Channel Codes

Classical and Modern

William E. Ryan
and **Shu Lin**

CAMBRIDGE

CAMBRIDGE www.cambridge.org/9780521848688

This page intentionally left blank

Channel Codes

Channel coding lies at the heart of digital communication and data storage, and this detailed introduction describes the core theory as well as decoding algorithms, implementation details, and performance analyses.

Professors Ryan and Lin, known for the clarity of their writing, provide the latest information on modern channel codes, including turbo and low-density parity-check (LDPC) codes. They also present detailed coverage of BCH codes, Reed–Solomon codes, convolutional codes, finite-geometry codes, and product codes, providing a one-stop resource for both classical and modern coding techniques.

The opening chapters begin with basic theory to introduce newcomers to the subject, assuming no prior knowledge in the field of channel coding. Subsequent chapters cover the encoding and decoding of the most widely used codes and extend to advanced topics such as code ensemble performance analyses and algebraic code design. Numerous varied and stimulating end-of-chapter problems, 250 in total, are also included to test and enhance learning, making this an essential resource for students and practitioners alike.

William E. Ryan is a Professor in the Department of Electrical and Computer Engineering at the University of Arizona, where he has been a faculty member since 1998. Before moving to academia, he held positions in industry for five years. He has published over 100 technical papers and his research interests include coding and signal processing with applications to data storage and data communications.

Shu Lin is an Adjunct Professor in the Department of Electrical and Computer Engineering, University of California, Davis. He has authored and co-authored numerous technical papers and several books, including the successful *Error Control Coding* (with Daniel J. Costello). He is an IEEE Life Fellow and has received several awards, including the Alexander von Humboldt Research Prize for US Senior Scientists (1996) and the IEEE Third-Millenium Medal (2000).

Channel Codes

Classical and Modern

WILLIAM E. RYAN
University of Arizona

SHU LIN
University of California, Davis

CAMBRIDGE
UNIVERSITY PRESS

CAMBRIDGE UNIVERSITY PRESS
Cambridge, New York, Melbourne, Madrid, Cape Town, Singapore,
São Paulo, Delhi, Dubai, Tokyo

Cambridge University Press
The Edinburgh Building, Cambridge CB2 8RU, UK

Published in the United States of America by Cambridge University Press, New York

www.cambridge.org
Information on this title: www.cambridge.org/9780521848688

First published in print format 2009

ISBN-13 978-0-511-64182-4 eBook (NetLibrary)

ISBN-13 978-0-521-84868-8 Hardback

Contents

Preface

The title of this book, *Channel Codes: Classical and Modern*, was selected to reflect the fact that this book does indeed cover both classical and modern channel codes. It includes BCH codes, Reed–Solomon codes, convolutional codes, finite-geometry codes, turbo codes, low-density parity-check (LDPC) codes, and product codes. However, the title has a second interpretation. While the majority of this book is on LDPC codes, these can rightly be considered to be both classical (having been first discovered in 1961) and modern (having been rediscovered circa 1996). This is exemplified by David Forney's statement at his August 1999 IMA talk on codes on graphs,"It feels like the early days." As another example of the classical/modern duality, finite-geometry codes were studied in the 1960s and thus are classical codes. However, they were rediscovered by Shu Lin *et al.* circa 2000 as a class of LDPC codes with very appealing features and are thus modern codes as well. The classical and modern incarnations of finite-geometry codes are distinguished by their decoders: one-step hard-decision decoding (classical) versus iterative soft-decision decoding (modern).

