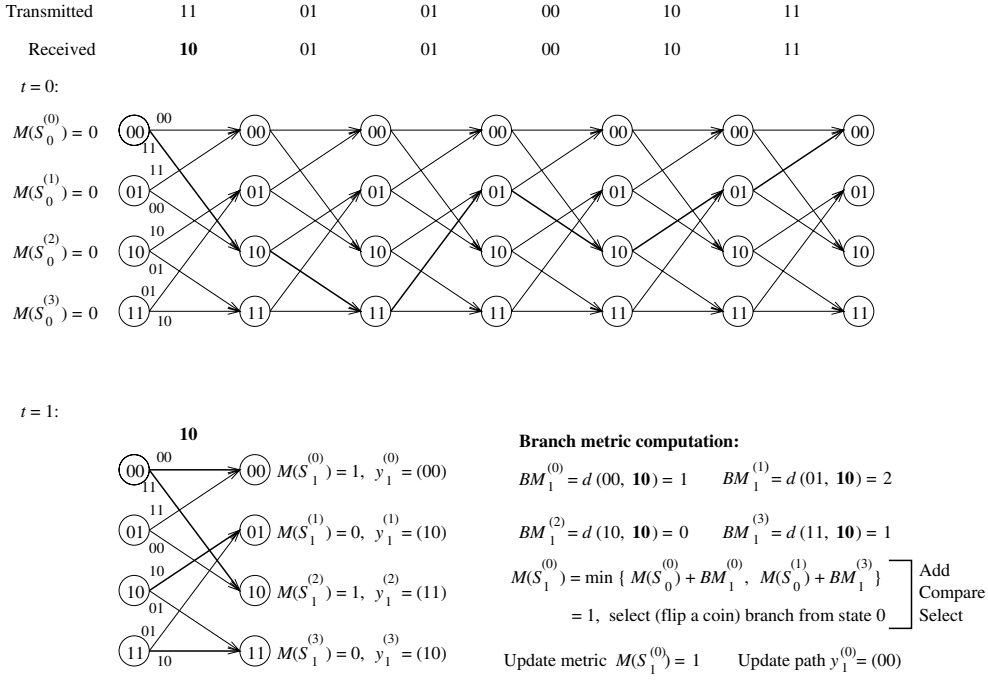


Figure 5.12 VD operation for Example 5.5.1, at $i=0$ and $i=1$.

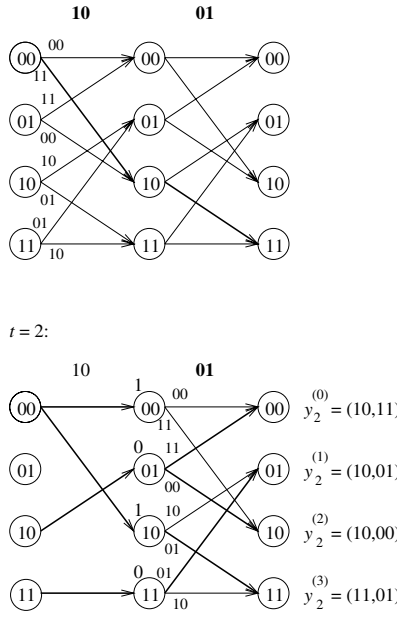


Figure 5.13 VD operation for Example 5.5.1, at $i = 2$.

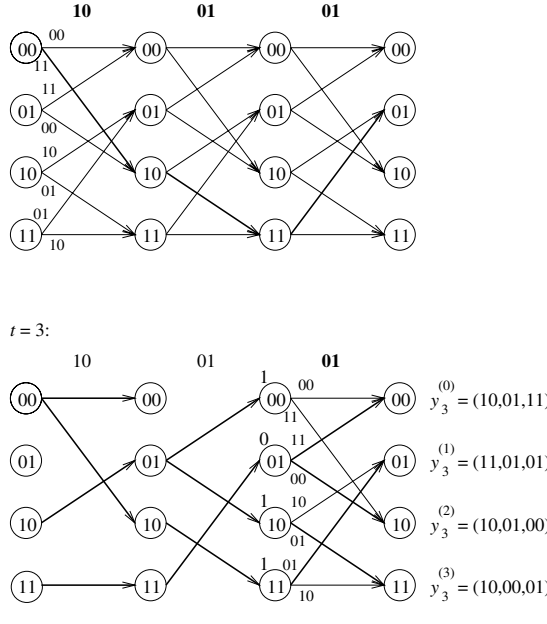


Figure 5.14 VD operation for Example 5.5.1, at $i = 3$.

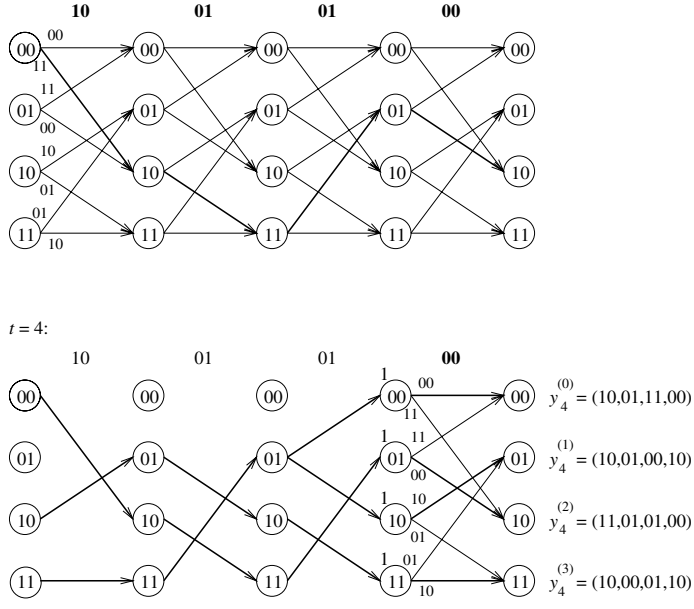


Figure 5.15 VD operation for Example 5.5.1, at $i = 4$.

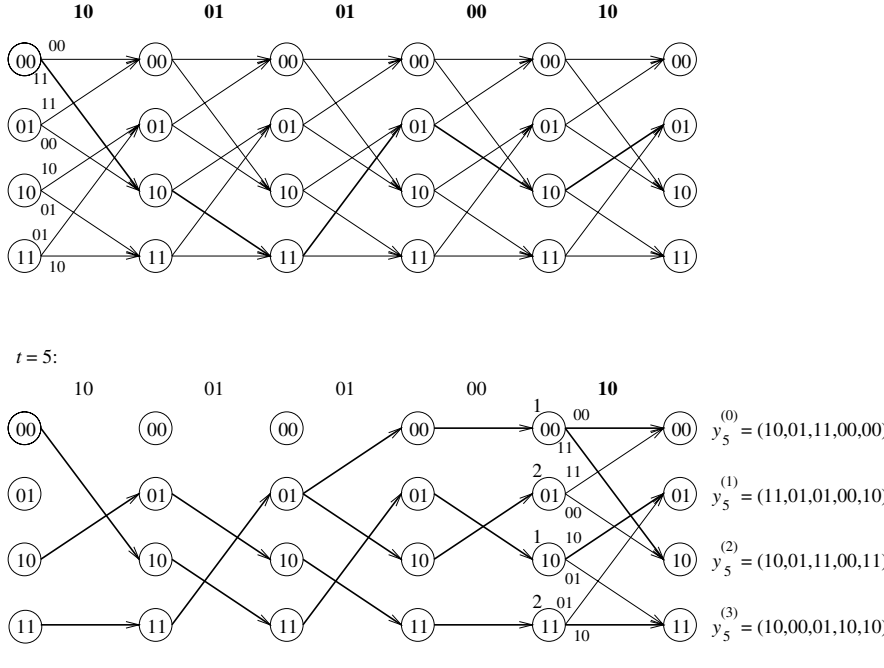


Figure 5.16 VD operation for Example 5.5.1, at $i = 5$.

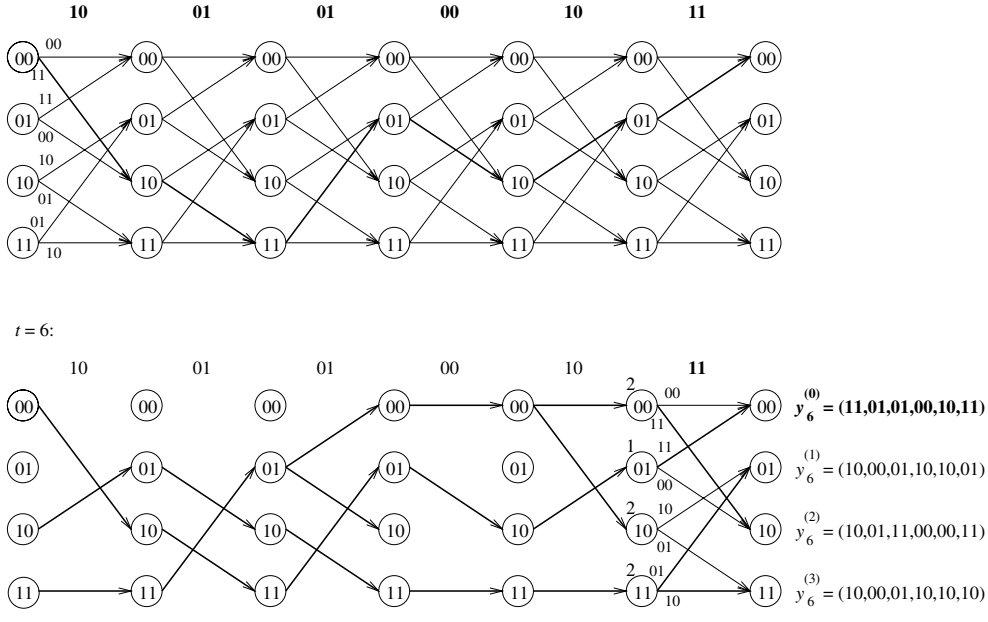


Figure 5.17 VD operation for Example 5.5.1, at $i = 6$.

therefore random and give no information. For this reason, the value of L contributes to the decoding delay and is also known as the *decoding depth*. Moreover, provided that L is large enough ($L \gg \ell$, where $\ell > 5m$ for rate-1/2 binary codes), the decoded bit can either output from the path with the lowest metric or always output from the zero state path ($\bar{y}^{(0)}$). The latter method is easier to implement and does not result in loss of performance. The programs on the ECC web site that implement MLD using the Viterbi algorithm work in this fashion.

Also note that in Example 5.5.1 the branch labels (output) were stored in the survivor paths. This was done in order to facilitate understanding of the algorithm. In a practical implementation, however, it is the corresponding *information bits* that are stored. This is discussed in the following text, in connection with path memory management.

Synchronization

Branch symbols must be properly *aligned* with the received symbols. Any misalignment can be detected by monitoring the value of a random variable associated with the VD. Two commonly used *synchronization variables* are (1) path metric growth, and (2) channel BER estimates. The statistics of these variables give an indication of abnormal decoding behavior.

Assume that the received sequence is not properly received, that is, the n-bit branch labels $\bar{v}[i]$ in the decoder are not properly aligned, or synchronized, with the received sequence $\bar{r}[i]$.

Example 5.2.2 *Figure 5.18 shows an example for a rate-*$1/2$ *in which the received sequence* $\bar{r}$ *is not synchronized with the reference coded sequence* $\bar{v}$.

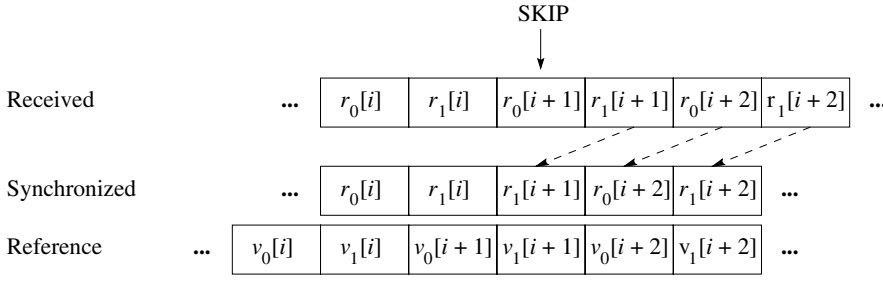


Figure 5.18 Example of branch misalignment in a Viterbi decoder.

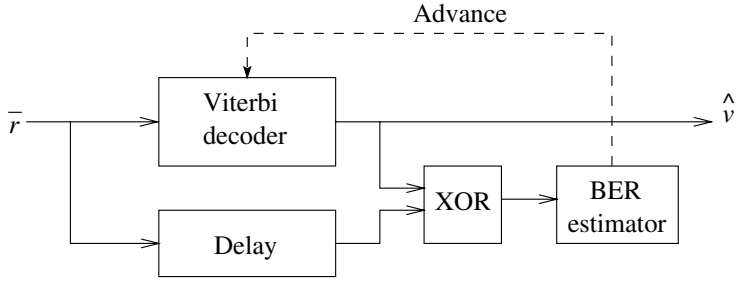


Figure 5.19 Channel error rate estimation for a BSC.

In other words, not all the bits in the received subsequence $\bar{r}[i]$ belong to the same trellis stage in the decoder. In this case, the two events that may occur are (1) the path metrics are close to each other and grow rapidly and (2) the estimated channel BER approaches $1/2$. Figure 5.19 shows the block diagram of a VD and a BER monitor.

A synchronization stage needs to be added, external to the decoder itself, whose function is to advance the reference sequence $\bar{v}$ in the decoder until the statistics return to normal. This can be done by skipping received symbols (a maximum of $n-1$ times) until the synchronization variables indicate normal decoding behavior. This is indicated in Figure 5.18 of Example 5.5.2 for the case of a rate-$1/2$ convolutional code.

Metric normalization

As the VD operates continuously, the path metrics will grow proportional to the length on the received sequence. To avoid overflow or saturation (depending on the number representation used), the metrics need to be normalized. There are basically two methods of doing this. Both rely on the following two properties of the Viterbi algorithm:

1. The MLD path selection depends only on the *metric differences*.

2. The metric differences are bounded.

- **Threshold method:**

  To normalize the metrics, the value of the smallest metric

  $$M_{\min} = \min_{0 \le k \le 2^m - 1} \left\{ M(S_i^{(k)}) \right\}$$

  is compared to a threshold T at each decoding stage. If $M_{\min} > T$, then T is subtracted from all the metrics. Clearly, this does not affect the selection process because the metric differences remain the same. This is a method that can be easily implemented in software on a generic processor.

- **Modular arithmetic method:**

  The metrics are computed modulo N, so that they lie within the range $[0, N-1]$, where $N = 2\Delta_{\max}$ and $\Delta_{\max}$ is the maximum difference in survivor path metrics. Obviously, $\Delta_{\max}$ depends on the range of values received from the channel. From the two properties of the Viterbi algorithm, it can be shown that the same MLD path is selected by the VD when computing path metrics with modular arithmetic. In particular, it is possible to use *two's complement arithmetic*[7] in such a way that overflow can occur but it does not affect the selection process. For details, see Hekstra (1989) and Onyszchuk *et al.* (1993). This method is favored in hardware implementations of VDs.

Path memory management

In a VD, survivor paths and their metrics need to be stored and updated. In a continuous operation mode, while updating the survivor paths at stage i, earlier portions of the survivor paths merge with high probability at stage $i - L$, where L is the decoding depth. The estimated information bits are taken from the (single) portion of the merged paths at stage $i - L$. There are different techniques to extract the information bits. Two of the most common are (1) register exchange and (2) traceback memory.

- **Register exchange:**

  This method is the easiest to implement in software. All the survivor paths are updated at each iteration of the Viterbi algorithm. Therefore, the information bits can be read directly from the survivor paths. However, if this technique is implemented in hardware, the decoding speed would be low because of the excessive number of times the path memory is read and written. To simplify control flow instructions, a *circular pointer* of length L can be used. With a circular pointer, at decoding stage i, the position $(i - L)$ in memory (to output the decoded bit) is equal to the position $(i + 1)$ modulo L.

- **Traceback memory:**

  This technique is favored in implementations in hardware. The survivor paths are composed of *decision values*, which indicate *state transitions* in order to trace back the survivor paths and reconstruct a *sequences of states* in reverse order.

[7] Arithmetic using the additive group $\{-2^{m-1}, 1 - 2^{m-1}, \ldots, 2^{m-1} - 1\}$ of m-bit integers (Hekstra 1989).

The traceback memory can be organized as a rectangular array, with rows indexed by k, for all the trellis states $S_i^{(k)}$, $k = 0, 1, \ldots, 2^m - 1$. At decoding stage i, for each trellis state $S_i^{(j)}$, $0 \leq j \leq 2^m - 1$, the *traceback memory* (or TB RAM) is written, with the rightmost bit of the previous state $S_{i-1}^{(b_j)}$, $\ell \in \{1, 2\}$ associated with the winning branch.

The traceback method trades off memory for decoding speed, since it writes only one bit per state per decoding stage, instead of L bits per state per decoding stage, as in the register exchange method. The information bits are decoded by reading the state transitions in reverse order with an encoder replica (or looking up a state transition table). More memory (number of columns if organized in a rectangular array) is required for continuous operation because information bits are read out (decoded)and new decision values written simultaneously.

Example 5.5.3 *Continuing with Example 5.5.1, using the traceback technique, at the end of the $i = 6$ received pair of bits from the channel, the traceback memory would have the contents shown in Table 5.2.*

The traceback memory is read, starting from the last bit (e.g., use a "last in, first out"), $i = 6$ (in general $T = L$). The row address is given by the state with the best metric. This is state $S^{(0)}$ in the example. The transition bit b_6 (in general, b_L) is read. To read the next transition bit b_5, the row address (ROW) at $i = 5$ ($i = L - 1$) is obtained by shifting the address k at $i = 6$ ($i = L$) and appending the previous decision bit b_6 (b_L). This can be stated in the following C language instruction:

```
ROW[j] = ((ROW[j+1] << 1) && MASK) ^ TB_RAM[ROW[j+1]][j+1];
```

(where `MASK` $= 2^m - 1$.*)*

Continuing in this way, the following state sequence is obtained:

$$S_6^{(0)} \to S_5^{(1)} \to S_4^{(2)} \to S_3^{(1)} \to S_2^{(3)} \to S_1^{(2)} \to S_0^{(0)}.$$

From this sequence, the corresponding information sequence can be recovered by reversing the state transitions: $\hat{u} = \begin{pmatrix} 1 & 1 & 0 & 1 & 0 & 0 \end{pmatrix}$.

For high-speed continuous VD operation with the traceback method, the traceback memory must be partitioned into several blocks. In this way, while one block is being written with decisions at the current stage i, another block is read (decoded) at a time $i - L$, $L > \ell$, and another (possibly more than one) block for intermediate stages is used for performing traceback tracing. This memory partitioning improves speed but increases the latency (or decoding delay) of the decoder (Boo *et al.* 1997; Collins 1992).

Table 5.2 Tracing back the path that ended in state $S^{(0)}$.

k	$i = 1$	$i = 2$	$i = 3$	$i = 4$	$i = 5$	$i = 6$
0	0	1	1	0	0	**1**
1	0	1	**1**	0	**0**	1
2	**0**	1	1	**1**	0	0
3	1	**0**	0	1	1	1

ACS

For memory-m rate-$1/n$ binary convolutional codes, the basic trellis element is a *butterfly*, as shown in Figure 5.20. The ACS operation is implemented using this structure over 2^{m-1} pairs, that is, $j = 0, 1, 2, \ldots, 2^{m-1} - 1$. Therefore, the ACS operation can be done either serially (a loop on j in software) or in parallel, with 2^{m-1} ACS units, one per butterfly. If the code is *antipodal*, then the generators of the code have one unit in the first and last positions. In this case, the labels of the branches incident to a state $S^{(2j)}$ are the same as those incident to state $S^{(2j+1)}$ in a butterfly. Moreover, the label of a branch incident to a state $S^{(2j)}$ is equal to the complement of the label of the other branch.[8] Using these facts, a clever technique (Fossorier and Lin 2000) based on branch metric differences was proposed.

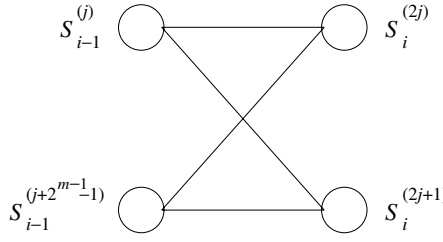


Figure 5.20 Trellis butterfly of a memory-m rate-$1/n$ binary convolutional code.

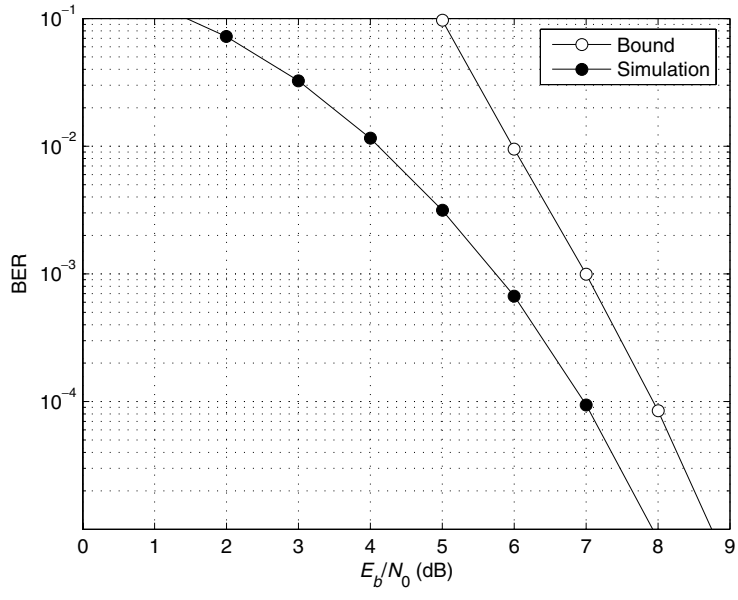


Figure 5.21 Simulation and bound on the bit error probability of a memory-2 rate-1/2 convolutional code. Binary transmission over an AWGN channel with hard decisions.

[8]The complement of a bit a is $1 + a$ mod 2.

For rate-$1/n$ codes, the result is a VD with a *compare-add-select* (CAS) architecture and a lower complexity of implementation than the conventional ACS approach.

Example 5.5.4 *Continuing with the example binary memory-2 rate-1/2 convolutional encoder with generators* $(7, 5)$*, simulation results using a VD with hard-decision metrics are shown in Figure 5.21. The union bound in Equation (5.20) was used with* $p = Q\left(\sqrt{\frac{2E_s}{N_0}}\right)$*. In the simulations, the noise power is computed on the basis of the energy-per-symbol-to noise ratio* E_s/N_0*, with* $E_s = RE_b$*.*

5.6 Punctured convolutional codes

Punctured convolutional codes were introduced in Clark and Cain (1981). Puncturing is the process of systematically deleting, or not sending, some output bits of a low-rate encoder.[9] Since the trellis structure of the low-rate encoder remains the same, the number of information bits per sequence does not change. As a result, the output sequences belong to a higher-rate *punctured convolutional (PC) code*. The following discussion focuses on codes obtained from a binary memory-m rate-$1/n$ convolutional code.

A *puncturing matrix* P specifies the rules of deletion of output bits. P is a $k \times n_p$ binary matrix, with binary symbols p_{ij} that indicate whether the corresponding output bit is transmitted ($p_{ij} = 1$) or not ($p_{ij} = 0$). An encoder of a PC code is shown in Fig. 5.22. Generally, the puncturing rule is periodic. A rate-k/n_p PC encoder based on a rate-$1/n$ encoder has a puncturing matrix P that contains ℓ zero entries, where $n_p = kn - \ell$, $0 \leq \ell < kn$.

Example 5.6.1 *A memory-2 rate-2/3 convolutional code can be constructed by puncturing the output bits of the encoder of Figure 5.1, according to the puncturing matrix*

$$P = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}.$$

The corresponding encoder is depicted in Figure 5.23. A coded sequence

$$\bar{v} = (\ldots, v_i^{(0)} v_i^{(1)}, v_{i+1}^{(0)} v_{i+1}^{(1)}, v_{i+2}^{(0)} v_{i+2}^{(1)}, v_{i+3}^{(0)} v_{i+3}^{(1)}, \ldots)$$

of the rate-$1/2$ *encoder is transformed into a code sequence*

$$\bar{v}_p = (\ldots, v_i^{(0)} v_i^{(1)}, v_{i+1}^{(0)}, v_{i+2}^{(0)} v_{i+2}^{(1)}, v_{i+3}^{(0)}, \ldots),$$

that is, every other bit of the second output is not transmitted.

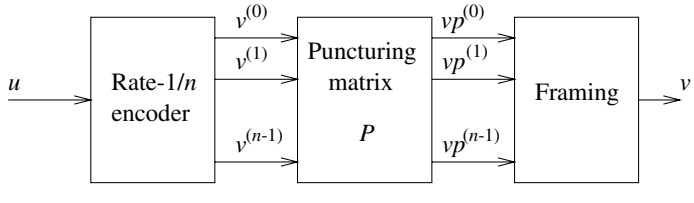


Figure 5.22 Encoder of a punctured code based on a rate-$1/n$ code.

[9] Sometimes referred to as the *mother code*.

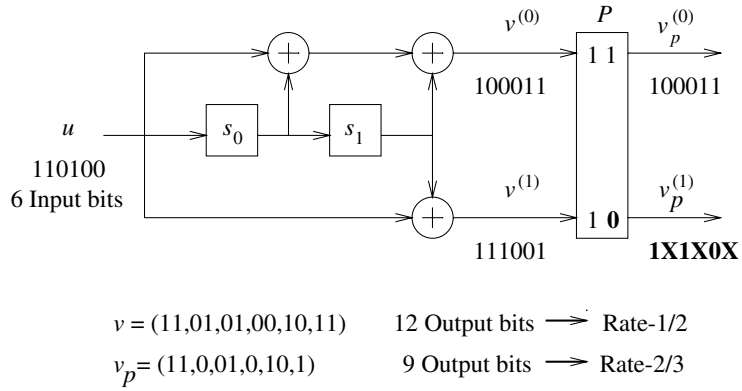


Figure 5.23 Encoder of a memory-2 rate-2/3 PC code.

One of the goals of puncturing is that the same decoder can be used for a variety of high-rate codes. One way to achieve decoding of a PC code using the VD of the low-rate code is by inserting "deleted" symbols in the positions that were not sent. This process is known as *depuncturing*. The "deleted" symbols are marked by a special flag. This can be done, for example, by using an additional bit and setting it to 1 in the position of an erased bit. If a position is flagged, then the corresponding received symbol is not taken into account in the branch metric computation.[10]

In a software implementation, an alternative method is for the decoder to check the entries $p_{m,j}$ of matrix P in the transmission order, $m = 0, 1, \ldots, n-1$, $j = i, i+1, \ldots, i+k-1$. If $p_{m,j} = 0$, then the current received symbol is not used to update the branch metric. The decoder advances a pointer to the next branch symbol in the trellis and repeats the test on $p_{m,j}$. Otherwise, $p_{m,j} = 1$ and the received symbol are used to update the branch metric. Every time the branch pointer is advanced, a check is made to determine whether all the symbols in this branch have been processed. If they have, then the ACS operation is performed. This method is used in some of the programs that implement Viterbi decoding of PC codes on the ECC web site.

Table 5.3 shows puncturing matrices employed with the de facto standard memory-6 rate-1/2 convolutional code with generators $(\bar{g}_0, \bar{g}_1) = (171,133)$. In the table, $v_m^{(i)}$ indicates the output, at time m, associated with generator $\bar{g}_i$, $i = 0, 1$. Other puncturing matrices can be found in Yasuda *et al.* (1984), Cain *et al.* (1979).

Example 5.6.2 *(Continuation of Example 5.6.1.) Let $\bar{u}$ be the same as in Example 5.5.1, and suppose transmission over a BSC introduces an error in the second position. Coded symbols $v^{(1)}[i]$, $i = 0, 2, 4, \ldots$, are not transmitted. The decoder knows this and inserts deleted symbols, denoted by E. When computing branch distances, the E symbols are ignored. The operation of a VD with deleted symbols is depicted in Figures 5.24 and 5.25 for $i = 1$ and $i = 2$, respectively. It is left as an exercise to finish the example.*

[10]Some authors write "the branch metric is zero" as an equivalent statement.

Table 5.3 Puncturing matrices for the de facto standard rate-1/2 code.

Rate	Pattern	Coded sequence	d_f
1/2	$\begin{pmatrix}1\\1\end{pmatrix}$	$v_i^{(0)}, v_i^{(1)}$	10
2/3	$\begin{pmatrix}1 & 0\\1 & 1\end{pmatrix}$	$v_i^{(0)}, v_i^{(1)}, v_{i+1}^{(1)}$	6
3/4	$\begin{pmatrix}1 & 0 & 1\\1 & 1 & 0\end{pmatrix}$	$v_i^{(0)}, v_i^{(1)}, v_{i+1}^{(1)}, v_{i+2}^{(0)}$	5
5/6	$\begin{pmatrix}1 & 0 & 1 & 0 & 1\\1 & 1 & 0 & 1 & 0\end{pmatrix}$	$v_i^{(0)}, v_i^{(1)}, v_{i+1}^{(1)}, v_{i+2}^{(0)}, v_{i+3}^{(1)}, v_{i+4}^{(0)}$	4
7/8	$\begin{pmatrix}1 & 0 & 0 & 0 & 1 & 0 & 1\\1 & 1 & 1 & 1 & 0 & 1 & 0\end{pmatrix}$	$v_i^{(0)}, v_i^{(1)}, v_{i+1}^{(1)}, v_{i+2}^{(1)}, v_{i+3}^{(1)}, v_{i+4}^{(0)}, v_{i+5}^{(1)}, v_{i+6}^{(0)}$	3

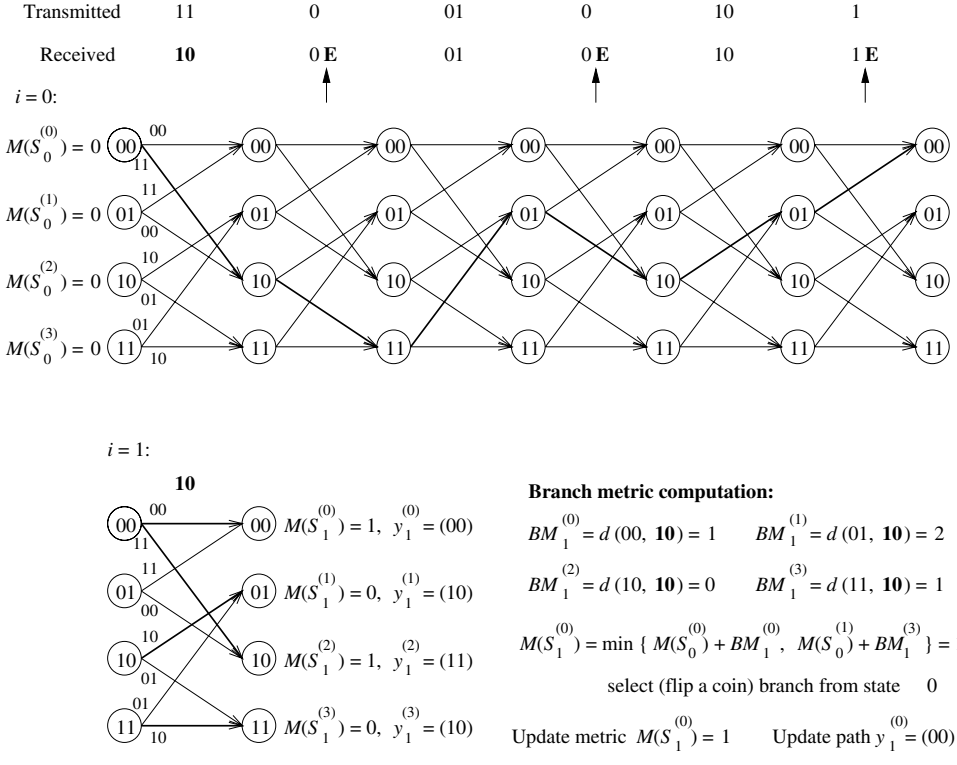
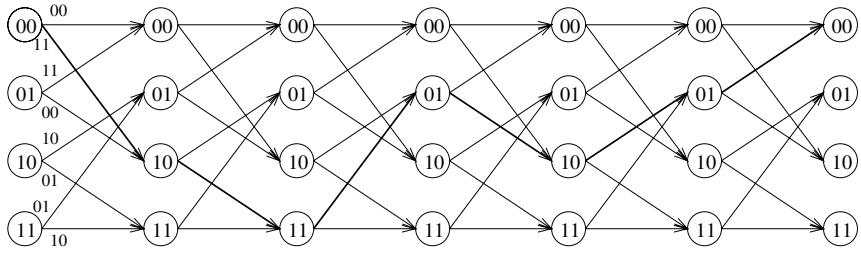


Figure 5.24 VD operation for a memory-2 rate-2/3 PC code, $i = 1$.

Transmitted	11	0	01	0	10	1
Received	**10**	**0E**	01	0E	10	1E



$i = 2$:

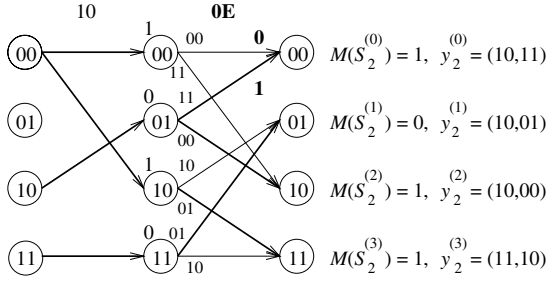


Figure 5.25 VD operation for a memory-2 rate-2/3 PC code, $i = 2$.

5.6.1 Implementation issues related to punctured convolutional codes

For a system that employs PC codes, the resynchronization process will take longer, as the output symbols are spread over several trellis branches. Moreover, practical systems use a multiplicity of puncturing patterns of different lengths. The decoder therefore may have to make a guess as to the specific rate and puncturing pattern being received. Other points to consider when implementing PC codes are as follows:

1. The *decoding depth L must be increased* as more output bits are punctured. This is because high-rate codes (e.g., rate 7/8) have low minimum Hamming distances between coded sequences (e.g., $d_{\min} = 3$) and, therefore, the survivor paths take longer to converge.

2. An *additional level of synchronization* with the received sequence is needed to align the puncturing pattern.

The fact that the puncturing patterns have different lengths for different rates may be solved by employing the same *puncturing period*. This is one of the features of *rate-compatible* PC (RCPC) codes.

5.6.2 RCPC codes

RCPC codes were introduced by Hagenauer (1988). These codes are constructed from a low-rate code by periodic puncturing. Let M denote the puncturing period. Then, as before, the puncturing matrices are $n \times M$ binary matrices. In general $M > kn - \ell$.

To construct a family of RCPC codes, the branch outputs of a high-rate PC code should be used by the lower-rate PC codes. This is achieved as follows. Let $P^{(H)}$ and $P^{(L)}$ denote matrices of a high-rate code C_H and a low-rate code C_L, respectively, both derived from the same lowest-rate code C. Then codes C_H and C_L are *rate compatible* if the following condition holds:

$$p_{ij}^{(H)} = 1 \rightarrow p_{ij}^{(L)} = 1.$$

An equivalent condition is

$$p_{ij}^{(L)} = 0 \rightarrow p_{ij}^{(H)} = 0.$$

Example 5.6.3 *Let C be a rate-$1/2$ convolutional code. Let $M = 4$. Then matrices $P(1)$ through $P(4)$ generate RCPC codes of rates 4/5, 4/6, 4/7 and 4/8 (=1/2), respectively.*

$$P(1) = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix}, \qquad P(2) = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & \mathbf{1} & 0 & 1 \end{pmatrix},$$

$$P(3) = \begin{pmatrix} 1 & 1 & 1 & \mathbf{1} \\ 1 & 1 & 0 & 1 \end{pmatrix}, \qquad P(4) = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & \mathbf{1} & 1 \end{pmatrix}.$$

RCPC codes find applications in automatic repeat request (ARQ) systems because of their incremental redundancy nature, as well as in constructing variable-rate codes and *unequal error protection* codes.

Problems

1. Let C denote the binary rate-1/2 RSC encoder C with generators $(1, 5/7)$.

   (a) Using its modified state diagram, find the complete WES $T(x, y, z)$ of C.

   (b) What is the value of the free distance of C?

   (a) Determine the coded sequence $\bar{v}$ that corresponds to the information sequence $\bar{u} = (01011)$ and sketch the associated path in the trellis.

2. Consider the binary memory-3 rate-1/2 convolutional encoder with generators $(15, 17)$.

   (a) Sketch the encoder's circuit and its state diagram.

   (b) Find its WES. What is the minimum free distance?

   (c) With transmission over a BSC, estimate the performance of this encoder and compare it with that of the memory-2 rate-1/2 convolutional encoder with generators $(5, 7)$.

3. Using the zero-tail construction and the binary memory-2 rate-1/2 convolutional encoder with generators $(5, 7)$, construct a binary linear block code C of dimension $k = 5$. Determine

   (a) the length and minimum Hamming distance of code C.

   (b) the WDS $A(x)$ of C.

   (c) the error performance of C with binary transmission over an AWGN channel.

4. Repeat the previous problem using the binary memory-3 rate-1/2 convolutional encoder with generators $(15, 17)$.

5. Repeat the previous problem using the RSC encoder of problem 1.

6. Show that the conditional probabilities $p_{\bar{R}|\bar{V}}(\bar{r}|\bar{v})$ with transmission over a BSC and an AWGN channel are given by Equations (5.22) and (5.23), respectively.

7. Show that puncturing always results in a decrease of the value of the free distance of the resulting PC code.

8. Show that the memory-2 rate-2/3 PC code, with mother code being the binary memory-2 rate-1/2 convolutional encoder with generators $(5, 7)$, has free distance $d_f = 6$.

9. Simulate the performance of each of the five memory-6 PCCs with puncturing matrices given in Table 5.3, with binary transmission over an AWGN channel, and compare it with the bound in Equation (5.20).

6

Modifying and combining codes

In this chapter, several techniques are presented that allow modification of an existing code, or the combination of several codes, in order to achieve flexibility in the design of error correcting codes (ECC). Many of the best codes known to date have been obtained not from members of a known family of codes, but from modifying and combining codes (Brouwer and Verhoeff 1993).

6.1 Modifying codes

In the following, let C denote a linear block (n, k, d) code over $GF(q)$ with generator matrix G and parity-check matrix H.

6.1.1 Shortening

Shortened cyclic codes were described in Section 3.1.5. Let s be an integer, $0 \leq s < k$. In general, a linear shortened $(n - s, k - s, d_s)$ code C_s has distance $d_s \geq d$. A generator matrix of C_s can be obtained from the generator matrix G of the original code C as follows. Assume that G is in systematic form, that is,

$$G = \left(I_k | P\right). \tag{6.1}$$

Then a $(k - s) \times (n - s)$ generator matrix G_s of the shortened code C_s is obtained by removing s columns of the identity matrix I_k and the s rows that correspond to where the selected columns elements are nonzero. This is best illustrated by an example.

Example 6.1.1 *Consider the Hamming* $(7, 4, 3)$ *code, presented in Example 2.1.1. This code has*

$$G = \begin{pmatrix} \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{1} \\ \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{1} & \mathbf{1} \\ \mathbf{0} & \mathbf{0} & 1 & 0 & 1 & 1 & 0 \\ \mathbf{0} & \mathbf{0} & 0 & 1 & 0 & 1 & 1 \end{pmatrix}. \tag{6.2}$$

The Art of Error Correcting Coding, Second Edition Robert H. Morelos-Zaragoza
 ISBN: 0-470-01558-6

To construct a shortened $(5, 2, 3)$ *code, any two among the four leftmost columns of* G *can be removed. Suppose that the first and second columns of* G *are removed, and that the first and second rows, corresponding to the nonzero elements of the columns, are removed. These rows and columns are indicated by boldface types in (6.2). The remaining entries of* G *form the matrix*

$$G_s = \begin{pmatrix} 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 \end{pmatrix}.$$

It is instructive to examine carefully the standard array of the shortened (5,2,3) code of Example 6.1.1 above, in order to understand the enhancement in the error-correcting capability of a shortened code with respect to the original code.

Example 6.1.2 *The standard array for the shortened* $(5, 2, 3)$ *code of Example 6.1.1 is given in Table 6.1.*

From the standard array, it follows that, although the minimum distance of the code is $d_s = 3$*, there are two error patterns of Hamming weight two, namely* 11000 *and* 01100*, that can be corrected.*

Note that shortening a code reduces the length and dimension of the code, at the same time maintains the same redundancy, so that more error patterns can be corrected. This can be explained from the Hamming bound for a t-error-correcting linear block (n, k, d) code, the inequality (1.26) of which is repeated here for convenience,

$$2^{n-k} \geq \sum_{\ell=0}^{t} \binom{n}{\ell}. \tag{6.3}$$

With respect to the original (n, k, d) code, the shortened $(n - s, k - s, d_s)$ has the same redundancy. Therefore, the left-hand side of (6.3) has the same value. In other words, *the number of cosets does not change.* On the other hand, for $s > 0$, the right-hand side becomes smaller. In other words, *the number of error patterns of Hamming weight up to* t *decreases.*

If the original code does not meet the Hamming bound (6.3) with equality (in the binary case, only the Hamming codes, the Golay code, the single parity-check (SPC) codes and the repetition codes do), then the quantity $2^{n-k} - \sum_{i=0}^{t} \binom{n}{i}$ represents the additional patterns

Table 6.1 Standard array for a shortened $(5, 2, 3)$ code.

$\bar{s}$	$\bar{u} = 00$	$\bar{u} = 10$	$\bar{u} = 01$	$\bar{u} = 11$
000	00000	10110	01011	11101
110	10000	00110	11011	01101
011	01000	11110	00011	10101
100	00100	10010	01111	11001
010	00010	10100	01001	11111
001	00001	10111	01010	11100
101	11000	01110	10011	00101
111	01100	11010	00111	10001

of weight greater than t that the original code can correct. The number of additional error patterns of Hamming weight greater than t that the shortened code can correct is given by

$$\Delta_t = 2^{n-k} - \sum_{\ell=0}^{t} \binom{n-s}{\ell} - \left[2^{n-k} - \sum_{\ell=0}^{t} \binom{n}{\ell} \right] = \sum_{\ell=0}^{t} \left[\binom{n}{\ell} - \binom{n-s}{\ell} \right], \tag{6.4}$$

which is equal to the difference in the volume of Hamming spheres of radius t when going from n to $n - s$ coordinates (or dimensions).

6.1.2 Extending

Extending a code C is achieved by adding ϵ parity-check symbols. Given a binary linear (n, k, d) code C, the extended $(n + \epsilon, k, d_{\text{ext}})$ code C_{ext} has minimum distance $d_{\text{ext}} \geq d$. In terms of the parity-check matrix H of code C, ϵ rows and columns are added to it, to obtain an $(n - k + \epsilon) \times (n + \epsilon)$ extended parity-check matrix,

$$H_{\text{ext}} = \begin{pmatrix} h_{1,1} & \cdots & h_{1,\epsilon} & \cdots & h_{1,n+\epsilon} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ h_{\epsilon,1} & \cdots & h_{\epsilon,\epsilon} & \cdots & h_{\epsilon,n+\epsilon} \\ & & & & \\ \vdots & \ddots & \vdots & & H \\ & & & & \\ h_{n-k+\epsilon,1} & \cdots & h_{n-k+\epsilon,\epsilon} & & \end{pmatrix}. \tag{6.5}$$

The most common way of extending a code is by adding one overall parity-check symbol. In this case, the following parity-check matrix is obtained,

$$H_{\text{ext}} = \begin{pmatrix} 1 & 1 & \cdots & 1 \\ 0 & & & \\ 0 & & H & \\ 0 & & & \end{pmatrix}. \tag{6.6}$$

The resulting code C_{ext} is an $(n + 1, k, d_{\text{ext}})$ code. If the minimum distance of the original code is odd, then $d_{\text{ext}} = d + 1$.

Example 6.1.3 *Let C be the Hamming (7,4,3) code. Then the extended (8,4,4) code C_{ext} has the parity-check matrix*

$$H_{\text{ext}} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{pmatrix}. \tag{6.7}$$

After permutation of columns, this is also the generator matrix of the $RM(1, 3)$ code of Example 2.3.2.

6.1.3 Puncturing

A puncturing technique was discussed in Section 5.6, in connection with convolutional codes. More generally, puncturing of linear block codes consists of the removal of parity-check symbols, to obtain a linear block $(n-p, k, d_p)$ code C_p with the minimum distance $d_p \leq d$. The rate of the code increases since the dimension (or number of information symbols) does not change while the redundancy (or number of parity-check symbols) is reduced.

Puncturing is achieved by removing certain columns of the parity-check matrix H of the original code, to obtain matrix H_p of C_p. This technique is similar to shortening of the dual code. If

$$H = \begin{pmatrix} \bar{p}_0 & \bar{p}_2 & \dots & \bar{p}_{k-1} | I_{n-k} \end{pmatrix},$$

denotes the parity-check matrix of C, then removing any p columns among the $n-k$ rightmost columns from H, and p rows corresponding to the nonzero values of the selected columns, results in an $(n-k-p) \times (n-p)$ matrix

$$H_p = \begin{pmatrix} \bar{p}'_0 & \bar{p}'_1 & \dots & \bar{p}'_{k-1} | \bar{I}_{j_0} & \bar{I}_{j_1} & \dots & \bar{I}_{j_{k-p-1}} \end{pmatrix},$$

where $\bar{I}_j$ denotes a column of weight equal to one, and $\bar{p}'_j$ denotes a column after the removal of the p rows described above.

Example 6.1.4 *Consider the* $(5, 2, 3)$ *code* C_s *from Example 6.1.1. Its parity-check matrix is equal to*

$$H_s = \begin{pmatrix} \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{0} \\ 1 & 1 & \mathbf{0} & 1 & 0 \\ 0 & 1 & \mathbf{0} & 0 & 1 \end{pmatrix}.$$

Deleting the third column and the top row of H_s *gives the parity-check matrix* H_p *of a punctured* $(4, 2, 2)$ *code* C_p*,*

$$H_p = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix},$$

which is the same LUEP code as in Examples 1.3.1 and 1.3.4, up to a permutation of the first and second code positions.

6.1.4 Augmenting, expurgating and lengthening

There are two additional techniques to modify an existing linear code that in general may produce nonlinear codes. These techniques are *augmenting* and *expurgating* and are very useful in analyzing families of linear block codes, such as the Reed–Muller and Bose–Chauduri–Hocquenghem (BCH) codes.

Augmenting a code means adding more codewords to it. An obvious way to do this and still get a linear code is to add more (linearly independent) rows to the generator matrix. This is equivalent to forming a *supercode* by taking the union of *cosets* of code C. The resulting code C_{aug} has dimension $(n, k+\delta, d_{\text{aug}})$, where δ is the number of rows added to G. The minimum distance of C_{aug} is $d_{\text{aug}} \leq d$. Let $G = \begin{pmatrix} \bar{g}_0^\top & \dots & \bar{g}_{k-1}^\top \end{pmatrix}^\top$ be a

generator matrix of C, where $\bar{g}_j$ denotes a vector of length n, $0 \leq j < n$, and $\bar{x}^\top$ denotes the *transpose* of a vector $\bar{x}$. Then a generator matrix of an augmented code is given by

$$G_{\text{aug}} = \begin{pmatrix} \bar{p}_0 \\ \vdots \\ \bar{p}_{\delta-1} \\ \hline \\ G \end{pmatrix} \tag{6.8}$$

Augmenting techniques are closely related to *coset decomposition* (Forney 1988) of codes. This is discussed below in connection with techniques related to the direct sum of codes, including the $|u|u+v|$-*construction*. A common way of augmenting a linear block (n, k, d) code C is to add to it the all-one codeword if it is not already a codeword. Alternatively, if the all-one codeword belongs to C, then C can be decomposed as

$$C = C_0 \bigcup \{\bar{1} + C_0\},$$

for an $(n, k-1, d_0)$ subcode $C_0 \subset C$, where $d_0 = d - 1$.

Expurgating is the process of removing codewords from a code. In general, this will produce a nonlinear code. One way to get a linear code is to remove rows from the generator matrix G of the original code. This results in an $(n, k-\ell, d_{\text{exp}})$ code C_{exp} with $d_{\text{exp}} \geq d$. It is interesting to note that if the code is systematic, then expurgating is equivalent to shortening.

Example 6.1.5 *Consider the Hamming* $(7, 4, 3)$ *code, with*

$$G = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{pmatrix}. \tag{6.9}$$

Removing the first row results in a $(7, 3, 3)$ *code, for which the first bit is always zero. In other words, the code is equivalent to a* $(6, 3, 3)$ *code.*

It is important to note that when expurgating a code, the choice of the generator matrix, or the *basis of the original code*, is important in order to obtain codes with high minimum distance. This is illustrated in the following example.

Example 6.1.6 *As shown in Example 3.6.1, the Hamming* $(7, 4, 3)$ *code contains seven codewords of weight 3 and seven codewords of weight 4. Selecting four linearly independent codewords, three of weight 4 and one of weight 3, the following generator matrix is obtained,*

$$G' = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \end{pmatrix}. \tag{6.10}$$

Matrix G' *generates the same Hamming* $(7, 4, 3)$ *of the previous example. However, if the top row of* G' *is removed, the generator matrix of a maximum-length-sequence (MLS) (or simplex)* $(7, 3, 4)$ *code is obtained.*

Table 6.2 Summary of basic techniques of modifying a code.

Technique	Action	Code Parameters
Shortening	Removing information symbols	$(n-\ell, k-\ell, d_s \geq d)$
Lengthening	Adding information/parity-check symbols	$(n+\ell, k+\ell, d_s \leq d)$
Extending	Adding parity-check symbols	$(n+\ell, k, d_{\text{ext}} \geq d)$
Puncturing	Removing parity-check symbols	$(n-\ell, k, d_p \leq d)$
Augmenting	Adding codewords (cosets)	$(n, k+\ell, d_{\text{aug}} \leq d)$
Expurgating	Removing codewords (cosets)	$(n, k-\ell, d_{\text{exp}} \geq d)$

The code *lengthening* technique is performed by adding ℓ columns and ℓ rows to the generator matrix, and ℓ additional parity-check symbols, so that ℓ additional information symbols are introduced. In particular, one way to lengthen a code is to add one information symbol and one overall parity-check symbol (MacWilliams and Sloane 1977).

Table 6.2 summarizes the techniques presented in this section.

6.2 Combining codes

In this section, several methods of combining codes are presented. Code combining techniques are very powerful, as evidenced by the appearance in 1993 of *turbo codes* (Berrou *et al.* 1993). In the following text, unless otherwise specified, let C_i denote a linear block (n_i, k_i, d_i) code, $i = 1, 2$.

6.2.1 Time sharing of codes

Consider two codes C_1 and C_2. Then the *time-sharing* of C_1 and C_2 consists of transmitting a codeword $\bar{c}_1 \in C_1$ followed by a codeword $\bar{c}_2 \in C_2$,

$$|C_1|C_2| = \{(\bar{c}_1, \bar{c}_2) : \bar{c}_i \in C_i,\ i = 1, 2\}. \tag{6.11}$$

The result of time sharing of m linear block (n_i, k_i, d_i) codes, $i = 1, 2, \ldots, m$, is an (n, k, d) code, with parameters

$$n = \sum_{i=1}^{m} n_i, \quad k = \sum_{i=1}^{m} k_i, \quad \text{and} \quad d = \min_{1 \leq i \leq m} \{d_i\}. \tag{6.12}$$

Let G_i denote the generator matrix of *component code* C_i, for $i = 1, 2, \ldots, m$. Then the generator matrix of the code obtained from time sharing is

$$G_{\text{TS}} = \begin{pmatrix} G_1 & & & \\ & G_2 & & \\ & & \ddots & \\ & & & G_m \end{pmatrix}, \tag{6.13}$$

where the blank entries represent zeros.

Time sharing is sometimes referred to as "direct sum" (MacWilliams and Sloane 1977) or "concatenation" (Robertson 1994). In this book, however, concatenation of codes has a different interpretation, and is discussed in Section 6.2.5.