It has been 60 years since the publication in 1948 of Claude Shannon's celebrated *A Mathematical Theory of Communication*, which founded the fields of channel coding, source coding, and information theory. Shannon proved the existence of channel codes which ensure reliable communication provided that the information rate for a given code did not exceed the so-called capacity of the channel. In the first 45 years that followed Shannon's publication, a large number of very clever and very effective coding systems had been devised. However, none of these had been demonstrated, in a practical setting, to closely approach Shannon's theoretical limit. The first breakthrough came in 1993 with the discovery of turbo codes, the first class of codes shown to operate near Shannon's capacity limit. A second breakthrough came circa 1996 with the rediscovery of LDPC codes, which were also shown to have near-capacity performance. (These codes were first invented in 1961 and mostly ignored thereafter. The state of the technology at that time made them impractical.) Because it has been over a decade since the discovery of turbo and LDPC codes, the knowledge base for these codes is now quite mature and the time is ripe for a new book on channel codes.

This book was written for graduate students in engineering and computer science as well as research and development engineers in industry and academia. We felt compelled to collect all of this information in one source because it has

been scattered across many journal and conference papers. With this book, those entering the field of channel coding, and those wishing to advance their knowledge, conveniently have a single resource for learning about both classical and modern channel codes. Further, whereas the archival literature is written for experts, this textbook is appropriate for both the novice (earlier chapters) and the expert (later chapters). The book begins slowly and does not presuppose prior knowledge in the field of channel coding. It then extends to frontiers of the field, as is evident from the table of contents.

The topics selected for this book, of course, reflect the experiences and interests of the authors, but they were also selected for their importance in the study of channel codes – not to mention the fact that additional chapters would make the book physically unwieldy. Thus, the emphasis of this book is on codes for binary-input channels, including the binary-input additive white-Gaussian-noise channel, the binary symmetric channel, and the binary erasure channel. One notable area of omission is coding for wireless channels, such as MIMO channels. However, this book is useful for students and researchers in that area as well because many of the techniques applied to the additive white-Gaussian-noise channel, our main emphasis, can be extended to wireless channels. Another notable omission is soft-decision decoding of Reed–Solomon codes. While extremely important, this topic is not as mature as those in this book.

Several different course outlines are possible with this book. The most obvious for a first graduate course on channel codes includes selected topics from Chapters 1, 2, 3, 4, 5, and 7. Such a course introduces the student to capacity limits for several common channels (Chapter 1). It then provides the student with an introduction to just enough algebra (Chapter 2) to understand BCH and Reed–Solomon codes and their decoders (Chapter 3). Next, this course introduces the student to convolutional codes and their decoders (Chapter 4). This course next provides the student with an introduction to LDPC codes and iterative decoding (Chapter 5). Finally, with the knowledge gained from Chapters 4 and 5 in place, the student is ready to tackle turbo codes and turbo decoding (Chapter 7). The material contained in Chapters 1, 2, 3, 4, 5, and 7 is too much for a single-semester course and the instructor will have to select a preferred subset of that material.

For a more advanced course centered on LDPC code design, the instructor could select topics from Chapters 10, 11, 12, 13, and 14. This course would first introduce the student to LDPC code design using Euclidean geometries and projective geometries (Chapter 10). Then the student would learn about LDPC code design using finite fields (Chapter 11) and combinatorics and graphs (Chapter 12). Next, the student would apply some of the techniques from these earlier chapters to design codes for the binary erasure channel (Chapter 13). Lastly, the student would learn design techniques for nonbinary LDPC codes (Chapter 14).

As a final example of a course outline, a course could be centered on computer-based design of LDPC codes. Such a course would include Chapters 5, 6, 8, and 9. This course would be for those who have had a course on classical channel

codes, but who are interested in LDPC codes. The course would begin with an introduction to LDPC codes and various LDPC decoders (Chapter 5). Then, the student would learn about various computer-based code-design approaches, including Gallager codes, MacKay codes, codes based on protographs, and codes based on accumulators (Chapter 6). Next, the student would learn about assessing the performance of LDPC code ensembles from a weight-distribution perspective (Chapter 8). Lastly, the student would learn about assessing the performance of (long) LDPC codes from a decoding-threshold perspective via the use of density evolution and EXIT charts (Chapter 9).

All of the chapters contain a good number of problems. The problems are of various types, including those that require routine calculations or derivations, those that require computer solution or computer simulation, and those that might be characterized as a semester project. The authors have selected the problems to strengthen the student's knowledge of the material in each chapter (e.g., by requiring a computer simulation of a decoder) and to extend that knowledge (e.g., by requiring the proof of a result not contained in the chapter).

We wish to thank, first of all, Professor Ian Blake, who read an early version of the entire manuscript and provided many important suggestions that led to a much improved book.

We also wish to thank the many graduate students who have been a tremendous help in the preparation of this book. They have helped with typesetting, computer simulations, proofreading, and figures, and many of their research results can be found in this book. The students (former and current) who have contributed to W. Ryan's portion of the book are Dr. Yang Han, Dr. Yifei Zhang, Dr. Michael (Sizhen) Yang, Dr. Yan Li, Dr. Gianluigi Liva, Dr. Fei Peng, Shadi Abu-Surra, Kristin Jagiello (who proofread eight chapters), and Matt Viens. Gratitude is also due to Li Zhang (S. Lin's student) who provided valuable feedback on Chapters 6 and 9. Finally, W. Ryan acknowledges Sara Sandberg of Luleå Institute of Technology for helpful feedback on an early version of Chapter 5. The students who have contributed to S. Lin's portion of the book are Dr. Bo Zhou, Qin Huang, Dr. Ying Y. Tai, Dr. Lan Lan, Dr. Lingqi Zeng, Jingyu Kang, and Li Zhang; Dr. Bo Zhou and Qin Huang deserve special appreciation for typing S. Lin's chapters and overseeing the preparation of the final version of his chapters.

We thank Professor Dan Costello, who sent us reference material for the convolutional LDPC code section in Chapter 15, Dr. Marc Fossorier, who provided comments on Chapter 14, and Professor Ali Ghrayeb, who provided comments on Chapter 7.

We are grateful to the National Science Foundation, the National Aeronautics and Space Administration, and the Information Storage Industry Consortium for their many years of funding support in the area of channel coding. Without their support, many of the results in this book would not have been possible. We also thank the University of Arizona and the University of California, Davis for their support in the writing of this book.

We also acknowledge the talented Mrs. Linda Wyrgatsch whose painting on the back cover was created specifically for this book. We note that the paintings on the front and back covers are classical and modern, respectively.

Finally, we would like to give special thanks to our wives (Stephanie and Ivy), children, and grandchildren for their continuing love and affection throughout this project.

William E. Ryan
University of Arizona

Shu Lin
University of California, Davis

Web sites for this book:
www.cambridge.org/9780521848688
http://www.ece.arizona.edu/~ryan/ChannelCodesBook/

1 Coding and Capacity

1.1 Digital Data Communication and Storage

Digital communication systems are ubiquitous in our daily lives. The most obvious examples include cell phones, digital television via satellite or cable, digital radio, wireless internet connections via Wi-Fi and WiMax, and wired internet connection via cable modem. Additional examples include digital data-storage devices, including magnetic ("hard") disk drives, magnetic tape drives, optical disk drives (e.g., CD, DVD, blu-ray), and flash drives. In the case of data-storage, information is communicated from one point in time to another rather than one point in space to another. Each of these examples, while widely different in implementation details, generally fits into a common digital communication framework first established by C. Shannon in his 1948 seminal paper, *A Mathematical Theory of Communication* [1]. This framework is depicted in Figure 1.1, whose various components are described as follows.

Source and *user* (or *sink*). The information source may be originally in analog form (e.g., speech or music) and then later digitized, or it may be originally in digital form (e.g., computer files). We generally think of its output as a sequence of bits, which follow a probabilistic model. The user of the information may be a person, a computer, or some other electronic device.