Example 6.2.1 *Let C_1 be a repetition* $(4, 1, 4)$ *code and C_2 be a Hamming* $(7, 4, 3)$ *code. Then time sharing of C_1 and C_2 results in a linear block* $(11, 5, 3)$ *code with generator matrix*

$$G_{\mathrm{TS}} = \begin{pmatrix} G_1 & 0 \\ 0 & G_2 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{pmatrix}.$$

The time-sharing technique has been widely used in communication systems that require a variable amount of error protection, or *unequal error protection* (UEP), using the rate-compatible punctured convolution (RCPC) codes of Section 5.6.2. See Hagenauer (1988). Also note that time sharing m times the same code is equivalent to repeating the transmission of a codeword m times. This is discussed more in detail when product codes are discussed, in Section 6.2.4.

6.2.2 Direct sums of codes

Let C_i denote a linear block (n, k_i, d_i) code, $1 \le i \le m$. A direct-sum code C_{DS} is defined as

$$C_{\mathrm{DS}} = \{\bar{v} | \bar{v} = \bar{v}_1 + \bar{v}_1 + \cdots + \bar{v}_m, \quad \bar{v}_i \in C_i, \quad i = 1, 2, \ldots, m\}.$$

This technique may increase dimension. However, it generally reduces the distance. Let G_i denote the generator matrix of component code C_i, for $i = 1, 2, \ldots, m$. Then the generator matrix of the code C_{DS}, obtained from the direct sum of these component codes, denoted as $C_{\mathrm{DS}} = C_1 + C_2 + \cdots + C_m$, is

$$G_{\mathrm{DS}} = \begin{pmatrix} G_1 \\ G_2 \\ \vdots \\ G_m \end{pmatrix}. \tag{6.14}$$

Code C_{DS} is a linear block (n, k, d) code with $k \le k_1 + k_2 + \cdots + k_m$ and $d \le \min_i\{d_i\}$.

Example 6.2.2 *Let C_1 be the repetition* $(4, 1, 4)$ *code and C_2 be a linear block* $(4, 2, 2)$ *code with generator matrix*

$$G_2 = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix}.$$

(This code consists of codewords formed by sending twice a 2-bit message.) Then code $C_{\mathrm{DS}} = C_1 + C_2$ is an SPC $(4, 3, 2)$ *code, with generator matrix,*

$$G_{\mathrm{DS}} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix}.$$

The direct-sum technique can be used not only to combine codes of smaller dimension, but also to *decompose* a code into a *union of subcodes* $C_i \subset C$, such that C can be expressed as the direct sum of the component subcodes. This is discussed in more detail in Section 6.2.5.

Trivially, every linear block (n, k, d) code C with generator matrix G can be decomposed into k linear block $(n, 1, d_i)$ subcodes C_i, $1 \leq i \leq k$. Each subcode has a generator matrix equal to a row $\bar{g}_i$, of G, $0 \leq i < k$. However, there are known construction techniques that allow decomposition of codes into subcodes of larger dimension and known minimum distances. This is the topic of the next section.

6.2.3 The $|u|u + v|$-construction and related techniques

Combining time sharing and direct sum gives interesting construction methods. The first one is the $|u|u + v|$-*construction* (Plotkin 1960), defined as follows. On the basis of two codes, C_1 and C_2, of length $n = n_1 = n_2$, the $|u|u + v|$-construction gives a code[1] $C \triangleq |C_1|C_1 + C_2|$ with generator matrix

$$G = \begin{pmatrix} G_1 & G_1 \\ 0 & G_2 \end{pmatrix}. \tag{6.15}$$

The parameters of C are $(2n, k_1 + k_2, d)$. The minimum distance of C is $d = \min\{2d_1, d_2\}$, which follows from the fact that, for two binary vectors $\bar{x}$ and $\bar{y}$,

$$\mathrm{wt}(\bar{x} + \bar{y}) \leq \mathrm{wt}(\bar{x}) - \mathrm{wt}(\bar{y}).$$

The $|u|u + v|$-construction allows to express RM codes in a convenient way.

Reed–Muller codes

$$\mathrm{RM}(r + 1, m + 1) = |\mathrm{RM}(r + 1, m)|\mathrm{RM}(r + 1, m) + \mathrm{RM}(r, m)|. \tag{6.16}$$

This interpretation of RM codes has been used to develop efficient soft-decision decoding methods, taking advantage of the "recursive" structure of this class of codes (see Forney (1988), Schnabl and Bossert (1995) and references therein). In Section 6.2.5, this construction will be shown to be a particular case of a more powerful combining technique. The $|u|u + v|$-construction is also called the *squaring construction* in Forney (1988).

Example 6.2.3 *Let* C_1 *be* RM(1, 2)*, an SPC (4,3,2) code, and* C_2 *be* RM(0, 2)*, the repetition* $(4, 1, 4)$ *code. Then* $C = |C_1|C_1 + C_2|$ *is* RM(1, 3)*, an extended Hamming* $(8, 4, 4)$ *code with generator matrix*

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

[1] If the lengths are not equal, say, $n_1 > n_2$, then append $n_2 - n_1$ zeros to the end of codewords of code C_2.

Construction X

This is a generalization of the $|u|u+v|$-construction (Sloane *et al.* 1972). Let C_i denote a linear block (n_i, k_i, d_i) code, for $i = 1, 2, 3$. Assume that C_3 is a subcode of C_2, so that $n_3 = n_2$, $k_3 \leq k_2$ and $d_3 \geq d_2$. Assume also that the dimension of C_1 is $k_1 = k_2 - k_3$. Let $\left(G_2^\top \quad G_3^\top\right)^\top$ and G_3 be the generator matrices of code $C_2 \supset C_3$ and subcode C_3, respectively. Note that G_2 is a *set of coset representatives* of C_3 in C_2 (Forney 1988). Then the code C_X with generator matrix

$$G_X = \begin{pmatrix} G_1 & G_2 \\ 0 & G_3 \end{pmatrix} \tag{6.17}$$

is a linear block $(n_1 + n_2, k_1 + k_2, d_X)$ code with $d_x = \min\{d_3, d_1 + d_2\}$.

Example 6.2.4 *Let C_1 be an SPC $(3, 2, 2)$ code, and C_2 be an SPC $(4, 3, 2)$ code whose subcode is C_3, a repetition $(4, 1, 4)$ code. Then*

$$G_1 = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix}, \quad \begin{pmatrix} G_2 \\ G_3 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix},$$

and $G_3 = \begin{pmatrix} 1 & 1 & 1 & 1 \end{pmatrix}$. Construction X results in code $C_X = |C_1|C_2 + C_3|$ with generator matrix

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

and is an MLS $(7, 3, 4)$ code. This code is equivalent to the code obtained from the Hamming $(7, 4, 3)$ code by expurgating one message symbol, as in Example 6.1.6.

Construction X3

Extending further the idea of using coset representatives of subcodes in a code, this method combines three codes, one of them with two levels of *coset decomposition* into subcodes, as follows (Sloane *et al.* 1972). Let C_3 be a linear block (n_1, k_3, d_3) code, where

$$k_3 = k_2 + a_{23} = k_1 + a_{12} + a_{23}.$$

C_3 is constructed as the union of $2^{a_{23}}$ disjoint cosets of a linear block (n_1, k_2, d_2) code, C_2, with $k_2 = k_1 + a_{12}$. In turn, C_2 is the union of $2^{a_{12}}$ disjoint cosets of a linear block (n_1, k_1, d_1) code, C_1. Then each codeword in C_3 can be written as $\bar{x}_i + \bar{y}_i + \bar{v}$, with $\bar{v} \in C_1$, where $\bar{x}_i$ is a coset representative of C_2 in C_3 and $\bar{y}_i$ is a coset representative of C_1 in C_2. Let C_4 and C_5 be two linear block (n_4, a_{23}, d_4) and (n_5, a_{12}, d_5) codes, respectively. The linear block $(n_1 + n_4 + n_5, k_3, d_{X3})$ code C_{X3} is defined as

$$C_{X3} \stackrel{\Delta}{=} \{|\bar{x}_i + \bar{y}_i + \bar{v}|\bar{w}|\bar{z}| : \bar{x}_i + \bar{y}_i + \bar{v} \in C_3, \quad \bar{w} \in C_4, \quad \text{and} \quad \bar{z} \in C_5\},$$

and has a minimum distance $d_{X3} = \min\{d_1, d_2 + d_4, d_3 + d_5\}$. A generator matrix of C_{X3} is

$$G_{X3} = \begin{pmatrix} G_1 & 0 & 0 \\ G_2 & G_4 & 0 \\ G_3 & 0 & G_5 \end{pmatrix},$$

where (G_1), $\left(G_1^\top \quad G_2^\top\right)^\top$ and $\left(G_1^\top \quad G_2^\top \quad G_3^\top\right)^\top$ are the generator matrices of codes C_1, C_2 and C_3, respectively.

Example 6.2.5 *Let C_1, C_2 and C_3 be* $(64, 30, 14)$, $(64, 36, 12)$ *and* $(64, 39, 10)$ *extended BCH codes, respectively, and let C_4 and C_5 be* $(7, 6, 2)$ *and* $(7, 3, 4)$ *SPC and maximum-length codes, respectively. Construction X3 results in a* $(78, 39, 14)$ *code. This code has higher rate (four more information bits) than a shortened* $(78, 35, 14)$ *code obtained from the extended BCH* $(128, 85, 14)$ *code.*

Generalizations of constructions X and X3 and their use in designing good families of codes are presented in Fossorier and Lin (1997a), Kasahara *et al.* (1975), MacWilliams and Sloane (1977), Sloane *et al.* (1972), Sugiyama *et al.* (1978). The application of these techniques to construct LUEP codes was considered in Morelos-Zaragoza and Imai (1998), van Gils (1983).

6.2.4 Products of codes

In this section, the important method of code combination known as product, is presented. The simplest method to combine codes is by serial connection. That is, the output of a first encoder is taken as the input of a second encoder, and so on. This is illustrated for two encoders, in Figure 6.1. This is a straightforward method to form a *product code*. Although simple, this *direct-product* method produces very good codes. Very low-rate convolutional codes can be constructed by taking products of binary convolutional codes and block repetition codes.

Example 6.2.6 *Consider the de facto standard memory-6 rate-1/2 convolutional encoder with generators* $(171, 133)$ *and free distance $d_f = 10$. The output of this encoder is combined serially with time sharing of repetition* $(2, 1, 2)$ *and* $(3, 1, 3)$ *codes, namely* $|(2, 1, 2)|(2, 1, 2)|$ *and* $|(3, 1, 3)|(3, 1, 3)|$. *In other words, every coded bit is repeated two or three times, respectively.*

These two schemes produce two codes: a binary memory-6 rate-1/4 convolutional code and a binary memory-6 rate-1/6 convolutional code, with generators $(171, 171, 133, 133)$ *and* $(171, 171, 171, 133, 133, 133)$, *and free distances $d_f = 20$ and $d_f = 30$, respectively. These codes are optimal (Dholakia 1994) in the sense that they have the largest free distance for a given number of states. This seems to be the first time that they have been expressed in terms of these generators.*

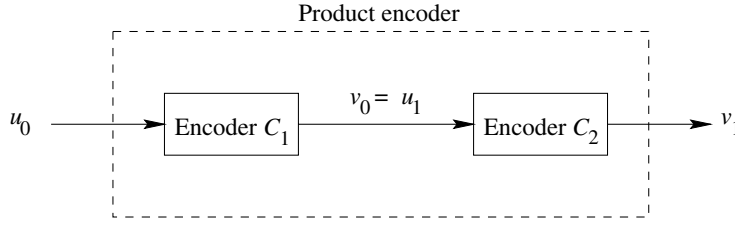


Figure 6.1 Block diagram of an encoder of a product code.

However, except for the case of two encoders where the second encoder is the time sharing of repetition codes, important questions arise when considering a serial connection[2] between two encoders: how is the output of the first encoder fed into the second encoder? In the following text, let C_1 denote the *outer code* and C_2 denote the *inner code*. Either C_1, or C_2 or both can be convolutional or block codes. If G_1 and G_2 are the generator matrices of the component codes, then the generator matrix of the product code is the Kronecker product, $G = G_1 \otimes G_2$.

In 1954, Elias (1954) introduced product (or iterated) codes. The main idea is as follows. Assume that both C_1 and C_2 are systematic. The codewords of the *inner code* C_1 are arranged as rows of a rectangular array with n_1 columns, one per code symbol in a codeword of C_1. After k_2 rows have been filled, the remaining $n_2 - k_2$ rows are filled with redundant symbols produced, on a column-by-column basis, by the *outer code* C_2. The resulting $n_2 \times n_1$ rectangular array is a codeword of the *product code* $C_P \triangleq C_1 \otimes C_2$. Figure 6.2 depicts the structure of a codeword of a two-dimensional product code. Extension to higher dimensions is straightforward.

The array codewords are transmitted on a column-by-column basis. With reference to the initial description of a product code (Figure 6.1), Elias' two-dimensional product codes can be interpreted as connecting the two encoders serially, with an *interleaver* in between. This is shown schematically in Figure 6.3.

As defined by Ramsey (1970), an interleaver is *a device that rearranges the ordering of a sequence of symbols in a one-to-one deterministic manner*. For the product of linear block codes, naturally, the device is known as a *block interleaver*. The interleaver describes a mapping $m_b(i, j)$ between the elements $a_{i,j}$ in a $k_2 \times n_1$ array, formed by placing k_2 codewords of C_1 as rows, and the elements $u_{m_b(i,j)}$ of an *information vector* $\bar{u} = (u_0 \quad u_1 \quad \ldots \quad u_{n_1 n_2 - 1})$.

The one-to-one and onto mapping induced by a $m_1 \times m_2$ block interleaver can also be expressed as a *permutation* $\Pi : i \mapsto \pi(i)$, acting on the set of integers modulo $m_1 m_2$.

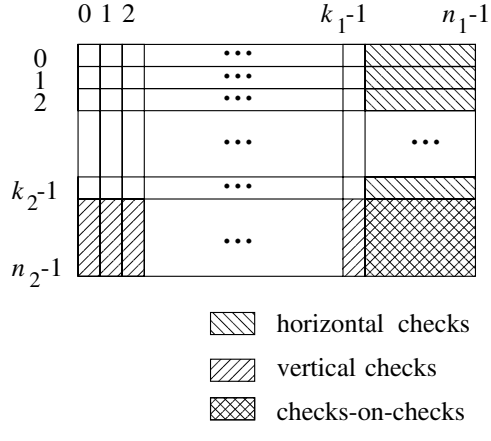


Figure 6.2 Codeword of a two-dimensional product code.

[2]This is also known in the literature as *serial concatenation*. However, in this chapter the term *concatenation* is used with a different meaning.

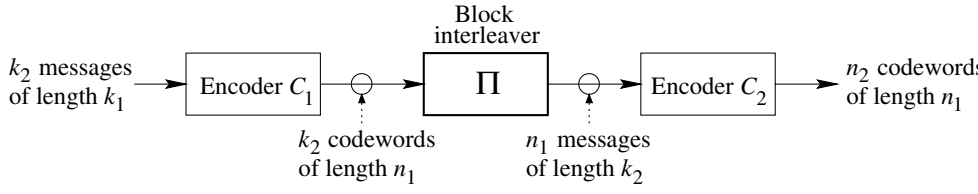


Figure 6.3 A two-dimensional product encoder with a block interleaver.

Writing the array as a one-dimensional vector, $\bar{u}$, by time sharing of (in that order) the first to the m_1-th row of the array,

$$\bar{u} = \begin{pmatrix} u_0 & u_1 & \dots & u_{m_1 m_2 - 1} \end{pmatrix},$$

the output of the block interleaver is read, via Π, as

$$\bar{u}_\pi \stackrel{\Delta}{=} \begin{pmatrix} u_{\pi(0)} & u_{\pi(1)} & \dots & u_{\pi(m_1 m_2 - 1)} \end{pmatrix}, \tag{6.18}$$

where

$$\pi(i) = m_2(i \bmod m_1) + \left\lfloor \frac{i}{m_1} \right\rfloor. \tag{6.19}$$

Example 6.2.7 *Let C_1 and C_2 be linear block SPC* $(5, 4, 2)$ *and* $(3, 2, 2)$ *codes, respectively. This results in a* $(15, 8, 4)$ *product code with the block interleaver shown in Figure 6.4. The permutation is given by*

$$\pi(i) = 5(i \bmod 2) + \left\lfloor \frac{i}{2} \right\rfloor,$$

and the vector $\bar{u} = (u_0, u_1, u_2, u_3, \dots, u_9)$ is mapped onto

$$\bar{u}_\pi = \begin{pmatrix} \begin{pmatrix} u_0 & u_5 \end{pmatrix} & \begin{pmatrix} u_1 & u_6 \end{pmatrix} & \dots & \begin{pmatrix} u_4 & u_9 \end{pmatrix} \end{pmatrix} = \begin{pmatrix} \bar{u}_0 & \bar{u}_1 & \dots & \bar{u}_4 \end{pmatrix}.$$

This is illustrated in Figure 6.5. The subvectors $\bar{u}_i = \begin{pmatrix} u_i u_{i+5} \end{pmatrix}$, $0 \le i < 5$, constitute information vectors to be encoded by C_1. Codewords are interpreted as two-dimensional arrays,

$$\bar{v} = \begin{pmatrix} a_{0,0} & a_{1,0} & a_{2,0} & a_{0,1} & a_{1,1} & \dots & a_{2,4} \end{pmatrix},$$

where the rows

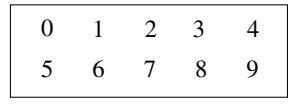
$$\begin{pmatrix} a_{\ell,0} & a_{\ell,1} & \dots & a_{\ell,4} \end{pmatrix} \in C_1, \ell = 0, 1, 2,$$

and the columns

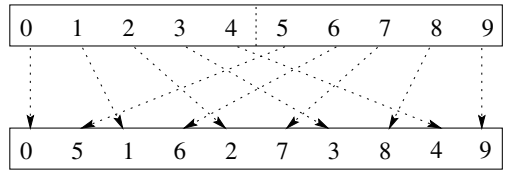
$$\begin{pmatrix} a_{0,\ell} & a_{1,\ell} & a_{2,\ell} \end{pmatrix} \in C_2, \ell = 0, 1, \dots 4.$$

	$j=0$	$j=1$	$j=2$	$j=3$	$j=4$
$i=0$	0	2	4	6	8
$i=1$	1	3	5	7	9

Figure 6.4 A 2-by-5 block interleaver.



(a)



(b)

Figure 6.5 (a) Codewords in C_1 as rows; (b) equivalent vector $\bar{u}$ and its permutation $\bar{u}_\pi$.

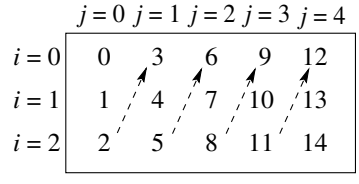


Figure 6.6 Mapping $m_b(i, j)$ of a 3-by-5 block interleaver.

The underlying ordering is depicted in Figure 6.6. The one-dimensional notation gives the same vector,

$$\bar{v} = \big((\bar{u}_0, v_0) \quad (\bar{u}_1, v_1) \quad \ldots \quad (\bar{u}_4, v_4)\big),$$

where $(\bar{u}_i, v_i) \in C_2$.

Example 6.2.8 *Let* C_1 *and* C_2 *be two binary SPC* $(3, 2, 2)$ *codes. Then* C_P *is a* $(9, 4, 4)$ *code. Although this code has one more redundant bit than an extended Hamming code (or the RM(1,3) code), it can correct errors very easily by simply checking the overall parity of the received rows and columns. Let the all-zero codeword be transmitted over a binary symmetric channel (BSC) and suppose that the received codeword is*

$$\bar{r} = \begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}.$$

Recall that the syndrome of a binary SPC $(n, n-1, 2)$ *code is simply the sum of the n bits. The second row and the first column will have nonzero syndromes, indicating the presence of an odd number of errors. Moreover, since the other columns and rows have syndromes equal to zero, it is concluded correctly that a single error must have occurred in the first bit of the*

second row (or the second bit of the first column). Decoding finishes upon complementing the bit in the located error position.

The code in Example 6.2.8 above is a member of a family of codes known as *array codes* (see, e.g., (Blaum 1990; Kasahara *et al.* 1976)). Being product codes, array codes are able to correct bursts of errors, in addition to single errors. Array codes have nice trellis structures (Honay and Markarian 1996), and are related to *generalized concatenated (GC) codes* (Honary *et al.* 1993), which are the topic of Section 6.2.5.

Let C_i be a linear block (n_i, k_i, d_i) code, $i = 1, 2$. Then the product $C_P = C_1 \otimes C_2$ is a linear block $(n_1 n_2, k_1 k_2, d_P)$ code, where $d_P = d_1 d_2$. In addition, C_P can correct all *bursts of errors* of length up to $b = \max\{n_1 t_2, n_2 t_1\}$, where $t_i = \lfloor (d_i - 1)/2 \rfloor$, for $i = 1, 2$. The parameter b is called the *burst error-correcting capability*.

Example 6.2.9 *Let C_1 and C_2 be two Hamming $(7, 4, 3)$ codes. Then C_P is a $(49, 16, 9)$ code that is capable of correcting up to 4 random errors and bursts of up to 7 errors.*

If the component codes are cyclic, then the product code is cyclic (Burton and Weldon 1965). More precisely, let C_i be a cyclic (n_i, k_i, d_i) code with generator polynomial $\bar{g}_i(x)$, $i = 1, 2$. Then the code $C_P = C_1 \otimes C_2$ is cyclic if the following conditions are satisfied:

1. The lengths of the codes C_i are relatively prime, that is, $an_1 + bn_2 = 1$, for two integers a and b;

2. The *cyclic mapping* $m_c(i, j)$ that relates the element $a_{i,j}$ in the rectangular array of Figure 6.2 with a coefficient $v_{m_c(i,j)}$ of a code polynomial $\bar{v}(x) = v_0 + v_1 + \cdots + v_{n_1 n_2 - 1} x^{n_1 n_2 - 1} \in C_P$, is such that

$$m_c(i, j) = \left[(j - i) \cdot bn_1 + i \right] \bmod n_1 n_2, \tag{6.20}$$

for $m_c(i, j) = 0, 1, \ldots, n_1 n_2 - 1$.

When these two conditions are satisfied, the generator polynomial of the cyclic code C_P is given by

$$\bar{g}(x) = \mathrm{GCD}\left(\bar{g}_1(x^{bn_2}) \bar{g}_2(x^{an_1}), x^{n_1 n_2} + 1 \right). \tag{6.21}$$

Example 6.2.10 *An example of the cyclic mapping for $n_1 = 5$ and $n_2 = 3$ is shown in Figure 6.7. In this case, $(-1)5 + (2)3 = 1$, so that $a = -1$ and $b = 2$. Consequently, the mapping is given by*

$$m_c(i, j) = (6j - 5i) \bmod 15.$$

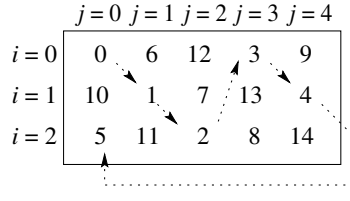


Figure 6.7 Cyclic mapping m_c for $n_1 = 5$, $n_2 = 3$.

As a check, if $i = 1$ *and* $j = 2$*, then* $m_c(1, 2) = (12 - 5) \bmod 15 = 7$*; if* $i = 2$ *and* $j = 1$*, then* $m_c(2, 1) = (6 - 10) \bmod 15 = -4 \bmod 15 = 11$.

The mapping $m_c(i, j)$ indicates *the order in which the digits of the array are transmitted* (Burton and Weldon 1965). This is not the same as the column-by-column order of the block interleaver for a conventional product code. The mapping described by (6.20) is referred to as a *cyclic interleaver*. Other classes of interleavers are discussed in Section 6.2.5.

With the appearance of turbo codes (Berrou *et al.* 1993) in 1993, there has been intense research activity in novel interleaver structures that perform a pseudorandom arrangement of the codewords of C_1, prior to encoding with C_2. In the next section, interleaved codes are presented. Chapter 8 discusses classes of interleaver structures that are useful in iterative decoding techniques of product codes.

Block interleaved codes

A special case of product code is obtained when the second encoder is the trivial $(n_2, n_2, 1)$ code. In this case, codewords of C_1 are arranged as rows of an n_2-by-n_1 rectangular array and transmitted column-wise, just as in a conventional product code. The value $I = n_2$ is known as the *interleaving degree* (Lin and Costello 2005) or *interleaving depth*.

The resulting block interleaved code, henceforth denoted as $C_1^{(n_2)}$, can be decoded using the same decoding algorithm of C_1, after reassembling a received word, column by column and decoding it row by row. Figure 6.8 shows the schematic of a codeword of an interleaved code, where $\begin{pmatrix} v_{i,0} & v_{i,1} & \dots & v_{i,n_0} \end{pmatrix} \in C_1$, for $0 \leq i < n_2$.

If the error-correcting capability of C_1 is $t_1 = \lfloor (d_1 - 1)/2 \rfloor$, then $C_1^{(n_2)}$ can correct any single error burst of length up to $b = t_1 n_2$. This is illustrated in Figure 6.9. Recall that the transmission order is column by column. If a burst occurs, but it does not affect more than b_1 positions per row, then it can be corrected by C_1. The maximum length of such a burst of errors is n_2 times b_1. Moreover, if code C_1 can already correct (or detect) any single burst of length up to b_1, then $C_1^{(n_2)}$ can correct (or detect) any single burst of length up to $b_1 n_2$.

If C_1 is a cyclic code, then it follows from (6.21) that $C_1^{(n_2)}$ is a cyclic code with generator polynomial $\bar{g}_1(x^{n_2})$ (Lin and Costello 2005; Peterson and Weldon 1972). This applies to shortened cyclic codes as well, and the following result holds ((Peterson and Weldon 1972), p. 358):

> *Interleaving a shortened cyclic* (n, k) *code to degree* ℓ *produces a shortened* $(n\ell, k\ell)$ *code whose burst error-correcting capability is* ℓ *times that of the original code.*

$$\begin{array}{cccc} v_{0,0} & v_{0,1} & \cdots & v_{0,n_1\text{-}1} \\ v_{1,0} & v_{1,1} & \cdots & v_{1,n_1\text{-}1} \\ \cdots & \cdots & & \cdots \\ v_{n_2\text{-}1,0} & v_{n_2\text{-}1,1} & \cdots & v_{n_2\text{-}1,n_1\text{-}1} \end{array}$$

Figure 6.8 Codeword of a block interleaved code of degree $I = n_2$.

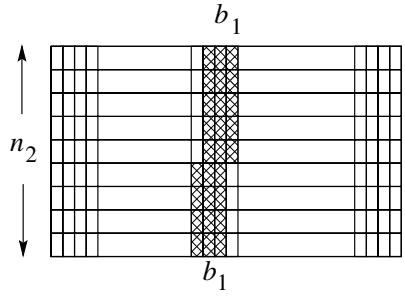


Figure 6.9 A correctable error burst in a block interleaved codeword.

Finally, note that the error-correcting capability of a product code, $t_P = \lfloor (d_1 d_2 - 1)/2 \rfloor$, can only be achieved if a carefully designed decoding method is applied.

Most of the decoding methods for product codes use a *two-stage decoding* approach. In the first stage, an errors-only algebraic decoder for the row code C_1 is used. Then *reliability weights* are assigned to the decoded symbols, based on the number of errors corrected. The more errors are corrected, the less reliable the corresponding estimated codeword $\hat{v}_1 \in C_1$ is.

In the second stage, an errors-and-erasures algebraic decoder for the column code C_2 is used, with an increasing number of erasures declared in the *least reliable positions* (those positions for which the reliability weights are the smallest), until a sufficient condition on the number of corrected errors is satisfied. This is the approach originally proposed in Reddy and Robinson (1972), Weldon (1971). The second decoding stage is usually implemented with the generalized minimum distance (GMD) algorithm, which is discussed in Section 7.6. More details on decoding of product codes can be found in Chapter 8.

6.2.5 Concatenated codes

In 1966, Forney (1966a) introduced a clever method of combining two codes, called *concatenation*. The scheme is illustrated in Figure 6.10. Concatenated codes[3] that are based on outer Reed–Solomon codes and inner convolutional codes have been[4] perhaps the most popular choice of ECC schemes for digital communications to date. In general, the outer code, denoted as C_1, is a nonbinary linear block (N, K, D) code over $GF(2^k)$. The codewords of C_1 are stored in an interleaver memory. The output bytes read from the interleaver are then passed through an encoder for an inner code, C_2. The inner code C_2 can be either a block code or a convolutional code. When block codes are considered, and C_2 is a binary linear block (n, k, d) code, the encoder structure is shown in Figure 6.10. Let $C = C_1 \star C_2$ denote the concatenated code with C_1 as the outer code and C_2 as the inner code. Then C is a binary linear block (Nn, Kk, Dd) code.

The purpose of the interleaver between the outer and the inner code is twofold. First, it serves to convert the bytes of size k into vectors of the same dimension (number of information bits) as the inner code, be it binary or nonbinary, a linear block (n, k', d)

[3] Also referred to by some authors as *cascaded* codes.

[4] Before the arrival of turbo codes and low-density parity-check (LDPC) codes.

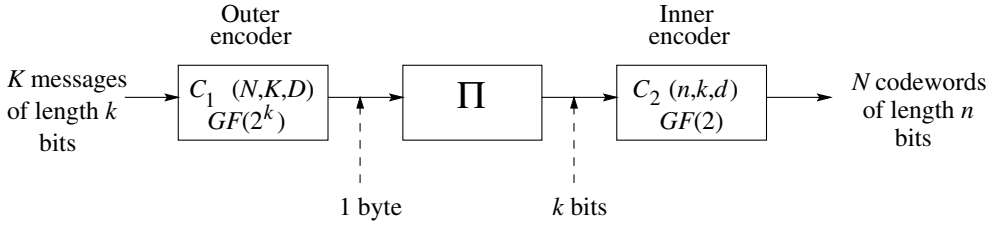


Figure 6.10 An encoder of a concatenated code.

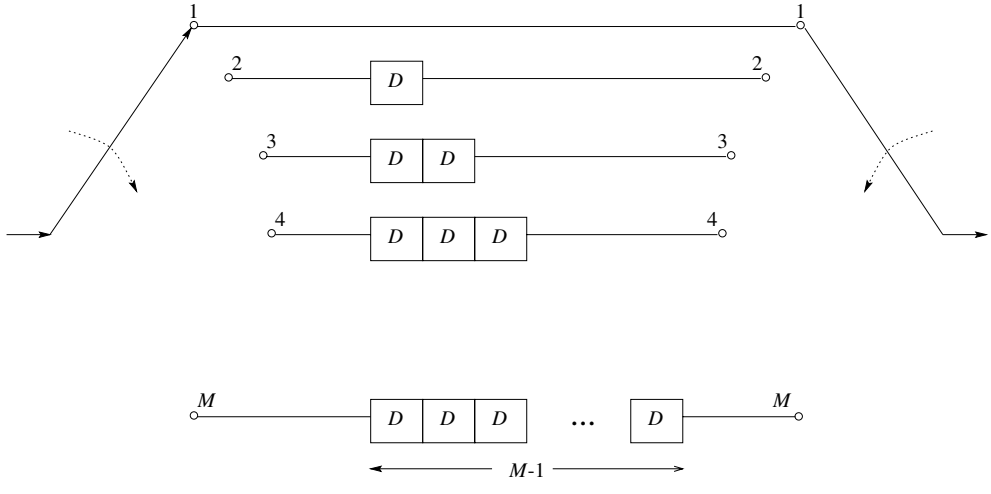


Figure 6.11 A convolutional interleaver.

code or a rate-k'/n convolutional code, for which in general $k' \neq k$. On the other hand, as discussed in the previous section, interleaving allows breaking of bursts of errors. This is useful when concatenated schemes with inner convolutional codes are considered, because the Viterbi decoder tends to produce bursts of errors (Chao and Yao 1996; Morris 1992). There are several types of interleavers that are used in practice. The most popular one appears to be the *convolutional interleaver* (Forney 1971), which is a special case of a *Ramsey interleaver* (Ramsey 1970). The basic structure of a convolutional interleaver is shown in Figure 6.11. The deinterleaver structure is identical, with the exception that the switches are initially in position M and rotate in the opposite direction.

An important advantage of concatenated codes (and product codes) is that decoding can be based on the decoding of each component code. This results in a dramatic reduction in complexity, compared to a decoder for the entire code.

Example 6.2.11 *Let C_1 be a* $(7, 5, 3)$ *RS code*[5] *with zeros* $\{1, \alpha\}$*, where* α *is a primitive element of* $GF(2^3)$*, and* $\alpha^3 + \alpha + 1 = 0$*. Let C_2 be the MLS* $(7, 3, 4)$ *code of Example 6.2.4.*

[5]RS codes are the topic of Chapter 4.

Then $C = C_1 \star C_2$ is a binary linear block $(49, 15, 12)$ *code. This code has six information bits less than a shortened* $(49, 21, 12)$ *code obtained from the extended BCH* $(64, 36, 12)$ *code. However, it is simpler to decode. Let*

$$\bar{v}(x) = (x^4 + \alpha^4)\bar{g}(x) = \alpha^5 + x + \alpha^4 x^2 + \alpha x^4 + \alpha^3 x^5 + x^6$$

be a codeword in the RS $(7, 5, 3)$ *code, where* $\bar{g}(x) = x^2 + \alpha^3 x + \alpha$.

Using the table on page 49, the elements of $GF(2^3)$ *can be expressed as vectors of 3 bits. A 3-by-7 array whose columns are the binary vector representations of the coefficients of the code polynomial* $\bar{v}(x)$ *is obtained. Then encoding by the generator polynomial of* C_2 *is applied to the columns to produce 4 additional rows of the codeword array. For clarity, the following systematic form of the generator matrix of* C_2 *is used, which is obtained after exchanging the third and sixth columns of G in Example 6.2.4,*

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

Figure 6.12 shows the codeword array corresponding to $\bar{v} \in C_1$.

6.2.6 Generalized concatenated codes

In 1974, Blokh and Zyablov (1974) and Zinov'ev (1976) introduced the powerful class of GC codes. This is a family of ECC that can correct both random errors and random bursts of errors. As the name implies, GC codes generalize Forney's concept of concatenated codes, by the introduction of a *subcode hierarchy* (or subcode partition) of the inner code C_I and several outer codes, one for each *partition level*. The GC construction combines the concepts of direct sum, or coset decomposition, and concatenation. Before defining the codes, some notation is needed.

A linear block (n, k, d) code C is said to be *decomposable* with respect to its linear block (n, k_i, d_i) subcodes C_i, $1 \leq i \leq M$, if the following conditions are satisfied:

α^5	1	α^4	0	α	α^3	1
1	0	1	0	0	0	0
1	0	1	0	1	1	0
1	1	0	0	0	1	1
0	1	1	0	0	1	1
0	1	1	0	1	0	1
0	0	0	0	1	1	0
1	1	0	0	1	0	1

Figure 6.12 A codeword in the concatenated code $C_1 \star C_2$, with C_1 the RS$(7, 5, 3)$ code over $GF(2^3)$ and C_2 a binary cyclic $(7, 3, 4)$ code.

(Sum) $C = C_1 + C_2 + \cdots + C_M$.

(D) For $\bar{v}_i \in C_i$, $1 \leq i \leq M$,

$$\bar{v}_1 + \bar{v}_2 + \cdots + \bar{v}_M = \bar{0}$$

if and only if

$$\bar{v}_1 = \bar{v}_2 = \cdots = \bar{v}_M = \bar{0}.$$

For $1 \leq i \leq M$, let C_{Ii} be a linear block (n_I, k_{Ii}) code over $GF(q)$ such that

(D_I) for $\bar{u}_i \in C_{Ii}$ with $1 \leq i \leq M$, $\bar{u}_1 + \bar{u}_2 + \cdots + \bar{u}_M = \bar{0}$, if and only if $\bar{u}_i = \bar{0}$ for $1 \leq i \leq M$.

Let δ_i denote the minimum Hamming distance of the direct-sum code

$$C_{Ii} + C_{Ii+1} + \cdots + C_{IM}.$$

Let C_{Oi} be a linear block (n_O, k_{Oi}, d_{Oi}) code over $GF(q^{k_{Ii}})$ and let $C_i^\star = C_{Oi} \star C_{Ii}$. The GC code C is defined as the direct sum

$$C \triangleq C_1^\star + C_2^\star + \cdots + C_M^\star. \tag{6.22}$$

Then, condition (D) on C follows from the condition (D_I). The minimum Hamming distance d of C is lower bounded (Takata *et al.* 1994) as

$$d \geq \min_{1 \leq i \leq M} \delta_i d_{Oi}. \tag{6.23}$$

As with (single level) concatenated codes, a main advantage of GC codes is that multistage decoding up to the distance given by the right-hand side of 6.23 is possible (Morelos-Zaragoza *et al.* 1999; Takata *et al.* 1994). Figure 6.13 shows the structure of an encoder for a GC code.

Unequal error protection

Another advantage of this class of codes is that it is relatively easy to coordinate the distances of the outer and inner codes to obtain linear block or convolutional codes with *unequal error protection* capabilities (Masnick and Wolf 1967). If the direct-sum conditions above are satisfied, and in addition, the products of the minimum distances satisfy the following inequalities,

$$\delta_1 d_{O1} \geq \delta_2 d_{O2} \geq \cdots \geq \delta_M d_{OM}, \tag{6.24}$$

then codewords in correspondence with $k_{Oi}k_{Ii}$ symbols over $GF(q)$ will have an error-correcting capability, $\lfloor (\delta_i d_{Oi} - 1)/2 \rfloor$, that decreases with the level i, for $1 \leq i \leq M$. As a result, the messages encoded in the top (low values of i) partition levels will have enhanced error-correcting capabilities, compared to those associated with the lowest partition levels. Constructions of this type are reported in Dettmar *et al.* (1995), Morelos-Zaragoza and Imai (1998).

A construction

Let a linear block (n_I, k_1, d_1) code C_1 over $GF(q)$ be *partitioned* into a chain of M (n_I, k_i, d_i) subcodes C_i, $i = 2, 3, \ldots, M+1$, such that

$$C_1 \supset C_2 \supset \cdots \supset C_{M+1},$$

where, for convenience, $C_{M+1} \triangleq \{\bar{0}\}$, and $d_{M+1} \triangleq \infty$.

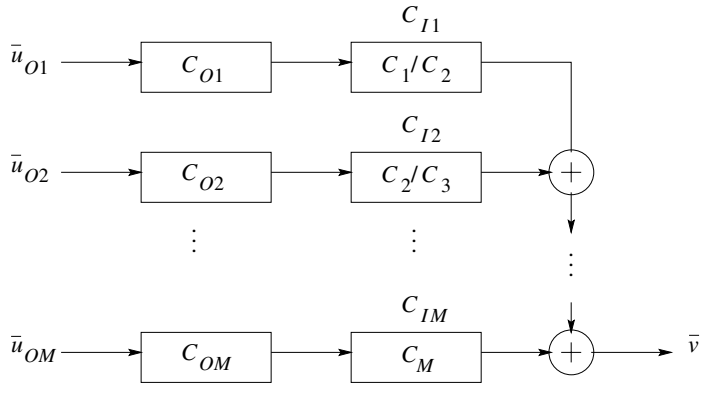


Figure 6.13 Encoder structure of a generalized concatenated code with M partition levels.

Let $C_{Ii} = [C_i/C_{i+1}]$ denote a linear block (n_I, k_{Ii}, δ_i) subcode of C_i, which is a *set of coset representatives* of C_{i+1} in C_i, of dimension $k_{Ii} = k_i - k_{i+1}$ and minimum Hamming distance $\delta_i \geq d_i$. Then C_1 has the following *coset decomposition* (Forney 1988)

$$C_1 = C_{I1} + C_{I2} + \cdots + C_{IM}. \tag{6.25}$$

Let C_{Oi} denote a linear block (n_O, k_{Oi}, d_{Oi}) code C_{Oi} over $GF(q^{k_{Ii}})$, where

$$k_{Ii} = \dim(C_i/C_{i+1}) = k_i - k_{i+1}, \quad i = 1, 2, \ldots, M.$$

Then the direct sum of concatenated codes

$$C = C_{O1} \star C_{I1} + C_{O2} \star C_{I2} + \cdots + C_{OM} \star C_{IM}$$

is an $(n_O n_I, k, d)$ linear block code of dimension and minimum Hamming distance, (Blokh and Zyablov 1974),

$$k = \sum_{i=1}^{M} k_{Ii} k_{Oi}, \qquad \text{and} \qquad d \geq \min_{1 \leq i \leq M} \{\delta_i d_{Oi}\}, respectively. \tag{6.26}$$

Note that equality holds in (6.26) when C_{Ii}, $1 \leq i \leq M$, contains the all-zero codeword.

The choice of component codes is governed by the dimensions of the coset representatives. Since, in general, the dimensions of the coset representatives are distinct, the outer codes can be selected as RS codes or shortened RS codes.

Binary RM codes are good candidates for inner codes, because they have the following *subcode property*,

$$\mathrm{RM}(r, m) \subset \mathrm{RM}(r + 1, m),$$

for $0 \leq r < m$, and $m \geq 2$.

Example 6.2.12 *Consider the r-th order RM codes of length 8,* $\mathrm{RM}(r, 3)$. *Then*

$$RM(3, 3) \supset RM(2, 3) \supset RM(1, 3) \supset RM(0, 3) \supset \{\bar{0}\},$$

and it follows from (6.25) that

$$RM(3,3) = [RM(3,3)/RM(2,3)] + [RM(2,3)/RM(1,3)] + [RM(1,3)/RM(0,3)] + RM(0,3).$$

This decomposition can also be appreciated from the generator matrix of $RM(3,3)$*, expressed as*

$$G = \begin{pmatrix} G_1 \\ \\ G_2 \\ \\ G_3 \\ \\ G_4 \end{pmatrix} = \begin{pmatrix} 0&0&0&0&0&0&0&1 \\ \\ 0&0&0&0&0&0&1&1 \\ 0&0&0&0&0&1&0&1 \\ 0&0&0&1&0&0&0&1 \\ \\ 0&0&0&0&1&1&1&1 \\ 0&0&1&1&0&0&1&1 \\ 0&1&0&1&0&1&0&1 \\ \\ 1&1&1&1&1&1&1&1 \end{pmatrix},$$

where G_i *has been defined as the generator matrix of the set of coset representatives of* $RM(3-i,3)$ *in* $RM(3-i+1,3)$*, or* $[RM(3-i+1,m)/RM(3-i,m)]$*, for* $i = 1, 2, 3$*, and where* G_4 *is the generator matrix of* $RM(0,3)$*.*

According to this partition of $RM(3,3)$*, a GC code can be designed with up to four levels. Note that the outer codes should be over* $GF(2)$*, for the first and fourth partition levels, and over* $GF(2^3)$ *for the second and third partition levels.*

Also, some of the subcodes of $RM(3,3)$ *themselves can be used as inner codes to obtain a GC code with a reduced number of levels. If* $RM(2,3)$ *is used as the inner code of a GC code, then the number of partition levels is three, as the generator matrix of* $RM(2,3)$ *is obtained from that of* $RM(3,3)$ *by removing* G_1 *(the top row).*

Let $RM(2,3)$ *be selected as the inner code, and an RS* $(8,1,8)$ *code* C_{O1} *over* $GF(2^3)$*, an RS* $(8,5,4)$ *code* C_{O2} *over* $GF(2^3)$*, and a binary linear SPC* $(8,7,2)$ *code* C_{O3} *as the outer code. This gives a binary linear GC* $(64,25,16)$ *code which has one more information bit than the binary extended BCH* $(64,24,16)$ *code.*

It is now shown how Reed–Muller codes can be expressed as GC codes. Recall from Section 6.2.2, Equation (6.16), that the $(r+1)$-th order RM code of length 2^{m+1} can be expressed as

$$\mathrm{RM}(r+1, m+1) = |\mathrm{RM}(r+1,m)|\mathrm{RM}(r+1,m) + \mathrm{RM}(r,m)|.$$

Let $G(r,m)$ denote the generator matrix of $\mathrm{RM}(r,m)$. Then

$$\begin{aligned} G(r+1,m+1) &= \begin{pmatrix} G(r+1,m) & G(r+1,m) \\ 0 & G(r,m) \end{pmatrix} \\ &= G(r+1,m)\begin{pmatrix} 1&1 \\ 0&0 \end{pmatrix} + G(r,m)\begin{pmatrix} 0&0 \\ 0&1 \end{pmatrix}. \end{aligned} \tag{6.27}$$

This expression holds for each r-th order RM code, with $r \geq 1$ and $m > 1$, so that a recursion is obtained. From (6.27) it follows that $\mathrm{RM}(r+1, m+1)$ is a GC code

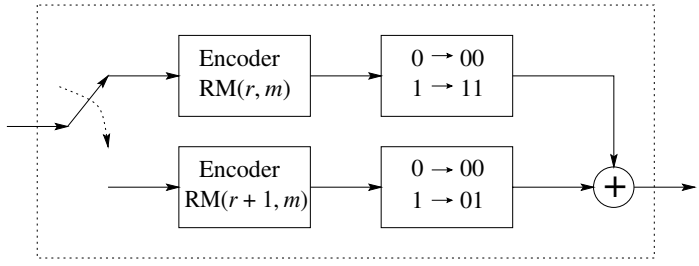


Figure 6.14 Encoder for a binary $RM(r+1, m+1)$ code as a two-level GC code.

with a $(2, 2, 1)$ code as the inner code and its partition into cosets of the repetition $(2, 1, 2)$ code, and outer codes $\mathrm{RM}(r, m)$ and $\mathrm{RM}(r+1, m)$, respectively. An encoder for $\mathrm{RM}(r+1, m+1)$ is shown in Figure 6.14. This recursive structure of RM codes is advantageous when designing soft-decision decoding algorithms for RM codes,[6] which can be based on simple repetition and parity-check codes. This is precisely what was done in (Schnabl and Bossert 1995).

Problems

1. Construct a binary linear $(17, 11, 3)$ shortened code based on a binary Hamming code. Determine the number of error patterns of Hamming weight greater than 1 that this code can correct.

2. Give an argument to support the observation that shortening a linear code may result in an LUEP code.

3. Show that if the minimum distance d of a binary linear (n, k, d) is even, then the minimum distance of the extended code – obtained by adding one overall parity-check bit – is equal to d.

4. Let C be the binary MLS (simplex) $(7, 3, 4)$ code. Determine the parity-check matrix of the extended code C_{ext} obtained from C. Show that C_{ext} is a proper subcode of the binary extended Hamming $(8, 4, 4)$ code.

5. Show that the binary Hamming $(7, 4, 3)$ is the union of cosets of the binary MLS (simplex) $(7, 3, 4)$.

6. Show that the binary linear $(11, 5, 3)$ code in Example 6.2.1 is an LUEP code capable of detecting any two errors in those codewords associated with one information bit (the bit encoded with the repetition code) and correcting any single error pattern.

7. Construct a linear UEP $(24, 12, 3)$ code capable of correcting two errors in $k_1 = 4$ information bits and correcting one error in the remaining $k_2 = 8$ information bits. (Hint: Use BCH codes of length 15, shortening and time sharing.)

[6] Soft-decision decoding algorithms are presented in the next chapter.

8. Show that the direct sum $C = C_1 \oplus C_2$, using a repetition $(2^m, 1, 2^m)$ code C_1 and an extended simplex $(2^m, m, 2^{m-1})$ code C_2, is the first-order RM code of length 2^m.

9. Find the generator matrix of the code obtained from construction X with C_1 as an SPC $(4, 3, 2)$ code and C_2 as an SPC $(8, 7, 2)$ code that contains as a proper subcode an extended Hamming $(8, 4, 4)$ code C_3.

10. Show that the generator matrix G of the direct product of two linear codes of generator matrices G_1 and G_2 is equal to the Kronecker product $G = G_1 \otimes G_2$.

11. Show that the direct-product code $C_P = C_1 \otimes C_2$ can correct all bursts of errors of length up to $b = \max\{n_1 t_2, n_2 t_1\}$, where $t_i = \lfloor (d_i - 1)/2 \rfloor$, for $i = 1, 2$.

12. Design the decoder of an interleaved cyclic code C^I with generator polynomial $\bar{g}(x^I)$ that uses the decoder of the original cyclic code C with generator polynomial $\bar{g}(x)$. Apply this result to an interleaved cyclic $(21, 12, 3)$ code based on a cyclic Hamming $(7, 4, 3)$ code interleaved to degree $I = 3$. Give a block diagram of the decoder.

13. Show that the minimum distance of a concatenated code with an outer (N, K, D) code and an inner (n, k, d) is equal to Dd.

14. Let $C = C_1 \star C_2$, where C_1 is an RS $(7, 5, 3)$ code over $GF(2^3)$ and C_2 is a binary SPC $(4, 3, 2)$ code. Then C is a binary $(28, 15, 6)$ code. This code has 2 bits less than a shortened $(28, 17, 6)$ code that is obtained from an extended BCH $(32, 21, 6)$ code. Devise a decoding algorithm for this code based on a two-stage approach, where the positions in the RS code where the overall parity-check bit fails are erased. Use an errors-and-erasures decoding algorithm for the RS code.

7

Soft-decision decoding

In this chapter, decoding with *soft information* from the channel is considered. For simplicity of exposition, binary transmission over an additive white Gaussian noise (AWGN) channel is assumed. In Chapter 9, the general case of multilevel transmission is discussed. To provide a motivation for the use of soft-decision decoding, it can be argued that the noise environment in data retrieval or signal reception is continuous in nature, not discrete. This means that the received symbols are (quantized) *real numbers* (voltages, currents, etc.), and not binary or $GF(2^m)$ symbols.

When making hard decisions on the received symbols, on a symbol-by-symbol basis, errors may be introduced. This is illustrated in Figure 7.1.

Basically, there are two methods of decoding an error correcting code, based on a received real-valued sequence.

1. *Hard-decision decoding (HDD)*:

   When hard decisions are made on the channel values, errors are introduced. The goal of HDD is to correct binary errors induced in the hard-decision process. The first part of the book was devoted to different techniques of HDD for linear block codes, cyclic codes, and convolutional codes.

2. *Soft-decision decoding (SDD)*:

   In this case, received values from the channel are directly processed by the decoder in order to estimate a code sequence. A special case of SDD is maximum-likelihood decoding (MLD), in which the *most likely* code sequence is output by the decoder. An important point to keep in mind here is that SDD needs to know the statistics of the channel noise.

In general, SDD is computationally more intensive than HDD. There are two main reasons for this. One is that SDD needs to operate on real numbers. In a practical application, these numbers will be *quantized* to a finite number of bits. In some binary transmission systems over an AWGN channel, it is known that eight quantization levels, or three bits, are adequate to achieve practically the same performance as with unquantized channel values (Massey 1974; Onyszchuk *et al.* 1993).

The Art of Error Correcting Coding, Second Edition Robert H. Morelos-Zaragoza
 ISBN: 0-470-01558-6

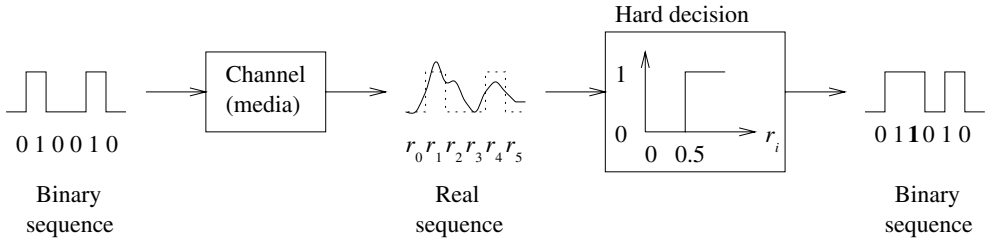


Figure 7.1 Example of an error introduced by hard-decision decoding.

The second reason for the increased complexity in SDD is that the *a posteriori* statistics of the coded symbols, given the received values, need to be computed. However, in return for the increase in implementation cost there is potentially much better performance to be attained. As shown in Chapter 1, with binary transmission over an AWGN channel, to get the same error performance, *SDD requires 2 to 3 dB less* signal-to-noise power ratio than HDD. This is to say that, with SDD, the transmitted power can be 50% to 63% lower compared to HDD, which translates into smaller transmit antennas or, alternatively, smaller receive antennas for the same transmission power.

7.1 Binary transmission over AWGN channels

For memoryless AWGN channels, the *a posteriori* probability of the received values, r_i, given that code symbols v_i are sent, is given by

$$p_{R|V}(r_i|v_i) = p_N(r_i - m(v_i)) = \frac{1}{\sqrt{\pi N_0}} e^{-(r_i - m(v_i))^2/N_0}. \tag{7.1}$$

As shown in Chapters 1 and 5, the MLD metrics become the *squared Euclidean distances*, $D^2(r_i, m(v_i)) = (r_i - m(v_i))^2$. Let E denote the energy of a transmitted symbol. The mapping rule or modulation employed is polar or BPSK,[1]

$$m(v_i) = \begin{cases} \sqrt{E} & \text{if } v_i = 0; \\ -\sqrt{E} & \text{if } v_i = 1. \end{cases} \tag{7.2}$$

which can be expressed as $m(v_i) = (-1)^{v_i}\sqrt{E}$. With BPSK transmission over an AWGN channel, decoding metrics for binary transmission can be simplified by noticing that if $v_i = 1$ is transmitted, then

$$p(r_i|1) = p_{n_i}(r_i + 1) = \frac{1}{\sqrt{\pi N_0}} e^{-(r_i+1)^2/N_0}.$$

Taking the natural logarithm and removing the constant terms, it follows that the *log-likelihood metric* $-(r+1)^2$ is proportional to $-r$. If $v_i = 0$ then the metric is proportional to r. Therefore, it is important to note the following:

[1] Binary phase-shift keying.

With binary transmission over an AWGN channel, metric computation is reduced to changing the signs of the received values.

7.2 Viterbi algorithm with Euclidean metric

The Viterbi decoder (VD) can be used to decode convolutionally coded binary data, BPSK modulated and transmitted over an AWGN channel. With respect to the hard-decision VD algorithm, studied in Section 5.5, two changes are required:

1. Branch-metric generator (BMG) stage

   For an AWGN channel, as shown in the previous section, the metric is proportional to the correlation between the received and the candidate code sequence. Therefore, instead of Euclidean distances, *correlation* metrics can be used.

2. Add-compare-select (ACS) stage

   Instead of minimizing the distance, the VD seeks to *maximize* the correlation metric.

Example 7.2.1 *Consider a memory-2 rate-1/2 convolutional encoder with generators (7,5). Suppose that the information sequence is* $\bar{u} = (110100)$. *The corresponding path is shown in Figure 7.2.*

The transmitted sequence (normalized with respect to $\sqrt{E}$*) is*

$$t(\bar{v}) = (-1, -1, 1, -1, 1, -1, 1, 1, -1, 1, -1, -1).$$

Let the received sequence, after transmission over an AWGN channel and quantization to three bits, be:

$$\bar{r} = (-4, -1, -1, -3, +2, -3, +3, +3, -3, +3, -3, +1).$$

Note that the hard-decision received sequence (given by the sign bits) is:

$$\bar{r}_H = (11, \underline{\mathbf{1}}1, 01, 00, 10, 1\underline{\mathbf{0}}),$$

so that the hard-decision detection process introduces two bit errors. The operation of the VD is illustrated in Figures 7.3 to 7.8. The evolution of the metric values with respect to the

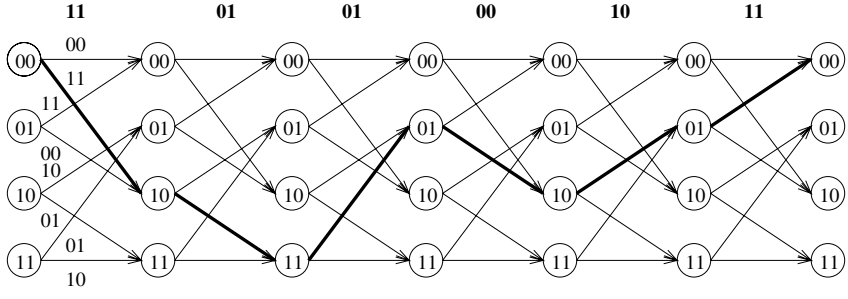


Figure 7.2 A path in the trellis corresponding to $\bar{u} = (110100)$.

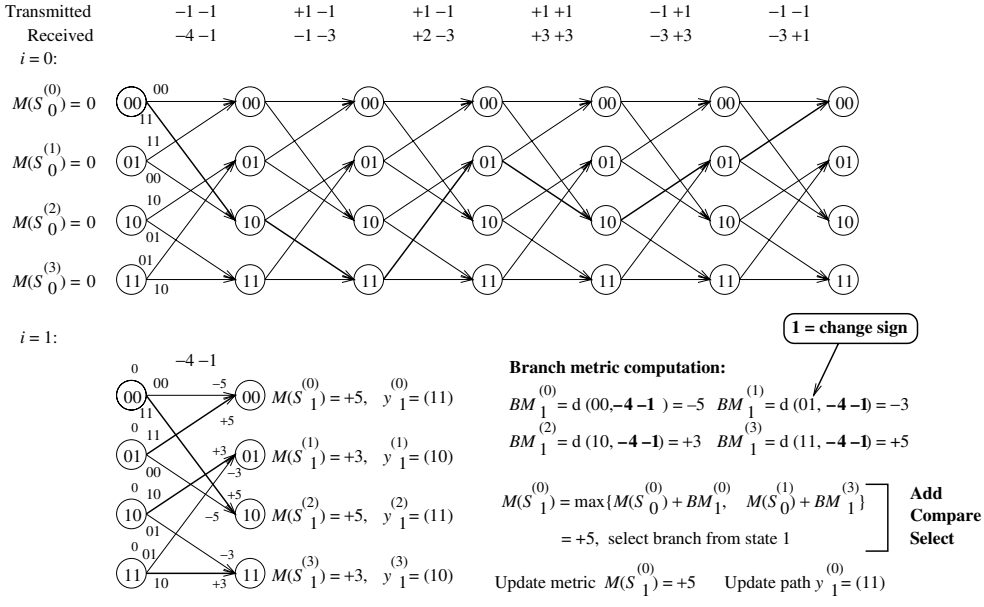


Figure 7.3 Soft-decision Viterbi decoder operation at $i = 1$.

decoding stages is shown in the following table:

State/Stage	$i = 0$	$i = 1$	$i = 2$	$i = 3$	$i = 4$	$i = 5$	$i = 6$
$S_i^{(0)}$	0	+5	+7	+6	+12	+14	+26
$S_i^{(1)}$	0	+3	+5	+12	+14	+24	+18
$S_i^{(2)}$	0	+5	+9	+8	+18	+14	+22
$S_i^{(3)}$	0	+3	+7	+14	+14	+20	+24

The decoded information sequence is $\hat{u} = \begin{pmatrix} 1 & 1 & 0 & 1 & 0 & 0 \end{pmatrix}$*, and two errors have been corrected.*

All the implementation issues related to Viterbi decoding, discussed in Sections 5.5.3 and 5.6.1, apply in the case of soft-decision (SD). In particular, metric normalization must be carefully taken into account.

7.3 Decoding binary linear block codes with a trellis

The Viterbi algorithm (VA) can also be applied to linear block codes. A *syndrome trellis* for a binary linear (N, K) block code C can be constructed from its parity-check matrix as follows (Wolf 1978). Let $(v_1, v_2, \ldots, v_N)$ denote a code word of C. At time i, $1 \leq i \leq N$,

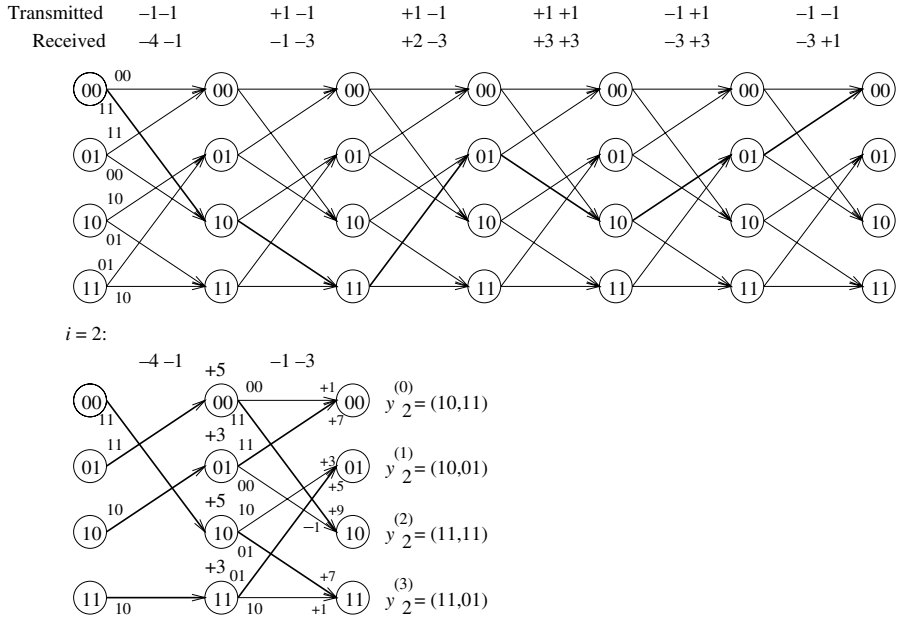


Figure 7.4 Soft-decision Viterbi decoder operation at $i = 2$.

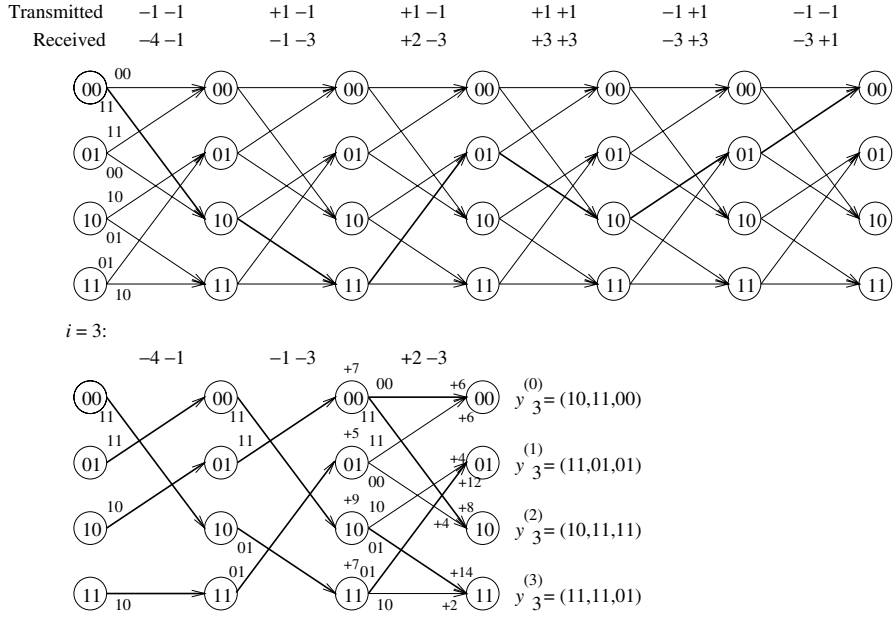


Figure 7.5 Soft-decision Viterbi decoder operation at $i = 3$.

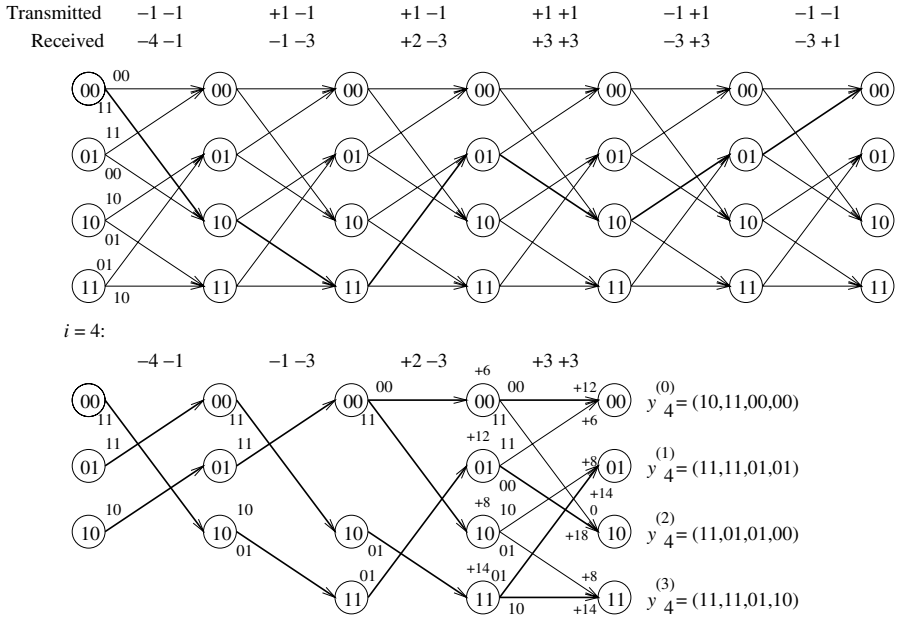


Figure 7.6 Soft-decision Viterbi decoder operation at $i = 4$.

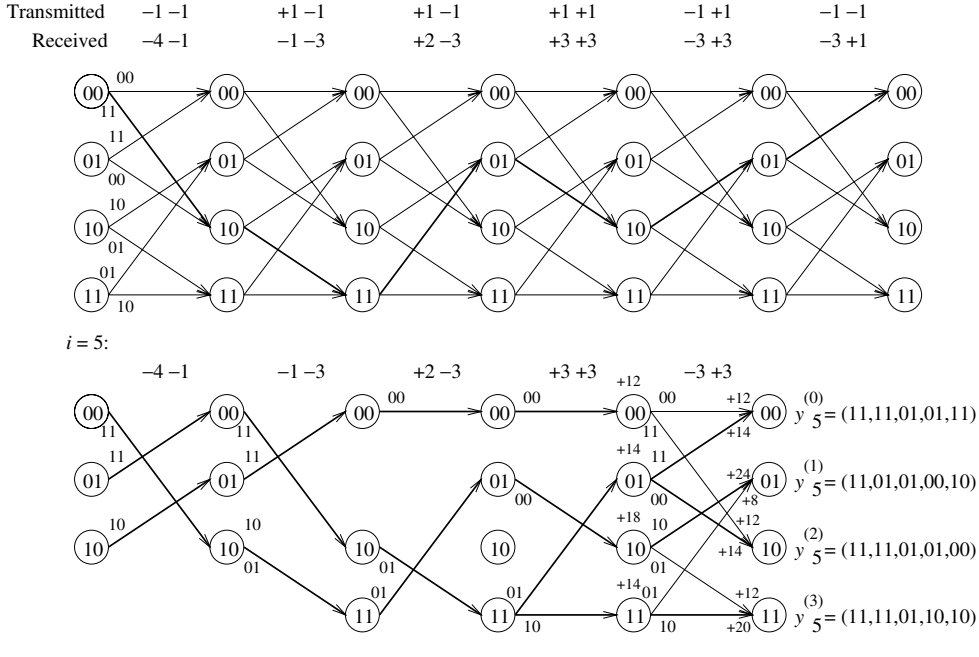


Figure 7.7 Soft-decision Viterbi decoder operation at $i = 5$.

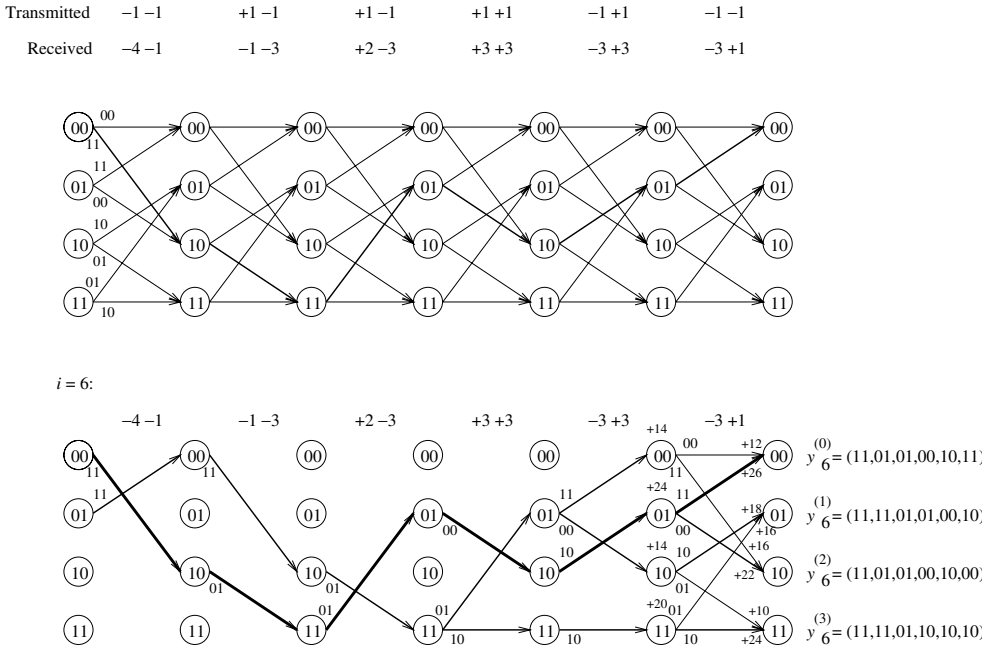


Figure 7.8 Soft-decision Viterbi decoder operation at $i = 6$.

states in the trellis are specified by the *partial syndromes*:

$$\bar{s}_i = \sum_{j=1}^{i} v_j \bar{h}_j, \tag{7.3}$$

where the sum is over $GF(2)$, $\bar{h}_j$ is the j-th column of H, and $\bar{s}_0^\top \triangleq (0\ 0\ \ldots\ 0)$. The maximum number of states in the syndrome trellis is $\min\left\{2^K, 2^{N-K}\right\}$. The syndrome trellis has the property that it has the smallest possible number of states. A trellis satisfying this condition is said to be a *minimal trellis*.

Example 7.3.1 *Consider a binary cyclic Hamming* $(7, 4, 3)$ *code with* $\bar{g}(x) = x^3 + x + 1$. *Then* $\bar{h}(x) = 1 + x + x^2 + x^4$, *and a parity-check matrix for* C *is*

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

To label a state $\bar{s}_j^\top = (s_0\ s_1\ s_2)$ *an integer* I_j *is assigned, such that*

$$I_j = s_0 + s_1 \times 2 + s_2 \times 2^2.$$

The trellis has a maximum of 8 states (since $2^{n-k} = 2^3$*) at times* $i = 3$ *and* $i = 4$, *and is shown in Figure 7.9.*

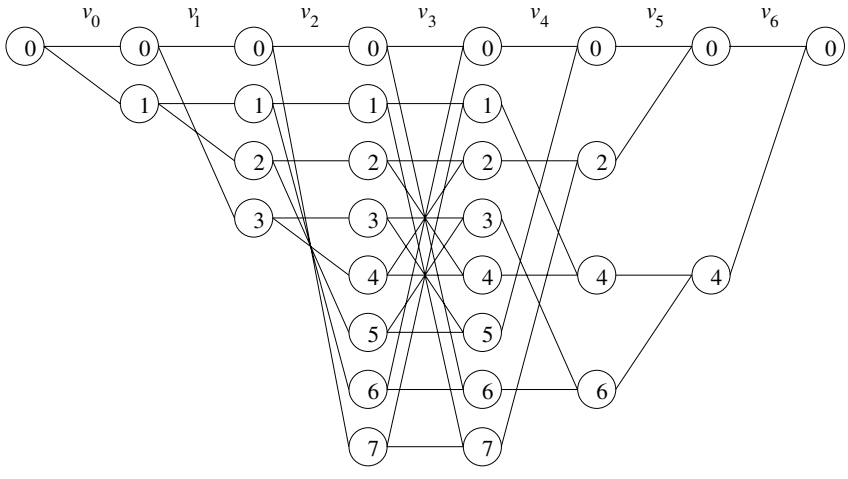


Figure 7.9 Trellis-structure of a Hamming (7, 4, 3) code.

It is interesting to note that with a syndrome trellis there is no need to label the branches with the coded bits. A transition between two states with the same label corresponds to coded bit 0, as seen from Equation (7.3): If the coded bit is 0, then the sum does not change.

Some observations about the trellis structure of linear block codes are the following: In general, the trellis of a binary cyclic code has an irregular structure. As a result, implementation of the VA will be more intricate than in the case of convolutional codes.

For some classes of codes, such as extended[2] BCH codes and Reed–Muller codes, the trellis can be divided into sections. This results in a more regular and symmetric trellis structure with many parallel subtrellises, which may be used to build very high-speed VDs for block codes (Honay and Markarian 1996; Lin *et al.* 1998).

7.4 The Chase algorithm

The Chase algorithm (Chase 1972) is a suboptimal decoding procedure that uses a *set or list of most likely error patterns*. These error patterns are selected on the basis of the *reliability* of the received symbols. Each error pattern is added to the hard-decision received word and decoded using a *hard-decision decoder*. Each decoded code word is scored by computing its *metric* with respect to the received SD sequence. The code word with the *best metric* is selected as the most likely.

Let C be a binary linear (N, K, d) block code, capable of correcting any combination of $t = \lfloor (d-1)/2 \rfloor$ or less random bit errors. Let $\bar{r} = (r_1, r_2, \ldots, r_N)$ be the received word from the output of the channel, $r_i = (-1)^{v_i} + w_i$, where w_i is a zero-mean Gaussian random variable with variance $N_0/2$, $i = 1, 2, \ldots, N$. The sign bits of the received values

[2]Extending codes is covered in Chapter 6.

represent the hard-decision received word,

$$\bar{z}_0 = \begin{pmatrix} z_{0,0} & z_{0,1} & \cdots & z_{0,N-1} \end{pmatrix}, \qquad z_{0,j} = \text{sgn}(r_j),\ 0 \le j < N, \tag{7.4}$$

where

$$\text{sgn}(x) = \begin{cases} 0, & \text{if } x \ge 0; \\ 1, & \text{otherwise.} \end{cases}$$

The *reliabilities* of the received channel values, for binary transmission over an AWGN channel, are the amplitudes $|r_i|$. The received symbol reliabilities are ordered with a sorting algorithm (e.g., *quick sort*). The output of the algorithm is a list of indexes I_j, $j = 1, 2, \ldots, N$, such that

$$|r_{I_1}| \le |r_{I_2}| \le \cdots \le |r_{I_N}|.$$

In the first round of decoding, the hard-decision received word $\bar{z}_0$ is fed into a hard-decision decoder. Let $\bar{v}_0$ denote the decoded code word, stored as the initial code word guess. Then the metric of $\bar{v}_0$ with respect to the received word $\bar{r}$

$$\nu_0 = \sum_{j=1}^{N} (-1)^{v_{0j}} \times r_j, \tag{7.5}$$

is computed and its value is stored as the maximum.

Chase classified his algorithm into three types, according to the error pattern generation:

- **Type-I**

  Test all error patterns within radius $(d - 1)$ from the received word.

- **Type-II**

  Test the error patterns with at most $\lfloor d/2 \rfloor$ errors located within the bit positions having the $\lfloor d/2 \rfloor$ lowest reliabilities. Compared to Type-I, the performance of Chase Type-II algorithm is only slightly inferior, while at the same time having a significantly reduced number of test patterns.

- **Type-III**

  Test those error patterns with i ones in the i least reliable bit positions, with i odd and $1 \le i \le d - 1$. It should be noted that this algorithm is closely related to the generalized minimum distance (GMD) decoding algorithm (see also (Michelson and Levesque 1985), p. 168). GMD decoding of RS codes is the topic of Section 7.6.

Because of its reduced complexity and good performance, Chase algorithm Type-II is the most popular among the three types above, and is described next. A flow diagram of Chase algorithm is shown in Figure 7.10.

Chase type-II algorithm

- For $i = 1, 2, \ldots, 2^t - 1$, an error pattern $\bar{e}_i$ is added to the hard-decision received word: $\bar{z}_i = \bar{e}_i \oplus \bar{z}_0$.

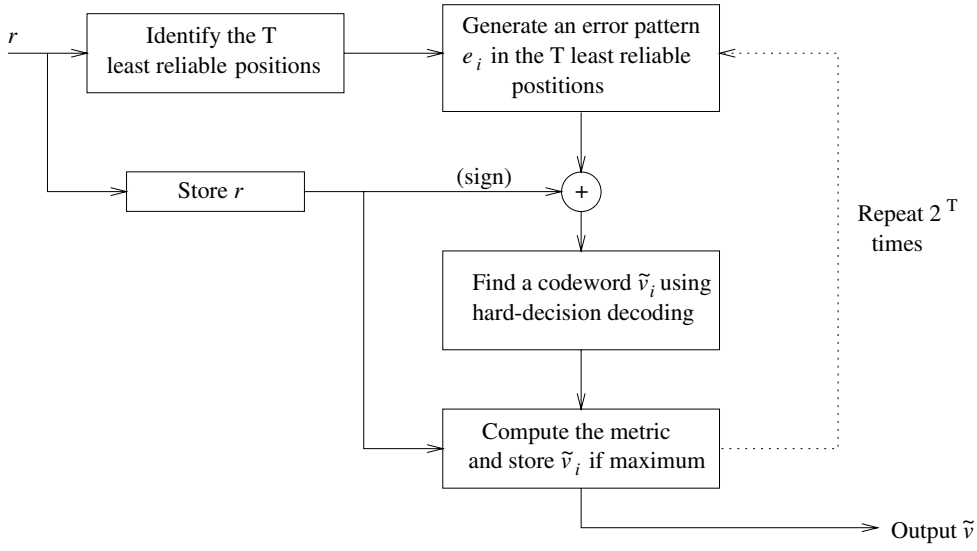


Figure 7.10 Flow diagram of Chase type-II decoding algorithm.

- The error patterns are generated among the t least reliable positions (LRPs), that is, positions $\{I_1, I_2, \ldots, I_t\}$, for which reliabilities (amplitudes) are the smallest.
- Each test vector $\bar{z}_i$ is input to a hard-decision decoder, producing a code word $\bar{v}_i$, $i = 1, 2, \ldots, 2^t - 1$.
- The metric is computed according to Equation (7.5) and if maximum, code word $\bar{v}_i$ stored as the most likely.

If desired, the algorithm can output a list of code words. This is a useful feature in producing soft-outputs for use of the Chase algorithm in iterative decoding of block turbo codes; see Section 8.2.3.

Example 7.4.1 *Consider the binary repetition* $(3, 1, 3)$ *code of Example 1.1.1. This code is capable of correcting* $t = 1$ *error. Assume binary transmission over an AWGN channel. maximum-likelihood (ML) decoding is achieved using correlation metrics. Let* $\bar{v} = (v_1, v_2, v_3)$ *denote a code word and* $\bar{r} = (r_1, r_2, r_3)$ *the corresponding received vector. The ML decoder computes the correlation metric*

$$\nu = \bar{r} \cdot m(\bar{v}) = r_1(-1)^{v_1} + r_2(-1)^{v_2} + r_3(-1)^{v_3},$$

and decides for the code word $\hat{v}_{ML}$ *for which the metric is largest. This is equivalent to checking the sign of the metric with the following rule:*

$$\hat{v}_{ML} = \begin{cases} (000), & \nu \geq 0; \\ (111), & \nu < 0. \end{cases}$$

Let $\bar{r} = (0.9, -0.1, -0.2)$ *be the received vector. Then* $\nu = 0.6$ *and* $\hat{v}_{ML} = (000)$.

Using Chase type-II algorithm, the hard-decision received word is

$$\bar{z}_0 = \text{sgn}(\bar{r}) = (011),$$

which is decoded as $\bar{v}_0 = (111)$. *This is the initial code word guess, which is stored together with the metric*

$$\nu_0 = -r_1 - r_2 - r_3 = -0.9 + 0.1 + 0.2 = -0.6$$

The reliabilities of the received values are sorted, resulting in $|r_2| < |r_3| < |r_1|$. *Therefore, the second position is the least reliable. The number of error patterns here is* $2^t = 1 : \{(010)\}$, *i.e., one error in the second position. This results in*

$$\bar{z}_1 = \bar{z}_0 \oplus (010) = (001),$$

which is decoded as $\bar{v}_1 = (000)$, *with metric*

$$\nu_1 = r_1 + r_2 + r_3 = 0.6.$$

Since $\nu_1 > \nu_0$, *the decoder decides that* $\hat{v} = \bar{v}_1 = (000)$ *is the most likely code word. In this case, the Chase type-II algorithm finds the ML code word successfully.*

7.5 Ordered statistics decoding

The ordered statistics decoding (OSD) algorithm (Fossorier and Lin 1995) is similar to the Chase algorithm in the sense of creating a list of error patterns and using hard-decision decoding. However, unlike the algorithm in the previous section, which deals with *least reliable code positions*, OSD considers a list of code words drawn from a set of (permuted) *most reliable information positions*.

Assume that an (N, K) linear block code C, with generator matrix G and minimum distance $d_H \geq 2t + 1$, is used with binary transmission over an AWGN channel. Let

$$\bar{c} = (c_1, c_2, \ldots, c_N)$$

denote a code word of C, and let

$$\bar{r} = (r_1, r_2, \ldots, r_N)$$

be the received sequence.

As in the Chase algorithm above, the decoding process begins by *reordering* the components of the received sequence in decreasing order of reliability value. Let

$$\bar{y} = (y_1, y_2, \ldots, y_N)$$

denote the resulting sequence, where $|y_1| \geq |y_2| \geq \cdots \geq |y_N|$. This reordering defines a *permutation mapping* λ_1 such that $\bar{y} = \lambda_1[\bar{r}]$.

The next step is to permute the columns of G using λ_1:

$$G' = \lambda_1[G] = \left(\bar{g}'_1 \quad \bar{g}'_2 \quad \cdots \quad \bar{g}'_N\right),$$

where $\bar{g}'_j$ denotes the j-th column of G'.

A *most reliable basis* is obtained in the next step of the algorithm as follows:

Starting from the first column of G', find the first K linearly independent (LI) columns with the largest associated reliability values. Then, these K LI columns are used as the first K columns of a new matrix G'', maintaining their reliability order. The remaining $(N-K)$ columns of G'' are also arranged in decreasing reliability order. This process defines a second permutation mapping λ_2, such that

$$G'' = \lambda_2\left[G'\right] = \lambda_2\left[\lambda_1\left[G\right]\right].$$

Applying the map λ_2 to $\bar{y}$ results in the reordered received sequence:

$$\bar{z} = \lambda_2\left[\bar{y}\right] = (z_1, z_2, \ldots, z_k, z_{k+1}, \ldots, z_N),$$

with $|z_1| \geq |z_2| \geq \cdots \geq |z_K|$ and $|z_{k+1}| \geq \cdots \geq |z_N|$. By performing elementary row operations on G'', a systematic form G_1 can be obtained:

$$G_1 = \begin{pmatrix} I_k & P \end{pmatrix} = \begin{pmatrix} 1 & 0 & \cdots & 0 & p_{1,1} & \cdots & p_{1,N-K} \\ 0 & 1 & \cdots & 0 & p_{2,1} & \cdots & p_{2,N-K} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 & p_{k,1} & \cdots & p_{k,N-K} \end{pmatrix}.$$

It is easy to see that the code generated by G_1 is equivalent to G, in the sense that the same code words are generated, with permuted code bit positions.

The next step is to perform *HDD* based on the k most reliable (MR) values of the permuted received sequence $\bar{z}$. Let $\bar{u}_a = (u_1, u_2, \ldots, u_K)$ denote the result. Then, the corresponding code word in C_1 can be computed as

$$\bar{v} = \bar{u}_a G_1 = \begin{pmatrix} v_0 & v_1 & \ldots & v_{K-1} & v_K & \ldots & v_N \end{pmatrix}.$$

The estimated code word $\bar{v}_{HD}$ in C can be obtained from $\bar{v}$, via the inverse mapping $\lambda_1^{-1}\lambda_2^{-1}$, as

$$\bar{v}_{HD} = \lambda_1^{-1}\left[\lambda_2^{-1}\left[\bar{v}\right]\right]. \tag{7.6}$$

On the basis of the hard-decision–decoded code word $\bar{v}$, the algorithm proceeds to *reprocess* it until either *practically optimum* or a predefined desired error performance is achieved.

Order-ℓ reprocessing

For $1 \leq i \leq \ell$, make all possible changes of i of the k MR bits of $\bar{v}$.

For each change, find the corresponding code word $\bar{v}'$ in C_1 and map it onto a binary real sequence $\bar{x}'$, via the mapping $\{0, 1\} \mapsto \{+1, -1\}$.

For each candidate code word generated, compute the squared Euclidean distance[3] between the generated sequence $\bar{x}'$ and the reordered received sequence $\bar{z}$. After all the $\sum_{i=0}^{\ell}\binom{K}{i}$ code words have been generated and their distances compared, the algorithm outputs the code word $\bar{v}^*$ which is closest to $\bar{z}$.

From $\bar{v}^*$, using the inverse map (7.6), the most likely code sequence $\bar{v}_{\mathrm{ML}}^*$ is computed.

[3]See equation (1.35) of Chapter 1.

Reprocessing can be done in a systematic manner to minimize the number of computations. In particular, as mentioned in Section 7.1, with binary transmission over an AWGN channel, there is no need to compute the Euclidean distance but rather the *correlation* between the generated code words and the reordered received sequence. Thus, the computation of the binary real sequence $\bar{x}'$ is not needed. Only additions of the permuted received values z_i, with sign changes given by the generated $\bar{v}^*$, are required.

Example 7.5.1 *Consider the binary Hamming* $(7, 4, 3)$ *code with generator matrix*

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

Suppose that the received vector, after binary transmission over an AWGN channel (the outputs of a matched filter), is given by

$$\bar{r} = (0.5, 0.3, 1.3, -0.1, 0.7, 0.6, 1.5).$$

OSD with order-1 reprocessing is considered next.

The permuted received vector based on reliability values is

$$\bar{y} = \Pi_1(\bar{r}) = \begin{pmatrix} 1.5 & 1.3 & 0.7 & 0.6 & 0.5 & 0.3 & -0.1 \end{pmatrix},$$

with $\Pi_1 = \begin{pmatrix} 5 & 6 & 2 & 7 & 3 & 4 & 1 \end{pmatrix}$. *The permuted generator matrix based on reliability values is*

$$G' = \Pi_1(G) = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 \end{pmatrix} \longrightarrow G' = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 \end{pmatrix}.$$

Therefore, $G_1 = \Pi_2(G') = G'$, *with* $\Pi_2 = I = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{pmatrix}$ *(the identity permutation). As a result,*

$$\Pi_2(y) = \begin{pmatrix} 1.5 & 1.3 & 0.7 & 0.6 & 0.5 & 0.3 & -0.1 \end{pmatrix}.$$

The corresponding hard-decision vector is $\bar{z} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$ *and the* $k = 4$ *most reliable values are* $\bar{u}_0 = \begin{pmatrix} 0 & 0 & 0 & 0 \end{pmatrix}$. *The initial code word is as follows:*

$$\bar{v}_0 = \bar{u}_0 G_1 = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}.$$

The decoding algorithm for order-1 reprocessing is summarized in the table below. The metric used is the correlation discrepancy:

$$\lambda(\bar{v}_\ell, \bar{z}) \stackrel{\Delta}{=} \sum_{i: v_{\ell,i} \neq z_i} |y_i|.$$

ℓ	$\bar{u}_\ell$	$\bar{v}_\ell$	$\lambda(\bar{v}_\ell, \bar{z})$
0	(0000)	(0000000)	0.1
1	(1000)	(1000111)	2.3
2	(0100)	(0100101)	1.8
3	(0010)	(0010011)	1.0
4	(0001)	(0001110)	1.4

The smallest metric corresponds to $\ell = 0$ *and it follows that the decoded code word is*

$$\bar{v}_{HD} = \Pi_1^{-1}\left(\Pi_2^{-1}(\bar{v}_0)\right) = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}.$$

7.6 Generalized minimum distance decoding

In 1966, Forney (1966b) introduced GMD decoding. The basic idea was to extend the notion of an erasure, by dividing the received values into *reliability classes*. The decoding strategy is similar to the Chase algorithm Type-III, with the use of *erasure patterns*. The GMD decoder works by declaring an increasing number of erasures within the $d - 1$ least reliable symbols, and testing a *sufficient condition* for the optimality of the decoded word, until the condition is satisfied or a maximum number of erasures have been considered.

Let C be a linear block (N, K, d) code. Assume that there is an *errors-and-erasures decoder* that is capable of decoding any combination of e errors and s erasures within the capability of the code, that is, $2e + s \leq d - 1$. Such decoders were presented in Section 3.5.6 for BCH codes and in Section 4.3.2 for RS codes.

Let $\bar{r} = (r_1, r_2, \ldots, r_N)$ be the received word from the output of the channel, where

$$r_i = (-1)^{c_i} + w_i,$$

and w_i is a zero-mean Gaussian random variable with variance $N_0/2$, $i = 1, 2, \ldots, N$.

Important: The GMD decoding algorithm below assumes that the received vector $\bar{r}$ has been clipped so that its components lie in the range $[-1, +1]$. That is, if the amplitude $|r_i| > 1$, then it is forced to one: $|r_i| = 1$, $i = 1, 2, \ldots, N$.

As before, the sign bits of the received values represent the hard-decision received word,

$$\bar{z} = \begin{pmatrix} z_1 & z_2 & \ldots & z_N \end{pmatrix}, \qquad z_j = \operatorname{sgn}(r_j),\ 1 \leq j \leq N.$$

As with Chase algorithms and the OSD algorithm, the *reliabilities* of the received channel values are sorted, producing a list of indexes I_j, $j = 1, 2, \ldots, N$, such that

$$|r_{I_1}| \leq |r_{I_2}| \leq \cdots \leq |r_{I_N}|.$$

In the first round of decoding, the hard-decision received word $\bar{z}$ is fed into an *errors-only* algebraic decoder. Let $\hat{v}$ denote the resulting estimated code word. The correlation metric of $\hat{v}$ with respect to the received word $\bar{r}$

$$\nu = \sum_{j=1}^{N} (-1)^{\hat{v}_j} \times r_j, \tag{7.7}$$

is computed. If the following sufficient condition is satisfied,

$$v > n - d, \tag{7.8}$$

then $\hat{v}$ is accepted as the most likely code word and decoding stops.

Otherwise, a new round of decoding is performed. This is accomplished by setting $s = 2$ erasures, in positions I_1 and I_2, and decoding the resulting word with an *errors-and-erasures* decoder. The correlation metric between $\bar{r}$ and the estimated code word $\hat{v}$ is computed, as in (7.7), and then the sufficient condition (7.8) tested.

This GMD decoding process continues, if necessary, every round increasing the number of erasures by two, $s = s + 2$, until the maximum number of erasures ($s_{\max} = d - 1$) in the LRPs are tried. If at the end of GMD decoding no code word is found, the output can be either an indication of a decoding failure or the hard-decision decoded code word $\hat{v}_0$ obtained with $s = 0$.

7.6.1 Sufficient conditions for optimality

The condition used in GMD decoding can be improved and applied to other decoding algorithms that output lists of code words, such as Chase and OSD. These algorithms are instances of *list decoding* algorithms. The acceptance criteria (7.8) is too restrictive, resulting in many code words rejected, possibly including the most likely (i.e., selected by true MLD) code word. Improved sufficient conditions on the optimality of a code word have been proposed. Without proofs, two such conditions are listed below. Before their description, some definitions are needed.

Let $\bar{x}$ represent a BPSK *modulated code word*, $\bar{x} = m(\bar{v})$, where $\bar{v} \in C$ and $x_i = (-1)^{v_i}$, for $1 \leq i \leq N$. See also (7.2). Let $S_e = \{i : \text{sgn}(x_i) \neq \text{sgn}(r_i)\}$ be the set of error positions, let $U = \{I_j, j = 1, 2, \ldots, d\}$ be the set of least reliable positions, and let the set of correct but least reliable positions be $T = \{i : \text{sgn}(x_i) = \text{sgn}(r_i),\ i \in U\}$. Then the *extended distance* or *correlation discrepancy* between a code word $\bar{v}$ and a received word $\bar{r}$ is defined as (Taipale and Pursley 1991)

$$d_e(\bar{v}, \bar{r}) = \sum_{i \in S_e} |y_i|. \tag{7.9}$$

Improved criteria for finding an optimum code word are based on upper bounds on (7.9) and increasing the cardinality of the sets of positions tested. Two improvements to Forney's conditions are:

- **Taipale-Pursley condition** (Taipale and Pursley 1991). There exists an optimal code word $\bar{x}_{\text{opt}}$ such that

$$d_e(\bar{x}_{\text{opt}}, \bar{r}) < \sum_{i \in T} |r_i|. \tag{7.10}$$

- **Kasami *et al.* condition** (Kasami *et al.* 1995). There exists an optimal code word $\bar{x}_{\text{opt}}$ such that

$$d_e(\bar{x}_{\text{opt}}, \bar{r}) < \sum_{i \in T_K} |r_i|, \tag{7.11}$$

  where $T_K = \{i : \text{sgn}(x_i) \neq \text{sgn}(r_i),\ i \in U\}$.

Good references to GMD decoding, its extensions, and combinations with Chase algorithms are Kaneko *et al.* (1994), Kamiya (2001), Tokushige *et al.* (2000), Fossorier and Lin (1997b), and Takata *et al.* (2001).

7.7 List decoding

List decoding was introduced by Elias and Wozencraft (see Elias (1991)). Most recently, list decoding of polynomial codes has received considerable attention, mainly caused by the papers written by Sudan and colleagues (Guruswami and Sudan 1999; Sudan 1997) on decoding RS codes beyond their error correcting capabilities. The techniques used, referred to as *Sudan algorithms*, use interpolation and factorization of bivariate polynomials over extension fields. Sudan algorithms can be considered extensions of the Welch-Berlekamp algorithm (Berlekamp 1996). These techniques have been applied to SD decoding of RS codes in Koetter and Vardy (2000).

7.8 Soft-output algorithms

The previous sections of this chapter have been devoted to decoding algorithms that output the most likely coded *sequence* or code word (or list of code words). However, since the appearance of the revolutionary paper on turbo codes in 1993 (Berrou *et al.* 1993), there is a need for decoding algorithms that output not only the most likely code word (or list of code words), but also an estimate of the *bit reliabilities* for further processing. In the field of error correcting codes, soft-output algorithms were introduced as early as 1962, when Gallager (1962) published his work on low-density parity-check (LDPC) codes[4], and later by Bahl *et al.* (1974). In both cases, the algorithms perform a *forward–backward* recursion to compute the reliabilities of the code symbols. In the next section, basic soft-output decoding algorithms are described. Programs to simulate these decoding algorithms can be found on the error correcting coding (ECC) web site.

In the following sections, and for simplicity of exposition, it is assumed that a linear block code, constructed by terminating a binary memory-m rate-$1/n$ convolutional code, is employed for binary transmission over an AWGN channel. It is also assumed that the convolutional encoder starts at the all-zero state $S_0^{(0)}$ and, after N trellis stages, ends at the all-zero state $S_N^{(0)}$.

7.8.1 Soft-output Viterbi algorithm

In 1989, the VA was modified to output bit reliability information (Hagenauer and Hoeher 1989). The *soft-output viterbi algorithm (SOVA)* computes the reliability, or soft-output, of the information bits as a *log-likelihood ratio* (LLR),

$$\Lambda(u_i) \triangleq \log\left(\frac{\Pr\{u_i = 1|\bar{r}\}}{\Pr\{u_i = 0|\bar{r}\}}\right), \tag{7.12}$$

where $\bar{r}$ denotes the received sequence.

[4]LDPC codes are covered in Chapter 8.

The operation of a SOVA decoder can be divided into two parts. In the first part, decoding proceeds as with the conventional VA, selecting the most likely coded sequence, $\hat{v}$, in correspondence with the path in the trellis with the maximum (correlation) metric, at stage n (see Section 5.5). In addition, the path metrics need to be stored at each decoding stage, and for each state. These metrics are needed in the last part of the algorithm, to compute the *soft outputs*. In the second part of SOVA decoding, the VA transverses the trellis backwards, and computes metrics and paths, starting at $i = N$ and ending at $i = 0$. It should be noted that in this stage of the SOVA algorithm there is no need to store the surviving paths, but only the metrics for each trellis state. Finally for each trellis stage i and $1 \leq i \leq N$, the soft outputs are computed.

Let $M_{\max}$ denote the (correlation) metric of the most likely sequence $\hat{v}$ found by the VA. The probability of the associated information sequence $\hat{u}$ given the received sequence, or *a posteriori probability* (APP), is proportional to $M_{\max}$, since

$$\Pr\{\hat{u}|\bar{r}\} = \Pr\{\hat{v}|\bar{r}\} \sim e^{M_{\max}}. \tag{7.13}$$

Without loss of generality, the APP of information bit u_i can be written as

$$\Pr\{u_i = 1|\bar{r}\} \sim e^{M_i(1)},$$

where $M_i(1) \stackrel{\Delta}{=} M_{\max}$. Let $M_i(0)$ denote the maximum metric of paths associated with the complement of information symbol u_i. Then it is easy to show that

$$\Lambda(u_i) \sim M_i(1) - M_i(0). \tag{7.14}$$

Therefore, at time i, the soft output can be obtained from the difference between the maximum metric of paths in the trellis with $\hat{u}_i = 1$ and the maximum metric of paths with $\hat{u}_i = 0$.

In the soft-output stage of the SOVA algorithm, at stage i, the most likely information symbol $u_i = a$, $a \in \{0, 1\}$ is determined and the corresponding maximum metric (found in the forward pass of the VA) is set equal to $M_i(u_i)$. The path metric of the best competitor, $M_i(u_i \oplus 1)$, can be computed as (Vucetic and J. Yuan 2000)

$$M_i(v_i \oplus 1) = \min_{k_1,k_2} \left\{ M_f(S_{i-1}^{(k_1)}) + BM_i^{(b_1)}(u_i \oplus 1) + M_b(S_i^{(k_2)}) \right\}, \tag{7.15}$$

where $k_1, k_2 \in \{0, 1, 2, \ldots, 2^m - 1\}$,

- $M_f(S_{i-1}^{(k_1)})$ is the path metric of the forward survivor at time $i - 1$ and state $S^{(k_1)}$,
- $BM_i^{(b_1)}(u_i \oplus 1)$ is the branch metric at time i for the complement information associated with a transition from state $S^{(k_1)}$ to $S^{(k_2)}$, and
- $M_b(S_i^{(k_2)})$ is the backward survivor path metric at time i and state $S^{(k_2)}$.

Finally, the soft output is computed as

$$\Lambda(u_i) = M_i(1) - M_i(0). \tag{7.16}$$

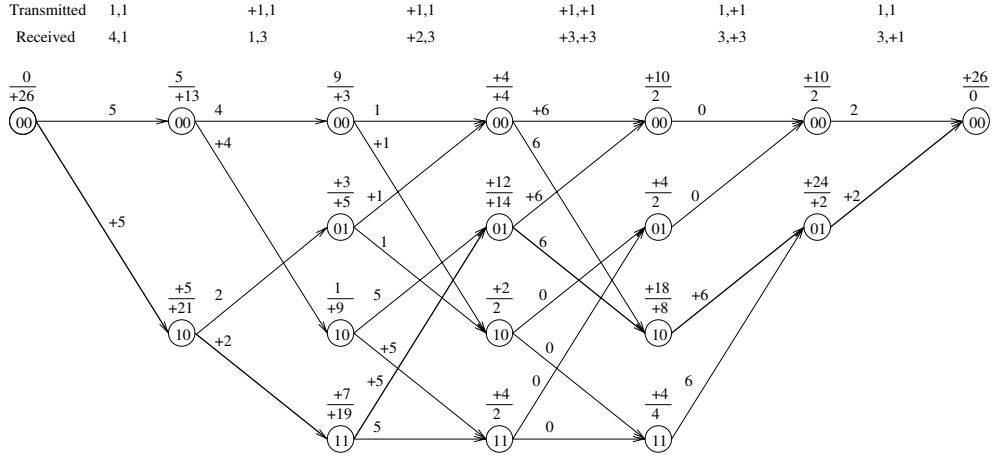


Figure 7.11 Trellis diagram used in SOVA decoding for Example 7.8.1.

Example 7.8.1 *Let C be a zero-tail* $(12, 4)$ *code obtained from a memory-2 rate-1/2 convolutional code with generators* $(7, 5)$. *The basic structure of the trellis diagram of this code is the same as in Example 7.2.1. Suppose the information sequence (including the tail bits) is* $\bar{u} = (110100)$*, and that the received sequence, after binary transmission over an AWGN channel, is*

$$\bar{r} = (-4, -1, -1, -3, +2, -3, +3, +3, -3, +3, -3, +1).$$

Figure 7.11 shows the trellis diagram for this code. For $i = 0, 1, \ldots, 6$*, each state has a label on top of it of the form*

$$\frac{M_f(S_i^{(m)})}{M_b(S_i^{(m)})}.$$

The branches are labeled with the branch metrics BM_i. *The soft outputs* $\Lambda(u_i)$ *are given by* $\{-16, -12, -2, +10, +22, +26\}$*, for* $i = 1, 2, \ldots, 6$*, respectively.*

Implementation issues

In a SOVA decoder, the VA needs to be executed twice. The forward processing is just as in conventional VD, with the exception that path metrics at each decoding stage need to be stored. The backward processing uses the VA but does not need to store the surviving paths; only the metrics at each decoding state. Note that both backward processing and SD computation can be done simultaneously. The backward processing stage does not need to store path survivors. In addition, the soft outputs need to be computed, after both the forward and backward recursions finish. Particular attention should be paid to the normalization of metrics at each decoding stage, in both directions. Other implementation issues are the same as in a VD, discussed in Sections 5.5.3 and 5.6.1.

The SOVA decoder can also be implemented as a *sliding window decoder*, like the conventional VD. By increasing the computation time, the decoder operates continuously, not on a block-by-block basis, without forcing the state of the encoder to return to the all-zero state periodically. The idea is the same as that used in the VD with traceback memory,

as discussed in Section 5.5.3, where forward recursion, traceback and backward recursion, and soft-output computations are implemented in several memory blocks (see also (Viterbi 1998)).

7.8.2 Maximum-a posteriori (MAP) algorithm

The Bahl-Cocke-Jelinek-Raviv (BCJR) algorithm (Bahl *et al.* 1974) is an optimal symbol-by-symbol MAP decoding method for linear block codes that minimizes the probability of a symbol error. The goal of this MAP decoder is to examine the received sequence $\bar{r}$ and to compute the a posteriori probabilities of the input information bits, as in (7.12). The MAP algorithm is described next, following closely the arguments in Bahl *et al.* (1974).

The state transitions (or branches) in the trellis have probabilities

$$\Pr\left\{S_i^{(m)} | S_{i-1}^{(m')}\right\}, \tag{7.17}$$

and for the output symbols $\bar{v}_i$,

$$q_i(x_i|m', m) \stackrel{\Delta}{=} \Pr\left\{x_i = x | S_{i-1}^{(m')}, S_i^{(m)}\right\}, \tag{7.18}$$

where $x = \pm 1$, and $x_i = m(v_i) = (-1)^{v_i}$, $0 < i \leq N$.

The sequence $\bar{x}$ is transmitted over an AWGN channel and received as a sequence $\bar{r}$, with transition probabilities

$$\Pr\{\bar{r}|\bar{x}\} = \prod_{i=1}^{N} p(\bar{r}_i|\bar{x}_i) = \prod_{i=1}^{N}\prod_{j=0}^{n-1} p(r_{i,j}|x_{i,j}), \tag{7.19}$$

where $p(r_{i,j}|x_{i,j})$ is given by (7.1).

Let $B_i^{(j)}$ be the set of branches connecting state $S_{i-1}^{(m')}$ to state $S_i^{(m)}$ such that the associated information bit $u_i = j$, with $j \in \{0, 1\}$. Then

$$\Pr\{u_i = j|\bar{r}\} = \sum_{(m',m)\in B_i^{(j)}} \Pr\left\{S_{i-1}^{(m')}, S_i^{(m)}, \bar{r}\right\} \stackrel{\Delta}{=} \sum_{(m',m)\in B_i^{(j)}} \sigma_i(m', m). \tag{7.20}$$

The value of $\sigma_i(m', m)$ in (7.20) is equal to

$$\sigma_i(m', m) = \alpha_{i-1}(m') \cdot \gamma_i^{(j)}(m', m) \cdot \beta_i(m), \tag{7.21}$$

where the joint probability $\alpha_i(m) \stackrel{\Delta}{=} \Pr\left\{S_i^{(m)}, \bar{r}_p\right\}$ is given recursively by

$$\alpha_i(m) = \sum_{m'} \alpha_{i-1}(m') \cdot \sum_{j=0}^{1} \gamma_i^{(j)}(m', m), \tag{7.22}$$

and is referred to as the *forward metric*.

The conditional probability $\gamma_i^{(j)}(m', m) \stackrel{\Delta}{=} \Pr\left\{S_i^{(m)}, \bar{r}|S_{i-1}^{(m')}\right\}$ is given by

$$\gamma_i^{(j)}(m', m) = \sum_{x} p_i(m|m')\Pr\left\{x_i = x|S_{i-1}^{(m')}, S_i^{(m)}\right\} \cdot \Pr\{r_i|x\} \tag{7.23}$$

where $p_i(m|m') = \Pr\left\{S_i^{(m)}|S_{i-1}^{(m')}\right\}$, which for the AWGN channel can be put in the form

$$\gamma_i^{(j)}(m', m) = \Pr\{u_i = j\} \cdot \delta_{ij}(m, m') \cdot \exp\left(-\frac{1}{N_0}\sum_{q=0}^{n-1}(r_{i,q} - x_{i,q})^2\right), \tag{7.24}$$

where $\delta_{ij}(m, m') = 1$ if $\{m', m\} \in B_i^{(j)}$, and $\delta_{ij}(m, m') = 0$ otherwise. $\gamma_i^{(j)}(m', m)$ is referred to as the *branch metric*.

The conditional probability $\beta_i(m) \triangleq \Pr\left\{\bar{r}_f|S_i^{(m)}\right\}$ is given by

$$\beta_i(m) = \sum_{m'} \beta_{i+1}(m') \cdot \sum_{j=0}^{1} \gamma_i^{(j)}(m', m), \tag{7.25}$$

and referred to as the *backward metric*.

Combining (7.25), (7.24), (7.22) , (7.21) and (7.12), the soft output (LLR) of information bit u_i is given by

$$\Lambda(u_i) = \log\left(\frac{\Pr\{u_i = 1|\bar{r}\}}{\Pr\{u_i = 0|\bar{r}\}}\right) = \log\left(\frac{\sum_m \sum_{m'} \alpha_{i-1}(m')\gamma_i^{(1)}(m', m)\beta_i(m)}{\sum_m \sum_{m'} \alpha_{i-1}(m')\gamma_i^{(0)}(m', m)\beta_i(m)}\right), \tag{7.26}$$

where the hard-decision output is given by $\hat{u}_i = \text{sgn}(\Lambda(u_i))$ and the reliability of the bit u_i is $|\Lambda(u_i)|$. The above equations can be interpreted as follows. A bidirectional Viterbi-like algorithm can be applied, just as in SOVA decoding (previous section). In the forward recursion, given the probability of a state transition at time i, the joint probability of the received sequence up to time i and the state at time i is evaluated. In the backward recursion, the probability of the received sequence from time $i + 1$ to time N, given the state at time i is computed. Then the soft output depends on the joint probability of the state transition and the received symbol at time i.

The MAP algorithm can be summarized as follows:

- **Initialization**

  For $m = 0, 1, \ldots, 2^m - 1$,

  $$\alpha_0(0) = 1, \quad \alpha_0(m) = 0, m \neq 0,$$

  For $m' = 0, 1, \ldots, 2^m - 1$,

  $$\beta_N(0) = 1, \quad \beta_N(m') = 0, m' \neq 0,$$

- **Forward recursion**

  For $i = 1, 2, \ldots, N$,

  1. For $j = 0, 1$, compute and store the branch metrics $\gamma_i^{(j)}(m', m)$ as in (7.24).
  2. For $m = 0, 1, \ldots, 2^m - 1$, compute and store the forward metrics $\alpha_i(m)$ as in (7.22).

- **Backward recursion**

  For $i = N-1, N-2, \ldots, 0$,

  1. Compute the backward metrics $\beta_i(m)$ as in (7.25), using the branch metric computed in the forward recursion.
  2. Compute the LLR $\Lambda(u_i)$ as in (7.26).

Implementation issues

The implementation of a MAP decoder is similar to that of a SOVA decoder, as both decoders perform forward and backward recursions. All the issues mentioned in the previous section apply to a MAP decoder. In addition, note that branch metrics depend on the noise power density N_0, which should be estimated to keep optimality. To avoid numerical instabilities, the probabilities $\alpha_i(m)$ and $\beta_i(m)$ need to be scaled at every decoding stage, such that $\sum_m \alpha_i(m) = \sum_m \beta_i(m) = 1$.

7.8.3 Log-MAP algorithm

To reduce the computational complexity of the MAP algorithm, the logarithms of the metrics may be used. This results in the so-called *log-MAP* algorithm.

From (7.22) and (7.25) (Robertson *et al.* 1995),

$$\log \alpha_i(m) = \log\left(\sum_{m'}\sum_{j=0}^{1} \exp\left(\log \alpha_{i-1}(m') + \log \gamma_i^{(j)}(m', m)\right)\right),$$
$$\log \beta_i(m) = \log\left(\sum_{m'}\sum_{j=0}^{1} \exp\left(\log \beta_{i+1}(m') + \log \gamma_i^{(j)}(m', m)\right)\right). \tag{7.27}$$

Taking the logarithm of $\gamma_i^{(j)}(m', m)$ in (7.24),

$$\log \gamma_i^{(j)}(m', m) = \delta_{ij}(m, m')\left\{\log \Pr\{u_i = j\} - \frac{1}{N_0}\sum_{q=0}^{n-1}(r_{i,q} - x_{i,q})^2\right\} \tag{7.28}$$

By defining $\bar{\alpha}_i(m) = \log \alpha_i(m)$, $\bar{\beta}_i(m) = \log \beta_i(m)$ and $\bar{\gamma}_i^{(j)}(m', m) = \log \gamma_i^{(j)}(m', m)$, (7.26) can be written as

$$\Lambda(u_i) = \log\left(\frac{\sum_m \sum_{m'} \exp\left(\bar{\alpha}_{i-1}(m') + \bar{\gamma}_i^{(0)}(m', m) + \bar{\beta}_i(m)\right)}{\sum_m \sum_{m'} \exp\left(\bar{\alpha}_{i-1}(m') + \bar{\gamma}_i^{(1)}(m', m) + \bar{\beta}_i(m)\right)}\right), \tag{7.29}$$

and an algorithm that works in the log-domain is obtained.

The following expression, known as the Jacobian logarithm (Robertson *et al.* 1995), is used to avoid the sum of exponential terms,

$$\log\left(e^{\delta_1} + e^{\delta_2}\right) = \max(\delta_1, \delta_2) + \log\left(1 + e^{-|\delta_1 - \delta_2|}\right), \tag{7.30}$$

The function $\log\left(1+e^{-|\delta_1-\delta_2|}\right)$ can be stored in a small look-up table (LUT), as only a few values (eight reported in Robertson *et al.* (1995)) are required to achieve practically the same performance as the MAP algorithm. Therefore, instead of several calls to slow (or hardware-expensive) $\exp(x)$ functions, simple LUT accesses give practically the same result.

7.8.4 Max-Log-MAP algorithm

A more computationally efficient, albeit suboptimal derivative of the MAP algorithm is the *Max-Log-MAP algorithm*. It is obtained as before, by taking the logarithms of the MAP metrics and using the approximation (Robertson *et al.* 1995)

$$\log\left(e^{\delta_1}+e^{\delta_2}\right)\approx\max\left(\delta_1,\delta_2\right), \tag{7.31}$$

which is equal to the first term on the right-hand side of (7.30). As a result, the LLR of information bit u_i is given by

$$\begin{aligned}\Lambda(u_i)\approx&\max_{m',m}\left\{\bar{\alpha}_{i-1}(m')+\bar{\gamma}_i^{(0)}(m',m)+\bar{\beta}_i(m)\right\}\\&-\max_{m',m}\left\{\bar{\alpha}_{i-1}(m')+\bar{\gamma}_i^{(1)}(m',m)+\bar{\beta}_i(m)\right\}.\end{aligned} \tag{7.32}$$

The forward and backward computations can now be expressed as

$$\begin{aligned}\bar{\alpha}_i(m)&=\max_{m'}\max_{j\in\{0,1\}}\left\{\bar{\alpha}_{i-1}(m')+\bar{\gamma}_i^{(j)}(m',m)\right\},\\\bar{\beta}_i(m)&=\max_{m'}\max_{j\in\{0,1\}}\left\{\bar{\beta}_{i+1}(m')+\bar{\gamma}_i^{(j)}(m',m)\right\}.\end{aligned} \tag{7.33}$$

For binary codes based on rate-$1/n$ convolutional encoders, in terms of decoding complexity (measured in number of additions and multiplications), the SOVA algorithm requires the least amount, about half of that of the max-log-MAP algorithm. The log-MAP algorithm is approximately twice as complex compared to the max-log-MAP algorithm. In terms of performance, it has been shown (Fossorier *et al.* 1998) that the max-log-MAP algorithm is equivalent to a modified SOVA algorithm. The log-MAP and MAP algorithms have the same best error performance.

7.8.5 Soft-output OSD algorithm

The OSD algorithm of Section 7.5 can be modified to output the symbol reliabilities (Fossorier and Lin 1998). This modification is referred to as the *soft-output OSD*, or SO-OSD. The SO-OSD algorithm is a two-stage order-i reprocessing. The first stage is the same as conventional OSD, determining the most likely code word $\bar{v}_{\mathrm{ML}}$ up to order-i reprocessing. To describe the second stage, the following definitions are required.

For each most reliable position j, $1\le j\le K$, define the code word $\bar{v}_{\mathrm{ML}}(j)$ obtained by complementing position j in $\bar{v}_{\mathrm{ML}}$,

$$\bar{v}_{\mathrm{ML}}(j)=\bar{v}_{\mathrm{ML}}\oplus\bar{e}(j),$$

where $\bar{e}(j)$ is the set of all length-K vectors of Hamming weight one. The vector $\bar{e}(j)$ is the *coset representative* of the partition of code C_1 (equivalent to the original code after reordering, as in OSD, see Section 7.5) into two sets of code words, having the j-th position equal to zero, or one in code words of C. These sets, after removing position j, become punctured subcodes of C_1, and are denoted $C(0)$ and $C(1)$. Let $C(j) = C(0)$.

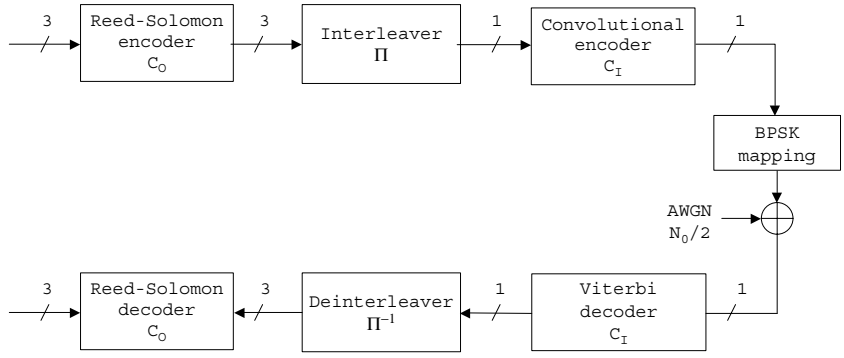
The SO-OSD algorithm consists of the following steps:

1. Determine the most likely code word $\bar{v}_{\mathrm{ML}}$ covered by order-i reprocessing.
2. For each subcode $C(j)$, $1 \le j \le K$,
   (a) Initialize all soft output values of the LRPs based on $\bar{v}_{\mathrm{ML}}$ and $\bar{v}_{\mathrm{ML}}(j)$. That is, compute
   $$L_j = r_j + \sum_{\substack{\ell \neq j \\ v_\ell(0) \neq v_\ell(1)}} m(v_\ell(1)) r_\ell, \tag{7.34}$$
   where $m(v_i) = (-1)^{v_i}$.
   (b) Determine the most likely code word $\bar{v}(j) \in C(j)$ covered by order-i reprocessing.
   (c) Evaluate L_j (7.34) on the basis of $\bar{v}_{\mathrm{ML}}$ and $\bar{v}(j)$.
   (d) Update soft output values of LRP in the positions of $\bar{v}_{\mathrm{ML}} \oplus \bar{v}(j)$ with L_j.
3. For $K + 1 \le j \le N$, choose the smallest output value associated with each LRP j.

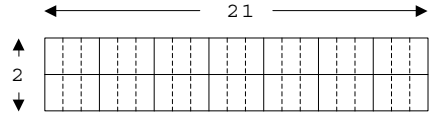
The performance of SO-OSD is the same as max-log-MAP decoding for many binary linear block codes of length up to 128 and high-rate codes (Fossorier and Lin 1998). In addition, scaling down the extrinsic values (7.34) improves the performance by a few tenths of a dB.

Problems

1. Let C be the binary memory-2 convolutional encoder with generators $(7, 5)$.
   (a) Simulate the performance of C over an AWGN channel with BPSK modulation and SD Viterbi decoding.
   (b) Use the bounds for hard- (HD) and soft-decision (SD) decoding and compare them with your simulations. What is the difference between required E_b/N_0 (dB) between HD and SD in order to achieve a probability of error $P_e \approx 10^{-4}$?
2. Repeat Problem 1 for the binary memory-2 recursive systematic convolutional (RSC) encoder with generators $(1, 5/7)$.
3. Let C_O be an RS $(7, 5, 3)$ code over $GF(2^3)$ and C_I be the binary rate-1/2 $K = 3$ non-systematic convolutional code with generators $\bar{g} = (7, 5)$. Consider a concatenated code $C_O \star C_I$ with C_O as outer code and C_I as inner code, as depicted below:



The interleaver Π is a standard 2×21 rectangular interleaver whose structure is depicted in the figure below:



The operation of the interleaver is as follows: Two RS code words – expressed as bits – are written in the array, row by row. The bits delivered to the binary convolutional encoder are then read from the array on a column-by-column basis.

The operation of the deinterleaver is similar, with a reversed order of writing and reading to and from the array: The received bits out of the VD are written to the array, two at a time and column-by-column. After 21 pairs of bits have been written, two words of length 21 bits (or 7 symbols over $GF(2^3)$) are read and delivered to the RS decoder.

(a) Evaluate the following bound on the performance of a concatenated coding scheme with an $RS(N, K, D)$ code C_O over $GF(2^m)$:

$$P_b \leq \frac{2^{m-1}}{2^m - 1} \sum_{i=t+1}^{n} \frac{i+t}{n} \binom{n}{i} P_s^i (1 - P_s)^{n-i},$$

where P_s is the symbol error probability at the input of the RS decoder,

$$P_s = 1 - (1 - p)^m,$$

and p is the probability of a bit error.

To estimate the value of p with the binary convolutional code C_I, use the bound expression for ML decoding discussed in class and in the homeworks.

You are asked to plot P_b as a function of E_b/N_0 (dB).

(b) Do a Monte Carlo simulation of the concatenated coding scheme and plot the resulting bit error rate (BER) in the same graph as P_b from part (a). How tight is the bound?

4. Consider the binary repetition $(5, 1, 5)$ code, denoted by C. In this problem you are asked to analyze the performance of C with binary transmission BPSK over an AWGN channel. Write computer programs to accomplish the following tasks:

   (a) Simulate the BER performance of ML decoding. (Hint: Look at the sign of the correlation $v = \sum_{i=1}^{5} r_i$, where $(r_1, r_2, r_3, r_4, r_5)$ is the received vector.)

   (b) Simulate the BER performance of Chase type-II algorithm. (Hint: $t = 2$.)

   (c) Plot the results of (a) and (b) together with the union bound.

5. Simulate the performance of SD Viterbi decoding of the binary rate-1/2 convolutional codes listed below and compare the plots. On the basis of your simulations, estimate the increment in coding gain that results from doubling the number of states of the encoder.

Constraint length, K	Generators (g_0, g_1)	d_{free}
4	(15,17)	6
5	(23,35)	7
6	(53,75)	8
7	(133,171)	10
9	(561,753)	12

6. Let C be a binary Hamming $(7, 4)$ code.

   (a) List all the 16 code words of C.

   (b) Build a computer model of an ML decoder of C for binary modulation over an AWGN channel. Assume that the mapping is "0" $\mapsto +1$ and "1" $\mapsto -1$. The decoder computes a total of 16 metrics, one per code word, and selects the code word with maximum metric as the ML code word. Simulate the performance of this ML decoder and compare with that of a hard-decision decoder.

   (c) Build a computer model of a soft-decision (SD) decoder of C using the Type-II Chase algorithm. Simulate the performance of this SD decoder and compare with the ML decoder.

   (d) Suppose that the received vector (from the outputs of the matched filter) is

   $$\bar{r} = (0.8, -0.1, 1.1, -0.2, 0.4, 0.6, 1.5).$$

   Using ordered-statistics decoding with order-1 reprocessing, determine the ML codeword.

7. Give a geometric interpretation of decoding of the binary repetition $(3, 1, 3)$ code with Chase type-II algorithm presented in Example 7.4.1.

8. Show that the Chase decoding algorithms (types 1 through 3) and ML decoding have the same asymptotic behavior. That is, the probability of error decreases exponentially with E_b/N_0 within a constant factor. (Hint: Section IV of Chase (1972).)

9. For the convolutional encoder and information sequence in Example 7.8.1, compute the soft outputs using the BCJR algorithm.

10. (Lin and Costello) Consider a memory-1 RSC encoder with generators $(1, 1/3)$. Using the zero-tail construction, a binary linear $(8, 3)$ code C is constructed.

    (a) What is the minimum distance of C?

    (b) Assume that the received sequence is

    $$\bar{r} = (+0.8, +0.1, +1.0, -0.5, -1.8, +1.1, +1.6, -1.6),$$

    which is normalized with respect to $\sqrt{E_s}$. Assume also that $E_s/N_0 = 1/4$. Using the BCJR (MAP) algorithm, determine the most likely information sequence.

8

Iteratively decodable codes

Iterative decoding may be defined as a technique employing a soft-output decoding algorithm that is iterated several times to improve the error performance of a coding scheme, with the aim of approaching true maximum-likelihood decoding (MLD) with least complexity. When the underlying error correcting code is well designed, increasing the number of iterations results in an improvement of the error performance.

Iterative decoding techniques date back to 1954 with the work of Elias (1954) on *iterated codes*. Later, in the 1960s, Gallager (1962) and Massey (1963) made important contributions. Iterative decoding in those days was referred to as *probabilistic decoding*. The main concept was then, as it is today, to maximize the *a posteriori probability* of a symbol being sent, given a noisy version of the coded sequence.

In this chapter, code constructions that can be iteratively decoded are presented. Elsewhere in the literature, these coding schemes have been named *turbo-like* codes. In terms of the application of iterative decoding algorithms, and for the purpose of this chapter, ECC schemes can be broadly classified into two classes:

Product codes

Examples of codes in this class include *parallel concatenated codes* or *turbo codes* and *serially concatenated codes*. They have found applications in third-generation (3G) cellular systems and deep-space communications. Members of this class are also block product codes, presented in Section 6.2.4.

Low-density parity-check (LDPC) codes

These are linear codes with the property that their parity-check matrix has a small ratio of number of nonzero elements to the total number of elements. LDPC codes have found applications in second-generation satellite broadcasting systems (DVB-S2), in gigabit ethernet systems (10Gbase-T), and are being considered in high-speed wireless LAN systems (IEEE 802.3an).

In both the above classes of codes, the component codes can be either convolutional or block codes, with systematic or nonsystematic encoding, or any combination thereof. To provide

The Art of Error Correcting Coding, Second Edition Robert H. Morelos-Zaragoza
 ISBN: 0-470-01558-6

a motivation for the study of iterative decoding techniques, the fundamental structure of turbo codes is discussed next.

Turbo codes

Over the past eight years, there has been an enormous amount of research effort dedicated to the analysis of iterative decoding algorithms and the construction of *iteratively decodable codes* or "turbo-like codes" that approach the *Shannon limit*.[1] Turbo codes were introduced in 1993 (Berrou *et al.* 1993). The results reported there sent a shock wave throughout the research community.

With binary transmission over an AWGN channel, a rate-1/2 coding scheme, based on a combination of two rate-1/2 16-state RSC codes, connected with an interleaver of size 256×256, achieved BER rates of 10^{-5} at a signal-to-noise ratio (SNR) per bit, or E_b/N_0 of 0.7 dB very close to the Shannon limit (0.2 dB).

The basic elements in the turbo coding scheme proposed in Berrou *et al.* (1993) are the following:

Coding structure

A two-dimensional product code[2] structure with constituent *recursive systematic* convolutional encoders.

Reliability-based decoding

Employ *soft-input soft-output (SISO)* maximum a posteriori (MAP) decoders for the component codes in order to generate *log-likelihood ratios*.

Iterative decoding

The use of *feedback* of part of the symbol reliabilities, in the form of *extrinsic information* from an outer (column) decoder to an inner (row) decoder, and vice versa.

Random permutation

A long *random interleaver* applied between the two encoders. Its main function is to ensure that, at each iteration, the component MAP decoders get independent estimates on the information symbols.

These four elements have been extensively studied over the years. References Heegard and Wicker 1999; ISTC 1997, 2000; JSAC-I 2001; JSAC-II 2001; Vucetic and J. Yuan 2000 continue to be a good sample of the research efforts.

Log-likelihood ratio (LLR) value

In the iterative decoding algorithms presented next, the LLR log-likelihood ratio (LLR) value becomes very important and is defined as

$$\Lambda(u) \triangleq \log\left(\frac{\Pr\{u=1\}}{\Pr\{u=0\}}\right) = \log\left(\frac{\Pr\{m(u)=-1\}}{\Pr\{m(u)=+1\}}\right) = \log\left(\frac{p_1}{p_0}\right),$$

[1] The Shannon limit refers to the capacity of a discrete-input continuous output channel.

[2] A product code is referred to in the original paper (Berrou *et al.* 1993) as a *parallel concatenated code*.

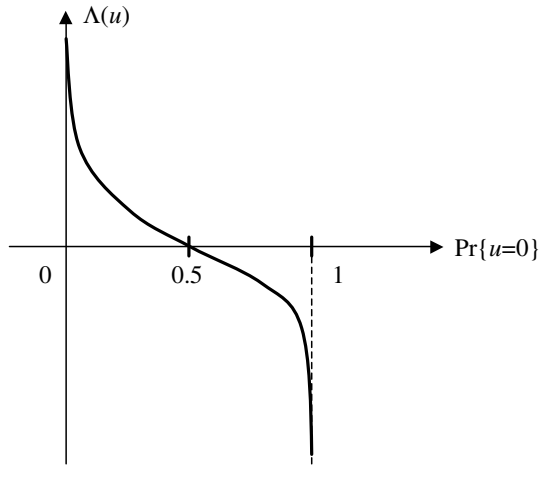


Figure 8.1 The LLR value as a function of the a priori bit probability.

where the modulated symbol

$$m(u) \stackrel{\Delta}{=} (-1)^u, \quad u \in \{0, 1\},$$

and

$$p_0 \stackrel{\Delta}{=} \Pr\{u = 0\}, \quad p_1 \stackrel{\Delta}{=} \Pr\{u = 1\}.$$

Then the a priori bit probabilities can be written in terms of the LLR value as follows (Figure 8.1):

$$p_0 = \frac{1}{1 + e^{\Lambda(u)}}, \quad p_1 = \frac{e^{\Lambda(u)}}{1 + e^{\Lambda(u)}}.$$

On the basis of its LLR value, the hard decision on a bit u is given by

$$\hat{u} = 1 \oplus \operatorname{sgn}(\Lambda(u)) = \begin{cases} 1, & \Lambda(u) \geq 0; \\ 0, & \Lambda(u) < 0. \end{cases}$$

The log-likelihood ratio value of a parity check (Hagenauer *et al.* 1996)

Let $p_i(j) \stackrel{\Delta}{=} \Pr\{x_i = j\}$, for $i = 1, 2$ and $j = 0, 1$. Then

$$\Pr\{x_1 \oplus x_2 = 1\} = p_1(0)p_2(1) + p_1(1)p_2(0),$$
$$\Pr\{x_1 \oplus x_2 = 0\} = p_1(0)p_2(0) + p_1(1)p_2(1),$$

and

$$\Lambda(x_1 \oplus x_2) = \log\left[\frac{e^{\Lambda(x_1)} + e^{\Lambda(x_2)}}{1 + e^{\Lambda(x_1)+\Lambda(x_2)}}\right].$$

8.1 Iterative decoding

The purpose of this section is to introduce basic concepts behind iterative decoding. Consider the a posteriori LLR[3] of an information symbol, Equation (7.12) repeated here for convenience,

$$\Lambda(u_i) \triangleq \log\left(\frac{\Pr\{u_i = 1|\bar{r}\}}{\Pr\{u_i = 0|\bar{r}\}}\right). \tag{8.1}$$

Let $C_i(0)$ and $C_i(1)$ denote the set of modulated coded sequences $\bar{x} = m(\bar{v}) \in m(C)$ such that the i-th position in $\bar{v} \in C$ is equal to zero and one, respectively. Recall that modulation was a one-to-one and onto mapping from $v \in \{0, 1\}$ to $x \in \{+1, -1\}$ as in Equation (7.2). Then the symbol LLR (8.1) can be written as

$$\begin{aligned}\Lambda(u_i) &= \log\left(\frac{\displaystyle\sum_{\bar{x}\in C_i(1)}\prod_{\ell=1}^{N} p(r_\ell, x_\ell)}{\displaystyle\sum_{\bar{x}\in C_i(0)}\prod_{\ell=1}^{N} p(r_\ell, x_\ell)}\right) \\ &= \log\left(\frac{p(r_i|x_i = -1)\Pr\{x_i = -1\}\displaystyle\sum_{\bar{x}\in C_i(1)}\prod_{\substack{\ell=1\\ \ell\neq i}}^{N} p(r_\ell, x_\ell)}{p(r_i|x_i = +1)\Pr\{x_i = +1\}\displaystyle\sum_{\bar{x}\in C_i(0)}\prod_{\substack{\ell=1\\ \ell\neq i}}^{N} p(r_\ell, x_\ell)}\right),\end{aligned} \tag{8.2}$$

from which it follows that the LLR of an information symbol can be expressed as

$$\Lambda(u_i) = \Lambda_{ci} + \Lambda_a(u_i) + \Lambda_{i,e}(C), \tag{8.3}$$

where Λ_{ci} is known as the *channel LLR* and given by

$$\begin{aligned}\Lambda_{ci} &= \log\left(\frac{p(r_i|x_i = -1)}{p(r_i|x_i = +1)}\right) \\ &= \begin{cases} -(-1)^{r_i}\log\left(\frac{1-p}{p}\right), & \text{for a BSC channel with parameter } p; \\ -\dfrac{4}{N_0}r_i, & \text{for an AWGN channel } (\sigma_n^2 = N_0/2); \text{ and} \\ -\dfrac{4}{N_0}a_i r_i, & \text{for a flat Rayleigh fading channel.} \end{cases}\end{aligned} \tag{8.4}$$

$$\Lambda_a(u_i) = \log\left(\frac{\Pr\{x_i = -1\}}{\Pr\{x_i = +1\}}\right) = \log\left(\frac{\Pr\{u_i = 1\}}{\Pr\{u_i = 0\}}\right) \tag{8.5}$$

[3]The LLR is also referred to as "L-value" by some authors.

is the *a priori LLR* of the information bit, and

$$\Lambda_{i,e}(C) = \log \left(\frac{\displaystyle\sum_{\bar{x} \in C_i(1)} \prod_{\substack{\ell=1 \\ \ell \neq i}}^{N} p(r_\ell, x_\ell)}{\displaystyle\sum_{\bar{x} \in C_i(0)} \prod_{\substack{\ell=1 \\ \ell \neq i}}^{N} p(r_\ell, x_\ell)} \right) \tag{8.6}$$

is the *extrinsic LLR*, which is specified by the constraints imposed by the code on the *other information symbols*.

Assume that binary transmission is performed over an AWGN channel and that binary rate-$1/n$ RSC encoders are used as components. In an iterative decoding procedure, the extrinsic information provided by $\Lambda_{i,e}(C)$ can be fed back to the decoder as a priori probability for a second round of decoding. In terms of forward–backward decoding using a trellis (see Section 7.8.2), the extrinsic LLR can be written as

$$\Lambda_{i,e}(C) = \log \left(\frac{\displaystyle\sum_{(m,m') \in B_i(1)} \alpha_{i-1}(m') \xi_i(m', m) \beta_i(m)}{\displaystyle\sum_{(m,m') \in B_i(0)} \alpha_{i-1}(m') \xi_i(m', m) \beta_i(m)} \right), \tag{8.7}$$

where $\alpha_i(m)$ and $\beta_i(m)$ are given by Equations (7.22) and (7.25), respectively.

A *modified branch metric* is needed to compute the extrinsic LLR

$$\xi_i(m', m) = \delta_{ij}(m, m') \exp \left(\frac{E}{N_0} \sum_{q=1}^{n-1} r_{i,q} x_{i,q} \right), \tag{8.8}$$

where, as before,

$$\delta_{ij}(m, m') = \begin{cases} 1, & \text{if } (m', m) \in B_i^{(j)} \\ 0, & \text{otherwise.} \end{cases}$$

This extrinsic LLR, for an information position i, does not contain any variable directly related to u_i, for $i = 1, 2, \ldots, N$. It should be noted that because of the assumption that encoding is systematic, the branch labels in the trellis are of the form $(u_i, v_{i,1}, \ldots, v_{i,n-1})$, and therefore the sum in the modified branch metric (8.8) starts at $q = 1$.[4]

In the more general case of a two-dimensional product coding scheme, the first (e.g., row) decoder produces $\Lambda_{i,e}^{(1)}(C)$ which is given to the second (e.g., column) decoder as a priori probability $\Lambda_a^{(2)}(u_i)$ to be used in the computation of the LLR of information symbol u_i. In other words, the extrinsic information provides a soft output that involves only soft inputs (reliabilities) that are not directly related to the information symbol u_i. Iterative decoding of product codes is the topic of the next section.

[4]Compare this with the expression of the branch metric $\gamma_i^{(j)}(m', m)$ in (7.24).

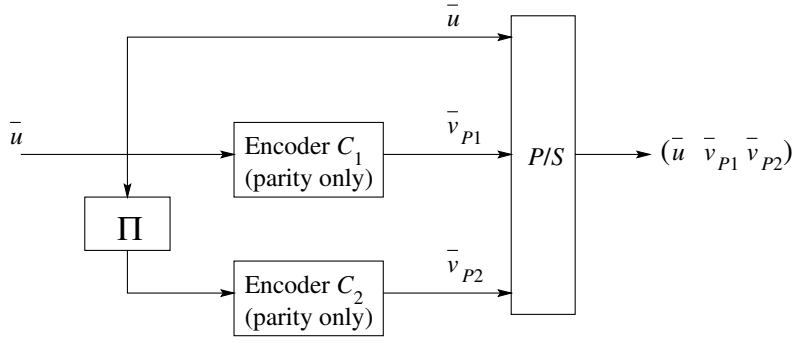


Figure 8.2 Encoder structure of a parallel concatenated (turbo) code.

8.2 Product codes

In this section, coding schemes based on products of codes with interleaving are presented. These schemes allow the use of simple decoders for the component codes. With iterative decoding, symbol-by-symbol MAP decoders of the component codes can exchange extrinsic LLR values, generally improving the reliability of the estimated information bits, with an increasing number of iterations.

8.2.1 Parallel concatenation: Turbo codes

Figure 8.2 shows the block diagram of an encoder of a *parallel concatenated code*, better known as a *turbo code* (Battail 1998; Berrou and Glavieux 1998; Divsalar and Pollara 1995; ITCG 2001). Two encoders are used, each delivering as output only redundant symbols. For instance, if rate-1/2 RSC codes are used, the encoders represent the term $\bar{g}_1(D)/\bar{g}_0(D)$ in the polynomial generator matrix $G(D)$. The inputs to the encoders are $\bar{u}$ and $\Pi\bar{u}$, where Π denotes a *permutation matrix*[5] associated with the interleaver. The output corresponding to an input symbol u_i is $\begin{pmatrix} u_i & v_{P1,i} & v_{P2,i} \end{pmatrix}$, with $v_{P1,i}$ and $v_{P2,i}$ representing the redundant symbols, for $i = 1, 2, \ldots, N$, where N is the block length.

The original turbo code scheme used RSC encoders. However, to examine this coding scheme, assume that the component codes are two binary *systematic* linear block (n_i, k_i) codes, $i = 1, 2$. The rate of the overall code is (Figure 8.3)

$$R = \frac{K}{N} = \frac{k_1 k_2}{n_1 n_2 - (n_1 - k_1)(n_2 - k_2)} = \frac{R_1 R_2}{R_1 + R_2 + R_1 R_2}, \tag{8.9}$$

where $R_i \stackrel{\Delta}{=} k_i/n_i$, with $i = 1, 2$. Let $G_i = \begin{pmatrix} I_i | P_i \end{pmatrix}$ denote the generator matrix of code C_i, $i = 1, 2$. Then the generator matrix of the parallel concatenated code C_{PC} can be put in the

[5] A permutation matrix is a binary matrix with only one nonzero entry per row and per column; if $\pi_{i,j} = 1$ then a symbol in position i at the input is sent to position j at the output.

following form:

$$G_{\mathrm{PC}} = \left(I_{k_1 k_2} \quad \begin{pmatrix} P_1 & & & \\ & P_1 & & \\ & & \ddots & \\ & & & P_1 \end{pmatrix} \quad \Pi \begin{pmatrix} P_2 & & & \\ & P_2 & & \\ & & \ddots & \\ & & & P_2 \end{pmatrix} \right)$$
$$= \left(I_{k_1 k_2} \mid P_1' \mid P_2' \right), \tag{8.10}$$

where Π is the permutation matrix associated with the interleaver, and P_i is the parity submatrix of code C_i, $i = 1, 2$. The number of times that P_1 appears in the middle part P_1' of G_{PC} is k_2, while the number of times that P_2 appears in the leftmost portion P_2' of G_{PC} is k_1. All other entries in P_1' and P_2' are zero. It follows that codewords of C_{PC} are of the form $\left(\bar{u} \mid \bar{u} P_1' \mid \bar{u} P_2'\right)$.

Example 8.2.1 *Let C_1 be a binary repetition $(2, 1, 2)$ code and C_2 be a binary SPC $(3, 2, 2)$ code. Then $k_1 k_2 = 2$. There is only one possible permutation matrix Π in this case. The parity submatrices and permutation matrix are*

$$P_1 = \begin{pmatrix} 1 \end{pmatrix}, \quad P_2 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \quad \Pi = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix},$$

respectively. From Equation (8.10), it follows that the generator matrix of the parallel concatenated $(5, 2)$ code C_{PC} is

$$G_{\mathrm{PC}} = \left(\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \quad \begin{pmatrix} (1) & 0 \\ 0 & (1) \end{pmatrix} \quad \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \end{pmatrix} \right)$$
$$= \left(\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \quad \begin{pmatrix} (1) & 0 \\ 0 & (1) \end{pmatrix} \quad \begin{pmatrix} 1 \\ 1 \end{pmatrix} \right)$$
$$= \begin{pmatrix} 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 \end{pmatrix}.$$

Note that the minimum distance is $d_{\mathrm{PC}} = 3$, which is less than that of the product of C_1 and C_2.

One way to interpret a parallel concatenated (turbo) code, from the point of view of linear block codes, is as a *punctured product code* in which the redundant symbols corresponding to the checks-on-checks are deleted. This interpretation is shown in Figure 8.3. The only essential difference between a turbo code and a block product code is that the interleaver is not simply a row-by-row, column-by-column interleaver, but one that introduces sufficient disorder in the information sequence so that iterative decoding works. When interpreted as a linear block code, a turbo code can be further interpreted as a *generalized concatenated code*[6], after replacing the puncturing operation with an equivalent multiplication by a binary matrix and a direct sum. Let M_1 be a $1 \times N$ matrix with $n_1 k_2$ successive ones followed by $N - n_1 k_2$ zeros. Let M_2 be another $1 \times N$ matrix with $N - (n_2 - k_2) k_1$ zeros followed by $(n_2 - k_2) k_1$ ones. Then a turbo encoder can be visualized as an encoder of a GC code as shown in Figure 8.4.

[6]See Section 6.2.6.

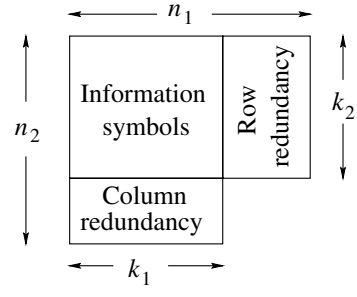


Figure 8.3 A parallel concatenated code interpreted as a *punctured product code*.

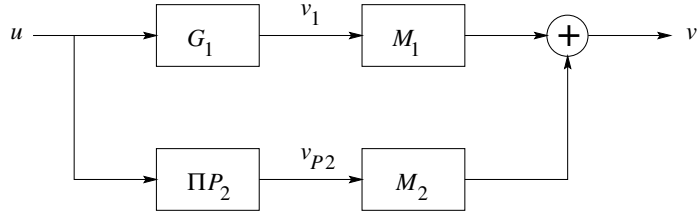


Figure 8.4 A turbo encoder as an encoder of a *generalized concatenated code*.

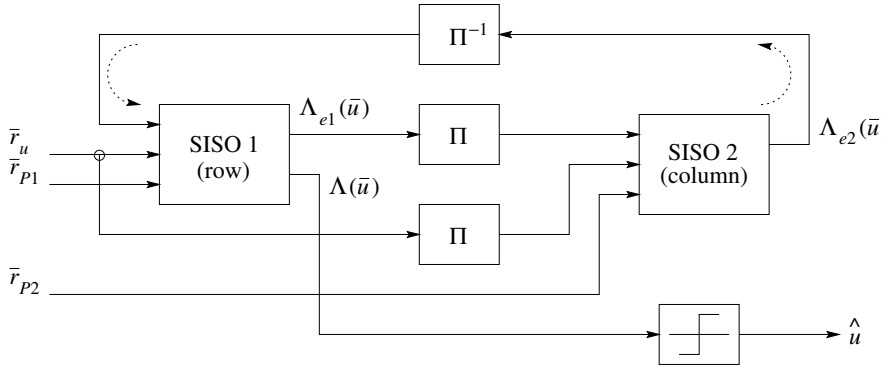


Figure 8.5 Block diagram of an iterative decoder for a parallel concatenated code.

Iterative decoding of parallel concatenated codes

The basic structure of an iterative decoder for a parallel concatenated coding scheme with two component codes is shown in Figure 8.5. Each iteration consists of two phases, one decoding phase per component decoder.

First phase

In the first decoding iteration, first phase, the soft-in soft-out (SISO) decoder for the first component code computes the a posteriori LLR (8.7) assuming equally likely symbols,

that is, $\Lambda_a(\bar{u}) = 0$. This decoder computes the extrinsic information for each information symbol, $\Lambda_{e1}(\bar{u})$, on the basis of the part of the received sequence that corresponds to the parity symbols, $\bar{r}_{P1}$, and sends the result to the second SISO decoder.

Second phase

In the second phase of the first decoding iteration, the permuted (or interleaved) extrinsic information from the first decoder is used as a priori LLR, $\Lambda_a(\bar{u}) = \Pi\Lambda_{e1}(\bar{u})$. Extrinsic information $\Lambda_{e2}(\bar{u})$ is computed on the basis of the part of the received sequence that corresponds to the parity symbols of the second component code, $\bar{r}_{P2}$, thus terminating the first decoding iteration. At this point, a decision can be made on an information symbol, on the basis of its a posteriori LLR $\Lambda(u)$.

In subsequent iterations, the first decoder uses the deinterleaved extrinsic information from the second decoder, $\Pi^{-1}\Lambda_{e2}(\bar{u})$, as a priori LLR for the computation of the soft-output (the a posteriori LLR), $\Lambda(\bar{u})$. This procedure can be repeated until either a stopping criterion is met (Hagenauer *et al.* 1996; Shao *et al.* 1999) or a maximum number of iterations is performed. It should be noted that making decisions on the information symbols after the first (row) decoder saves one deinterleaver.

As in Section 7.8.4, the iterative MAP decoding algorithm described above can be simplified by using the approximation of the "log" by the "max" as in Equation (7.31). Also, there is an iterative version of the SOVA algorithm presented in Section 7.8.1 that essentially computes the a posteriori LLR $\Lambda(\bar{u})$ directly and then, subtracting the channel LLR $\bar{\Lambda}_c$ and the extrinsic LLR $\Lambda_{ej}(\bar{u})$ from the other decoder, produces the extrinsic LLR (Feng and Vucetic 1997).

Example 8.2.2 *Consider the turbo* $(5, 2, 3)$ *code of Example 8.2.1. A codeword is arranged in a* 3×2 *array with the lower right corner position unused (or deleted). Suppose that the following codeword* $\bar{v}$ *is transmitted*

$$\bar{v} = \begin{vmatrix} u_1 & v_{11} \\ u_2 & v_{12} \\ v_{21} & \end{vmatrix} = \begin{vmatrix} 0 & 0 \\ 1 & 1 \\ 1 & \end{vmatrix}$$

mapped to

$$m(\bar{v}) = \begin{vmatrix} +1 & +1 \\ -1 & -1 \\ -1 & \end{vmatrix},$$

through an AWGN channel and received with the following channel LLR values (recall that $\Lambda_{c_i} = -\frac{4}{N_0} r_i$*):*

$$\Lambda_c = \begin{vmatrix} \Lambda_{c_1} & \Lambda(r_{11}) \\ \Lambda_{c_2} & \Lambda(r_{12}) \\ \Lambda(r_{21}) & \end{vmatrix} = \begin{vmatrix} -0.5 & +2.5 \\ -3.5 & +1.0 \\ +0.5 & \end{vmatrix}.$$

The hard decisions based on the channel LLR values are

$$\hat{v} = \begin{vmatrix} \hat{u}_1 & \hat{v}_{11} \\ \hat{u}_2 & \hat{v}_{12} \\ \hat{v}_{21} & \end{vmatrix} = \begin{vmatrix} 0 & 1 \\ 0 & 1 \\ 1 & \end{vmatrix}.$$

There are two errors (compare with codeword) that cannot be corrected.

Iteration # 1

Phase 1 (Row decoding)

$$\Lambda_{e_1}(u_1) = \Lambda(r_{11}) = \underline{+2.5}$$
$$\Lambda_{e_1}(u_2) = \Lambda(r_{12}) = \underline{+1.0}$$

Updated (partial) LLRs:

$$\Lambda'(u_1) \triangleq \Lambda_{c_1} + \Lambda_{e_1}(u_1) = -0.5 + 2.5 = \underline{+2.0}$$
$$\Lambda'(u_2) \triangleq \Lambda_{c_2} + \Lambda_{e_1}(u_2) = -3.5 + 1.0 = \underline{-2.5}$$

Phase 2 (Column decoding)

$$\Lambda_{e_2}(u_1) = \Lambda(u_2 \oplus r_{21}) = \log\left[\frac{e^{\Lambda'(u_2)} + e^{\Lambda(r_{21})}}{1 + e^{\Lambda'(u_2)+\Lambda(r_{21})}}\right] = \log\left[\frac{e^{-2.5} + e^{0.5}}{1 + e^{-2.0}}\right] = \underline{0.42}$$
$$\Lambda_{e_2}(u_2) = \Lambda(u_1 \oplus r_{21}) = \log\left[\frac{e^{+2.0} + e^{0.5}}{1 + e^{-2.5}}\right] = \underline{-0.38}$$

A posteriori LLRs

$$\Lambda(u_1) = \Lambda_{c_1} + \Lambda_{e_1}(u_1) + \Lambda_{e_2}(u_1) = \underline{+2.42}$$
$$\Lambda(u_2) = \Lambda_{c_2} + \Lambda_{e_1}(u_2) + \Lambda_{e_2}(u_2) = \underline{-2.88}$$

Iteration # 2

Phase 1 (Row decoding)

$$\Lambda_{e_1}(u_1) = \Lambda(r_{11}) + \Lambda_{e_2}(u_1) = +2.5 + 0.42 = \underline{+2.92}$$
$$\Lambda_{e_1}(u_2) = \Lambda(r_{12}) + \Lambda_{e_2}(u_2) = +1.0 - 0.38 = \underline{+0.62}$$

Updated (partial) LLRs:

$$\Lambda'(u_1) = \Lambda_{c_1} + \Lambda_{e_1}(u_1) = -0.5 + 2.92 = \underline{+2.42}$$
$$\Lambda'(u_2) = \Lambda_{c_2} + \Lambda_{e_1}(u_2) = -3.5 + 0.62 = \underline{-2.88}$$

Phase 2 (Column decoding)

$$\Lambda_{e_2}(u_1) = \Lambda(u_2 \oplus r_{21}) = \log\left[\frac{e^{-2.88} + e^{0.5}}{1 + e^{-2.38}}\right] = \underline{0.44}$$
$$\Lambda_{e_2}(u_2) = \Lambda(u_1 \oplus r_{21}) = \log\left[\frac{e^{+2.42} + e^{0.5}}{1 + e^{+2.92}}\right] = \underline{-0.41}$$

A posteriori LLRs

$$\Lambda(u_1) = \Lambda_{c_1} + \Lambda_{e_1}(u_1) + \Lambda_{e_2}(u_1) = \underline{+2.86}$$
$$\Lambda(u_2) = \Lambda_{c_2} + \Lambda_{e_1}(u_2) + \Lambda_{e_2}(u_2) = \underline{-3.29}$$

The reliability (magnitude of the LLR values) increases in the second iteration. On the basis of the a posteriori LLRs after two iterations, the hard decisions (HDs) are $\hat{u}_1 = 1$ *and* $\hat{u}_2 = 0$. *These are still incorrect. However, given the channel LLRs, the decoder was able to flip one information bit* (u_1) *so that most of the parity checks are satisfied. It should be noted that after the second iteration, the soft outputs changed because of the influence of the extrinsic LLR computed in the first iteration. In particular, the reliability of the first bit increases from* $|\Lambda(u_1)| = +2.42$ *to* $|\Lambda(u_1)| = +2.86$.

Example 8.2.3 *Consider now the product of two binary SPC (3,2,2) codes.*[7] *A codeword is arranged in a* 3×3 *array with the lower right corner position deleted. Suppose that the following codeword* $\bar{v}$ *is transmitted*

$$\hat{v} = \begin{vmatrix} u_1 & u_2 & v_{11} \\ u_3 & u_4 & v_{12} \\ v_{21} & v_{22} & \end{vmatrix} = \begin{vmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & \end{vmatrix},$$

mapped to

$$m(\hat{v}) = \begin{vmatrix} -1 & +1 & -1 \\ +1 & -1 & -1 \\ -1 & -1 & \end{vmatrix},$$

through an AWGN channel and with channel LLR values ($\Lambda_{c_i} = -\frac{4}{N_0} r_i = -2r_i$, *assuming that* $N_0 = 2$*):*

$$\Lambda_c = \begin{vmatrix} \Lambda_{c_1} & \Lambda_{c_2} & \Lambda(r_{11}) \\ \Lambda_{c_3} & \Lambda_{c_4} & \Lambda(r_{12}) \\ \Lambda(r_{21}) & \Lambda(r_{22}) & \end{vmatrix} = \begin{vmatrix} +1.5 & +0.1 & +2.5 \\ +0.2 & +0.3 & +2.0 \\ +6.0 & +1.0 & \end{vmatrix}.$$

The hard decisions based on the channel LLR values are:

$$\hat{v} = \begin{vmatrix} \hat{u}_1 & \hat{u}_2 & \hat{v}_{11} \\ \hat{u}_3 & \hat{u}_4 & \hat{v}_{12} \\ \hat{v}_{21} & \hat{v}_{22} & \end{vmatrix} = \begin{vmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & \end{vmatrix}.$$

There are two errors in $\bar{v}$ *that cannot be corrected as the minimum distance of the product code is 4.*

Iteration # 1

Phase 1 (Row decoding)

$$\Lambda_{e_1}(u_1) = \Lambda(u_2 \oplus r_{11}) \stackrel{\Delta}{=} \log\left[\frac{e^{\Lambda_{c_2}} + e^{\Lambda(r_{11})}}{1 + e^{\Lambda_{c_2} + \Lambda(r_{21})}}\right] = \log\left[\frac{e^{+0.1} + e^{+2.5}}{1 + e^{+2.6}}\right] = \underline{-0.08}$$

$$\Lambda_{e_1}(u_2) = \Lambda(u_1 \oplus r_{11}) = \log\left[\frac{e^{+1.5} + e^{+2.5}}{1 + e^{+4.0}}\right] = \underline{-1.2}$$

$$\Lambda_{e_1}(u_3) = \Lambda(u_4 \oplus r_{12}) = \log\left[\frac{e^{+0.3} + e^{+2.0}}{1 + e^{+2.3}}\right] = \underline{-0.23}$$

$$\Lambda_{e_1}(u_4) = \Lambda(u_3 \oplus r_{12}) = \log\left[\frac{e^{+0.2} + e^{+2.0}}{1 + e^{+2.2}}\right] = \underline{-0.15}$$

[7] This is the same example presented in Sklar (1997).

Updated (partial) LLRs:

$$\Lambda'(u_1) = \Lambda_{c_1} + \Lambda_{e_1}(u_1) = +1.5 - 0.08 = \underline{+1.42}$$
$$\Lambda'(u_2) = \Lambda_{c_2} + \Lambda_{e_1}(u_2) = +0.1 - 1.2 = \underline{-1.1}$$
$$\Lambda'(u_3) = \Lambda_{c_3} + \Lambda_{e_1}(u_3) = +0.2 - 0.23 = \underline{-0.03}$$
$$\Lambda'(u_4) = \Lambda_{c_4} + \Lambda_{e_1}(u_4 2) = +0.3 - 0.15 = \underline{-0.15}$$

Phase 2 (Column decoding)

$$\Lambda_{e_2}(u_1) = \Lambda(u_3 \oplus r_{21}) = \log\left[\frac{e^{-0.03} + e^{+6.0}}{1 + e^{+5.97}}\right] = \underline{+0.03}$$
$$\Lambda_{e_2}(u_2) = \Lambda(u_4 \oplus r_{22}) = \log\left[\frac{e^{+0.15} + e^{+1.0}}{1 + e^{+1.15}}\right] = \underline{-0.06}$$
$$\Lambda_{e_2}(u_3) = \Lambda(u_1 \oplus r_{21}) = \log\left[\frac{e^{+1.42} + e^{+6.0}}{1 + e^{+7.42}}\right] = \underline{-1.41}$$
$$\Lambda_{e_2}(u_4) = \Lambda(u_2 \oplus r_{22}) = \log\left[\frac{e^{-1.1} + e^{+1.0}}{1 + e^{-0.1}}\right] = \underline{+0.47}$$

Updated (partial) LLRs:

$$\Lambda''(u_1) = \Lambda_{c_1} + \Lambda_{e_2}(u_1) = +1.5 + 0.03 = \underline{+1.53}$$
$$\Lambda''(u_2) = \Lambda_{c_2} + \Lambda_{e_2}(u_2) = +0.1 - 0.06 = \underline{+0.04}$$
$$\Lambda''(u_3) = \Lambda_{c_3} + \Lambda_{e_2}(u_3) = +0.2 - 1.41 = \underline{-1.21}$$
$$\Lambda''(u_4) = \Lambda_{c_4} + \Lambda_{e_2}(u_4) = +0.3 + 0.47 = \underline{+0.77}$$

Iteration # 2

Phase 1 (Row decoding)

$$\Lambda_{e_1}(u_1) = \Lambda(u_2 \oplus r_{11}) \triangleq \log\left[\frac{e^{\Lambda''(u_2)} + e^{\Lambda(r_{11})}}{1 + e^{\Lambda''(u_2)+\Lambda(r_{21})}}\right] = \log\left[\frac{e^{+0.04} + e^{+2.5}}{1 + e^{+2.54}}\right] = \underline{-0.03}$$
$$\Lambda_{e_1}(u_2) = \Lambda(u_1 \oplus r_{11}) = \log\left[\frac{e^{+1.53} + e^{+2.5}}{1 + e^{+4.03}}\right] = \underline{-1.24}$$
$$\Lambda_{e_1}(u_3) = \Lambda(u_4 \oplus r_{12}) = \log\left[\frac{e^{+0.77} + e^{+2.0}}{1 + e^{+2.77}}\right] = \underline{-0.57}$$
$$\Lambda_{e_1}(u_4) = \Lambda(u_3 \oplus r_{12}) = \log\left[\frac{e^{-1.2} + e^{+2.0}}{1 + e^{+0.8}}\right] = \underline{+0.87}$$

Updated (partial) LLRs:

$$\Lambda'(u_1) = \Lambda_{c_1} + \Lambda_{e_1}(u_1) = +1.5 - 0.03 = \underline{+1.47}$$
$$\Lambda'(u_2) = \Lambda_{c_2} + \Lambda_{e_1}(u_2) = +0.1 - 1.24 = \underline{-1.14}$$
$$\Lambda'(u_3) = \Lambda_{c_3} + \Lambda_{e_1}(u_3) = +0.2 - 1.57 = \underline{-1.37}$$
$$\Lambda'(u_4) = \Lambda_{c_4} + \Lambda_{e_1}(u_4) = +0.3 + 0.87 = \underline{+1.17}$$

Phase 2 (Column decoding)

$$\Lambda_{e_2}(u_1) = \Lambda(u_3 \oplus r_{21}) = \log\left[\frac{e^{-1.37} + e^{+6.0}}{1 + e^{+4.63}}\right] = \underline{+1.36}$$
$$\Lambda_{e_2}(u_2) = \Lambda(u_4 \oplus r_{22}) = \log\left[\frac{e^{+1.17} + e^{+1.0}}{1 + e^{+2.17}}\right] = \underline{-0.50}$$
$$\Lambda_{e_2}(u_3) = \Lambda(u_1 \oplus r_{21}) = \log\left[\frac{e^{+1.47} + e^{+6.0}}{1 + e^{+7.47}}\right] = \underline{-1.46}$$
$$\Lambda_{e_2}(u_4) = \Lambda(u_2 \oplus r_{22}) = \log\left[\frac{e^{-1.14} + e^{+1.0}}{1 + e^{-0.14}}\right] = \underline{+0.49}$$

A posteriori LLR values (soft outputs) after two iterations:

$$\Lambda(u_1) = \Lambda_{c_1} + \Lambda_{e_1}(u_1) + \Lambda_{e_2}(u_1) = +1.5 - 0.03 + 1.36 = \underline{+2.83}$$
$$\Lambda(u_2) = \Lambda_{c_2} + \Lambda_{e_1}(u_2) + \Lambda_{e_2}(u_2) = +0.1 - 1.24 - 0.50 = \underline{-1.64}$$
$$\Lambda(u_3) = \Lambda_{c_3} + \Lambda_{e_1}(u_3) + \Lambda_{e_2}(u_3) = +0.2 - 1.57 - 1.46 = \underline{-2.83}$$
$$\Lambda(u_4) = \Lambda_{c_4} + \Lambda_{e_1}(u_4) + \Lambda_{e_2}(u_4) = +0.3 + 0.87 + 0.49 = \underline{+1.66}$$

As in Example 1, the reliability (absolute value of the LLRs) increases in the second iteration. On the basis of a posteriori LLR values after two iterations, the hard decisions are $\hat{u}_1 = 1$, $\hat{u}_2 = 0$, $\hat{u}_3 = 0$, *and* $\hat{u}_4 = 1$. *The errors have been corrected with a relatively high degree of confidence (reliability).*

The results are the same as those presented in Sklar (1997), although the numbers are different because here the LLR values are exact computations. Also, care must be taken with the signs of the LLR values, which differ because of a different definition.

Error performance

With iterative decoding, it is shown in Berrou *et al.* (1993) that as the number of iterations grows, the error performance improves. Typical of turbo coding schemes is the fact that increasing the number of iterations results in a monotonically decreasing improvement in coding gain. Increasing the number of iterations from 2 to 6 gives an improvement in SNR of 1.7 dB, whereas going from 6 to 18 iterations yields only a 0.3 dB improvement in coding gain.

Since the appearance of turbo codes, advances have taken place in understanding the BER behavior of turbo codes. At the time of writing this, there appears to be a consensus among researchers on why turbo codes offer such an excellent error performance:

- Turbo codes have a weight distribution that approaches, for long interleavers, that of random codes.
- Recursive convolutional encoders and proper interleaving map most of the low-weight information sequences into high-weight coded sequences.
- Systematic encoders allow the effective use of iterative decoding techniques utilizing constituent MAP decoders. Information symbol estimates are available directly from the channel.

In addition, a number of interesting semianalytical tools have appeared, in density evolution (Richardson and Urbanke 2001), Gaussian approximation (Gamal and Hammons 2001), and mutual information (ten Brink 1999, 2001) or SNR transfer (Divsalar *et al.* 2001) characteristics, to study the convergence properties of iterative decoding algorithms.

The art of interleaving

A critical component in achieving good performance with iterative decoding of a turbo code is the interleaver. In turbo codes, the interleaver serves three main purposes: (1) build very long codes with weight distributions that approach those of random codes, (2) help in the iterative decoding process by decorrelating the input LLRs to the SISO decoders as much as possible, and (3) proper termination of the trellis in a known state, after the transmission of short to medium length frames, to avoid *edge effects* that increase the multiplicity of low-weight paths in the trellises of the component codes. To emphasize, the specific type of interleaver becomes an important factor to consider as the frame lengths (or interleaver lengths) become relatively small, say, up to one thousand symbols. There is a wealth of publications devoted to interleaver design for turbo codes. In this section, a brief description of the basic interleaver types and pointers to the literature are given.

In 1970, several types of optimum interleavers were introduced (Ramsey 1970). In particular, an (n_1, n_2) interleaver was defined as a device that *"reorders a sequence so that no contiguous sequence of n_2 symbols in the reordered sequence contains any symbols that are separated by fewer than n_1 symbols in the original ordering."*

Let $\ldots, a_{\ell_1}, a_{\ell_2}, a_{\ell_3}, \ldots$ denote the output sequence from an (n_1, n_2) interleaver, where $\ell_1, \ell_2, \ldots$ are the positions of these symbols in the input sequence. Then the definition in the previous paragraph translates into the following condition:

$$|\ell_i - \ell_j| \geq n_1$$

whenever

$$|i - j| < n_2.$$

It can be shown that the deinterleaver of an (n_1, n_2) interleaver is itself an (n_1, n_2) interleaver. The delay of the interleaver and deinterleaver are both equal to the delay introduced by the overall interleaving–deinterleaving operation. Another important parameter of an interleaver is the amount of storage or memory required. Ramsey showed four types of (n_1, n_2) interleavers that are optimum in the sense of minimizing the delay and memory required to implement them. These interleavers are known as *Ramsey interleavers*. At about the same time, Forney (1971) proposed an interleaver with the same basic structure of an (n_1, n_2) Ramsey interleaver, known as a *convolutional interleaver*. Convolutional interleavers were discussed in Section 6.2.4 and have been applied to the design of good turbo coding schemes (Hall and Wilson 2001).

There are several novel approaches to the analysis and design of interleavers: one is based on a *random interleaver* with a *spreading property*. Such a structure was first proposed in Divsalar and Pollara (1993), together with a simple algorithm to construct *S-random interleavers*: Generate random integers in the range $[1, N]$, and use a constraint on the *interleaving distance*. This constraint is seen to be equivalent to the definition of a Ramsey $(S_2, S_1 + 1)$ interleaver (this is noted in Vucetic and J. Yuan (2000), pp. 211–213). Additional constraints, for example, based on the empirical correlation between successive

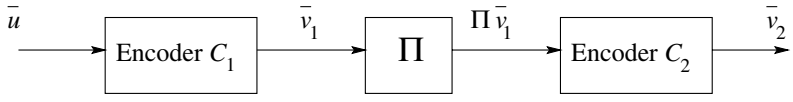


Figure 8.6 Block diagram of the encoder of a serially concatenated code.

extrinsic LLR values, are imposed to direct the selection of the positions of the permuted symbols at the output of the interleaver (Hokfelt *et al.* 2001; Sadjadpour *et al.* 2001). A second approach to the design of interleavers is to consider the overall turbo encoder structure and to compute its minimum distance and error coefficient (the number of coded sequences at minimum distance) (Breiling and Huber 2001; Garello *et al.* 2001). This gives an accurate estimation of the error floor in the medium-to-high SNR region. Other important contributions to the design of short interleavers for turbo codes are Barbulescu and Pietrobon (1994), Takeshita and Costello (2000).

8.2.2 Serial concatenation

Serial concatenation of codes was introduced in Benedetto *et al.* (1998). A block diagram of an encoder of a serial concatenation of two linear codes is shown in Figure 8.6. On the basis of the results from Section 6.2.4, and in particular comparing Figure 8.6 with Figure 6.3, the serial concatenation of two linear codes is easily recognized as a *product code*. Note that, as opposed to a turbo code, in a serially concatenated coding system there is no puncturing of redundant symbols.

The encoder of a serial concatenation of codes has the same structure as that of a product code. Following closely the notation in the original paper (Benedetto *et al.* 1998), the outer (p, k, d_1) code C_1 has a rate $R_1 = k/p$ and the inner (n, p, d_2) code C_2 has a rate $R_2 = p/n$. The codes are connected in the same manner as a block product code, with a block interleaver of length $L = mp$. This is achieved, as before, by writing m codewords of length p into the interleaver, and reading in a different order according to the permutation matrix Π. The sequence of L bits at the output of the interleaver is sent in blocks of p bits to the outer encoder. The rate of the overall $(N, K, d_1 d_2)$ code C_{SC} is $R_{\mathrm{SC}} = k/n$, where $N = nm$ and $K = km$.

The generator matrix of C_{SC} can be expressed as the product of the generator matrix of C_1, the $k_2 \times n_1$ permutation matrix Π of the interleaver, and the generator matrix of C_2, as follows:

$$G_{\mathrm{SC}} = \begin{pmatrix} G_1 & & & \\ & G_1 & & \\ & & \ddots & \\ & & & G_1 \end{pmatrix} \Pi \begin{pmatrix} G_2 & & & \\ & G_2 & & \\ & & \ddots & \\ & & & G_2 \end{pmatrix} = \left(G_1' \mid \Pi \mid G_2'\right), \qquad (8.11)$$

where G_i is the generator matrix of code C_i, $i = 1, 2$. The number of times that G_1 appears in the first factor G_1' of G_{SC} in Equation (8.11) is k_2, and the number of times that G_2 appears in the third factor G_2' of G_{SC} is n_1. All other entries in G_1' and G_2' are zero.

Example 8.2.4 *Let C_1 and C_2 be the same codes as in Example 8.2.1, that is, binary repetition* $(2, 1, 2)$ *and SPC* $(3, 2, 2)$ *codes, respectively. Then the serial concatenation or product*

of C_1 and C_2, C_{SC}, is a binary linear block $(6, 2, 4)$ *code. Note that the minimum distance of C_{SC} is larger than that of C_{PC} in Example 8.2.1. The generator matrices are* $G_1 = \begin{pmatrix} 1 & 1 \end{pmatrix}$ *and* $G_2 = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix}$. *Assuming that a conventional product code is employed, the permutation matrix, associated with a row-by-row and column-by-column interleaver, is*

$$\Pi = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

Therefore, the generator matrix of C_{SC} is

$$\begin{aligned} G_{SC} &= \begin{pmatrix} \begin{pmatrix} 1 & 1 \end{pmatrix} & \bar{0}_{12} \\ \bar{0}_{12} & \begin{pmatrix} 1 & 1 \end{pmatrix} \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix} & \bar{0}_{23} \\ \bar{0}_{23} & \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix} \end{pmatrix} \\ &= \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix} \begin{pmatrix} \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix} & \bar{0}_{23} \\ \bar{0}_{23} & \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix} \end{pmatrix} \\ &= \begin{pmatrix} \begin{pmatrix} 1 & 0 & 1 \end{pmatrix} & \begin{pmatrix} 1 & 0 & 1 \end{pmatrix} \\ \begin{pmatrix} 0 & 1 & 1 \end{pmatrix} & \begin{pmatrix} 0 & 1 & 1 \end{pmatrix} \end{pmatrix} \\ &= \left(\begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix} \quad \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix} \right), \end{aligned}$$

where $\bar{0}_{ij}$ denotes the $i \times j$ all-zero matrix.

The result can be verified by noticing that the last equality contains the generator matrix of the SPC $(3, 2, 2)$ *code* twice *because of the repetition* $(2, 1, 2)$ *code. It is also interesting to note that this is the smallest member of the family of repeat-and-accumulate codes (Divsalar et al. 1998).*

It should be noted that the main difference between the serial concatenated coding scheme and product coding discussed in Section 6.2.4 is that the interleaver was either a row-by-row column-by-column interleaver or a cyclic interleaver (if the component codes were cyclic). In contrast, as with turbo codes, the good performance of serial concatenation schemes generally depends on an interleaver that is chosen as "randomly" as possible.

Contrary to turbo codes, serially concatenated codes do not exhibit "interleaver gain saturation" (i.e., there is no error floor). Using a random argument for interleavers of length N, it can be shown that the error probability for a product code contains a factor $N^{-[(d_{Of}+1)/2]}$, where d_{Of} denotes the minimum distance of the outer code as opposed to a factor N^{-1} for parallel concatenated codes (Benedetto *et al.* 1998).

As a result, product codes outperform turbo codes in the SNR region where the error floor appears. At low SNR values, however, the better weight distribution properties of turbo codes (Perez *et al.* 1996) leads to better performance than product codes.

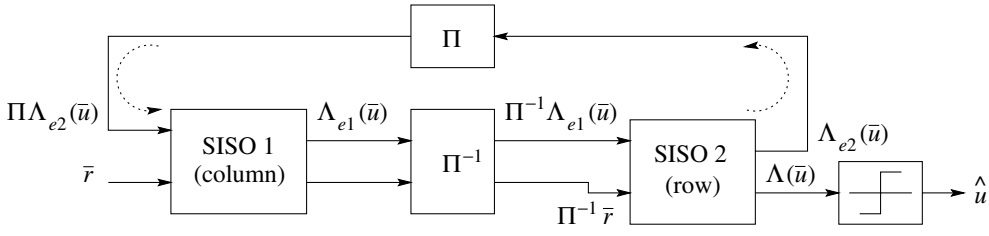


Figure 8.7 Block diagram of an iterative decoder for a serially concatenated code.

The following design rules were derived for the selection of component codes in a serially concatenated coding scheme[8] for component convolutional codes:

- The inner code must be an RSC code.
- The outer code should have a large and, if possible, odd value of minimum distance.
- The outer code may be a nonrecursive (FIR) nonsystematic convolutional encoder.

The last design criterion is needed in order to minimize the number of codewords of minimum weight (also known as the *error exponent*) and the weight input sequences resulting in minimum weight codewords.

Iterative decoding of serially concatenated codes

With reference to Figure 8.7, note that if the outer code is a nonsystematic convolutional, then it is not possible to obtain the extrinsic information from the SISO decoder (Benedetto *et al.* 1998). Therefore, contrary to the iterative decoding algorithm for turbo codes, in which only the LLR of the information symbols are updated, here the LLR of both information and code symbols are updated. The operation of the SISO decoder for the inner code remains unchanged. However, for the outer SISO decoder, the a priori LLR is always set to zero, and the LLR of both information and parity symbols is computed and delivered, after interleaving, to the SISO decoder for the inner code as a priori LLR for the next iteration. As with iterative decoding of turbo codes, there is a max-log-MAP based iterative decoding algorithm, as well as a version of SOVA that can be modified to become an approximated MAP decoding algorithm for iterative decoding of product codes (Feng and Vucetic 1997).

8.2.3 Block product codes

Although turbo codes and serial concatenations of RSC codes seem to have dominated the landscape of coding schemes where iterative decoding algorithms are applied, block product codes may also be used, as is evident from the discussion in the preceding text. In 1993, at the same conference where Berrou and colleagues introduced turbo codes, a paper was presented on iterative decoding of product and concatenated codes (Lodge *et al.* 1993). In particular, a three-dimensional product $(4096, 1331, 64)$ code, based on

[8]It should be noted that these criteria were obtained on the basis of *union bounds* on the probability of a bit error.

the extended Hamming $(16, 11, 4)$ code, with iterative MAP decoding was considered and shown to achieve impressive performance. One year later, near-optimum turbo-like decoding of product codes was introduced in Pyndiah *et al.* (1994) (see also (Pyndiah 1998)). There the product of linear block codes of relatively high rate, single- and double-error correcting extended BCH codes was considered. An iterative decoding scheme was proposed where the component decoders use the Chase type-II algorithm.[9] After a list of candidate codewords is found, LLR values are computed. This iterative decoding algorithm and its improvements are described in the next section.

Iterative decoding using Chase algorithm

In Pyndiah (1998) and Pyndiah *et al.* (1994), the Chase type-II decoding algorithm is employed to generate a list of candidate codewords that are close to the received word. Extrinsic LLR values are computed on the basis of the best two candidate codewords. If only one codeword is found, then an approximated LLR value is output by the decoder. Let C be a binary linear (N, k, d) block code capable of correcting any combination of $t = [(d-1)/2]$ or less random bit errors. Let $\bar{r} = (r_1, r_2, \ldots, r_N)$ be the received word from the output of the channel, $r_i = (-1)^{v_i} + w_i$, $\bar{v} \in C$, where w_i is a zero-mean Gaussian random variable with variance $N_0/2$. Chase type-II algorithm is executed on the basis of the received word $\bar{r}$, as described on page 151.

Three possible events can happen at the end of the Chase type-II algorithm:

1. two or more codewords, $\{\hat{v}_1, \ldots, \hat{v}_\ell\}$, $\ell \geq 2$, are found;
2. one codeword $\hat{v}_1$ is found; or
3. no codeword is found.

In the last event, the decoder may raise an uncorrectable error flag and output the received sequence as is. Alternatively, the number of error patterns to be tested can be increased until a codeword is found, as suggested in Pyndiah (1998).

Let $\mathcal{X}_j(\ell)$ denote the set of modulated codewords of C, found by Chase algorithm, for which the j-th component $x_j = \ell$, $\ell \in \{-1, +1\}$, for $1 \leq j \leq N$. By $\bar{x}_j(\ell)$, $\bar{y}_j(\ell) \in \mathcal{X}_j(\ell)$, denote respectively the closest and the next closest modulated codewords to the received word $\bar{r}$ in the Euclidean distance sense.

By using the log-max approximation $\log(e^a + e^b) \approx \max(a, b)$, the symbol LLR value (8.2) can be expressed as (Fossorier and Lin 1998; Pyndiah 1998)

$$\Lambda(u_j) \sim \Lambda'(u_j) = \frac{4E}{N_0} \left(|\bar{r} - \bar{y}_j(-1)|^2 - |\bar{r} - \bar{x}_j(+1)|^2 \right), \tag{8.12}$$

from which, after normalization and redefining $x_m = x_m(+1)$ and $y_m = y_m(-1)$, the *soft output* is

$$\Lambda'(u_j) = r_j + \sum_{\substack{x_m \neq y_m \\ m=1,\ m \neq j}}^{N} r_m x_m. \tag{8.13}$$

[9] Chase algorithms are discussed in Section 7.4.

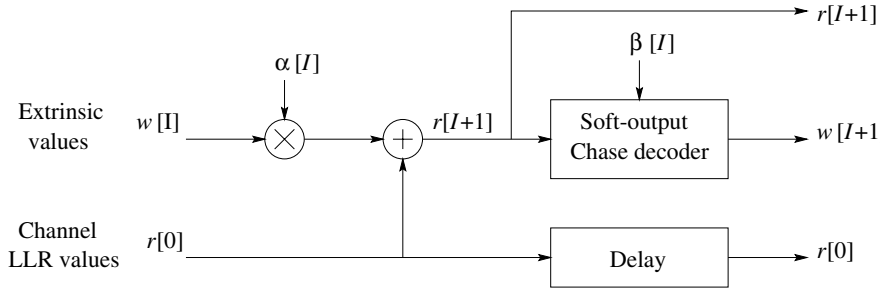


Figure 8.8 Block diagram of a soft-output Chase decoder.

The term

$$w_j = \sum_{\substack{x_m \neq y_m \\ m=1, m \neq j}}^{N} r_m x_m \tag{8.14}$$

is interpreted as a *correction term* to the soft-input r_j, which depends on the two modulated codewords closest to $\bar{r}$ and plays the same role as the *extrinsic LLR*. For $1 \leq j \leq N$, and each position j, the value w_j is sent to the next decoder as extrinsic LLR, with a *scaling factor* α_c, so that

$$r'_j = r_j + \alpha_c w_j, \tag{8.15}$$

is computed as the soft input at the next decoder. The factor α_c is used to compensate for the difference in the variances of the Gaussian random variables r_i and r'_j. A block diagram of the procedure for the generation of soft-input values and extrinsic LLR values is shown in Figure 8.8.

If for a given j-th position, $1 \leq j \leq N$, no pair of sequences $\bar{x}_j(+1)$ and $\bar{y}_j(-1)$ can be found by Chase algorithm, the use of the following symbol LLR has been suggested in Pyndiah (1998):

$$\Lambda'(u_j) = \beta_c x_j, \tag{8.16}$$

where β_c is a *correction factor* to compensate the approximation in the extrinsic information that has been estimated by simulations as

$$\beta_c = \left| \log \left(\frac{\Pr\{v_j \text{ correct}\}}{\Pr\{v_j \text{ incorrect}\}} \right) \right|, \tag{8.17}$$

that is, the magnitude of the LLR of the simulated symbol error rate. In Martin and Taylor (2000) and Picart and Pyndiah (1995), it is shown how the correction factors α and β can be computed *adaptively* on the basis of the statistics of the processed codewords. It should also be noted that the soft-output algorithm proposed in Fossorier and Lin (1998) and described in Section 7.5 can also be applied. Adaptive weights are also needed in this case to scale down the extrinsic LLR values.

To summarize, the iterative decoding method with soft outputs based on a set of codewords produced by a Chase type-II algorithm is as follows:

Step 0: Initialization

Set iteration counter $I = 0$. Let $\bar{r}[0] = \bar{r}$ (the received channel values).

Step 1: Soft inputs

For $j = 1, 2, \ldots, N$,

$$r_j[I+1] = r_j[0] + \alpha_c[I]\, w_j[I].$$

Step 2: Chase algorithm

Execute Chase type-II algorithm, using $\bar{r}[I+1]$. Let n_c denote the number of codewords found. If possible, save the two modulated codewords $\bar{x}$ and $\bar{y}$ closest to the received sequence.

Step 3: Extrinsic information

For $j = 1, 2, \ldots, N$,

- If $n_c \geq 2$

$$w_j[I+1] = \sum_{\substack{x_m \neq y_m \\ m=1, m\neq j}}^{N} r[I+1]_m x_m,$$

- Else

$$w_j[I+1] = \beta_c[I] x_j.$$

Step 4: Soft output

Let $I = I + 1$. If $I < I_{\max}$ (the maximum number of iterations) or a stopping criterion is not satisfied then go to Step 1.

Else compute the soft output:

For $j = 1, 2, \ldots, N$,

$$\Lambda(u_i) = \alpha_c[I]\, w_j[I] + r_j[0], \tag{8.18}$$

and stop.

For BPSK modulation, the values of α_c and β_c were computed for up to four iterations (eight values in total, two values for the first and second decoders) as (Pyndiah 1998)

$$\alpha_c = \begin{pmatrix} 0.0 & 0.2 & 0.3 & 0.5 & 0.7 & 0.9 & 1.0 & 1.0 \end{pmatrix},$$
$$\beta_c = \begin{pmatrix} 0.2 & 0.4 & 0.6 & 0.8 & 1.0 & 1.0 & 1.0 & 1.0 \end{pmatrix}.$$

Example 8.2.5 *Figure 8.9 shows the simulated error performance of the product of two identical Hamming codes, with component binary Hamming* $(2^m - 1, 2^m - 1 - m, 3)$ *codes, for* $3 \leq m \leq 7$. *The number of iterations was set to 4. A turbo code effect is observed clearly as the length of the code increases. The longer the code, the higher the code rate and the steeper the BER curve.*

Example 8.2.6 *Figure 8.10 shows the performance of a turbo product Hamming* $(15, 11)^2$ *code with the number of iterations as a parameter. As the number of iterations increases, the error performance improves. There is saturation after four iterations, in the sense that the performance improves only marginally with more iterations.*

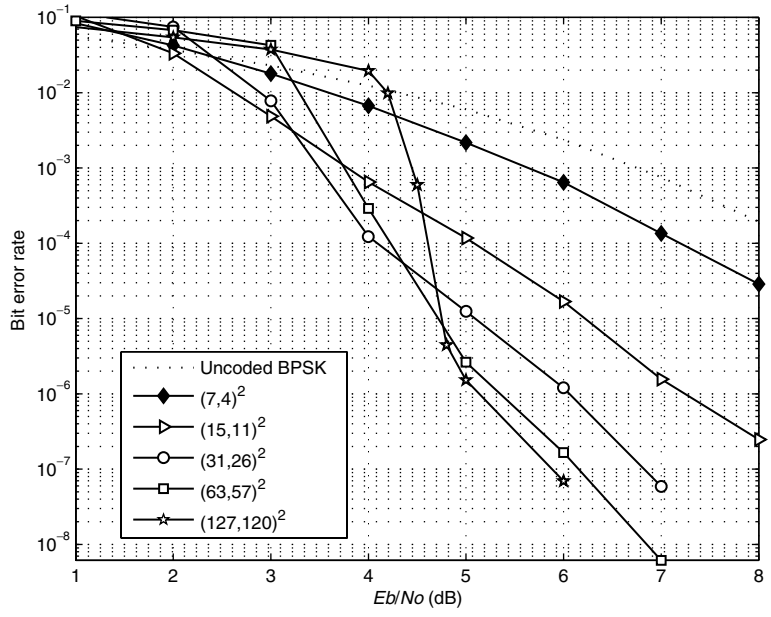


Figure 8.9 Error performance of turbo product Hamming codes with iterative decoding based on Chase type-II algorithm and four iterations.

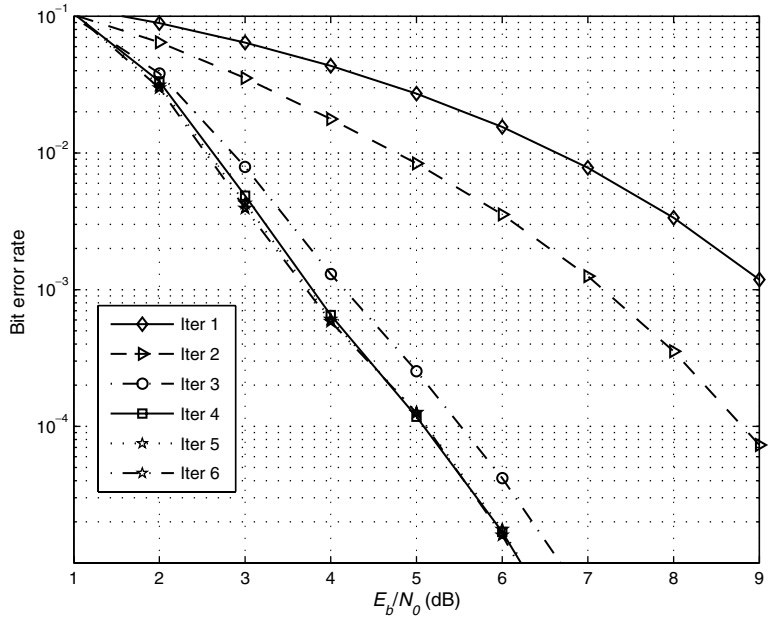


Figure 8.10 Performance of a turbo product Hamming $(15, 11)^2$ code with iterative decoding based on Chase type-II algorithm. Number of iterations as parameter.

8.3 Low-density parity-check codes

In 1962, Gallager in Gallager (1962) introduced a class of linear codes known as low-density parity-check (LDPC) codes and presented two iterative probabilistic decoding algorithms. Later, Tanner (1981) extended Gallager's probabilistic decoding algorithm to the more general case where the parity checks are defined by subcodes instead of simple single parity-check equations. Earlier, it was shown that LDPC codes have a minimum distance that grows linearly with the code length and that errors up to the minimum distance could be corrected with a decoding algorithm with almost linear complexity (Zyablov and Pinsker 1975).

In MacKay (1999) and MacKay and Neal (1999) it is shown that LDPC codes can get as close to the Shannon limit as turbo codes. Later in Richardson *et al.* (2001), irregular LDPC codes were shown to outperform turbo codes of approximately the same length and rate, when the block length is large. At the time of writing, the best rate-1/2 binary code, with a block length of 10,000,000, is an LDPC code that achieved a record 0.0045 dB away from the Shannon limit for binary transmission over an AWGN channel (Chung *et al.* 2001).

A *regular* LDPC code is a linear (N, k) code with parity-check matrix H having the Hamming weight of the columns and rows of H equal to J and K, respectively, with both J and K much smaller than the code length N. As a result, an LDPC code has a very sparse parity-check matrix. If the Hamming weights of the columns and rows of H are chosen in accordance to some nonuniform distribution, then *irregular* LDPC codes are obtained (Richardson *et al.* 2001). MacKay has proposed several methods to construct LDPC matrices by computer search (MacKay 1999).

8.3.1 Tanner graphs

For every linear (N, k) code C, there exists a bipartite graph with incidence matrix H. This graph is known as a *Tanner graph* (Sipser and Spielman 1996; Tanner 1981), named after its inventor. By introducing state nodes, Tanner graphs have been generalized to factor graphs (Forney 2001; Kschischang *et al.* 2001). The nodes of the Tanner graph of a code are associated with two kinds of variables and their LLR values.

The Tanner graph of a linear (N, k) code C has N *variable nodes* or *code nodes*, x_ℓ, associated with code symbols, and at least $N - k$ *parity nodes*, z_m, associated with the parity-check equations. For a regular LDPC code, the degrees of the code nodes are all equal to J and the degrees of the parity nodes are equal to K.

Example 8.3.1 *To illustrate the Tanner graph of a code, consider the Hamming* $(7, 4, 3)$ *code. Its parity check matrix is*[10]

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

The corresponding Tanner graph is shown in Figure 8.11. The way the code nodes connect to check nodes is dictated by the rows of the parity-check matrix.

[10] See Example 2.1.1 on page 28.

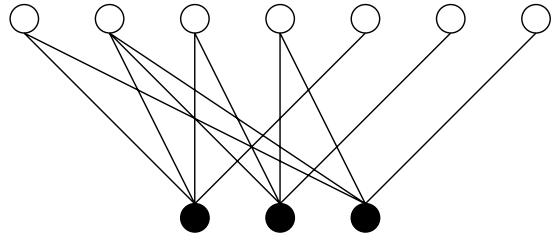


Figure 8.11 Tanner graph of a Hamming $(7, 4, 3)$ code.

The first row gives the parity-check equation $v_1 + v_2 + v_3 + v_5 = 0$. *As indicated before, variables* x_ℓ *and* z_m *are assigned to each code symbol and each parity-check equation, respectively. Therefore, the following parity-check equations are obtained,*

$$\begin{aligned} z_1 &= x_1 + x_2 + x_3 + x_5, \\ z_2 &= x_2 + x_3 + x_4 + x_6, \\ z_3 &= x_1 + x_2 + x_4 + x_7. \end{aligned}$$

From the topmost equation, code nodes x_1, x_2, x_3 *and* x_5 *are connected to check node* z_1. *Similarly, the columns of* H, *when interpreted as incidence vectors, indicate in which parity-check equations code symbols appear or* participate. *The leftmost column of* H *above,* $\begin{pmatrix} 1 & 0 & 1 \end{pmatrix}^\top$, *indicates that* x_1 *is connected to check nodes* z_1 *and* z_3.

Example 8.3.2 *The parity-check matrix in Gallager's paper (Gallager 1962),*

$$H = \left(\begin{array}{cccc|cccc|cccc|cccc|cccc} 1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&1&1&1&1&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1 \\ \hline 1&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&0&0&0&0 \\ 0&1&0&0&0&1&0&0&0&1&0&0&0&0&0&0&1&0&0&0 \\ 0&0&1&0&0&0&1&0&0&0&0&0&0&1&0&0&0&1&0&0 \\ 0&0&0&1&0&0&0&0&0&0&1&0&0&0&1&0&0&0&1&0 \\ 0&0&0&0&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&1 \\ \hline 1&0&0&0&0&1&0&0&0&0&0&1&0&0&0&0&0&1&0&0 \\ 0&1&0&0&0&0&1&0&0&0&1&0&0&0&0&1&0&0&0&0 \\ 0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&0&1&0 \\ 0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&1&0&0&0 \\ 0&0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&1 \end{array}\right),$$

is that of an LDPC $(20, 7, 6)$ *code with* $J = 3$ *and* $K = 4$. *Its Tanner graph is shown in Figure 8.12. Note that every code node is connected to exactly three parity-check nodes. In other words, the* degree *of the code nodes is equal to* $J = 3$. *Similarly, the degree of the parity nodes is equal to* $K = 4$.

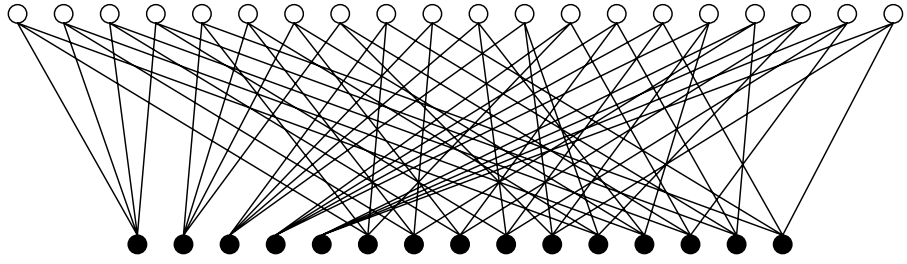


Figure 8.12 Tanner graph of a binary Gallager (20, 7, 6) code.

Tanner graphs can be used to estimate codewords of an LDPC code C with iterative probabilistic decoding algorithms on the basis of either hard or soft decisions. In the following text, the two basic iterative decoding algorithms introduced by Gallager are presented.

8.3.2 Iterative hard-decision decoding: The bit-flip algorithm

In his 1962 paper, Gallager gave the following algorithm (Gallager 1962)[11]:

> *The decoder computes all the parity checks and then changes any digit that is contained in more than some fixed number of unsatisfied parity-check equations. Using these new values, the parity checks are recomputed, and the process is repeated until the parity checks are all satisfied.*

The input of the algorithm is the hard-decision vector

$$\bar{r}_h = (\text{sgn}(r_1), \text{sgn}(r_2), \ldots, \text{sgn}(r_N)) ,$$

where $r_i = (-1)^{v_i} + w_i$, $\bar{v} \in C$, w_i denotes a zero-mean Gaussian random variable with variance $N_0/2$, $i = 1, 2, \ldots, N$, and $\text{sgn}(x) = 1$, if $x < 0$, and $\text{sgn}(x) = 0$ otherwise. Let T denote a threshold such that, if the number of unsatisfied parity-check equations where a code symbol v_i participates exceeds T, then the symbol is "flipped," $v_i = v_i \oplus 1$. The algorithm can be iterated several times until either all parity-check equations are satisfied or a prescribed maximum number of iterations is reached. We refer to this algorithm as iterative bit-flip (IBF) decoding. Figures 8.13 and 8.14 show simulation results of IBF decoding for the binary (20, 7, 6) Gallager code C_G and the binary Hamming (7, 4, 3) code C_H, respectively, with binary transmission over an AWGN channel. Threshold values were set to $T = 1, 2, 3$. In both cases, two iterations were performed.

Example 8.3.3 *For the binary* (20, 7, 6) *Gallager code, Figure 8.13 shows simulation results of IBF decoding. Note that the Hamming weight of each column of the parity-check matrix is three. This is the number of parity-check equations in which each coded bit is involved. With a threshold setting* $T = 1$*, too many bits are flipped, resulting in a high number of decoding errors. With* $T = 2$*, at least two checks need to fail for the bit to change. This gives the best performance. Setting* $T = 3$ *does not change any bit. In this case, the performance is the same as uncoded BPSK modulation, with an additional penalty of* $-10 \log_{10}(7/20) = 4.56$ *dB in* E_b/N_0 *because of the code rate.*

[11] This algorithm is also known as Gallager's algorithm A.

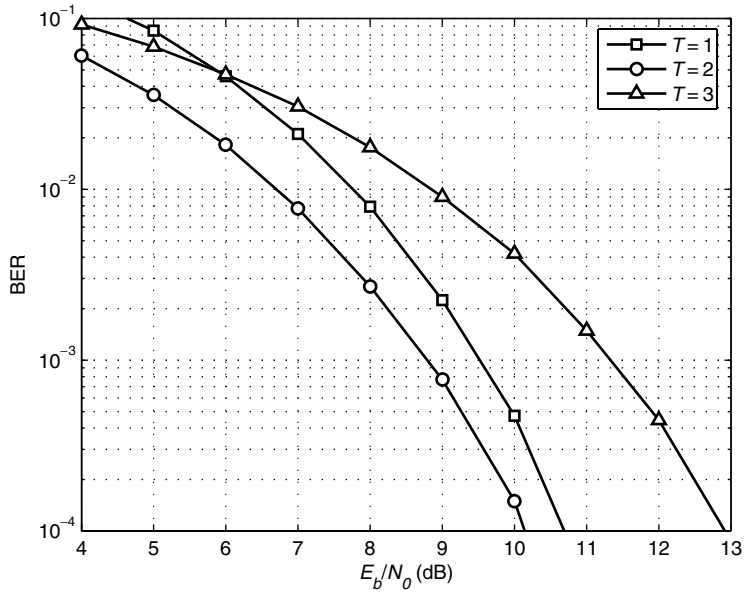


Figure 8.13 Performance of a binary (20, 7, 6) Gallager code with iterative bit-flip decoding and two iterations. Threshold value as parameter.

Example 8.3.4 *For the Hamming (7,4,3) code C_H, Figure 8.14 shows simulation results of IBF decoding and hard-decision decoding (denoted LUT in the figure). The performance of the IBF decoding algorithm is analyzed next. The parity-check matrix of code C_H is*

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

And the corresponding parity-check equations can be written as:

$$\begin{aligned} z_1 &= x_1 + x_3 + x_4 + x_5 \\ z_2 &= x_1 + x_2 + x_3 + x_6 \\ z_3 &= x_2 + x_3 + x_4 + x_7. \end{aligned}$$

For a threshold setting $T = 1$, the following table shows the bit-flipping process:

z_1	z_2	z_3	*Bits flipped*
0	0	0	*None*
1	1	0	x_1 *and* x_3
0	1	1	x_2 *and* x_3
1	1	1	x_3
1	0	1	x_3 *and* x_4
1	0	0	*None*
0	1	0	*None*
0	0	1	*None*

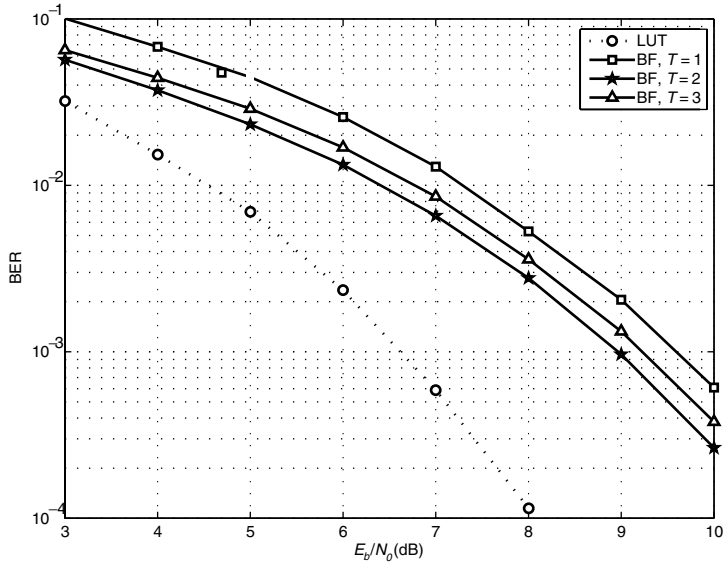


Figure 8.14 Performance of a binary Hamming (7, 4, 3) code with iterative bit-flip decoding and two iterations. Threshold value as parameter.

As a result, in the presence of a single error in all positions except the third, two *bits are complemented, and a single error will occur. In the case of a single error in the third position, all information bits are flipped, resulting in three additional errors. This explains why the performance is worse than the single-error correcting decoding algorithm using a look-up table (LUT).*

If $T = 2$ *then only when* $z_1 = 1$, $z_2 = 1$, *and* $z_3 = 1$ *is the third information bit changed. Otherwise, no bit is flipped. Consequently, only one out of seven single-error patterns are corrected.*

If $T = 3$, *then no bit is ever flipped. This is the same performance as uncoded BPSK modulation. The performance is the same as that of BPSK but shifted to the right by the rate loss* $-10\log_{10}(4/7) = 2.43$ *dB.*

The relatively poor performance of bit-flip decoding occurs because the underlying Tanner graph is not sufficiently connected. One variable node (x_3) *has degree equal to three, while the other variable nodes have degree equal to two. Moreover, the variable nodes associated with parity-check positions all have degree equal to one. To improve the performance of the code under IBF decoding, it is necessary to increase the node degrees. This can be done by increasing the number of parity-check equations.*

In Figures 8.15 to 8.17, the error performance of the Berlekamp–Massey (BM) algorithm and Gallager bit-flip (BF) algorithm is compared for the BCH (31, 26, 3), (31, 21, 5) and (31, 16, 7) codes, respectively. It is evident that, as the error-correcting capability of the code increases, the performance of the BF algorithm is inferior to that of the BM algorithm. On the other hand, in terms of computational complexity, the Gallager BF algorithm requires simple exclusive -OR gates and comparisons, as opposed to $GF(2^m)$ arithmetic processors in the case of the BM algorithm. This suggests that, for some high-rate codes such

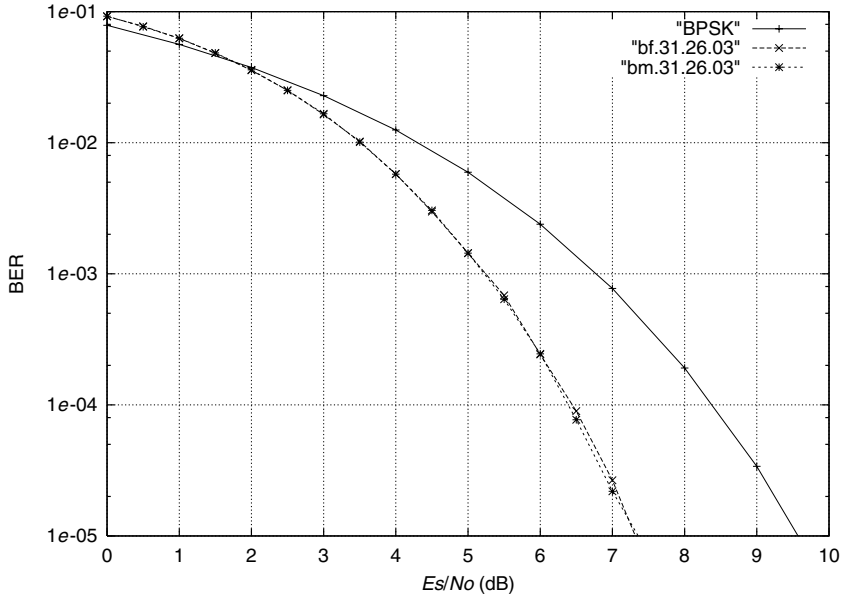


Figure 8.15 Performance of Berlekamp–Massey and Gallager bit-flip algorithms for the binary BCH $(31, 26, 3)$ code.

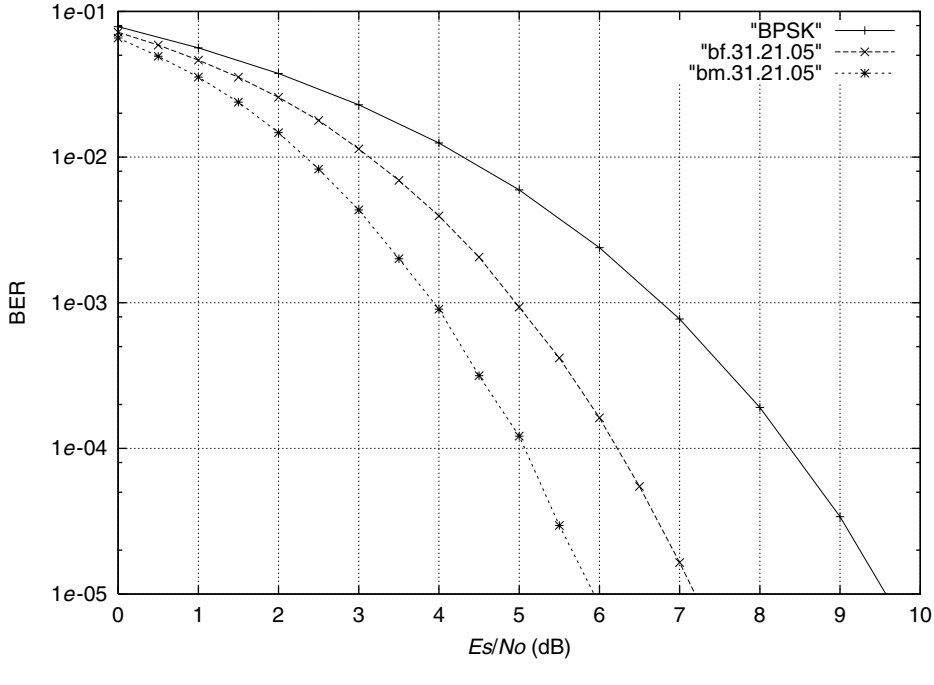


Figure 8.16 Performance of Berlekamp–Massey and Gallager bit-flip algorithms for the binary BCH $(31, 21, 5)$ code.

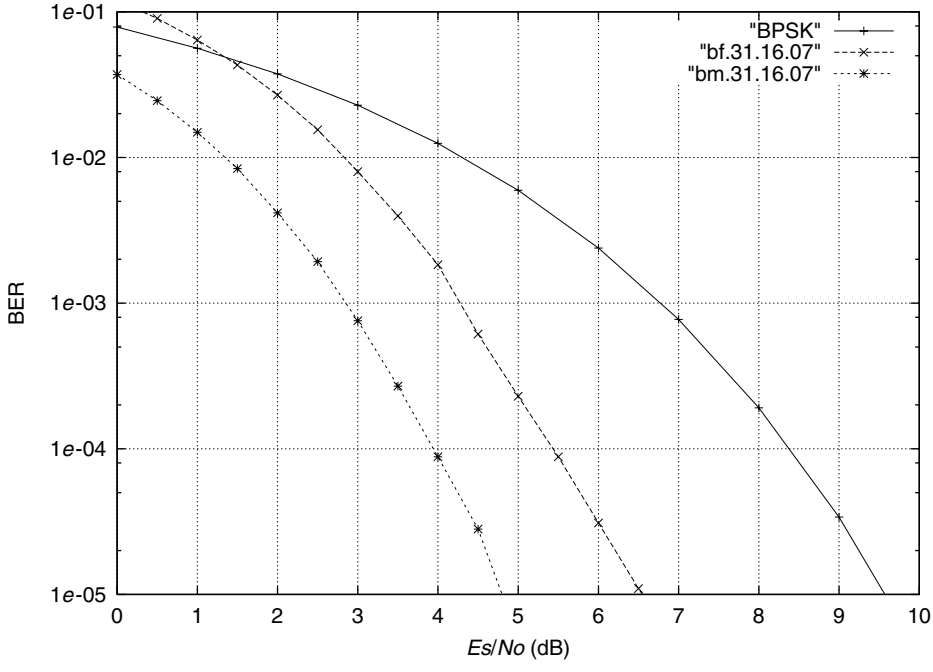


Figure 8.17 Performance of Berlekamp–Massey and Gallager bit-flip algorithms for the binary BCH (31, 16, 7) code.

as single- and double-error correcting BCH codes, the BF algorithm might be considered an alternative to the BM algorithm.

An additional feature of the BF algorithm, and the iterative probabilistic decoding algorithm presented below, is that its complexity depends only on the degrees of the nodes in the Tanner graph. In other words, for fixed values of J and K, the decoding complexity grows linearly with the code length.

8.3.3 Iterative probabilistic decoding: Belief propagation

In this section, an *iterative belief-propagation* (IBP) decoding algorithm is presented. This algorithm is also known in the literature as *Pearl's algorithm* (Pearl 1988) and the *sum-product algorithm* (Forney 2001; Kschischang *et al.* 2001; Wiberg 1996). Wiberg in his Ph.D. thesis (Wiberg 1996) has shown that the forward–backward, turbo and Gallager algorithm (described below) are all special cases of the sum-product algorithm. Furthermore, it was demonstrated (McEliece *et al.* 1998) that iterative decoding algorithms for turbo codes and product codes are particular cases of IBP decoding. The description below follows closely (MacKay 1999; Pearl 1988) for binary LDPC codes. The following notation is convenient in describing the algorithm. Let $h_{i,j}$ denote the entry of H in the i-th row and j-th column. Let

$$\mathcal{L}(m) = \{\ell : h_{m,\ell} = 1\} \tag{8.19}$$

denote the set of *code positions* that participate in the m-th parity-check equation, and let

$$\mathcal{M}(\ell) = \{m : h_{m,\ell} = 1\} \tag{8.20}$$

denote the set of *check positions* in which code position ℓ participates.

The algorithm iteratively computes two types of conditional probabilities:

$q^x_{m\ell}$, the probability that the ℓ-th bit of $\bar{v}$ has the value x, given the information obtained via the check nodes other than check node m.

$r^x_{m\ell}$, the probability[12] that a check node m is satisfied when bit ℓ is fixed to a value x and the other bits are independent with probabilities $q_{m\ell'}$, $\ell' \in \mathcal{L}(m) \setminus \ell$.

As noted in MacKay (1999) and Pearl (1988), the following IBP decoding algorithm would produce the exact a posteriori probabilities after some number of iterations *if the Tanner graph of the code contained no cycles*. In the following, binary transmission over an AWGN channel is assumed. Modulated symbols $m(v_i) = (-1)^{v_i}\sqrt{E_s}$ are transmitted over an AWGN channel and received as $r_i = m(v_i) + w_i$, where w_i is a Gaussian distributed random variable with zero mean and variance $N_0/2$, $1 \le i \le N$.

Initialization

For $\ell \in \{1, 2, \ldots, N\}$, initialize the a priori probabilities of the code nodes

$$p^1_\ell = \frac{1}{1 + \exp(r_\ell \frac{4}{N_0})} \tag{8.21}$$

and $p^0_\ell = 1 - p^1_\ell$. For every (ℓ, m) such that $h_{m,\ell} = 1$,

$$q^0_{m,\ell} = p^0_\ell, \qquad q^1_{m,\ell} = p^1_\ell. \tag{8.22}$$

Message passing

Step 1: Bottom-up (horizontal):

For each ℓ, m, compute

$$\delta r_{m,\ell} = \prod_{\ell' \in \mathcal{L}(m)\setminus\ell} (q^0_{m,\ell'} - q^1_{m,\ell'}), \tag{8.23}$$

and

$$r^0_{m,\ell} = (1 + \delta r_{m,\ell})/2, \qquad r^1_{m,\ell} = (1 - \delta r_{m,\ell})/2. \tag{8.24}$$

Step 2: Top-down (vertical):

For each ℓ, m, compute

$$q^0_{m,\ell} = p^0_\ell \prod_{m' \in \mathcal{M}(\ell)\setminus m} r^0_{m',\ell}, \qquad q^1_{m\ell} = p^1_\ell \prod_{m' \in \mathcal{M}(\ell)\setminus m} r^1_{m',\ell}, \tag{8.25}$$

[12]Note: This is an abuse of notation, because r_i denotes the i-th component of the received vector. However, it helps keep the same notation as most of the literature in the topic.

and normalize, with $\alpha = 1/(q^0_{m\ell} + q^1_{m\ell})$,

$$q^0_{m,\ell} = \alpha\, q^0_{m,\ell}, \qquad q^1_{m,\ell} = \alpha\, q^1_{m,\ell}. \tag{8.26}$$

For each ℓ, compute the a posteriori probabilities

$$q^0_\ell = p^0_\ell \prod_{m\in\mathcal{M}(\ell)} r^0_{m,\ell}, \qquad q^1_\ell = p^1_\ell \prod_{m\in\mathcal{M}(\ell)} r^1_{m,\ell}, \tag{8.27}$$

and normalize, with $\alpha = 1/(q^0_\ell + q^1_\ell)$,

$$q^0_\ell = \alpha\, q^0_\ell, \qquad q^1_\ell = \alpha\, q^1_\ell. \tag{8.28}$$

Decoding and soft outputs

For $i = 1, 2, \ldots N$, compute

$$\hat{v}_i = \operatorname{sgn}\left(q^0_i - \frac{1}{2}\right). \tag{8.29}$$

If $\hat{v}H^\top = \bar{0}$, then $\hat{v}$ is the estimated codeword and the soft outputs are

$$\Lambda(v_i) = \log(q^1_i) - \log(q^0_i), \quad 1 \le i \le N. \tag{8.30}$$

The algorithm stops.

Otherwise, return to Step 2. If the number of iterations exceeds a predetermined threshold, a decoding failure is declared. Output the received values as they are. The algorithm stops.

Figure 8.18 shows the performance of IBP decoding for the binary cyclic PG (273, 191,17) code. This code is a small member of the family of binary finite geometry LDPC codes recently introduced in Kuo *et al.* (2001). Also shown in the plot is the performance of the BF algorithm with a threshold equal to 8.

Notes

As proposed in Gallager (1962), the IBP decoding algorithm can be modified in a straightforward way to use LLR values instead of probabilities. This has the advantage that no normalization is required in the second step of message passing. In this case, the function

$$F(x) \triangleq \frac{e^x + 1}{e^x - 1} = \tanh(x), \tag{8.31}$$

or its inverse is needed in Step 1 of message passing. A program implementing this log-IBP decoder is available in the ECC web site.

Note that $F(x)$ can be implemented as a look-up table. Numerical results suggest that quantization of $F(x)$ to eight levels results in practically no performance loss compared with a floating point version. In Fossorier *et al.* (1999), several reduced-complexity versions of the IBP algorithm are presented that favorably trade off decoding complexity and error performance. Other interesting applications of iterative decoding algorithms include fast-correlation attacks in cryptanalysis (Golić 2001; Mihaljević and Golić 1991).

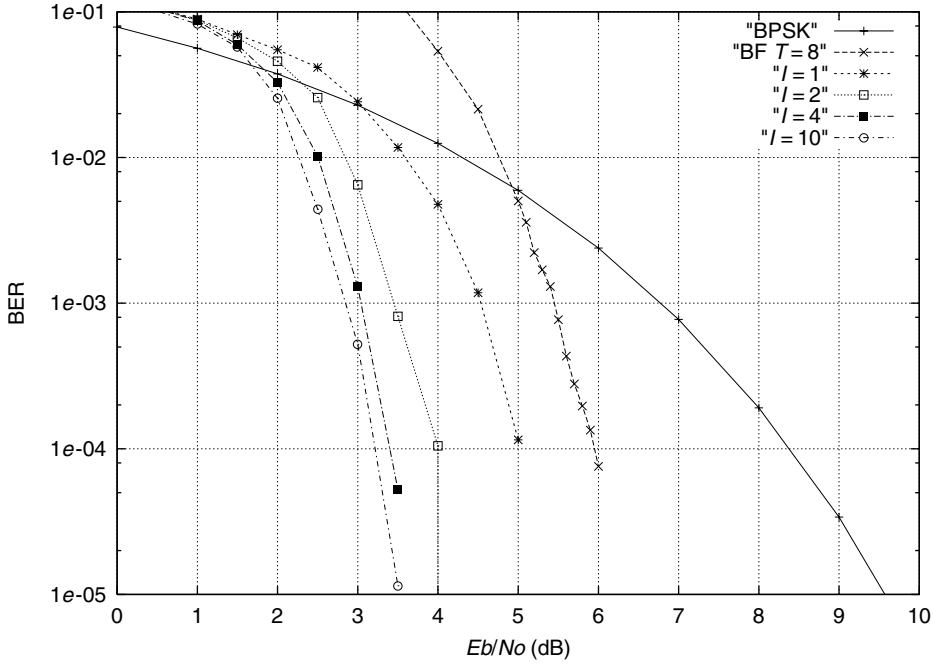


Figure 8.18 Performance of iterative decoding of a binary (273, 191) cyclic code.

It is noteworthy that LDPC codes can always detect the occurrence of decoding errors. This is in contrast to turbo codes and product codes based on convolutional codes that cannot detect many errors (MacKay 1999). LDPC codes have very low complexity compared with iterative decoding using component MAP decoders. This is true when complexity is measured in terms of the number of real additions and multiplications per block per iteration. However, it should be pointed out that IBP decoding gives the MLD codeword *whenever the Tanner graph contains no cycles*. Because for most practical codes the Tanner graph has relative short cycles (lengths 4 and 6), convergence of the IBP decoding algorithm is either not guaranteed or is slow (Forney 2001; Kschischang *et al.* 2001; Richardson *et al.* 2001). As a result, in general, IBP decoding algorithms for LDPC codes may require many more iterations than iterative decoding algorithms for product codes with MAP component decoders.

In terms of implementation of IBP decoding, the following are the two architectures. A fast parallel architecture can be realized with N X-processors for code nodes and M Z-processors for check nodes, with connections between them specified by multiple address computation units (ACU). This architecture is shown in Figure 8.19. In the figure, λ and π are LLR values associated with the conditional probabilities used in the IBP algorithm. Alternatively, only one X-processor and one Z-processor can be used and shared between nodes whose metrics are stored in two memories, a π-memory and a λ-memory, as shown in Figure 8.20. These architectures represent the extremes in the space/time tradeoff to find the best architecture.

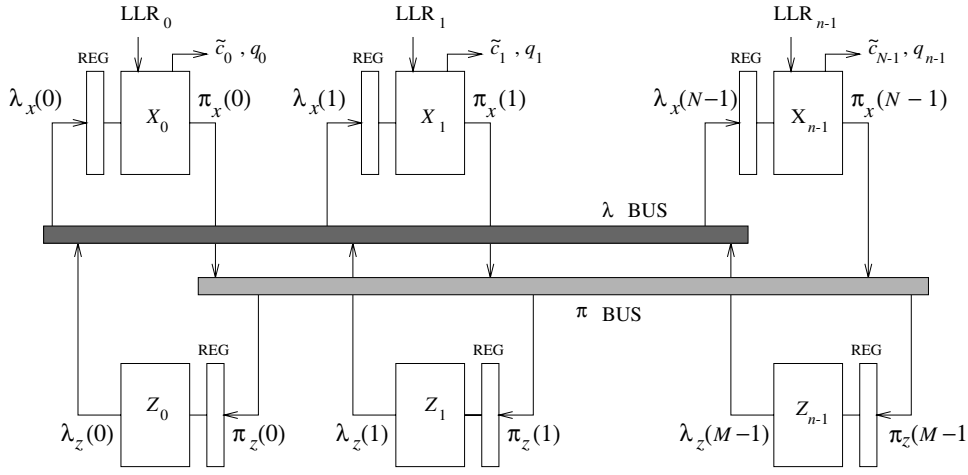


Figure 8.19 Parallel architecture of an IBP decoder.

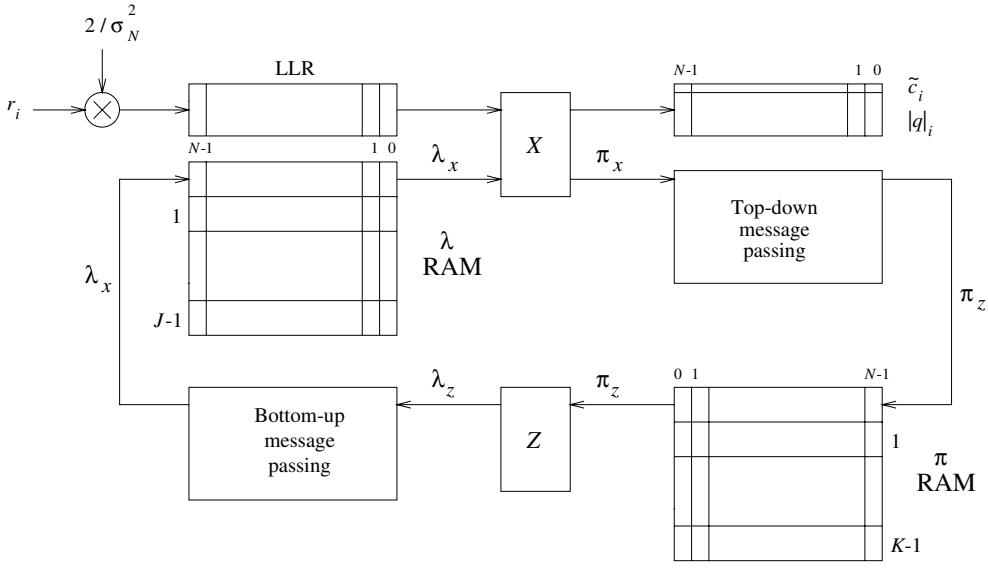


Figure 8.20 Serial architecture of an IBP decoder.

The number of computations in the IBP decoding algorithm can be reduced by making preliminary *hard decisions* that are based on the amplitudes of the channel LLR values (reliabilities). This idea appears in Frey and Kschischang (1998) and in Pearl (1988) as a method of dealing with short cycles in Bayesian networks (Tanner graphs). A set of highly reliable positions (HRP) can be selected by comparing the channel LLR values to a threshold T. If the reliability of a channel symbol exceeds T, then that code node is fixed to a HD value. As a result, the X-processor is not utilized in the decoding process. Instead,

this processor always gives as an output either the maximum probability (LLR) or a flag, indicating that this position is highly reliable. Consequently, the Z-processor to which the HD value is sent performs a smaller number of computations, because this input is highly reliable and not taken into account in the probability or LLR computations.

Problems

1. Show that the channel LLR values for a BSC, an AWGN channel, and a Rayleigh fading channel are as given in Eq. (8.4). (Caution: Note the mapping used in AWGN and Rayleigh fading channel and the definition of LLR.)

2. Consider a turbo code C_{PC} with constituent codes C_1, a binary SPC $(3, 2, 2)$ code, and C_2, a binary SPC $(4, 3, 2)$ code.

   (a) Determine the generator matrix and minimum distance of this code.

   (b) Devise an iterative decoder for C_{PC}.

   (c) Let the received channel reliabilities be

   $$\Lambda_c = \begin{vmatrix} -2.5 & +0.2 & -0.2 \\ +2.5 & +1.5 & -0.5 \\ -1.5 & -1.5 & +0.2 \\ +0.5 & +0.5 & \end{vmatrix}.$$

   After two iterations, determine the estimated codeword.

3. Devise an iterative decoding algorithm for block product codes that uses the SO-OSD algorithm introduced in Section 7.8.5.

4. Let C denote a binary Hamming (7,4,3) code with parity-check matrix

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

   (a) As explained in Section 8.3.2, BF decoding does not work for this choice of H. Use the Tanner graph associated with H to provide further insight into this malfunction. (Hint: Look at cycles of length 4 in the graph.)

   (b) Build an extended 7×7 parity-check matrix H_{ext} by taking all linear combinations of the rows of H. Note that the rows are all nonzero codewords of a binary MLS $(7, 3, 4)$ code.

   (c) Construct the Tanner graph with incidence matrix H_{ext}.

   (d) Simulate the performance of iterative BF decoding on the basis of the extended matrix H_{ext}. Analyze the performance on the basis of the structure of the Tanner graph, by considering single-error patterns in the received codeword.

5. (Lin) In general, the Tanner graph of a binary cyclic (n, k) code can be constructed from its parity-check polynomial $\bar{h}(x)$ by constructing an n-by-n extended parity-check matrix H whose rows are formed from the binary vectors corresponding to

all the n cyclic shifts of $\bar{h}(x)$. Show that $J = K = \mathrm{wt}_H(\bar{h}(x))$, that is, the degree of every node is equal to the Hamming weight of $\bar{h}(x)$.

6. On the basis of problem 5, construct the Tanner graph of a binary MLS $(7, 3, 4)$ code.

7. On the basis of problem 5, construct the Tanner graph of a binary $(15, 7, 5)$ BCH code.

8. On the basis of problem 5, construct the Tanner graph of a binary $(15, 5, 7)$ BCH code.

9. Construct the Tanner graph of a binary repetition $(5, 1, 5)$ code.

10. Let C denote the binary Gallager $(20, 7, 6)$ code.

    (a) Show that its minimum distance is 6.

    (b) (Hard) Show that the dimension of the linear space spanned by H (the dual code) is $n - k = 13$.

    (c) (Hard) Find a generator matrix G of C. Using G, devise a maximum-likelihood decoder that computes 128 correlation metrics and outputs the (ML) codeword corresponding to the highest metric.

11. A nice feature of the function $F(x)$ defined in (8.31) is that it is its own inverse. Therefore, only one look-up table is needed for implementing a log-IBP decoder. Prove that $F(F(x)) = 1$.

9

Combining codes and digital modulation

In discussing SDD, attention was focused on binary transmission, that is, using two possible transmitted symbols $\{-1, +1\}$. Let the *Nyquist bandwidth* of a transmission (or storage) signal be the rate R_s at which symbols are transmitted (or stored). For binary transmission, the two values of a bit, 0 and 1, are assigned to two values of the transmitted symbol, $+1$ and -1, respectively. Therefore, R_s bits/second require a Nyquist bandwidth of R_s Hz. The *spectral efficiency* μ of a binary transmission system equals 1 bit/sec/Hz (or 1 bit/symbol). Coded modulation is the joint design of error correcting codes and digital modulation formats to increase the bandwidth efficiency of a digital communication system.

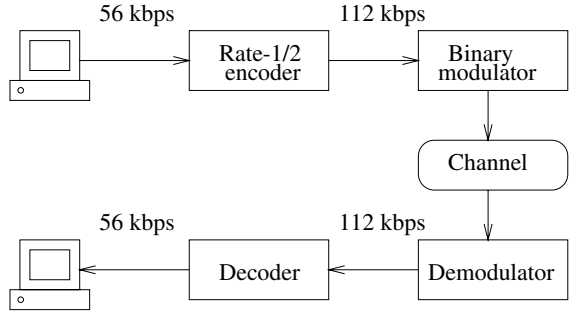
9.1 Motivation

Suppose an error correcting coding scheme is required to increase the reliability of the binary transmission (or storage) system. Let $R_c = k/n$ denote the rate of the code. Then the spectral efficiency is $\mu = R_c$ bps/Hz. Thus for a coded binary transmission system, a spectral efficiency $\mu \leq 1$ bps/Hz is achieved. This translates into a faster signaling rate or *larger bandwidth* for the same bit rate. Equivalently, the *bit rate has to be reduced* by a factor of $1/R_s$ so as to keep the transmitted symbol rate or bandwidth constant.

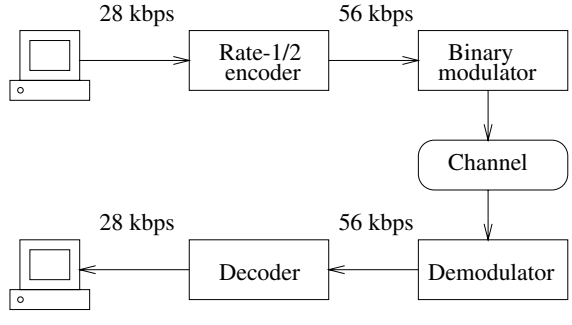
Example 9.1.1 *Assume that 56 kbps are desired to be transmitted over a channel. The required Nyquist bandwidth of an uncoded binary transmission system is 56 kHz. Suppose a rate-1/2 convolutional code is used. Then, with binary transmission, the spectral efficiency of the coded system is $\mu = 0.5$ bps/Hz. The effect of μ in data rate and signaling rate is illustrated in Figure 9.1.*

Increasing the bandwidth of the signal as in Figure 9.1 (a) is not a practical solution since typically the channel *bandwidth is expensive* (or limited, as in a telephone line). Decreasing the data rate (Figure 9.1 (b)) is a possible solution, but places a limit on the number of services or applications offered. In addition, the *increased transmission delay* may not be acceptable (e.g., voice or video).

The Art of Error Correcting Coding, Second Edition Robert H. Morelos-Zaragoza
 ISBN: 0-470-01558-6



(a) Keeping bit rate constant



(b) Keeping symbol rate constant

Figure 9.1 Effect of coding in *binary transmission* of information.

The fundamental question then is the following: how to increase the data rate without increasing the bandwidth (or symbol rate)? The answer, as given in Ungerboeck (1982) and Imai and Hirakawa (1977), is to use an expanded signal set (e.g., 2^{ν}-ary PSK or QAM digital modulation) and then to apply error correcting coding to increase the Euclidean distance between coded sequences.

9.1.1 Examples of signal sets

Several signal sets used in digital communication systems are shown in Figure 9.2. The I and Q axis represent *orthogonal signals* that are used in transmission.[1] In bandpass digital communication systems, generally

$$I = \cos(2\pi f_c t), \qquad Q = \sin(2\pi f_c t), \tag{9.1}$$

where I stands for *in-phase* and Q for *quadrature*. If a point in the IQ-plane has coordinates (x, y), then the transmitted signal is

$$s(t) = R\cos(2\pi f_c t + \phi), \qquad (k-1)T \leq t < kT, \tag{9.2}$$

[1]For simplicity of exposition, let $I \triangleq I(t)$ and $Q \triangleq Q(t)$.

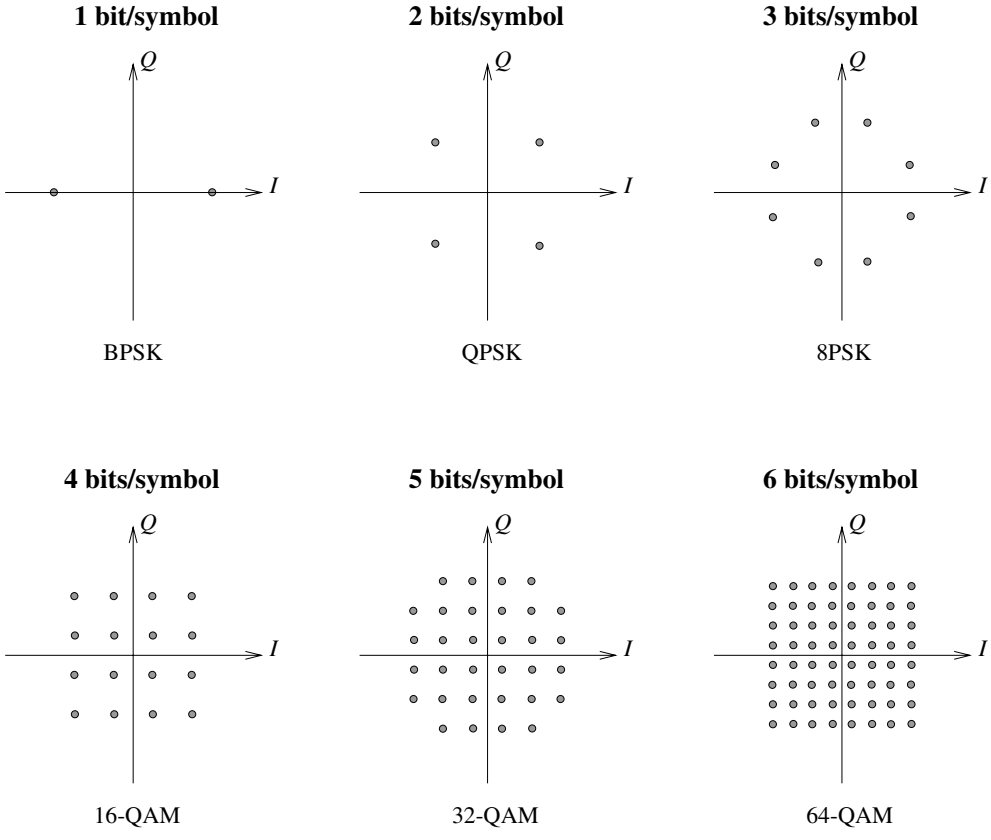


Figure 9.2 Examples of M-PSK and M-QAM signal constellations.

where $R = \sqrt{x^2 + y^2}$, $\phi = \tan^{-1}(y/x)$, and T is the symbol duration. In other systems (e.g., storage), orthogonal pulses may be used, such as those illustrated in Figure 9.3. The set of signal symbols in the two-dimensional IQ-plane is called a *constellation*, and symbols are called *signal points*.

From the viewpoint of digital signal processing, *modulation* is a *mapping*. That is, the process of assigning a ν-dimensional binary vector $\bar{b}$ to a signal point $(x(\bar{b}), y(\bar{b}))$ in the constellation. In previous chapters, only BPSK modulation was considered, in which case $\nu = 1$. For $\nu > 1$, there are many possible assignments of bits-to-signal points. That is, many ways to *label the signal points*. Figure 9.4 shows an example of a QPSK constellation ($\nu = 2$) with two different (nonequivalent) mappings.

Moving from binary modulation to 2^ν-ary modulation has the advantage that the number of bits per symbol is increased by a factor of ν, thus *increasing the spectral efficiency* of the system. On the other hand, the required average energy of the signal increases (QAM) or the distance between modulation symbols decreases (PSK). In practice, transmitted power is limited to a maximum value. This implies that the *signal points become closer* to each other. Recall that the probability of error in an AWGN channel between two signal points

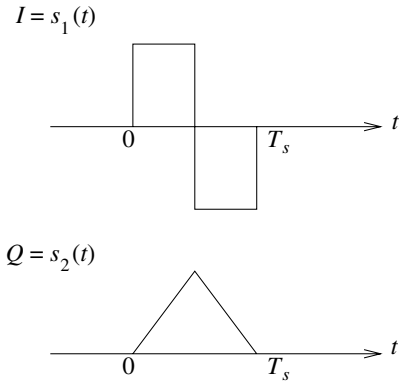
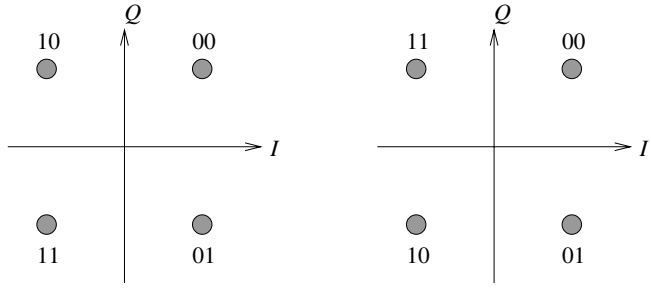


Figure 9.3 An example of two orthogonal pulses.



(a) GRAY MAP (b) NATURAL MAP

Figure 9.4 Two different mappings of a QPSK constellation.

separated by a Euclidean distance equal to D_e is (Haykin 2001; Proakis 2001)

$$\Pr(\epsilon) = Q\left(\sqrt{\frac{D_e^2}{2N_0}}\right), \tag{9.3}$$

where $Q(x)$ is given by Equation (1.2). As a result, a *higher probability of error* is experienced at the receiver end. In this sense, the function of error correcting coding is to reduce the probability of error $\Pr(\epsilon)$ and to improve the quality of the system.

9.1.2 Coded modulation

In 1974, Massey introduced the key concept of treating coding and modulation as a joint signal processing entity (Massey 1974) (Figure 9.5), that is, the *coordinated design* of error correcting coding and modulation schemes.

Two fundamental questions on combining coding and modulation arise:

1. How to construct the bits-to-symbols mapping?
2. How to assign coded *bit sequences* to coded *symbol sequences*?

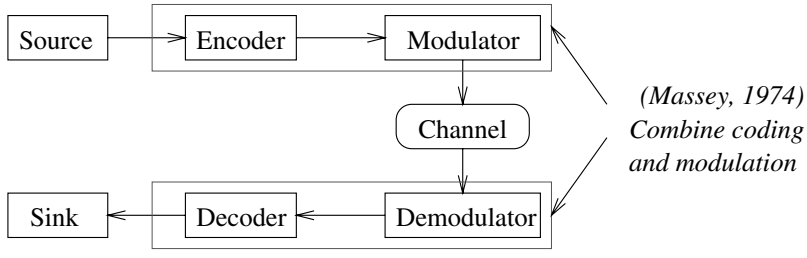


Figure 9.5 The idea of joint coding and modulation.

Two basic approaches were proposed in the 1970s to design coded modulation systems:

1. *Trellis-Coded Modulation (TCM) (Ungerboeck 1982)*

   Apply a natural mapping of bits to signals, through set partitioning. Given an underlying finite-state machine, assign symbol sequences to trellis paths. Perform *Viterbi decoding* at the receiver.

2. *Multilevel Coded Modulation (MCM) (Imai and Hirakawa 1977)*

   Apply a mapping of codewords to *bit positions*, through a binary partition. For 2^{ν}-ary modulation, use ν error correcting codes, one per label bit. Perform *multistage decoding* at the receiver.

In both TCM (Trellis Coded Modulation) and MCM (Multilevel Coded Modulation), the basic idea is to *expand the constellation* to obtain the *redundancy* needed for error correcting coding, and then to use coding to increase the *minimum Euclidean distance* between sequences of modulated signals.

9.1.3 Distance considerations

To illustrate how error correcting codes and digital modulation can be combined and the consequent increase in the minimum distance between signal sequences from an expanded signal constellation, consider the following.

Example 9.1.2 *A block-coded QPSK modulation scheme is shown in Figure 9.6. Codewords of the extended Hamming* $(8, 4, 4)$ *code are divided into four pairs of symbols and mapped to QPSK signal points with Gray mapping. A nice feature of Gray labeling of QPSK points is that the squared Euclidean distance between points is equal to twice the Hamming distance between their labels. As a result, a block-coded modulation scheme with* $\mu = 1$ *bit/symbol and a* minimum squared Euclidean distance *(MSED)* $D^2_{\min} = 8$ *is obtained. An uncoded system with the same spectral efficiency is BPSK, which has an MSED* $D^2_{unc} = 4$. *It follows that the* asymptotic coding gain *of this scheme is*

$$G = 10 \log_{10} \left(\frac{D^2_{\min}}{D^2_{unc}} \right) = 3 \text{ dB}. \tag{9.4}$$

Over an AWGN channel, this coded QPSK modulation requires half the power of uncoded BPSK modulation, to achieve the same probability of error $P(\epsilon)$. *Figure 9.7 shows simulation results of this scheme.*

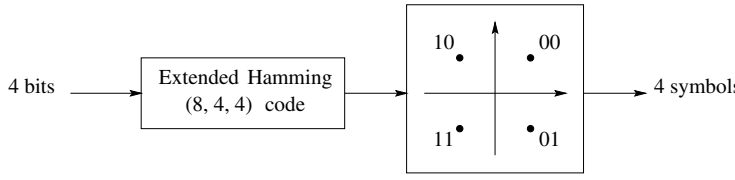


Figure 9.6 Block-coded QPSK modulation using an extended Hamming (8, 4, 4) code.

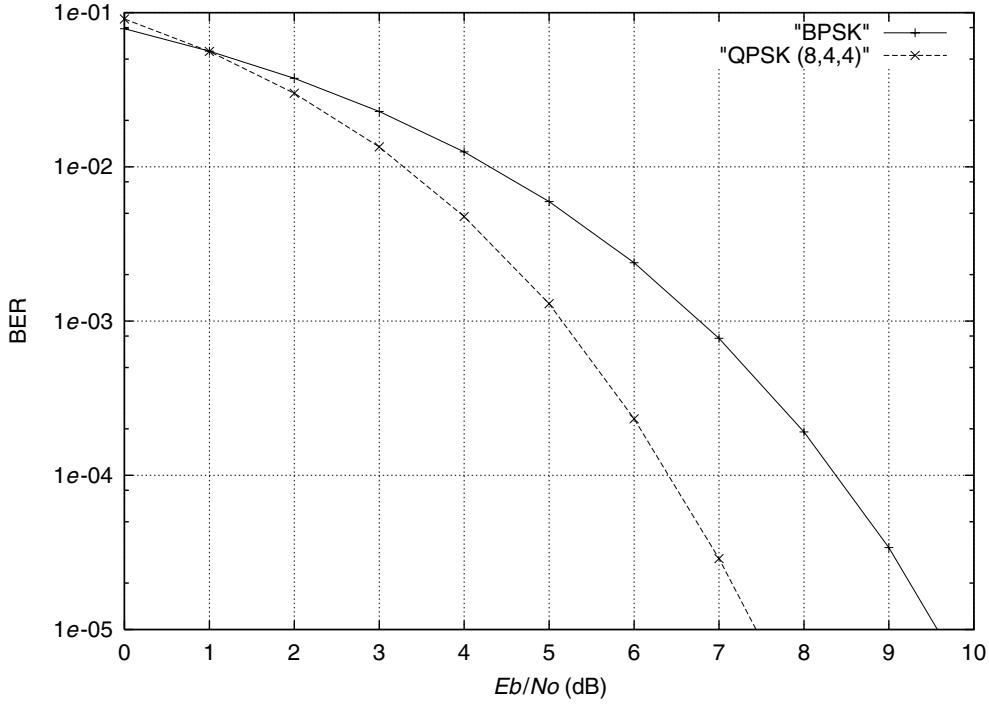


Figure 9.7 Simulation results of a block-coded QPSK modulation using an extended Hamming (8, 4, 4) code. AWGN channel.

Therefore, while the use of an expanded 2^{ν}-th modulation causes the distance between signal points to decrease, a properly selected error correcting code can make sequences of signal points at a minimum distance larger than that of an uncoded system, with the same spectral efficiency.

9.2 Trellis-coded modulation (TCM)

Proposed by Ungerboeck in 1976, the main idea in TCM is to perform *mapping by set partitioning* (Ungerboeck 1987a,b). A basic trellis structure, associated with the state transitions of a *finite-state machine*, is selected and signal subsets are mapped to trellis branches.

For systems that require high spectral efficiency, uncoded bits may be assigned to parallel branches in the trellis.

9.2.1 Set partitioning and trellis mapping

Bit labels assigned to the signal points are determined from a *partition* of the constellation. A 2^{ν}-th modulation signal set $\mathcal{S}$ is *partitioned* in ν levels. For $1 \leq i \leq \nu$, at the i-th partition level, the signal set is divided into two subsets $\mathcal{S}_i(0)$ and $\mathcal{S}_i(1)$ such that the *intraset distance*, δ_i^2, is maximized. A *label bit* $b_i \in \{0, 1\}$ is associated with the subset choice, $\mathcal{S}_i(b_i)$, at the i-th partition level. This partitioning process results in a *labeling* of the signal points. Each signal point in the set has a unique ν-bit label $b_1 b_2 \ldots b_\nu$, and is denoted by $s(b_1, b_2, \ldots, b_\nu)$. With this *standard (Ungerboeck) partitioning* of a 2^{ν}-th modulation signal constellation, the intraset distances are in nondecreasing order $\delta_1^2 \leq \delta_2^2 \leq \cdots \leq \delta_\nu^2$. This strategy corresponds to a *natural labeling* for M-PSK modulations, that is, binary representations of integers whose value increases clockwise (or counterwise). Figure 9.8 shows a natural mapping of bits to signals for the case of 8-PSK modulation, with $\delta_1^2 = 0.586$, $\delta_2^2 = 2$, and $\delta_3^2 = 4$.

Ungerboeck regarded the encoder *"simply as a finite-state machine with a given number of states and specified state transitions"*. He gave a set of pragmatic rules to map signal subsets and points to branches in a trellis. These rules can be summarized as follows:

Rule 1:

All subsets should occur in the trellis with equal frequency.

Rule 2:

State transitions that begin and end in the same state should be assigned subsets separated by the largest Euclidean distance.

Rule 3:

Parallel transitions are assigned signal points separated by the largest Euclidean distance (the highest partition levels).

The general structure of a TCM encoder is shown in Figure 9.9. In the general case of a rate $(\nu - 1)/\nu$ TCM system, the trellis structure is inherited from a $k/(k + 1)$ convolutional encoder. The uncoded bits introduce *parallel branches* in the trellis.

Example 9.2.1 *In this example, a 4-state rate-2/3 TCM system is considered. A constellation for 8-PSK modulation is shown in Figure 9.8. The spectral efficiency is $\mu = 2$ bits/symbol. A block diagram of the encoder is shown in Figure 9.10. The binary convolutional code is the same memory-2 rate 1/2 code that was used in Chapter 5. Note from Figure 9.11 that the trellis structure is the same as that of the binary convolutional code, with the exception that every branch in the original diagram is replaced by two parallel branches associated with the uncoded bit u_1.*

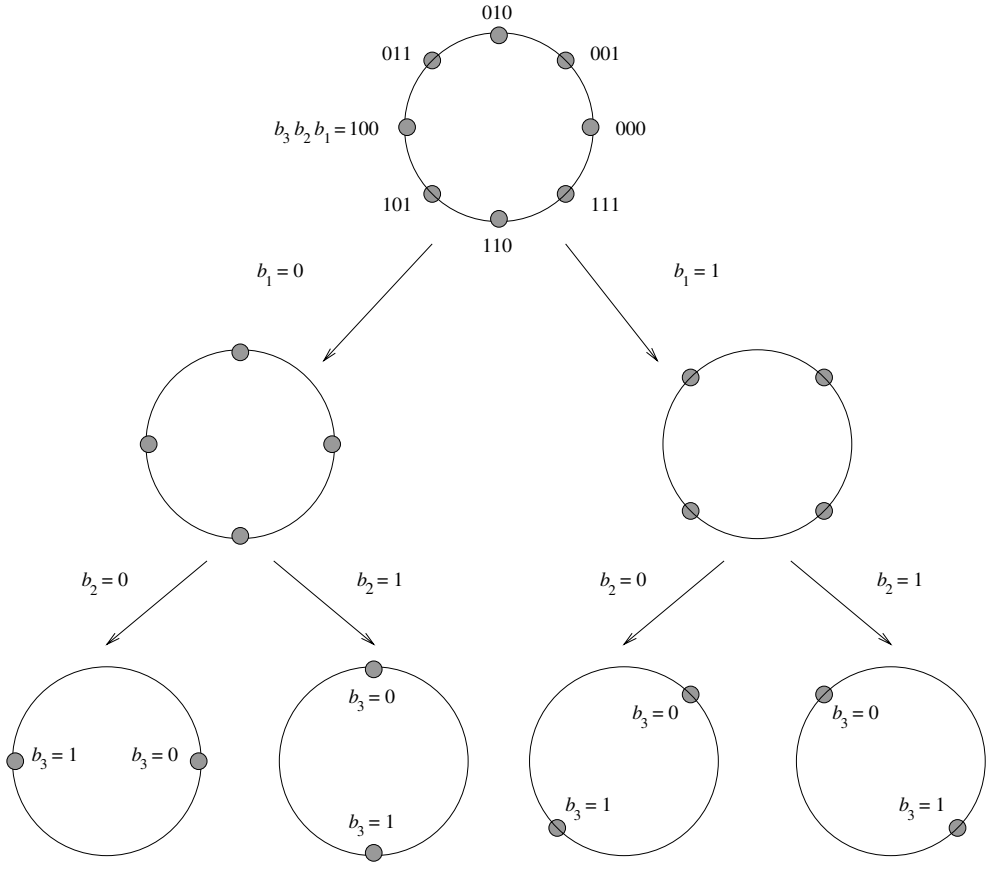


Figure 9.8 Natural mapping of an 8-PSK constellation.

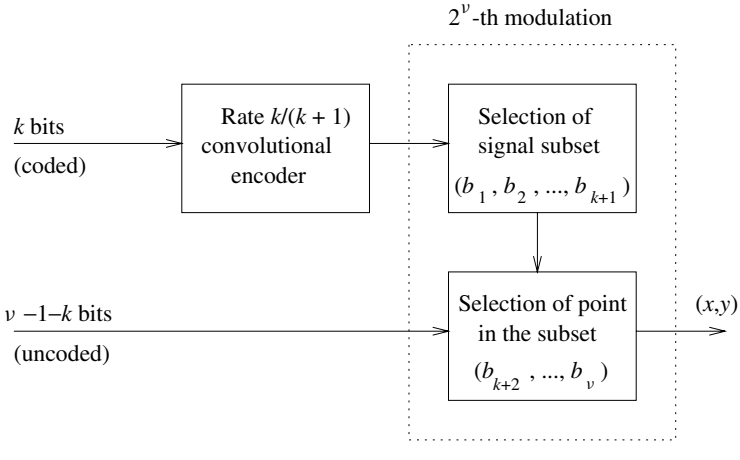


Figure 9.9 General encoder of rate-$(\nu - 1)/\nu$ trellis-coded modulation.

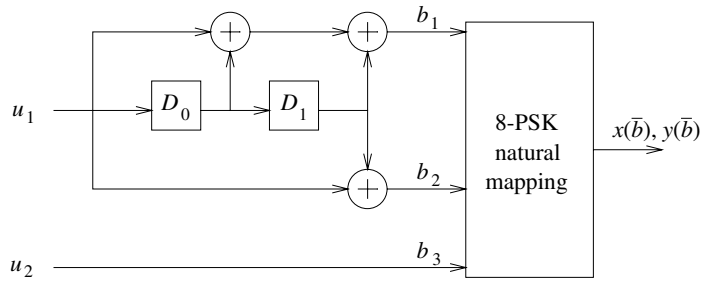


Figure 9.10 Encoder of a 4-state rate-2/3 trellis-coded 8-PSK modulation.

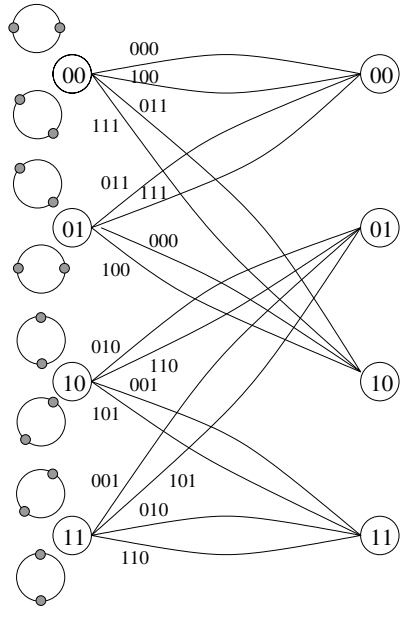


Figure 9.11 Trellis structure of a rate-2/3 trellis-coded 8-PSK modulation based on the encoder of Figure 9.10.

9.2.2 Maximum-likelihood decoding

The Viterbi algorithm[2] can be applied to decode the most likely TCM sequences, provided the branch metric generator is modified to include parallel branches. Also, the selection of the winning branch and surviving uncoded bits should be changed. The survivor path (or trace-back) memory should include the $(\nu - 1 - k)$ uncoded bits as opposed to just one bit for rate $1/n$ binary convolutional codes. It is also important to note that in 2^{ν}-th PSK or QAM modulation, the *correlation metrics* for *two-dimensional* symbols are of the form $x_p x_r + y_p y_r$, where (x_p, y_p) is a reference signal point in the constellation and (x_r, y_r) is

[2]The Viterbi algorithm is discussed in Sections 5.5 and 7.2.

the received signal point. All other implementation issues discussed in Sections 5.5 and 7.2 apply to TCM decoders.

9.2.3 Distance considerations and error performance

The error performance of TCM can be analyzed in the same way as for convolutional codes. That is, a weight-enumerating sequence can be obtained from the state diagram of the TCM encoder, as in Section 5.3. The only difference is that the powers are not integers (Hamming distances) but real numbers (Euclidean distances). Care needs to be taken of the fact that the state transitions contain parallel branches. This means that the labels of the modified state diagram contain two terms. See Biglieri *et al.* (1991).

Example 9.2.2 *Figure 9.12 shows the modified state diagram for the 4-state TC 8-PSK modulation of Example 9.2.1. The branches in the trellis have been labeled with integers corresponding to the eight phases of the modulation signals. To compute the weight enumerating sequence $T(x)$, the same procedure as in Section 5.3 is applied. Alternatively, by directly analyzing the trellis structure in Figure 9.13, it can be deduced that the MSED between coded sequences is*

$$D_C^2 = \min\{D_{\text{par}}^2, D_{\text{tre}}^2\} = 3.172,$$

which, when compared to an uncoded QPSK modulation system with the same spectral efficiency of 2 bits/symbol, gives an asymptotic coding gain of 2 dB.

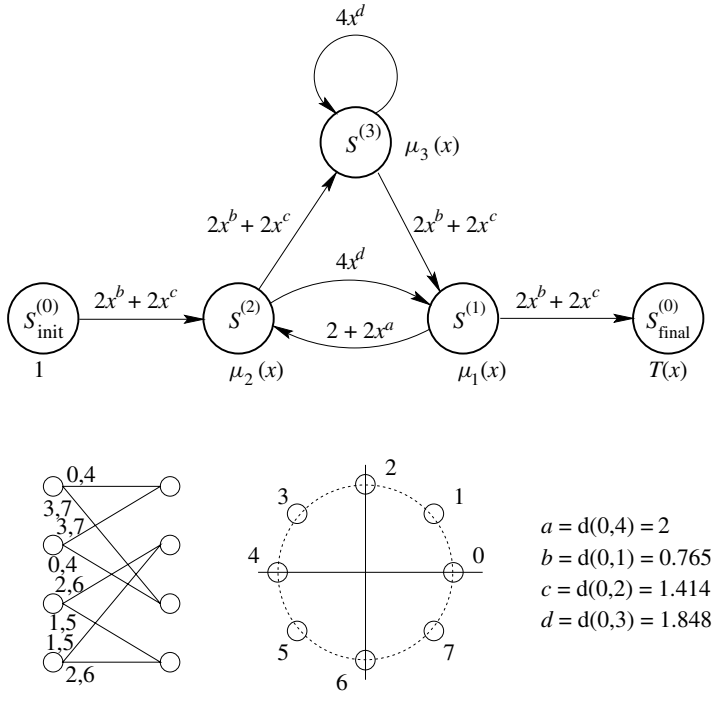


Figure 9.12 The modified state diagram of a 4-state TC 8-PSK modulation scheme.

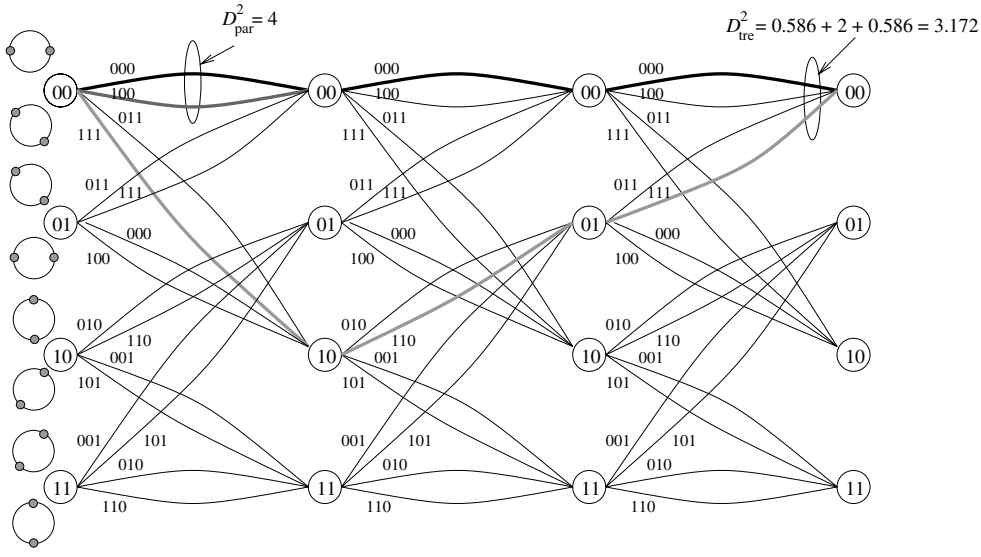


Figure 9.13 Two paths at minimum squared Euclidean distance in the trellis of Example 9.2.1.

9.2.4 Pragmatic TCM and two-stage decoding

For practical considerations, it was suggested in Viterbi *et al.* (1989), Zehavi and Wolf (1995) that the 2^{ν}-th modulation signal constellations be partitioned in such a way that the cosets at the top two partition levels are associated with the outputs of the standard memory-6 rate-1/2 convolutional encoder. This mapping leads to a *pragmatic TCM system*. With respect to the general encoder structure in Figure 9.9, the value of $k = 1$ is fixed, as shown in Figure 9.14. As a result, the trellis structure of pragmatic TCM remains the same, as opposed to Ungerboeck-type TCM, for all values of $\nu > 2$. The difference is that the number of parallel branches $\nu - 2$ increases with the number of bits per symbol. This suggests a two-stage decoding method in which, at the first stage, the parallel branches in the trellis "collapse" into a single branch and a conventional off-the-shelf Viterbi decoder (VD) used to estimate the coded bits associated with the two top partition levels. In a second decoding stage, based on the estimated coded bits and the positions of the received symbols, the uncoded bits are estimated. Figure 9.15 is a block diagram of a two-stage decoder of pragmatic TCM.

In Morelos-Zaragoza and Mogre (2001), a symbol transformation is applied to the input symbols that enables the use of a VD without changes in the branch metric computation stage. The decoding procedure is similar to that presented in Carden *et al.* (1994), Pursley and Shea (1997), with the exception that, with symbol transformation, the Viterbi algorithm can be applied as if the signals were BPSK (or QPSK) modulated. This method is described below for M-PSK modulation.

Specifically, let (x, y) denote the I and Q coordinates of a received M-PSK symbol with amplitude $r = \sqrt{x^2 + y^2}$ and phase $\phi = \tan^{-1}(y/x)$. On the basis of ϕ, a transformation is

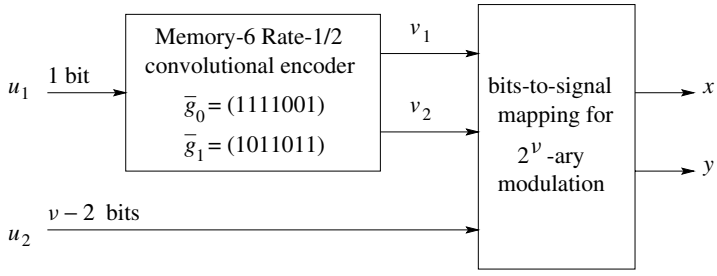


Figure 9.14 Block diagram of an encoder of pragmatic TCM.

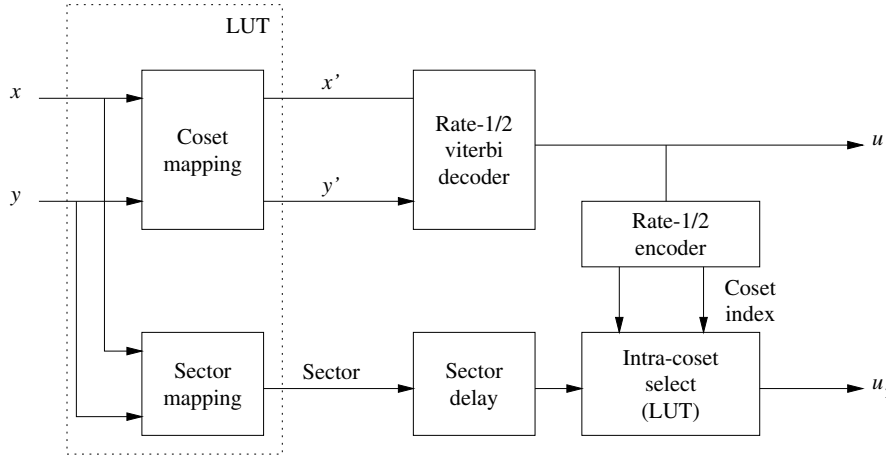


Figure 9.15 Block diagram of a two-stage decoder of pragmatic TCM.

applied such that the M-PSK points are mapped into "coset" points labeled by the outputs of a rate-1/ 2 64-state convolutional encoder.

For TCM with M-th PSK modulation, $M = 2^v$, $v > 2$, let ξ denote the number of *coded bits per symbol*,[3] where $\xi = 1, 2$. Then the following rotational transformation is applied to each received symbol, (x, y), to obtain an input symbol (x', y') to the VD,

$$\begin{aligned} x' &= r\cos\left[2^{v-\xi}(\phi - \Phi)\right], \\ y' &= r\sin\left[2^{v-\xi}(\phi - \Phi)\right], \end{aligned} \tag{9.5}$$

where Φ is a constant phase rotation of the constellation that affects all points equally. Under the transformation (9.5), a $2^{m-\xi}$-PSK coset in the original 2^m-PSK constellation "collapses" into a coset point in a 2^ξ-PSK coset constellation in the $x' - y'$ plane.

Example 9.2.3 *A rate-2/3 trellis-coded 8-PSK modulation with 2 coded bits per symbol is considered. Two information bits* (u_1, u_2) *are encoded to produce three coded bits* (u_2, v_2, v_1),

[3]The case $\xi = 2$ corresponds to conventional TCM with 8-PSK modulation. The case $\xi = 1$ is used in TCM with coded bits *distributed over two 8-PSK signals*, such as the rate-5/6 8-PSK modulation scheme proposed in the DVB-DSNG specification (ETSI 1997).

which are mapped onto an 8-PSK signal point, where (v_2, v_1) *are the outputs of the standard rate-1/2 64-state convolutional encoder.*[4] *The signal points are labeled by bits* (u_2, v_2, v_1) *and the pair* (v_2, v_1) *is the index of a coset of a BPSK subset in the 8-PSK constellation, as shown at the top of Figure 9.16.*

In this case $\phi' = 2\phi$ *and, under the rotational transformation, a BPSK subset in the original 8-PSK constellation collapses to a coset point of the QPSK coset constellation in the* $x' - y'$ *plane, as shown in Figure 9.16. Note that both points of a given BPSK coset have the same value of* ϕ'*. This is because their phases are given by* ϕ *and* $\phi + \pi$*.*

The output of the VD is an estimate of the coded information bit, u_1*. To estimate the uncoded information bit,* u_2*, it is necessary to reencode* u_1 *to determine the most likely coset index. This index and a sector in which the received 8-PSK symbol lies can be used to decode* u_2*. For a given coset, each sector S gives the closest point (indexed by* u_2*) in the BPSK pair to the received 8-PSK symbol. For example, if the decoded coset is (1,1) and the received symbol lies within sector 3, then* $u_2 = 0$*, as can be verified from Figure 9.16.*

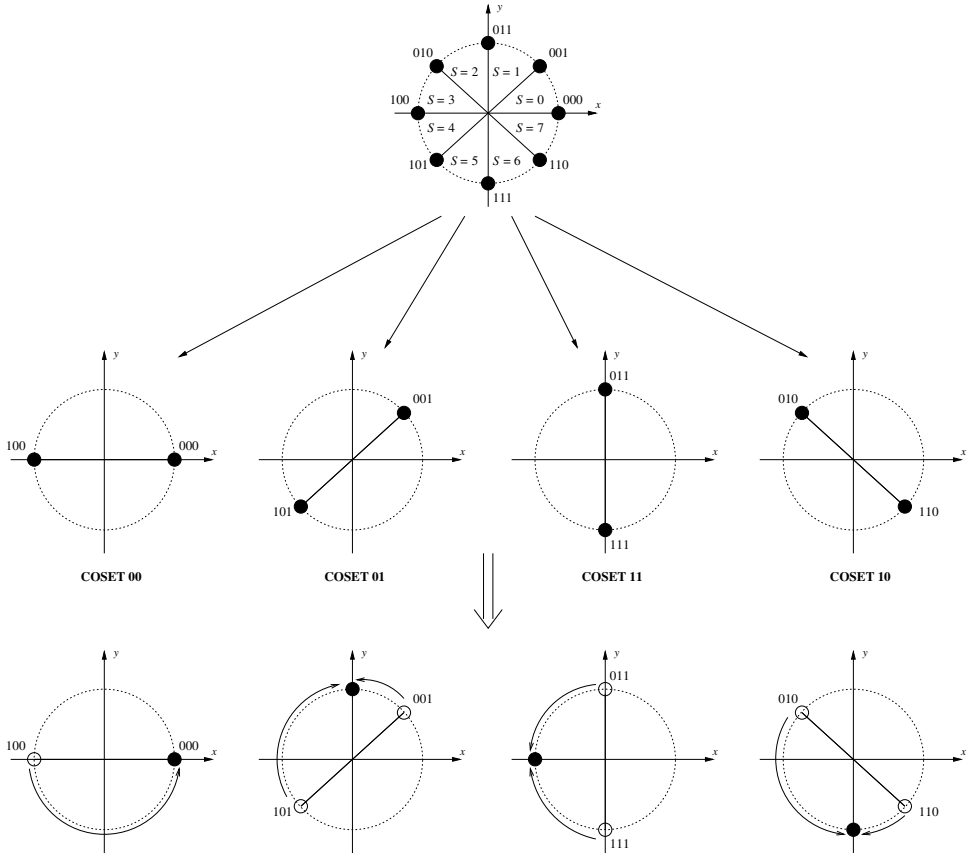


Figure 9.16 Partitioning of an 8-PSK constellation ($\xi = 2$) and coset points.

[4] v_1 and v_2 are the outputs from generators 171 and 133, in octal, respectively.

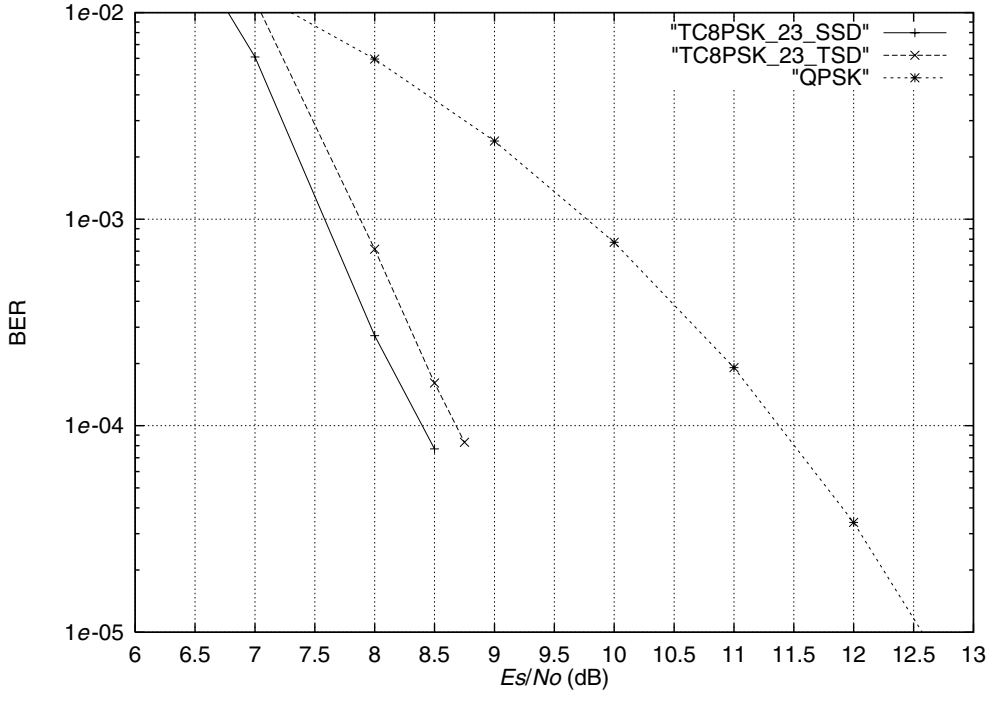


Figure 9.17 Simulation results of MLD versus two-stage decoding for pragmatic 8-PSK modulation.

Figure 9.17 shows simulation results of MLD (with the legend "TC8PSK_23_SSD") and two-stage decoding of pragmatic TC-8PSK (legend "TC8PSK_23_TSD"). With two-stage decoding, a loss in performance of only 0.2 dB is observed compared with MLD.

A similar transformation can be applied in the case of M-QAM; the difference is that the transformation is based solely on the I-channel and Q-channel symbols. That is, there is no need to compute the phase. An example is shown in Figure 9.18 for TC 16-QAM with $\xi = 2$ coded bits per symbol. The coded bits are now the indexes of cosets of QPSK subsets. The transformation of 16-QAM modulation ($\xi = 2$) is given by a kind of "modulo 4" operation:

$$x' = \begin{cases} x, & |x| \leq 2; \\ (x-4), & x \geq 2; \\ (x+4), & x < 2, \end{cases}$$

$$y' = \begin{cases} y, & |y| \leq 2; \\ (y-4), & y \geq 2; \\ (y+4), & y < 2. \end{cases}$$

Finally, it is interesting to note that a pragmatic TCM system with a turbo code as component code was recently proposed in Wahlen and Mai (2000).

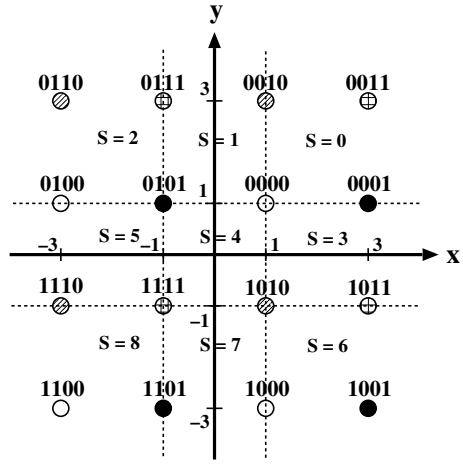


Figure 9.18 16-QAM constellation for pragmatic TCM with two-stage decoding.

9.3 Multilevel coded modulation

The idea in the MCM scheme of Imai and Hirakawa (1977) is to do a binary partition of a 2^ν-ary modulation signal set into ν levels. The components of codewords of ν binary component codes C_i, $1 \le i \le \nu$, are used to index the cosets at each partition level. One of the advantages of MCM is the flexibility of designing coded modulation schemes by coordinating the intraset Euclidean distances, δ_i^2, $i = 1, 2, \ldots, \nu$, at each level of set partitioning, and the minimum Hamming distances of the component codes. Wachsmann *et al.* (1999) have proposed several design rules that are based on capacity (by applying the chain rule of mutual information) arguments. Moreover, multilevel codes with long binary component codes, such as turbo codes or LDPC codes, were shown to achieve capacity (Forney 1998; Wachsmann *et al.* 1999).

It is also worthwhile noting that while binary codes are generally chosen as component codes, that is, the partition is binary, the component codes can be chosen from any finite field $GF(q)$ matching the partition of the signal set. Another important advantage of multilevel coding is that (binary) decoding can be performed separately at each level. This *multistage decoding* results in greatly reduced complexity, compared with MLD for the overall code.

9.3.1 Constructions and multistage decoding

For $1 \le i \le \nu$, let C_i denote a binary linear (n, k_i, d_i) code. Let $\bar{v}_i = (v_{i1}, v_{i2}, \ldots, v_{in})$ be a codeword in C_i, $1 \le i \le \nu$. Consider a *permuted time-sharing code* $\pi(|C_1|C_2|\ldots|C_\nu|)$, with codewords

$$\bar{v} = \begin{pmatrix} v_{11}v_{21}\ldots v_{\nu 1} & v_{12}v_{22}\ldots v_{\nu 2} & \ldots & v_{1n}v_{2n}\ldots v_{\nu n} \end{pmatrix}.$$

Each ν-bit component in $\bar{v}$ is the *label* of a signal in a 2^ν-ary modulation signal set $\mathcal{S}$. Then

$$s(\bar{v}) = (s(v_{11}v_{21}\ldots v_{\nu 1}), s(v_{12}v_{22}\ldots v_{\nu 2}), \ldots, s(v_{1n}v_{2n}\ldots v_{\nu n})$$

is a sequence of signal points in $\mathcal{S}$.

The following collection of signal sequences over $\mathcal{S}$,

$$\Lambda \triangleq \{s(\bar{v}) \,:\, \bar{v} \in \pi(|C_1|C_2|\ldots|C_\nu|)\},$$

forms *a ν-level modulation code over the signal set $\mathcal{S}$* or a ν-level coded 2^ν-ary modulation. The same definition can be applied to convolutional component codes.

The rate (spectral efficiency) of this coded modulation system, in bits/symbol, is

$$R = (k_1 + k_2 + \cdots + k_\nu)/n.$$

The MSED of this system, denoted by $D_C^2(\Lambda)$, is given by Imai and Hirakawa (1977)

$$D_C^2(\Lambda) \geq \min_{1\leq i\leq \nu} \{d_i \delta_i^2\}. \tag{9.6}$$

Example 9.3.1 *In this example, a three-level block coded 8-PSK modulation system is considered. The encoder structure is depicted in Figure 9.19. Assuming a unit-energy 8-PSK signal set, and with reference to Figure 9.8, note that the MSED at each partition level is* $\delta_1^2 = 0.586$, $\delta_2^2 = 2$ *and* $\delta_3^2 = 4$.

The MSED of this coded 8-PSK modulation system is

$$D_C^2(\Lambda) = \min\{d_1\delta_1^2, d_2\delta_2^2, d_3\delta_3^2\} = \min\{8 \times 0.586, 2 \times 2, 1 \times 4\} = 4,$$

and the coding gain is 3 dB with respect to uncoded QPSK. The trellises of the component codes are shown in Figure 9.20. The overall trellis is shown in Figure 9.21.

As mentioned in the preceding text, one of the advantages of multilevel coding is that multistage decoding can be applied. Figures 9.22 (a) and (b) show the basic structures used in the encoding and decoding of multilevel codes. Multistage decoding results in reduced complexity (e.g., measured as number of branches in trellis decoding), compared to MLD decoding (e.g., using the Viterbi algorithm and the overall trellis.) However, in multistage decoding, the decoders at early levels regard the later levels as uncoded. This results in more codewords at minimum distance, that is, an increase in *error multiplicity* or *the number of nearest neighbors*. The value of this loss depends on the choice of the component codes and the bits-to-signal mapping, and for BER $10^{-2} \sim 10^{-5}$ can be in the order of several dB.

Example 9.3.2 *In this example, multistage decoding of three-level coded 8-PSK modulation is considered. The decoder in the first stage uses the trellis of the first component code*

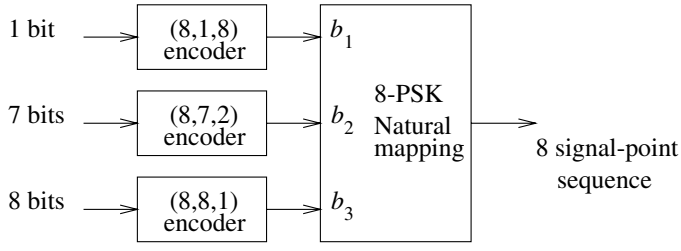


Figure 9.19 Example of MCM with 8-PSK modulation.

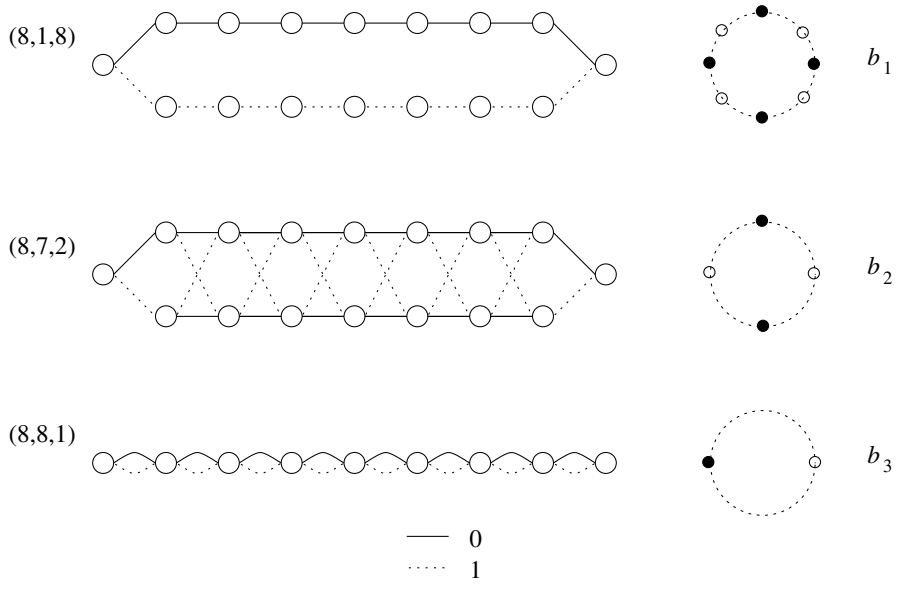


Figure 9.20 Trellises of component codes of an example MCM with 8-PSK modulation.

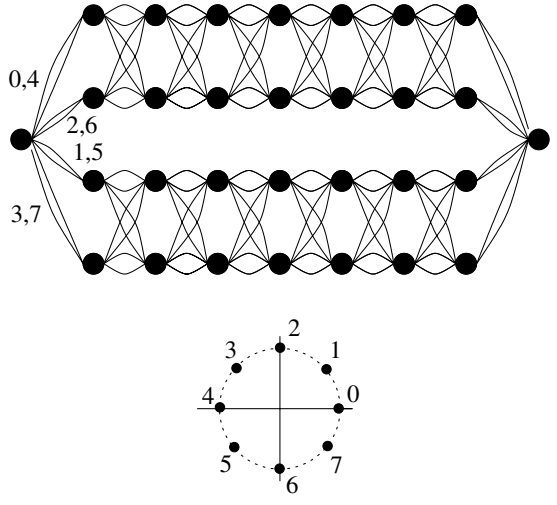
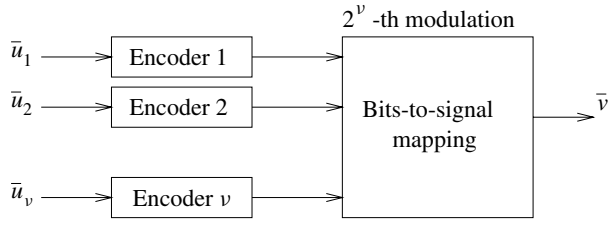


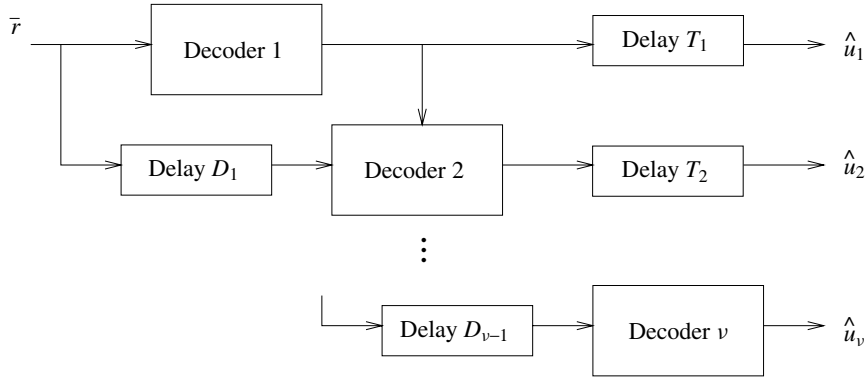
Figure 9.21 Overall trellis of an example MCM with 8-PSK modulation.

C_1. Branch metrics are the distances (correlations) from the subsets selected at the first partitioning level to the received signal sequence, as illustrated in Figure 9.23.

Once a decision is made in the first stage, it is passed on to the second stage. The decoder in the second stage uses the trellis of the second component code with information from the first stage. For 8-PSK modulation, if the decoded bit in the first stage is $b_1 = 0$, then



(a) Multilevel Encoder



(b) Multistage Decoder

Figure 9.22 Basic structures of an encoder and a decoder of multilevel coded modulation systems.

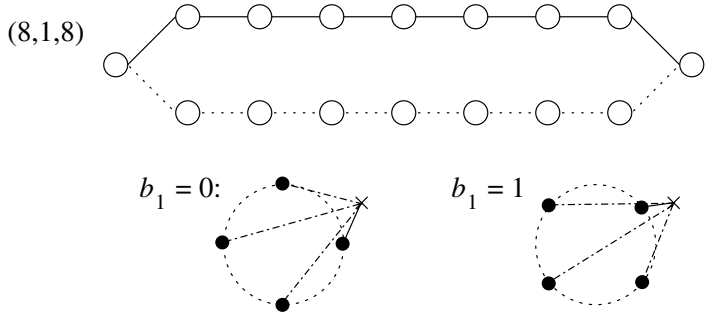


Figure 9.23 Trellis and symbols used in metric computations in the first decoding stage.

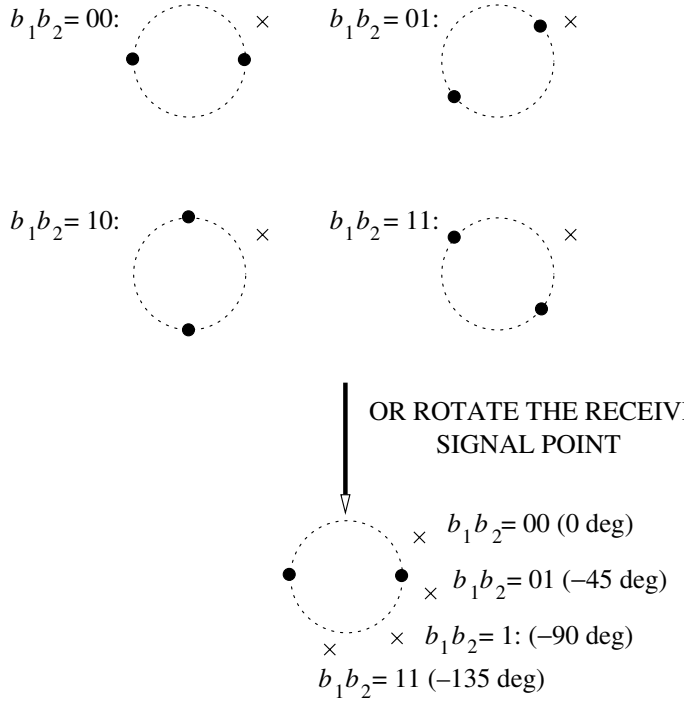


Figure 9.24 Trellis and symbols used in metric computations in the third decoding stage.

the received signal sequence is unchanged. If the decoded bit is $b_1 = 1$*, then the received signal is rotated by* 45°*. Again, branch metrics are distances (correlations) from the subsets selected at the second partitioning stage – given the decision at the first decoding stage – to the received signal sequence.*

Finally, on the basis of the decisions in the first two decoding stages, the decoder of the third component is used. The branch metrics are the same as for BPSK modulation. There are four rotated versions of the BPSK constellation, in accordance with the decisions in the first two decoding stages. Therefore, one approach is to rotate the received signal according to the decisions on b_1b_2 *and use the same reference BPSK constellation. This is illustrated in Figure 9.24.*

For medium to large code lengths, hybrid approaches may be the way to go for ultimate MCM performance, with powerful turbo codes used in the top partition levels and binary codes with hard-decision decoding assigned to lower partition levels. These combinations can achieve excellent performance (Wachsmann *et al.* 1999).

9.3.2 Unequal error protection with MCM

Because of its flexibility in designing the minimum Euclidean distances between coded sequences at each partition level, MCM is an attractive scheme to achieve *unequal error protection* (UEP). However, great care has to be exercised in choosing the bits-to-signal

mapping so that the desired UEP capabilities are not destroyed. This issue was investigated in Isaka *et al.* (2000), Morelos-Zaragoza *et al.* (2000), where several partitioning approaches were introduced that constitute generalizations of the block (Wachsmann *et al.* 1999) and (Ungerboeck 1982) partitioning rules.

In these *hybrid partitioning* approaches, some partition levels are nonstandard while at other levels partitioning is performed using Ungerboeck's rules (Ungerboeck 1982). In this manner, a good trade-off is obtained between error coefficients and intralevel Euclidean distances. To achieve UEP capabilities, the Euclidean distances at each partition level are chosen such that

$$d_1\delta_1^2 \geq d_2\delta_2^2 \geq \cdots \geq d_\nu\delta_\nu^2. \tag{9.7}$$

For $1 \leq i \leq \nu$, let $\bar{v}_i(\bar{u}_i)$ be the codeword of C_i in correspondence to a k_i-bit message vector $\bar{u}_i$, and let $\bar{s} = \bar{s}(\bar{u})$ and $\bar{s}' = \bar{s}(\bar{u}')$ denote coded 2^ν-th modulation signal sequences corresponding to message vectors $\bar{u} = (\bar{u}_1, \bar{u}_2, \ldots, \bar{u}_\nu)$ and $\bar{u}' = (\bar{u}'_1, \bar{u}'_2, \ldots, \bar{u}'_\nu)$, respectively. The *Euclidean separations* (Yamaguchi and Imai 1993) between coded sequences at the i-th partition level, for $i = 1, \ldots, \nu$, are defined as

$$\mathbf{s}_i \stackrel{\Delta}{=} \min\left\{d\left(\bar{s}, \bar{s}'\right) : \bar{u}_i \neq \bar{u}'_i, \bar{u}_j = \bar{u}'_j, j < i\right\}, \tag{9.8}$$

with $\mathbf{s}_1 = d_1\delta_1^2$, $\mathbf{s}_2 = d_2\delta_2^2$, ..., $\mathbf{s}_\nu = d_\nu\delta_\nu^2$. For transmission over an AWGN channel, the set of inequalities (9.7) results in message vectors with decreasing error protection levels.

It is known (Wachsmann *et al.* 1999) that Ungerboeck's partitioning rules (Ungerboeck 1982) are inappropriate for multistage decoding of MCMs, at low to medium signal-to-noise ratios, because of the large number of nearest neighbor sequences (NN) in the first decoding stages.

Example 9.3.3 *Figure 9.25 shows simulation results of the performance of a three-level coded 8-PSK modulation with the* $(64, 18, 22)$, $(64, 57, 4)$ *and* $(64, 63, 2)$ *extended BCH codes (ex-BCH codes) as component codes* C_i, $i = 1, 2, 3$, *respectively. The Euclidean separations are* $\mathbf{s}_1 = 12.9$, $\mathbf{s}_2 = \mathbf{s}_3 = 8$, *for 18 and 120 information bits, respectively (asymptotic coding gains of 8.1 dB and 6 dB, respectively). The adverse effects of the number of NN (or error coefficient) in the first decoding stage are such that the coding gains are greatly reduced.*

In the following text, a UEP scheme based on nonstandard partitioning is presented. The reader is referred to (Isaka *et al.* 2000; Morelos-Zaragoza *et al.* 2000; Wachsmann *et al.* 1999) for details on multilevel coding design for both conventional (equal error protection) and UEP purposes.

Nonstandard partitioning

The block partitioning (Wachsmann *et al.* 1999) shown in Figure 9.26 (a) is used to construct three-level coded 8-PSK modulation schemes with UEP. In the figure, the color black is used to represent signal points whose label is of the form $\mathbf{0}b_2b_3$, with $b_2, b_3 \in \{0, 1\}$. Similarly, the color white is used for points with labels $\mathbf{1}b_2b_3$. A circle indicates that the label is of the form $b_1\mathbf{0}b_3$, $b_1, b_3 \in \{0, 1\}$, while a square is used to represent signal points with labels $b_1\mathbf{1}b_3$.

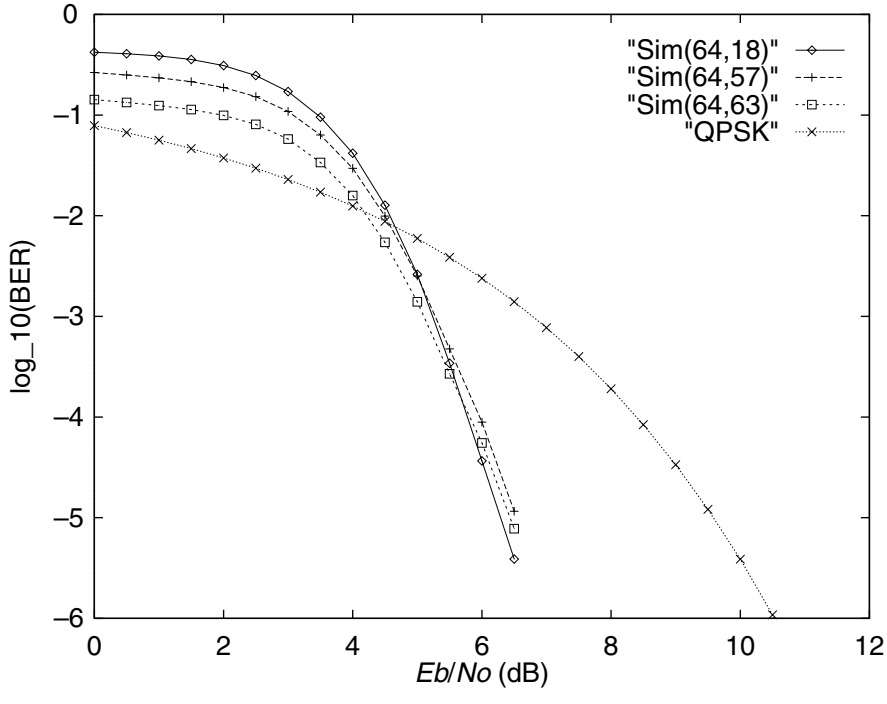


Figure 9.25 Simulation results of a three-level coded 8-PSK modulation with Ungerboeck mapping.

It can be seen from Figure 9.26 (b) that to determine the value of the first label bit, b_1, only the X-coordinate is sufficient. If a signal point is on the left half plane ($X < 0$), then it corresponds to $b_1 = 0$; otherwise it corresponds to $b_1 = 1$. In the same way, the Y-coordinate is sufficient to determine the value of the second label bit b_2. If a signal point lies in the upper half plane ($Y > 0$), then $b_2 = 0$; otherwise $b_2 = 1$. This property of block partitioning allows the first and second levels to be decoded *independently* or *in parallel*. A similar observation led to the development of *parallel decoding* (PD) for multilevel codes with Gray mapping in Schramm (1997).

Multistage decoding

In the first and second decoding stages, the decision variable is just the projection of the received signal sequence onto the X or Y axis, respectively. Figure 9.26 (c) shows a block diagram of a multistage decoder for a three-level coded 8-PSK modulation with block partitioning. The decoders for the first and second stages operate independently on the in-phase and quadrature component of the received signal sequences, $\bar{r}_x$ and $\bar{r}_y$, respectively. Once decisions are made as to the estimates of the corresponding codewords, $\hat{v}_1$ and $\hat{v}_2$, they are passed on to the third decoding stage.

Let $\hat{v}_i = (\hat{v}_{i1}, \hat{v}_{i2}, \ldots, \hat{v}_{in}) \in C_i$ be the decoded codeword at the i-th stage, $i = 1, 2$. Before the third-stage decoding, each two-dimensional coordinate (r_{xj}, r_{yj}) of the received

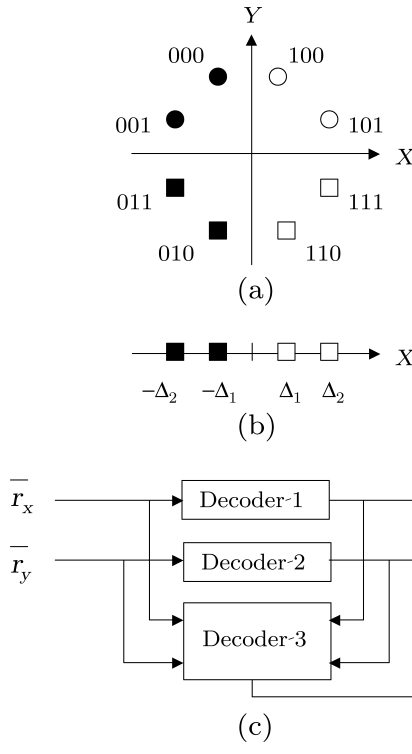


Figure 9.26 An 8-PSK constellation with block partitioning: (a) labeling; (b) X coordinate projections; (c) decoder structure. ($\bar{c}_i$ denote estimated codewords in C_i, $i = 1, 2, 3$.)

Table 9.1 Transformation of received values for third-stage decoding.

$\hat{v}_{1j}$	$\hat{v}_{2j}$	r'_{xj}
0	0	$-\sqrt{2}/2\,(r_{xi} - r_{yi})$
0	1	$-\sqrt{2}/2\,(r_{xi} + r_{yi})$
1	1	$\sqrt{2}/2\,(r_{xi} - r_{yi})$
1	0	$\sqrt{2}/2\,(r_{xi} + r_{yi})$

signal $\bar{r} = (\bar{r}_x, \bar{r}_y)$ is projected onto a one-dimensional coordinate r'_{xj}, $1 \le j \le n$. These r'_{xj} values are the decision variables used by the decoder of C_3. The projection depends on the decoded quadrant, which is indexed by the pair $(\hat{v}_{1j}, \hat{v}_{2j})$, $1 \le j \le n$, as shown in Table 9.1.

This is a scaled rotation of $\bar{r}$ by $\pi/4$, so that the *rotated sequence* $\bar{r}' = (r'_{x1}, r'_{x2}, \ldots, r'_{xn})$ can be decoded using a soft-decision procedure for component code C_3. Note that, unlike Ungerboeck partitioning, the independence between the first and second levels in block partitioning results in *no error propagation* from the first decoding stage to the second.

For $i = 1, 2, \ldots, \nu$, let $A_w^{(i)}$ denote the number of codewords in C_i of weight w. Assuming systematic encoding, a union bound on the bit error probability of the first decoding stage can be written as (Isaka *et al.* 2000; Morelos-Zaragoza *et al.* 2000)

$$P_{b1}^{(NS)} \leq \sum_{w=d_1}^{n} \frac{w}{n} A_w^{(1)} 2^{-w} \sum_{i=0}^{w} \binom{w}{i} Q\left(\sqrt{\frac{2RE_b}{N_0} d_P^2(i)}\right), \tag{9.9}$$

where $d_P^2(i) = \frac{1}{w}[i\Delta_1 + (w-i)\Delta_2]^2$. The probability of a bit error in the second decoding stage upper is also bound by (9.9) using the same arguments above.

The bound (9.9) can be compared with a similar one for the Ungerboeck's partitioning (UG) strategy:

$$P_{b1}^{(UG)} \leq \sum_{w=d_1}^{n} \frac{w}{n} A_w^{(1)} 2^w Q\left(\sqrt{\frac{2RE_b}{N_0} w\Delta_1^2}\right). \tag{9.10}$$

From (9.9) and (9.10), it is observed that while Ungerboeck's partitioning increases exponentially the effect of the nearest neighbor sequences by a factor of 2^w, the block partitioning has for $d_P^2(w) = w\Delta_1^2$ an error coefficient term, 2^{-w}, that *decreases exponentially* with the distances of the first-level component code. As a result, for practical values of E_b/N_O, the block partitioning may yield, at the first stage, a real coding gain *even greater than the asymptotic coding gain*. This is a desirable feature of the coded modulation with UEP.

For nonstandard partitioning (NS), the second level is generally designed to have a larger coding gain than the third level. Under this assumption, a good approximation is obtained by assuming that decoding decisions in the first and the second decoding stages are correct.

$$P_{b3}^{(NS)} \lesssim \sum_{w=d_3}^{n} \frac{w}{n} A_w^{(3)} Q\left(\sqrt{\frac{2RE_b}{N_0} w\Delta_1^2}\right). \tag{9.11}$$

Example 9.3.4 *A three-level 8-PSK modulation for UEP can be constructed using extended BCH* $(64, 18, 22)$*,* $(64, 45, 8)$ *and* $(64, 63, 2)$ *codes as the first-, second- and third-level codes, respectively. This coding scheme has a rate equal to* 1.97 *bits per symbol and can be compared with uncoded QPSK modulation, which has approximately the same rate (a difference of only 0.06 dB). Simulation results are shown in Figure 9.27.* $S1(n, k)$ *and* $UB(n, k)$ *denote simulations and upper bounds. A large coding gain of 8.5 dB is achieved at the BER of* 10^{-5} *for 18 most important bits (14.3%) encoded in the first level. In the second and third stages, the corresponding values of coding gain are 2.5 dB and* -4.0 *dB, respectively.*[5]

9.4 Bit-interleaved coded modulation

In Caire *et al.* (1996, 1998) the ultimate approach to pragmatic coded modulation is presented. The system consists of binary encoding followed by a pseudorandom *bit interleaver*. The output of the interleaver is grouped in blocks of ν bits that are assigned, via a *Gray mapping*, to points in a 2^ν-ary modulation constellation. The capacity of this *bit-interleaved*

[5]Note that at this BER, the simulated coding gain at the first decoding stage is even greater than the asymptotic coding gain (8.1 dB) because of the reduced error coefficients.

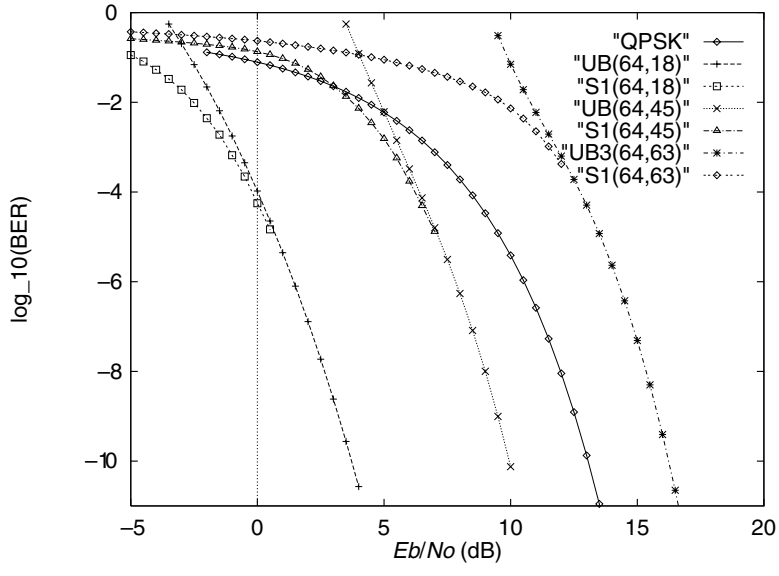


Figure 9.27 Simulation results of a three-level coded 8-PSK modulation with UEP capability. BCH component codes and block partitioning.

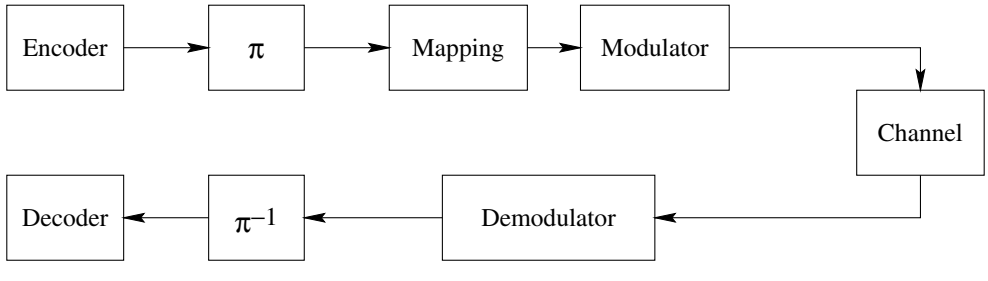


Figure 9.28 A bit-interleaved coded modulation system.

coded modulation (BICM) scheme has been shown to be surprisingly close to the capacity of TCM, when Gray mapping is employed. Moreover, over flat Rayleigh fading channels, BICM outperforms a CM with symbol interleaving (Caire *et al.* 1998). A block diagram of a BICM system is shown in Figure 9.28.

Let $\mathcal{X}$ denote a 2^{ν}-ary signal constellation with minimum distance $D_{\min}$. Let $\ell^i(x)$ be the i-th bit of the label of a signal point x, and let $\mathcal{X}_b^i \subset \mathcal{X}$ be the subset of signal points with labels such that the i-th bit has a value $b \in \{0, 1\}$, for $i = 1, 2, \ldots, \nu$.

9.4.1 Gray mapping

A one-to-one and onto binary map m from $\{0, 1\}^{\nu}$ to $\mathcal{X}$ is said to be a Gray mapping if, for all $i = 1, 2, \ldots, \nu$, and $b \in \{0, 1\}$, each $x \in \mathcal{X}_b^i$ has one nearest neighbor $y \in \mathcal{X}_{b'}^i$ at the most at distance $D_{\min}$, where $b' = b \oplus 1$. Gray mapping is the key component of a BICM

system. Its main function is – ideally – to produce an equivalent channel that has ν parallel, independent and memoryless, binary channels. Each channel corresponds to a position in the label of a signal $x \in \mathcal{X}$. For each codeword at the output of the binary encoder, the interleaver assigns at random a position in the label of the signals to transmit the coded bits.

9.4.2 Metric generation: De-mapping

Before the description of how metrics for an MLD decoder are generated, some notation is needed. Let r denote the channel output after transmission of x. Assuming uniform input distribution, the conditional probability of r given $\ell^i(x) = b$ is

$$p(r|\ell^i(x) = b) = \sum_{x \in \mathcal{X}} p(r|x)\, p(x|\ell^i(x) = b) = 2^{-(\nu-1)} \sum_{x \in \mathcal{X}_b^i} p(r|x). \tag{9.12}$$

Let i denote the position of the coded bit v_j in the label of $x_{\pi(j)}$. At each time j, let v_j be a code symbol and $x_{\pi(j)}$ be the interleaved signal point, received as $r_{\pi(j)}$ after transmission over a noisy channel.

The receiver then produces *bit metrics*

$$\lambda^i(r_{\pi(j)}, b) = \log \left(\sum_{x \in \mathcal{X}_b^i} p(r_{\pi(j)}|x) \right), \tag{9.13}$$

for $b = 0, 1$ and $i = 1, 2, \ldots, \nu$.

An MLD algorithm, such as the Viterbi algorithm, uses the above metrics and makes decisions on the basis of the rule

$$\hat{v_i} = \arg \max_{\bar{v} \in C} \sum_j \lambda^i(r_{\pi(j)}, v_j). \tag{9.14}$$

As before, a max-log approximation of (9.13) is possible, resulting in the approximated bit metric,

$$\lambda_{\mathrm{a}}^i(r_{\pi(j)}, b) = \max_{x \in \mathcal{X}_b^i} \log p(r_{\pi(j)}|x). \tag{9.15}$$

9.4.3 Interleaving

With transmission over an AWGN channel, a short interleaver will suffice. The main purpose is to break the correlation introduced by the 2^ν-ary modulation signal set, which carries ν bits per signal. Therefore, an interleaver of length equal to a few times ν is enough to approach best performance (Caire *et al.* 1998). Note that this interleaver has nothing to do with the interleaver that a turbo code or a block product code would use.

9.5 Turbo trellis-coded modulation

Conceptually, there are various approaches to the combination of turbo codes, or product codes with interleaving, and digital modulation: pragmatic coded modulation (Le Goff *et al.* 1994), turbo TCM with *symbol interleaving* (Robertson and Wörz 1995, 1998) and turbo TCM with *bit interleaving* (Benedetto *et al.* 1996).

9.5.1 Pragmatic turbo TCM

Motivated by the extraordinary performance of turbo coding schemes, another pragmatic coded modulation scheme was introduced by Le Goff *et al.* (1994). Its block diagram is shown in Figure 9.29. The main feature is, as in pragmatic TCM, the use of the turbo encoder and decoder operating as in binary transmission mode. This requires careful computation of the *bit metrics*, as in the case of BICM.

9.5.2 Turbo TCM with symbol interleaving

In Robertson and Wörz (1995), recursive systematic convolutional encoders such as those in Ungerboeck (1982) are proposed to be used as components in an overall coding system similar to that of turbo codes. A block diagram of this scheme is shown in Figure 9.30. As can be seen from the diagram, interleaving operates on symbols of ν bits, instead of on bits for binary turbo codes. There is a need to puncture *redundant symbols*, because of the two paths of modulated signal points. A careful component code (no parallel transitions) and interleaver design (even positions to even positions, odd-to-odd; or even-odd and odd-even) are required. In terms of iterative decoding, it should be noted that the systematic component cannot be separated from the extrinsic one since they are transmitted together in

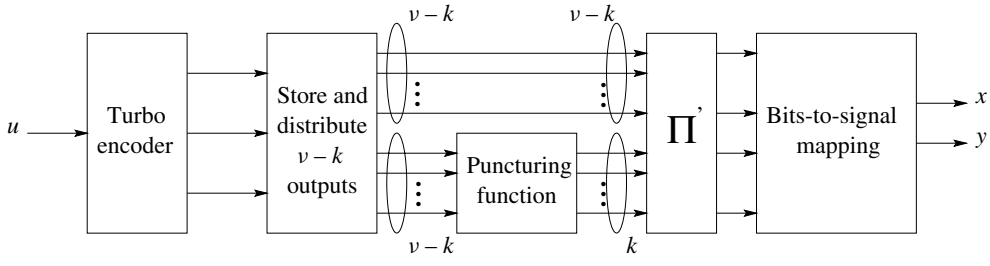


Figure 9.29 Combination of a turbo encoder and digital modulation (Le Goff *et al.* 1994).

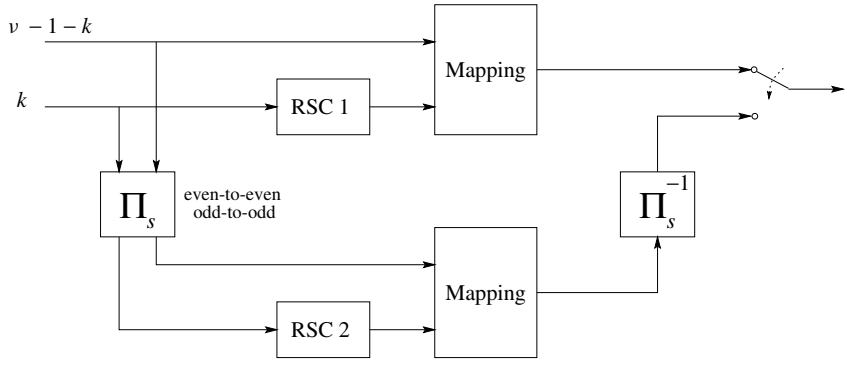


Figure 9.30 The encoder structure of a turbo TCM scheme with symbol interleaving.

one symbol. However, the log-likelihood ratio (LLR) can be separated into an a priori and a systematic-and-extrinsic part. Care must be taken so that the information is not used more than once in the component decoders. This is the reason why redundant symbol puncturing, in the form of a selector, is needed at the output of the encoder (Robertson and Wörz 1998). Figure 9.31 shows a block diagram of an iterative decoder for turbo TCM.

9.5.3 Turbo TCM with bit interleaving

In 1996, (Benedetto *et al.* 1996) proposed symbol puncturing rules such that the outputs of the encoder contain the information bits only once. Moreover, as opposed to symbol interleaving and puncturing of redundant symbols, in this scheme multiple *bit interleavers* are employed. A block diagram of the encoder structure is shown in Figure 9.32, for the case of two component codes.

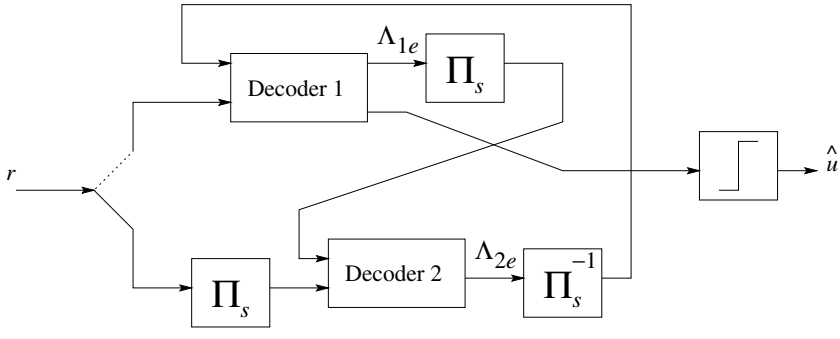


Figure 9.31 An iterative decoder for turbo TCM with symbol interleaving.

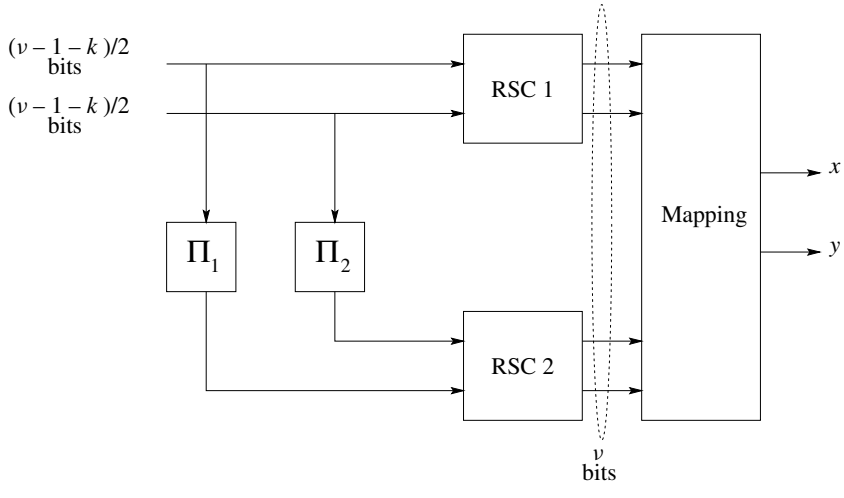


Figure 9.32 Encoder for turbo TCM with bit interleaving.

MAP decoding and bit metrics

The decoder structure for turbo TCM with bit interleaving is similar to that of binary turbo codes. The main difference is that conversion of LLR values from bits to symbols, and from symbols to bits, needs to be performed between decoders (Benedetto *et al.* 1996; Vucetic and J. Yuan 2000). For decoding of turbo TCM with bit interleaving, the LLR values that are computed per bit need to be converted to a symbol level a priori probability. Also, the a priori probabilities per symbol need to be converted to bit level extrinsic LLR values as follows:

Let $\mathcal{X}$ denote the 2^{ν}-ary modulation signal set. The extrinsic information of a bit b_i, with $i = 1, 2, \ldots, \nu$, of a symbol $x(\bar{b}) \in \mathcal{X}$ with label $\bar{b} = (b_1, b_2, \ldots, b_\nu)$, is computed as

$$\Lambda_e(b_i) = \log \left(\frac{\sum\limits_{x(\bar{b}), b_i = 1} e^{\Lambda_e(x(\bar{b}))}}{\sum\limits_{x(\bar{b}), b_i = 0} e^{\Lambda_e(x(\bar{b}))}} \right). \tag{9.16}$$

Similarly, the a priori symbol probability can be computed from the extrinsic LLR value at the bit level through the expression,

$$\Pr\left(\bar{b} = (b_1, b_2, \ldots, b_\nu)\right) = \prod_{i=1}^{\nu} \frac{e^{b_i \Lambda_e(b_i)}}{1 + e^{\Lambda_e(b_i)}}. \tag{9.17}$$

Problems

1. Enumerate all coded sequences for the block coded QPSK modulation in Example 9.1.2.
2. Compute the WES of the 4-state TC 8-PSK modulation scheme of Example 9.2.2. Estimate the performance of this scheme over an AWGN channel.
3. Consider the 4-state encoder of Example 9.2.2 and the 8-PAM constellation shown in Figure 9.33.
   (a) Design a TC 8-PAM modulation scheme. Determine the value of D_C^2.
   (b) Express d as a function of E_s, the average energy-to-noise ratio. Using the WES, estimate the performance of this TCM scheme over an AWGN channel.

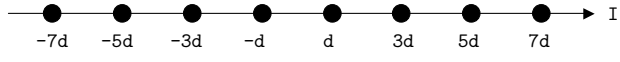


Figure 9.33 An 8-PAM signal constellation.

4. Design a 4-state 8-PSK TCM scheme with $\mu = 2$ bits/symbol and $D_C^2 = 4$.
5. Design a block coded QPSK modulation scheme of length 16 and $\mu = 1$ bit/symbol with $D_{\min}^2 = 16$. What is the coding gain of this scheme? (Hint: Consider codes of length 32.)

6. Construct a three-level modulation code Λ of the shortest length using 8-PSK modulation with $\mu \geq 1.5$ and $D_C^2(\Lambda) \geq 16$.

7. Construct a four-level modulation code Λ using a unit-energy 16-QAM signal constellation and four component binary extended BCH (ex-BCH) codes of length 16 with the highest possible rate μ. Determine the MSED $D_C^2(\Lambda)$ and the coding gain of this system.

8. Repeat the previous problem with ex-BCH codes of length 32.

9. One way to provide UEP capabilities is to design an asymmetric signal constellation. This is done, for example, in the European DVB-T standard for terrestrial broadcasting of digital TV.

   (a) Determine the average energy E_s of the 16-QAM constellation shown in Fig. 9.34. Assume that $d_2 = 2d_1$. Express d_1 and d_2 as a function of E_s.

   (b) For $1 \leq i \leq 4$, compute the probabilities of error $P\{e\}_i$ for each label bit b_i in the 16-QAM constellation shown in Fig. 9.34. Assume transmission over an AWGN channel with double-sided power spectral density $N_0/2$.

   (c) Plot $P\{e\}_i$, $1 \leq i \leq 4$, as a function of the energy-to-noise ratio E_s/N_0 in dB.

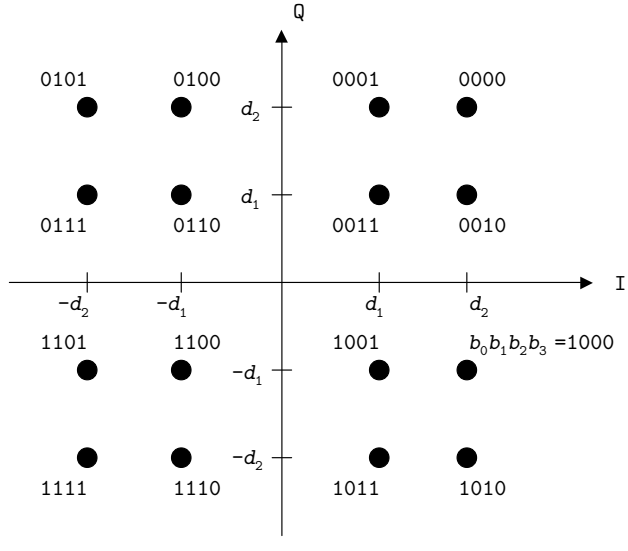


Figure 9.34 An asymmetric 16-QAM signal constellation.

10. Asymmetric signal constellations provide an additional parameter in MCM for UEP. Design an MCM scheme that provides two-levels of UEP, using ex-BCH codes of length 32 and the 16-QAM signal constellation shown in Fig. 9.34. Specify the number of bits and associated coding gain, for each error protection level.

Appendix A

Weight distributions of extended BCH codes

In this appendix, the weight distributions of all extended BCH codes of length up to 128 are presented. The first row of a table indicates the parameters of the code "n,k,d" (this is also the name of a file containing the weight distribution in the ECC web site.) Subsequent rows of a table list the weight w and the number of codewords of this weight A_w. These codes are symmetric, in the sense that the relation $A_w = A_{n-w}$, for $0 \leq w \leq n/2$, holds. Consequently, only half of the distribution is listed.

A.1 Length 8

```
wd.8.1.8
8 1

wd.8.4.4
4 14
8 1

wd.8.7.2
2 28
4 70
```

A.2 Length 16

```
wd.16.05.08
8  30

wd.16.07.06
```

The Art of Error Correcting Coding, Second Edition Robert H. Morelos-Zaragoza
 ISBN: 0-470-01558-6

```
6  48
8  30

wd.16.11.04
4  140
6  448
8  870

wd.16.15.02
2  120
4  1820
6  8008
8  12870
```

A.3 Length 32

```
wd.32.06.16
16 62

wd.32.11.12
12 496
16 1054

wd.32.16.08
8  620
12 13888
16 36518

wd.32.21.06
6  992
8  10540
10 60512
12 228160
14 446400
16 603942

wd.32.26.04
4  1240
6  27776
8  330460
10 2011776
12 7063784
14 14721280
16 18796230

wd.32.31.02
2  496
4  35960
```

```
6  906192
8  10518300
10 64512240
12 225792840
14 471435600
16 601080390
```

A.4 Length 64

```
wd.64.07.32
32 126

wd.64.10.28
28 448
32 126

wd.64.16.24
24 5040
28 12544
32 30366

wd.64.18.22
22 4224
24 5040
26 24192
28 12544
30 69888
32 30366

wd.64.24.16
16 2604
18 10752
22 216576
24 291648
26 1645056
28 888832
30 4419072
32 1828134

wd.64.30.14
14 8064
16 30828
18 631680
20 1128960
22 14022144
24 14629440
26 105057792
28 65046016
```

```
30 282933504
32 106764966

wd.64.36.12
12 30240
14 354816
16 3583020
18 27105792
20 145061280
22 603113472
24 1853011776
26 4517259264
28 8269968448
30 12166253568
32 13547993382

wd.64.39.10
10 13888
12 172704
14 2874816
16 29210412
18 214597824
20 1168181280
22 4794749760
24 14924626752
26 35889146496
28 66620912960
30 96671788416
32 109123263270

wd.64.45.08
8  27288
10 501760
12 12738432
14 182458368
16 1862977116
18 13739292672
20 74852604288
22 306460084224
24 956270217000
26 2294484111360
28 4268285380352
30 6180152832000
32 6991765639110

wd.64.51.06
6  20160
8  1067544
10 37051840
12 801494400
```

```
14 11684617344
16 119266575708
18 879321948288
20 4789977429888
22 19616032446528
24 61193769988008
26 146864398476096
28 273137809339136
30 395577405119232
32 447418802536902

wd.64.57.04
4  10416
6  1166592
8  69194232
10 2366570752
12 51316746768
14 747741998592
16 7633243745820
18 56276359749120
20 306558278858160
22 1255428754917120
24 3916392495228360
26 9399341113166592
28 17480786291963792
30 25316999607653376
32 28634752793916486

wd.64.63.02
2  2016
4  635376
6  74974368
8  4426165368
10 151473214816
12 3284214703056
14 47855699958816
16 488526937079580
18 3601688791018080
20 19619725782651116
22 80347448443237936
24 250649105469666110
26 601557853127198720
28 1118770292985240200
30 1620288010530347000
32 1832624140942591500
```

A.5 Length 128

```
wd.128.008.064
64  254

wd.128.015.056
56 8128
64 16510

wd.128.022.048
48 42672
56 877824
64 2353310

wd.128.029.044
44 373888
48 2546096
52 16044672
56 56408320
60 116750592
64 152623774

wd.128.036.032
32 10668
36 16256
40 2048256
44 35551872
48 353494848
52 2028114816
56 7216135936
60 14981968512
64 19484794406

wd.128.043.032
32 124460
36 8810752
40 263542272
44 4521151232
48 44899876672
52 262118734080
56 915924097536
60 1931974003456
64 2476672341286

wd.128.050.028
28 186944
32 19412204
36 113839296
40 33723852288
44 579267441920
```

```
48 5744521082944
52 33558415333632
56 117224663972352
60 247312085243776
64 31699236111910

wd.128.057.024
24 597408
28 24579072
32 2437776684
36 141621881856
40 4315318568736
44 74150180302848
48 73528925007168
52 4295496356229120
56 15004724612905792
60 31655991621445632
64 4574965317267238

wd.128.064.022
22 243840
24 6855968
26 107988608
28 1479751168
30 16581217536
32 161471882796
34 1292241296640
36 9106516329984
38 53383279307904
40 278420690161824
42 1218666847725184
44 4782630191822848
46 15858705600596992
48 47425684161326912
50 120442185147493376
52 277061634654099456
54 543244862505775360
56 967799721857135168
58 1473287478189735168
60 2041819511308530688
62 2421550630907043328
64 2617075886216910118

wd.128.071.020
20 2674112
22 37486336
24 839699616
26 13825045248
28 188001347136
30 2140095182336
```

```
32 20510697927468
34 166689980438016
36 1156658661471040
38 6886497209935616
40 35363776220195360
42 157207798773129984
44 607468163067994304
46 2045773679068686336
48 6023796954778012480
50 15537040516548126720
52 35191124114633006464
54 70078589269156969984
56 122925566952088660288
58 190054082758956107264
60 259342737902840355456
62 312380032198035579904
64 332409207867786543910

wd.128.078.016
16 387096
18 5462016
20 213018624
22 539859840
24 107350803840
26 1766071867392
28 24074650400768
30 273932927993856
32 2625267567169884
34 2133648518951040
36 14805286631892608
38 881470039149213696
40 4526561735332554624
42 20122606565844068352
44 77755925658495682560
46 261859003134276581376
48 771046023044966543784
50 1988741249124011372544
52 4504463828911859699712
54 8970059328813665832960
56 15734472710169831412480
58 24326922690137187741696
60 33195872219449244835 84
62 39984644079892337086464
64 42548378876302513514950

wd.128.085.014
14 341376
16 22121368
18 856967552
20 27230880768
```

```
22 680417833472
24 13721772977024
26 226128254847488
28 3081454360189952
30 35064826913355520
32 336014520825141340
34 2731238665152128768
36 1894961228051341184
38 112834993226032103936
40 579364846705294996864
42 2575849616631486204416
44 995215572807115388211 2
46 33519982404512223401600
48 98687914666573428364840
50 25457429624880015992 2816
52 576536456040619165149184
54 11482371298197878949785 6
56 2138905488918250206574 08
58 31140346847427153938155 52
60 42488140880205307904225 28
62 51183444087494928984115 2
64 54458627033734445178254 78

wd.128.092.012
12 1194816
14 45646848
16 2751682584
18 110071456768
20 3484410778688
22 8709939355008
24 1756359917165952
26 28944450656120832
28 394426389988237184
30 4488297727663171584
32 43009842715896693084
34 349598717578587531264
36 2425549189872597678976
38 14442886028067639783424
40 7415866532060415580416
42 3297089066350487847690 24
44 12738753386272545590976
46 42905597780091321977640 96
48 12632047099619818751639976
50 32585525337307036591291392
52 73796631469243277615111 04
54 14697442214886651424308 4288
56 25777786830680023693247232
58 39859662823272583189523712
60 54384794596123339347265 4592
```

```
62 655148393268075658872238080
64 69770096246413149145713094

wd.128.099.010
10 796544
12 90180160
14 6463889536
16 347764539928
18 14127559573120
20 44575475469248
22 11149685265467776
24 224811690627712384
26 370489537782191104
28 50486556173121673600
30 574502176730571255552
32 5505259786944679990620
34 44748635720273383143168
36 31047029627999439297536
38 184868941730134924789904
40 94923091237319118515669 76
42 42202740212894624045103744
44 16356041742389991882232512
46 54919165360291990896148 4160
48 16169020228032633502641 49928
50 41709472585828650199604 80640
52 94459687920413917959509 26784
54 18812726104659846681453 12896
56 32995567020535162782224 34304
58 51020368602278287044715 99232
60 69612536825819432117261 21216
62 83858994648317780352552315392
64 89224971989631194512677986758

wd.128.106.008
8  774192
10 105598976
12 11361676032
14 828626841600
16 44515013174520
18 1808265733435392
20 57056968214853376
22 1427159096213901312
24 28775892186952836240
26 474226642406696116224
28 646227907 1735110418944
30 735362788164337729290 24
32 704673252880779235687452
34 57278253704580994610380 80
36 39740197928028063063904768
38 23663224541420383808194 9696
```

```
40 12150155678013131756971523O4
42 54019507474336274569812664O2
44 208711733423668728595660147O0
46 70296531663247684816378728448
48 20696345891289102627719816877O
50 53388124911384079711522346188O
52 12090840053465919059416834368OO
54 24080289414648567100618556825OO
56 42234325785062185551285581214OO
58 65306071812806596550178516664O2
60 89104047134463252502559431096O2
62 10733951315294301174491841282048
64 11420796414343588424136158689350

wd.128.113.006
6  341376
8  87288624
10 13842455424
12 1448180487936
14 106141978256640
16 5697211389035256
18 231462916338818304
20 7303265469631124224
23 182676478544670888576
24 3683313800412335283600
26 60701011366229993420928
28 827171718544587565763072
30 941264369344488003313920O
32 90198176361260636112668700
34 7331616474282651537211008OO
36 50867453347742987955355059200
38 30288927413044862466125137280
40 15552199267849917590594150712O
42 69144969567169308745863446208O
44 26715101878223949636712940352OO
46 89979560528975061271236222656OO
48 26491322740844548554720461440200
50 68336799886586511592317937195776
52 15476275268432893523092165715174O
54 30822770450757203268200050751040O
56 54059937004867236324247368436404O
58 83591771920411452688890815493235O
60 11405318033208722013148761000990O2
62 13739457683579788462150741772974O8
64 14618619410356520132002732324864O0

wd.128.120.004
4  85344
6  42330624
8  11170182384
```

```
10 1772228014592
12 185359804775712
14 13586256544975872
16 729242357526446712
18 29627257927486958592
20 934817955092922629344
22 23382589365749366429184
24 471464166034059302122704
26 776972945617456205621606 4
28 10587797997047686927538550 4
30 12048183927667968257894707 20
32 11545366574237052418777217820
34 938446908707985400524343603 20
36 651103402851220082586931517920
38 387698270886939719010380968 1920
40 199068150628486994621400586 02480
42 885055610459752751522003146 06080
44 341953304041268345847846829 061280
46 11517383747708802174418396617 16480
48 339088931082809748767980761 3566280
50 87471103854830912553230500187 47392
52 19809632343594061105640384790579552
54 39453146176969302060067713110615552
56 69196719366229927819863672056678992
58 10699746805812685444195630142066 2272
60 14598807082507138963311965426639 7216
62 17586505834982158571541139235791 2576
64 18711832845256314920999104434444 9600

wd.128.127.002
2  8128
4  10668000
6  5423611200
8  1429702652400
10 226846154180800
12 23726045489546400
14 1739040916651367936
16 93343021201262198784
18 3792289018215984005120
20 119656698232656988471296
22 29929714389103547934310 40
24 60347413251942500075044864
26 99452537039201205626496 6144
28 13552381436214964486037045248
30 15421675427417015798741755 4944
32 14778069215022799216777342 48448
34 12012120431462387730048509542400
36 83341235564955678220181980577792
38 49625378673528400858712792629 2480
40 25480723280446307864089382470287 36
```

42 113287118138848348320467110173081 60
44 437700229172823588450176894553292 80
46 147422511970672750843485296833593344
48 434033831785996763487994252512722944
50 111963012934183589493510144979369984 0
52 253563293998003974172479085064539340 8
54 505000271065207171732982239898632192 0
56 885718007887743250057105214786450227 2
58 136956759114402370135324746965843968 00
60 186864730656091439657122556760408719 36
62 225107274687771671436711720814798438 40
64 239511460419281039376887104289825095 68

Bibliography

L. R. Bahl, J. Cocke, F. Jelinek and J. Raviv, "Optimal Decoding of Linear Codes for Minimizing Symbol Error Rate," *IEEE Trans. Inf. Theory*, vol. IT-20, pp. 284–287, Mar. 1974.

A. S. Barbulescu and S. S. Pietrobon, "Interleaver Design for Turbo Codes," *Electron. Lett.*, vol. 30, no. 25, pp. 2107–2108, Dec. 1994.

G. Battail, "A Conceptual Framework for Understanding Turbo Codes," *IEEE J. Sel. Areas Commun.*, vol. 16, no. 2, pp. 245–254, Feb. 1998.

S. Benedetto and E. Biglieri, *Principles of Digital Transmission*, Kluwer Academic/Plenum Publishers, 1999.

S. Benedetto, D. Divsalar, G. Montorsi and F. Pollara, "Parallel Concatenated Trellis Coded Modulation," *Proceedings of the 1996 IEEE International Conference on Communications (ICC'96)*, pp. 974–978, Dallas, Texas, 1996.

S. Benedetto, D. Divsalar, G. Montorsi and F. Pollara, "Serial Concatenation of Interleaved Codes: Performance Analysis, Design, and Iterative Decoding," *IEEE Trans. Inf. Theory*, vol. 44, no. 3, pp. 909–926, May 1998.

S. Benedetto and G. Montorsi, "Unveiling Turbo Codes: Some Results on Parallel Concatenated Coding Schemes," *IEEE Trans. Inf. Theory*, vol. 42, no. 2, pp. 409–428, Mar. 1996.

E. R. Berlekamp, *Algebraic Coding Theory*, rev. ed., Aegean Park Press, 1984.

E. R. Berlekamp, "Bounded Distance + 1 Soft-Decision Reed-Solomon Decoding," *IEEE Trans. Inf. Theory*, vol. 42, no. 3, pp. 704–720, May 1996.

C. Berrou and A. Glavieux, "Near Optimum Error Correcting Coding and Decoding," *IEEE Trans. Commun.*, vol. 44, no. 10, pp. 1261–1271, Oct. 1996.

C. Berrou and A. Glavieux, "Reflections on the Prize Paper: 'Near Optimum Error-Correcting Coding and Decoding: Turbo Codes'," *IEEE Inf. Theory Soc. News*, vol. 48, no. 2, pp. 24–31, June 1998.

C. Berrou, A. Glavieux and P. Thitimajshima, "Near Shannon Limit Error-Correcting Coding and Decoding: Turbo-Codes," *Proceedings of the 1993 IEEE International Conference on Communications (ICC'93)*, pp. 1064–1070, Geneve, Switzerland, May 1993.

E. Biglieri, D. Divsalar, P. J. McLane and M. K. Simon, *Introduction to Trellis-Coded Modulation with Applications*, Macmillan Publishing, 1991.

R. E. Blahut, *Theory and Practice of Error Control Codes*, Addison-Wesley, 1984.

M. Blaum, "A Family of Efficient Burst-Correcting Array Codes," *IEEE Trans. Inf. Theory*, vol. 36, no. 3, pp. 671–675, May 1990.

E. L. Blokh and V. V. Zyablov, "Coding of Generalized Concatenated Codes," *Probl. Pered. Inform.*, vol. 10, no. 1, pp. 45–50, 1974.

The Art of Error Correcting Coding, Second Edition Robert H. Morelos-Zaragoza

ISBN: 0-470-01558-6

M. Boo, F. Arguello, J. D. Bruguera, R. Doallo and E. L. Zapata, "High-Performance VLSI Architecture for the Viterbi Algorithm," *IEEE Trans. Commun.*, vol. 45, no. 2, pp. 168–176, Feb. 1997.

M. Bossert, *Channel Coding for Telecommunications*, John Wiley & Sons, 1999.

M. Breiling and J. B. Huber, "Combinatorial Analysis of the Minimum Distance of Turbo Codes," *IEEE Trans. Inf. Theory*, vol. 47, no. 7, pp. 2737–2750, Nov. 2001.

A. E. Brouwer and T. Verhoeff, "An Updated Table of Minimum-Distance Bounds for Binary Linear Codes," *IEEE Trans. Inf. Theory*, vol. 39, no. 2, pp. 662–677, Mar. 1993.

H. O. Burton and E. J. Weldon, Jr., "Cyclic Product Codes," *IEEE Trans. Inf. Theory*, vol. IT-11, no. 3, pp. 433–439, July 1965.

J. B. Cain, G. C Clark and J. M. Geist, "Punctured Convolutional Codes of Rate $(n-1)/n$ and Simplified Maximum Likelihood Decoding," *IEEE Trans. Inf. Theory*, vol. IT-25, pp. 97–100, Jan. 1979.

G. Caire, G. Taricco and E. Biglieri, "Capacity of Bit-Interleaved Channels," *Electron. Lett.*, vol. 32, pp. 1060–1061, June 1996.

G. Caire, G. Taricco and E. Biglieri, "Bit-Interleaver Coded Modulation," *IEEE Trans. Inf. Theory*, vol. 44, no. 3, pp. 927–946, May 1998.

F. Carden, M. D. Ross, B. T. Kopp and W. P. Osborn, "Fast TCM Decoding: Phase Quantization and Integer Weighting," *IEEE Trans. Commun.*, vol. 42, no. 4, pp. 808–812, Apr. 1994.

C.-C. Chao and Y.-L. Yao, "Hidden Markov Models for the Burst Error Statistics of Viterbi Decoding," *IEEE Trans. Commun.*, vol. 44, no. 12, pp. 1620–1622, Dec. 1996.

D. Chase, "A Class of Algorithms for Decoding Block Codes with Channel Measurement Information," *IEEE Trans. Inf. Theory*, vol. IT-18, pp. 170–182, 1972.

P. Chaudhari and A. K. Khandani, "Using the Fourier Transform to Compute the Weight Distribution of a Binary Linear Block Code," *IEEE Commun. Lett.*, vol. 5, no. 1, pp. 22–24, Jan. 2001.

S.-Y. Chung, G. D. Forney, Jr., T. J. Richardson and R. Urbanke, "On the Design of Low-Density Parity-Check Codes within 0.0045 dB of the Shannon Limit," *IEEE Commun. Lett.*, vol. 5, no. 2, pp. 58–60, Feb. 2001.

G. Clark and J. Cain, *Error-Correction Coding for Digital Communications*, Plenum Press, 1981.

O. M. Collins, "The Subtleties and Intricacies of Building a Constraint Length 15 Convolutional Decoder," *IEEE Trans. Commun.*, vol. 40, no. 12, pp. 1810–1819, Nov. 1992.

T. M. Cover and J. A. Thomas, *Elements of Information Theory*, John Wiley & Sons, 1991.

Y. Desaki, T. Fujiwara and T. Kasami, "A Method for Computing the Weight Distribution of a Block Code by Using Its Trellis Diagram," *IEICE Trans. Fundamentals*, vol. E77-A, pp. 1230–1237, Aug. 1994.

Y. Desaki, T. Fujiwara and T. Kasami, "The Weight Distribution of Extended Binary Primitive BCH Codes of Length 128," *IEEE Trans. Inf. Theory*, vol. 43, no. 4, pp. 1364–1371, July 1997.

U. Dettmar, G. Yan and U. K. Sorger, "Modified Generalized Concatenated Codes and Their Application to the Construction and Decoding of LUEP Codes," *IEEE Trans. Inf. Theory*, vol. 41, no. 5, pp. 1499–1503, Sept. 1995.

A. Dholakia, *Introduction to Convolutional Codes With Applications*, Kluwer, 1994.

D. Divsalar, S. Dolinar and F. Pollara, "Iterative Turbo Decoder Analysis Based on Density Evolution," *IEEE J. Sel. Areas Commun.*, vol. 19, no. 5, pp. 891–907, May 2001.

D. Divsalar, H. Jin and R. J. McEliece, "Coding Theorems for Turbo-Like Codes," *Proceedings of the 1998 Allerton Conference on Communications, Control, and Computing*, pp. 210–219, University of Illinois, Urbana-Champaign, IL, 1998.

D. Divsalar and F. Pollara, "Turbo Codes for PCS Applications," *Proceedings of the 1993 IEEE International Conference on Communications (ICC'93)*, pp. 1064–1070, Geneva, Switzerland, May 1993.

D. Divsalar and F. Pollara, "Turbo Codes for Deep-Space Communications," *JPL TDA Progress Report 42-120*, NASA Jet Propulsion Laboratory, pp. 29–39, Feb. 1995.

H. El Gamal and A. R. Hammons, Jr., "Analyzing the Turbo Decoder Using the Gaussian Approximation," *IEEE Trans. Inf. Theory*, vol. 47, no. 2, pp. 671–686, Feb. 2001.

P. Elias, "Error-Free Coding," *IRE Trans.*, vol. PGIT-4, pp. 29–37, 1954.

P. Elias, "Coding for Noisy Channels," *IRE Conv. Rec.*, vol. 3, pt. 4, pp. 37–46, 1955; Also in E. R. Berlekamp, ed., *Key Papers in the Development of Coding Theory*, pp. 48–55, IEEE Press, 1974.

P. Elias, "Error-Correcting Codes for List Decoding," *IEEE Trans. Inf. Theory*, vol. 37, no. 1, pp. 5–12, Jan. 1991.

EN 310 210, ETSI, "Digital Video Broadcasting (DVB): Framing Structure, Channel Coding and Modulation for Digital Satellite News Gathering (DSNG) and Other Contribution Applications by Satellite," Mar. 1997.

W. Feng and B. Vucetic, "A List Bidirectional Soft Output Decoder of Turbo Codes," *Proceedings of the International Symposium on Turbo Codes and Related Topics*, pp. 288–292, Brest, France, Sept. 3-5, 1997.

G. D. Forney, Jr., *Concatenated Codes*, Research Monograph 37, MIT Press, 1966a.

G. D. Forney, "Generalized Minimum Distance Decoding," *IEEE Trans. Inf. Theory*, vol. IT-12, pp. 125–131, April 1966b.

G. D. Forney, "Convolutional Codes I: Algebraic Structure," *IEEE Trans. Inf. Theory*, vol. IT-16, no. 6, pp. 720–738, Nov. 1970.

G. D. Forney, Jr., "Burst Correcting Codes for the Classic Bursty Channel," *IEEE Trans. Commun. Technol.*, vol. COM-19, pp. 772–281, 1971.

G. D. Forney, Jr., "On Decoding BCH Codes," *IEEE Trans. Inf. Theory*, vol. IT-11, pp. 393–403, Oct. 1965; Also in E. R. Berlekamp, ed., *Key Papers in the Development of Coding Theory*, pp. 149–155, IEEE Press, 1974.

G. D. Forney, Jr., "Coset Codes II: Binary Lattices and Related Codes," *IEEE Trans. Inf. Theory*, vol. 24, no. 5, pp. 1152–1187, Sept. 1988.

G. D. Forney and G. Ungerboeck, "Modulation and Coding for Linear Gaussian Channels," *IEEE Trans. Inf. Theory*, *Special Commemorative Issue*, vol. 44, no. 6, pp. 2384–2415, Oct. 1998.

G. D. Forney, "Codes on Graphs: Normal Realizations," *IEEE Trans. Inf. Theory*, vol. 47, no. 2, pp. 520–548, Feb. 2001.

M. P. C. Fossorier, F. Burkert, S. Lin and J. Hagenauer, "On the Equivalence Between SOVA and Max-Log-MAP Decodings," *IEEE Commun. Lett.*, vol. 2, no. 5, pp. 137–139, May 1998.

M. P. C. Fossorier and S. Lin, "Soft-Decision Decoding on Linear Block Codes Based on Ordered Statistics," *IEEE Trans. Inf. Theory*, vol. 41, no. 5, pp. 1379–1396, Sept. 1995.

M. P. C. Fossorier and S. Lin, "Some Decomposable Codes: The $|a + x|b + x|a + b + x|$ Construction," *IEEE Trans. Inf. Theory*, vol. 43, no. 5, pp. 1663–1667, Sept. 1997a.

M. P. C. Fossorier and S. Lin, "Complementary Reliability-Based Soft-Decision Decoding of Linear Block Codes Based on Ordered Statistics," *IEEE Trans. Inf. Theory*, vol. 42, no. 5, pp. 1667–1672, Sep. 1997b.

M. P. C. Fossorier and S. Lin, "Soft-Input Soft-Output Decoding of Linear Block Codes Based on Ordered Statistics," *Proceedings of the 1998 IEEE Global Telecommunications Conference (GLOBECOM'98)*, pp. 2828–2833, Sydney, Australia, Nov. 1998.

M. P. C. Fossorier and S. Lin, "Differential Trellis Decoding of Convolutional Codes," *IEEE Trans. Inf. Theory*, vol. 46, no. 3, pp. 1046–1053, May 2000.

M. P. C. Fossorier, S. Lin and D. J. Costello, Jr., "On the Weight Distribution of Terminated Convolutional Codes," *IEEE Trans. Inf. Theory*, vol. 45, no. 5, pp. 1646–1648, July 1999.

M. P. C. Fossorier, S. Lin and D. Rhee, "Bit-Error Probability for Maximum-Likelihood Decoding of Linear Block Codes and Related Soft-Decision Decoding Methods," *IEEE Trans. Inf. Theory*, vol. 44, no. 7, pp. 3083–3090, Nov. 1998.

M. P. C. Fossorier, M. Mihaljević and H. Imai, "Reduced Complexity Iterative Decoding of Low-Density Parity Check Codes Based on Belief Propagation," *IEEE Trans. Commun.*, vol. 47, no. 5, pp. 673–680, May 1999.

B. J. Frey and F. R. Kschischang, "Early Detection and Trellis Splicing: Reduced-Complexity Iterative Decoding," *IEEE J. Sel. Areas Commun.*, vol. 16, no. 2, pp. 153–159, Feb. 1998.

T. Fujiwara, T. Kasami, A. Kitai and S. Lin, "On the Undetected Error Probability of Shortened Hamming Codes," *IEEE Trans. Commun.*, vol. COM-33, pp. 570–574, June 1985.

R. G. Gallager, "Low-Density Parity-Check Codes," *IRE Trans. Inf. Theory*, vol. 8, no. 1, pp. 21–28, Jan. 1962.

R. Garello, P. Perleoni and S. Benedetto, "Computing the Free Distance of Turbo Codes and Serially Concatenated Codes with Interleavers: Algorithms and Applications," *IEEE J. Sel. Areas Commun.*, vol. 19, no. 5, pp. 800–812, May 2001.

M. J. E. Golay, "Notes on Digital Coding" *Proc. IRE*, vol. 37, p. 657, June 1949; Also in E. R. Berlekamp, ed., *Key Papers in the Development of Coding Theory*, p. 13, IEEE Press, 1974.

J. D. Golić, "Iterative Optimum Symbol-by-Symbol Decoding and Fast Correlation Attacks," *IEEE Trans. Inf. Theory*, vol. 47, no. 7, pp. 3040–3049, Nov. 2001.

D. Gorenstein and N. Zierler, "A Class of Error Correcting Codes in p^m Symbols," *J. SIAM*, vol. 9, pp. 207–214, June 1961.

V. Guruswami and M. Sudan, "Improved Decoding of Reed-Solomon and Algebraic-Geometry Codes," *IEEE Trans. Inf. Theory*, vol. 45, no. 6, pp. 1757–1767, Sep. 1999.

J. Hagenauer, "Rate-Compatible Punctured Convolutional Codes (RCPC Codes) and their Applications," *IEEE Trans. Commun.*, vol. 36, no. 4, pp. 389–400, April 1988.

J. Hagenauer and P. Hoeher, "A Viterbi Algorithm with Soft-Decision Outputs and Its Applications," *Proceedings of the 1989 IEEE Global Telecommunications Conference (GLOBECOM'89)*, pp. 47.1.1–47.1.7, Dallas, Texas, 1989.

J. Hagenauer, E. Offer and L. Papke, "Iterative Decoding of Binary Block and Convolutional Codes," *IEEE Trans. Inf. Theory*, vol. 42, no. 2, pp. 429–445, Mar. 1996.

E. K. Hall and S. G. Wilson, "Stream-Oriented Turbo Codes," *IEEE Trans. Inf. Theory*, vol. 47, no. 5, pp. 1813–1831, July 2001.

R. W. Hamming, "Error Detecting and Error Correcting Codes," *Bell Syst. Tech. J.*, vol. 29, pp. 147–160, 1950; Also in E. R. Berlekamp, ed., *Key Papers in the Development of Coding Theory*, pp. 9–12, IEEE Press, 1974.

S. Haykin, *Digital Communications*, 4th ed., John Wiley and Sons, 2001.

C. Heegard and S. B. Wicker, *Turbo Coding*, Kluwer Academic Press, 1999.

A. P. Hekstra, "An Alternative to Metric Rescaling in Viterbi Decoder," *IEEE Trans. Commun.*, vol. 37, no. 11, pp. 1220–1222, Nov. 1989.

I. N. Herstein, *Topics in Algebra*, 2nd ed., John Wiley and Sons, 1975.

J. Hokfelt, O. Edfors and T. Maseng, "A Turbo Code Interleaver Design Criterion Based on the Performance of Iterative Decoding," *IEEE Commun. Lett.*, vol. 5, no. 2, pp. 52–54, Feb. 2001.

B. Honay and G. S. Markarian, *Trellis Decoding of Block Codes: A Practical Approach*, Kluwer, 1996.

B. Honary, G. S. Markarian and P. G. Farell, "Generalised Array Codes and Their Trellis Structure," *Electron. Lett.*, vol. 28, no. 6, pp. 541–542, Mar. 1993.

M. Y. Hsiao, "A Class of Optimal Minimum Odd-Weight-Column SEC-DED Codes," *IBM J. Res. Dev.*, vol. 14, July 1970.

H. Imai and S. Hirakawa, "A New Multilevel Coding Method Using Error-Correcting Codes," *IEEE Trans. Inf. Theory*, vol. IT-23, no. 3, pp. 371–377, May 1977.

M. Isaka, M. P. C. Fossorier, R. H. Morelos-Zaragoza, S. Lin and H. Imai, "Multilevel Coded Modulation for Unequal Error Protection and Multistage Decoding. Part II: Asymmetric Constellations," *IEEE Trans. Commun.*, vol. 48, no. 5, pp. 774–784, May 2000.

Ecole Nationale Superieure des Telecommunications de Bretagne, *Proceedings of the International Symposium on Turbo Codes and Related Topics*, Brest, France, September 3–5, 1997.

Ecole Nationale Superieure des Telecommunications de Bretagne, *Proceedings of the Second International Symposium on Turbo Codes and Related Topics*, Brest, France, September 4–7, 2000.

Institute of Electrical and Electronics Engineers, Inc., "Special Issue on Codes and Graphs and Iterative Decoding Algorithms," *IEEE Trans. Inf. Theory*, vol. 47, no. 2, Feb. 2001.

M. C. Jeruchim, P. Balaban and K. S. Shanmugan, *Simulation of Communication Systems*, Plenum Press, 1992.

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

Institute of Electrical and Electronics Engineers, Inc., "Special Issue on the Turbo Principle: from Theory to Practice," Part I, *IEEE J. Sel. Areas Commun.*, vol. 19, no. 5, May 2001.

Institute of Electrical and Electronics Engineers, Inc., "Special Issue on the Turbo Principle: from Theory to Practice," Part II, *IEEE J. Sel. Areas Commun.*, vol. 19, no. 9, Sept. 2001.

N. Kamiya, "On Algebraic Soft-Decision Decoding Algorithms for BCH Codes," *IEEE Trans. Inf. Theory*, vol. 47, no. 1, pp. 45–58, Jan. 2001.

T. Kaneko, T. Nishijima, H. Inazumi and S. Hirasawa, "An Efficient Maximum-Likelihood Decoding Algorithm for Linear Block Codes with Algebraic Decoder," *IEEE Trans. Inf. Theory*, vol. 40, no. 2, pp. 320–327, Mar. 1994.

M. Kasahara, Y. Sugiyama, S. Hirasawa and T. Namekawa, "A New Class of Binary Codes Constructed on the Basis of BCH Codes," *IEEE Trans. Inf. Theory*, vol. IT-21, pp. 582–585, 1975.

M. Kasahara, Y. Sugiyama, S. Hirasawa and T. Namekawa, "New Classes of Binary Codes Constructed on the Basis of Concatenated and Product Codes," *IEEE Trans. Inf. Theory*, vol. IT-22, no. 4, pp. 462–468, July 1976.

T. Kasami, "A Decoding Procedure for Multiple-Error-Correcting Cyclic Codes," *IEEE Trans. Inf. Theory*, vol. IT-10, pp. 134–138, 1964.

T. Kasami, T. Koumoto, T. Takata, T. Fujiwara and S. Lin, "The Least Stringent Sufficient Condition on the Optimality of Suboptimally Decoded Codewords," *Proceedings of the 1995 IEEE International Symposium on Information Theory (ISIT'95)*, p. 470, Whistler, Canada, 1995.

T. Kasami, S. Lin and W. W. Peterson, "Polynomial Codes," *IEEE Trans. Inf. Theory*, vol. IT-14, no. 6, pp. 807–814, Nov. 1968.

P. Kazakov, "Fast Calculation of the Number of Minimum-Weight Words of CRC Codes," *IEEE Trans. Inf. Theory*, vol. 47, no. 3, pp. 1190–1195, March 2001.

R. Koetter and A. Vardy, "Algebraic Soft-Decision Decoding of Reed-Somon Codes," *Proceedings of the 2000 IEEE International Symposium on Information Theory (ISIT'00)*, p. 61, Sorrento, Italy, June 25–30, 2000.

R. R. Kschischang, B. J. Frey and H.-A. Loeliger, "Factor Graphs and the Sum-Product Algorithm," *IEEE Trans. Inf. Theory*, vol. 47, no. 2, pp. 498–519, Feb. 2001.

Y. Kuo, S. Lin and M. P. C. Fossorier, "Low-Density Parity-Check Codes Based on Finite Geometries: A Rediscovery and New Results," *IEEE Trans. Inf. Theory*, vol. 47, no. 7, pp. 2711–2736, Nov. 2001.

L. H. C. Lee, *Convolutional Coding: Fundamentals and Applications*, Artech House, 1997.

S. Le Goff, A. Glavieux and C. Berrou, "Turbo-Codes and High-Spectral Efficiency Modulation," *Proceedings of the 1994 IEEE International Conference on Communications (ICC'94)*, pp. 645–649, New Orleans, Louisiana, 1994.

S. Lin and D. J. Costello, Jr., *Error Control Coding: Fundamentals and Applications*, 2nd ed., Prentice-Hall, 2005.

S. Lin, T. Kasami, T. Fujiwara and M. Fossorier, *Trellises and Trellis-Based Decoding Algorithms for Linear Block Codes*, Kluwer Academic Press, 1998.

J. Lodge, R. Young, P. Hoeher and J. Hagenauer, "Separable MAP 'Filters' for the Decoding of Product and Concatenated Codes," *Proceedings of the 1993 IEEE International Conference on Communications (ICC'93)*, pp. 1740–1745, Geneva, Switzerland, May 1993.

D. J. C. MacKay, "Good Error-Correcting Codes Based on Very Sparse Matrices," *IEEE Trans. Inf. Theory*, vol. 45, no. 2, pp. 399–432, Mar. 1999.

D. J. C. MacKay and R. M. Neal, "Near Shannon Limit Performance of Low Density Parity Check Codes," *Electron. Lett.*, vol. 32, pp. 1645–1646, 1996.

F. J. MacWilliams and N. J. A. Sloane, *The Theory of Error Correcting Codes*, North-Holland, 1977.

D. Mandelbaum, "Decoding Beyond the Designed Distance of Certain Algebraic Codes," *Inf. Control*, vol. 35, pp. 209–228, 1977.

P. A. Martin and D. P. Taylor, "On Adaptive Reduced-Complexity Iterative Decoding," *Proceedings of the 2000 IEEE Global Telecommunications Conference (GLOBECOM'00)*, pp. 772–776, San Francisco, California, 2000.

B. Masnick and J. Wolf, "On Linear Unequal Error Protection Codes," *IEEE Trans. Inf. Theory*, vol. IT-13, no. 4, pp. 600–607, July 1967.

J. L. Massey, *Threshold Decoding*, MIT Press, 1963.

J. L. Massey, "Shift Register Synthesis and BCH Decoding," *IEEE Trans. Inf. Theory*, vol. IT-15, no. 1, pp. 122–127, Jan. 1969.

J. L. Massey, "Coding and Modulation in Digital Communications," *Proceedings of the International Zurich Seminar on Digital Communications*, pp. E2(1)-E2(4), Zurich, Switzerland, 1974.

J. L. Massey, "The How and Why of Channel Coding," *Proceedings of the International Zurich Seminar on Digital Communications*, pp. 67–73, Zurich, Switzerland, 1984.

J. L. Massey and M. K. Sain, "Codes, Automata and Continuous Systems: Explicit Interconnections," *IEEE Trans. Aut. Cont.*, vol. AC-12, pp. 644–650, 1967

R. J. McEliece, *The Theory of Information and Coding*, Addison-Wesley, 1977.

R. J. McEliece, D. J. C. MacKay and J.-F. Cheng, "Turbo Decoding as an Instance of Pearls' 'Belief Propagation' Algorithm," *IEEE J. Sel. Areas Commun.*, vol. 16, no. 2, pp. 140–152, Feb. 1998.

J. E. Meggit, "Error Correcting Codes for Correcting Bursts of Errors," *IBM J. Res. Develop.*, vol. 4, no. 3, pp. 329–334, 1960.

A. M. Michelson and A. H. Levesque, *Error-Control Techniques for Digital Communications*, John Wiley and Sons, 1985.

M. J. Mihaljević and J. D. Golić, "A Comparison of Cryptoanalitic Principles Based on Iterative Error Correction," *Advances in Cryptology–EUROCRYPT'91 (Lecture Note in Computer Science)*, vol. 547, pp. 527–531, Springer-Verlag, 1991.

T. K. Moon, *Error Correction Coding*, John Wiley and Sons, 2005.

R. H. Morelos-Zaragoza, M. P. C. Fossorier, S. Lin and H. Imai, "Multilevel Coded Modulation for Unequal Error Protection and Multistage Decoding. Part I: Symmetric Constellations," *IEEE Trans. Commun.*, vol. 48, no. 2, pp. 204–213, Feb. 2000.

R. H. Morelos-Zaragoza, T. Fujiwara, T. Kasami and S. Lin, "Constructions of Generalized Concatenated Codes and Their Trellis-Based Decoding Complexity," *IEEE Trans. Inf. Theory*, vol. 45, no. 2, pp. 725–731, Mar. 1999.

R. H. Morelos-Zaragoza and H. Imai, "Binary Multilevel Convolutional Codes with Unequal Error Protection Capabilities," *IEEE Trans. Commun.*, vol. 46, no. 7, pp. 850–853, July 1998.

R. H. Morelos-Zaragoza and A. Mogre, "A Two-Stage Decoder for Pragmatic Trellis-Coded M-PSK Modulation Using a Symbol Transformation," *IEEE Trans. Commun.*, vol. 49, no. 9, pp. 1501–1505, Sept. 2001.

J. M. Morris, "Burst Error Statistics of Simulated Viterbi Decoded BPSK on Fading and Scintillating Channels," *IEEE Trans. Commun.*, vol. 40, no. 1, pp. 34–41, Jan. 1992.

I. M. Onyszchuk, K.-M. Cheung and O. Collins, "Quantization Loss in Convolutional Decoding," *IEEE Trans. Commun.*, vol. 41, no. 2, pp. 261–265, Feb. 1993.

J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*, Morgan Kaufmann, 1988.

L. C. Perez, J. Seghers and D. J. Costello, Jr., "A Distance Spectrum Interpretation of Turbo Codes," *IEEE Trans. Inf. Theory*, vol. 42, no. 6, pp. 1698–1709, Nov. 1996.

W. W. Peterson, "Encoding and Error-Correction Procedures for the Bode-Chaudhuri Codes," *IRE Trans. Inf. Theory*, vol. IT-6, pp. 459–470, Sept. 1960; Also in E.R. Berlekamp, ed., *Key Papers in the Development of Coding Theory*, pp. 109–120, IEEE Press, 1974.

W. W. Peterson and E. J. Weldon, Jr., *Error Correcting Codes*, 2nd ed., MIT Press, 1972.

A. Picart and R. M. Pyndiah, "Adapted Iterative Decoding of Product Codes," *Proceedings of the 1995 IEEE Global Telecommunications Conference (GLOBECOM'95)*, pp. 2357–2362, Singapore, 1995.

M. Plotkin, "Binary Codes with Specified Minimum Distances," *IEEE Trans. Inf. Theory*, vol. IT-6, pp. 445–450, 1960.

O. Pretzel, *Codes and Algebraic Curves*, Oxford Lecture Series in Mathematics and Its Applications, vol. 8, 1998.

J. G. Proakis, *Digital Communications*, 4th ed., McGraw-Hill, 2001.

M. B. Pursley and J. M. Shea, "Bit-by-Bit Soft-Decision Decoding of Trellis-Coded M-DPSK Modulation," *IEEE Commun. Lett.*, vol. 1, no. 5, pp. 133–135, Sept. 1997.

R. M. Pyndiah, "Near-Optimum Decoding of Product Codes: Block Turbo Codes," *IEEE Trans. Commun.*, vol. 46, no. 8, pp. 1003–1010, Aug. 1998.

R. M. Pyndiah, A. Glavieux, A. Picart and S. Jacq, "Near Optimum Decoding of Product Codes," *Proceedings of the 1994 IEEE Global Telecommunications Conference (GLOBECOM'94)*, vol. 1, pp. 339–343, San Francisco, CA, Dec. 1994.

J. L. Ramsey, "Realization of Optimum Interleavers," *IEEE Trans. Inf. Theory*, vol. IT-16, no. 3, pp. 338–345, May 1970.

S. M. Reddy and J. P. Robinson, "Random Error and Burst Correction by Iterated Codes," *IEEE Trans. Inf. Theory*, vol. IT-18, no. 1, pp. 182–185, Jan. 1972.

I. S. Reed and X. Chen, *Error-Control Coding for Data Networks*, Kluwer Academic Press, 1999.

I. Reed and G. Solomon, "Polynomial Codes over Certain Finite Fields," *SIAM J. Appl. Math.*, Vol. 8, pp. 300–304, 1960; Also in E.R. Berlekamp, ed., *Key Papers in the Development of Coding Theory*, pp. 70–71, IEEE Press, 1974.

T. J. Richardson, M. A. Shokrollahi and R. L. Urbanke, "Design of Capacity-Approaching Irregular Low-Density Parity-Check Codes," *IEEE Trans. Inf. Theory*, vol. 47, no. 2, pp. 619–637, Feb. 2001.

T. J. Richardson and R. L. Urbanke, "The Capacity of Low-Density Parity-Check Codes Under Message-Passing Decoding," *IEEE Trans. Inf. Theory*, vol. 47, no. 2, pp. 599–618, Feb. 2001.

P. Robertson, "Improving Decoder and Code Structure of Parallel Concatenated Recursive Systematic (Turbo) Codes," *Proceedings of the IEEE International Conference on Universal Personal Communications (ICUPC'94)*, pp. 183–187, San Diego, California, 1994.

P. Robertson, E. Villebrun and P. Hoeher, "A Comparison of Optimal and Sub-Optimal MAP Decoding Algorithms Operating in the Log Domain," *Proceedings of the 1995 IEEE International Conference on Communication (ICC'95)*, pp. 1009–1013, Seattle, Washington, 1995.

P. Robertson and T. Wörz, "Coded Modulation Scheme Employing Turbo Codes," *Electron. Lett.*, vol. 31, pp. 1546–1547, Aug. 1995.

P. Robertson and T. Wörz, "Bandwidth-Efficient Turbo Trellis-Coded Modulation Using Punctured Component Codes," *IEEE J. Sel. Areas Commun.*, vol. 16, no. 2, pp. 206–218, Feb. 1998.

H. R. Sadjadpour, N. J. A. Sloane, M. Salehi and G. Nebe, "Interleaver Design for Turbo Codes," *IEEE J. Sel. Areas Commun.*, vol. 19, no. 5, pp. 831–837, May 2001.

G. Schnabl and M. Bossert, "Soft-Decision Decoding of Reed-Muller Codes as Generalized Multiple Concatenated Codes," *IEEE Trans. Inf. Theory*, vol. 41, no. 1, pp. 304–308, Jan. 1995.

P. Schramm, "Multilevel Coding with Independent Decoding on Levels for Efficient Communication on Static and Interleaved Fading Channels," *Proceedings of the IEEE 1997 International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC'97)*, pp. 1196–1200, 1997.

C. Shannon, "A Mathematical Theory of Communication," *Bell Syst. Tech. J.*, vol. 27, pp. 379–423 and 623–656, 1948; Also in D. Slepian, ed., *Key Papers in the Development of Information Theory*, pp. 5–29, IEEE Press, 1974.

R. Shao, S. Lin and M. P. C. Fossorier, "Two Simple Stopping Criteria for Turbo Decoding," *IEEE Trans. Commun.*, vol. 47, no. 8, pp. 1117–1120, Aug. 1999.

R. C. Singleton, "Maximum Distance q-nary Codes," *IEEE Trans. Inf. Theory*, vol. IT-10, pp. 116–118, 1964.

M. Sipser and D. A. Spielman, "Expander Codes," *IEEE Trans. Inf. Theory*, vol. 42, no. 6, pp. 1710–1722, Nov. 1996.

B. Sklar, "A Primer on Turbo Code Concepts," *IEEE Commun. Mag.*, vol. 35, no. 12, pp. 94–102, Dec. 1997.

D. Slepian, "A Class of Binary Signaling Alphabets," *Bell. Sys. Tech. J.*, vol. 35, pp. 203–234, Jan. 1956.

N. J. A. Sloane, S. M. Reddy and C.-L. Chen, "New Binary Codes," *IEEE Trans. Inf. Theory*, vol. IT-18, no. 4, pp. 503–510, July 1972.

M. Sudan, "Decoding of Reed-Solomon Codes Beyond the Error-Correction Bound," *J. Complexity*, vol. 12, pp. 180–193, Dec. 1997.

Y. Sugiyama, Y. Kasahara, S. Hirasawa and T. Namekawa, "A Method for Solving Key Equation for Goppa Codes," *Inf. Control*, vol. 27, pp. 87–99, 1975.

Y. Sugiyama, M. Kasahara, S. Hirasawa and T. Namekawa, "Further Results on Goppa Codes and Their Applications to Constructing Efficient Binary Codes," *IEEE Trans. Inf. Theory*, vol. IT-22, pp. 518–526, 1978.

D. J. Taipale and M. B. Pursley, "An Improvement to Generalized Minimum Distance Decoding," *IEEE Trans. Inf. Theory*, vol. 37, no. 1, pp. 167–172, Jan. 1991.

T. Takata, Y. Yamashita, T. Fujiwara, T. Kasami and S. Lin, "On a Suboptimum Decoding of Decomposable Block Codes," *IEEE Trans. Inf. Theory*, vol. 40, no. 5, pp. 1392–1406, Sept. 1994.

O. Y. Takeshita and D. J. Costello, Jr., "New Deterministic Interleaver Designs for Turbo Codes," *IEEE Trans. Inf. Theory*, vol. 46, no. 6, pp. 1988–2006, Sept. 2000.

H. Tang, Y. Liu, M. P. C. Fossorier and S. Lin, "On Combining Chase-2 and GMD Decoding Algorithms for Nonbinary Block Codes," *IEEE Commun. Lett.*, vol. 5, no. 5, pp. 209–211, May 2001.

R. M. Tanner, "A Recursive Approach to Low Complexity Codes," *IEEE Trans. Inf. Theory*, vol. IT-27, no. 5, pp. 533–547, Sept. 1981.

S. ten Brink, "Convergence of Iterative Decoding," *Electron. Lett.*, vol. 35, pp. 806–808, May 1999.

S. ten Brink, "Convergence Behaviour of Iteratively Decoded Parallel Concatenated Codes," *IEEE Trans. Inf. Theory*, vol. 49, no. 10, pp. 1727–1737, Oct. 2001.

H. Tokushige, T. Koumoto and T. Kasami, "An Improvement to GMD-like Decoding Algorithms," *Proceedings of the 2000 IEEE International Symposium on Information Theory (ISIT'00)*, p. 396, Sorrento, Italy, 2000.

G. Ungerboeck, "Channel Coding with Multilevel/Phase Signals," *IEEE Trans. Inf. Theory*, vol. IT-28, no. 1, pp. 55–67, Jan. 1982.

G. Ungerboeck, "Trellis-Coded Modulation with Redundant Signal Sets I: Introduction," *IEEE Commun. Mag.*, vol. 25, no. 2, pp. 5–11, Feb. 1987a.

G. Ungerboeck, "Trellis-Coded Modulation with Redundant Signal Sets II: State of the Art," *IEEE Commun. Mag.*, vol. 25, no. 2, pp. 12–21, Feb. 1987b.

W. J. van Gils, "Two Topics on Linear Unequal Error Protection Codes: Bounds on Their Length and Cyclic Code Classes," *IEEE Trans. Inf. Theory*, vol. IT-29, no. 6, pp. 866–876, Nov. 1983.

S. A. Vanstone and P. C. van Oorschot, *An Introduction to Error Correcting Codes with Applications*, Kluwer Academic Publishers, 1989.

A. J. Viterbi, "Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm," *IEEE Trans. Inf. Theory*, vol. IT-13, pp. 260–269, April 1967.

A. J. Viterbi, "Convolutional Codes and Their Performance in Communication Systems," *IEEE Trans. Commun.*, vol. COM-19, no. 4, pp. 751–772, 1971.

A. J. Viterbi, "An Intuitive Justification and a Simplified Implementation of the MAP Decoder for Convolutional Codes," *IEEE J. Sel. Areas Commun.*, vol. 16, no. 2, pp. 260–264, Feb. 1998.

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

A. J. Viterbi, J. K. Wolf, E. Zehavi and R. Padovani, "A Pragmatic Approach to Trellis-Coded Modulation," *IEEE Commun. Mag.*, vol. 27, no. 7, pp. 11–19, July 1989.

J. von zur Gathen and J. Gerhard, *Modern Computer Algebra*, Cambridge University Press, 1999.

B. Vucetic and J. Yuan, *Turbo Codes: Principles and Applications*, Kluwer Academic, 2000.

U. Wachsmann, R. F. H. Fischer and J. B. Huber, "Multilevel Codes: Theoretical Concepts and Practical Design Rules," *IEEE Trans. Inf. Theory*, vol. 45, no. 5, pp. 1361–1391, July 1999.

B. E. Wahlen and C. Y. Mai, "Turbo Coding Applied to Pragamtic Trellis-Coded Modulation," *IEEE Commun. Lett.*, vol. 4, no. 2, pp. 65–67, Feb. 2000.

E. J. Weldon, Jr., "Decoding Binary Block Codes on Q-ary Output Channels," *IEEE Trans. Inf. Theory*, vol. IT-17, no. 6, pp. 713–718, Nov. 1971.

N. Wiberg, "Codes and Decoding on General Graphs," Ph.D. dissertation, Department of Electrical Engineering, Linköping University, 1996.

S. B. Wicker, *Error Control Systems for Digital Communication and Storage*, Prentice-Hall, 1995.

S. B. Wicker and V. K. Bhargava, ed., *Reed-Solomon Codes and Their Applications*, IEEE Press, 1994.

S. G. Wilson, *Digital Modulation and Coding*, Prentice-Hall, 1996.

J. K. Wolf, "Efficient Maximum-Likelihood Decoding of Linear Block Codes Using a Trellis," *IEEE Trans. Inf. Theory*, vol. IT-24, no. 1, pp. 76–80, Jan. 1978.

J. K. Wolf and A. J. Viterbi, "On the Weight Distribution of Linear Block Codes formed From Convolutional Codes," *IEEE Trans. Commun.*, vol. 44, no. 9, pp. 1049–1051, Sep. 1996.

J. M. Wozencraft and I. M. Jacobs, *Principles of Communication Engineering*, John Wiley and Sons, 1965.

K. Yamaguchi and H. Imai, "A New Block Coded Modulation Scheme and Its Soft Decision Decoding," *Proceedings of the 1993 IEEE International Symposium on Information Theory (ISIT'93)*, p. 64, San Antonio, Texas, 1993.

Y. Yasuda, K. Kashiki and Y. Hirata, "High-Rate Punctured Convolutional Codes for Soft Decision Viterbi Decoding," *IEEE Trans. Commun.*, vol. COM-32, no. 3, pp. 325–328, March 1984.

E. Zehavi and J. K. Wolf, "P^2 Codes: Pragmatic Trellis Codes Utilizing Punctured Convolutional Codes," *IEEE Commun. Mag.*, vol. 33, no. 2, pp. 94–99, Feb. 1995.

V. A. Zinov'ev, "Generalized Cascade Codes," *Prob. Pered. Inform.*, vol. 12, no. 1, pp. 5–15, 1976.

V. V. Zyablov and M. S. Pinsker, "Estimation of the Error-Correction Complexity for Gallager Low-Density Codes," *Prob. Pered. Inform.*, vol. 11, no. 1, pp. 26–36, 1975.

Index

The Art of Error Correcting Coding, Second Edition Robert H. Morelos-Zaragoza

ISBN: 0-470-01558-6