Source encoder and *source decoder*. The encoder is a processor that converts the information source bit sequence into an alternative bit sequence with a more efficient representation of the information, i.e., with fewer bits. Hence, this operation is often called *compression*. Depending on the source, the compression can be *lossless* (e.g., for computer data files) or *lossy* (e.g., for video, still images, or music, where the loss can be made to be imperceptible or acceptable). The source decoder is the encoder's counterpart which recovers the source sequence exactly, in the case of lossless compression, or approximately, in the case of lossy compression, from the encoder's output sequence.

Channel encoder and *channel decoder*. The role of the channel encoder is to protect the bits to be transmitted over a channel subject to noise, distortion, and interference. It does so by converting its input into an alternate sequence possessing redundancy, whose role is to provide immunity from the various

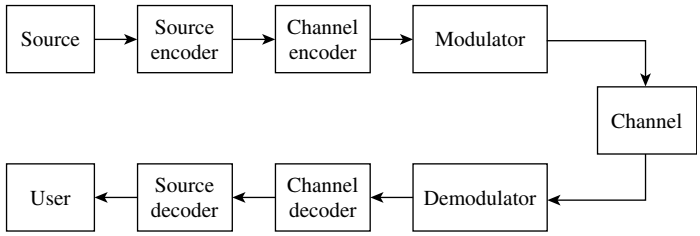


Figure 1.1 Basic digital communication- (or storage-) system block diagram due to Shannon.

channel impairments. The ratio of the number of bits that enter the channel encoder to the number that depart from it is called the *code rate*, denoted by R, with $0 < R < 1$. For example, if a 1000-bit codeword is assigned to each 500-bit information word, $R = 1/2$, and there are 500 redundant bits in each codeword. The function of the channel decoder is to recover from the channel output the input to the channel encoder (i.e., the compressed sequence) in spite of the presence of noise, distortion, and interference in the received word.

Modulator and *demodulator*. The modulator converts the channel-encoder output bit stream into a form that is appropriate for the channel. For example, for a wireless communication channel, the bit stream must be represented by a high-frequency signal to facilitate transmission with an antenna of reasonable size. Another example is a so-called modulation code used in data storage. The modulation encoder output might be a sequence that satisfies a certain runlength constraint (runs of like symbols, for example) or a certain spectral constraint (the output contains a null at DC, for example). The demodulator is the modulator's counterpart which recovers the modulator input sequence from the modulator output sequence.

Channel. The channel is the physical medium through which the modulator output is conveyed, or by which it is stored. Our experience teaches us that the channel adds noise and often interference from other signals, on top of the signal distortion that is ever-present, albeit sometimes to a minor degree. For our purposes, the channel is modeled as a probabilistic device, and examples will be presented below. Physically, the channel can included antennas, amplifiers, and filters, both at the transmitter and at the receiver at the ends of the system. For a hard-disk drive, the channel would include the write head, the magnetic medium, the read head, the read amplifier and filter, and so on.

Following Shannon's model, Figure 1.1 does not include such blocks as encryption/decryption, symbol-timing recovery, and scrambling. The first of these is optional and the other two are assumed to be ideal and accounted for in the probabilistic channel models. On the basis of such a model, Shannon showed that a channel can be characterized by a parameter, C, called the *channel capacity*, which is a measure of how much information the channel can convey, much like

the capacity of a plumbing system to convey water. Although C can be represented in several different units, in the context of the channel code rate R, which has the units information bits per channel bit, Shannon showed that codes exist that provide arbitrarily reliable communication provided that the code rate satisfies $R < C$. He further showed that, conversely, if $R > C$, there exists no code that provides reliable communication.

Later in this chapter, we review the capacity formulas for a number of commonly studied channels for reference in subsequent chapters. Prior to that, however, we give an overview of various channel-coding approaches for error avoidance in data-transmission and data-storage scenarios. We then introduce the first channel code invented, the (7,4) Hamming code, by which we mean a code that assigns to each 4-bit information word a 7-bit codeword according to a recipe specified by R. Hamming in 1950 [2]. This will introduce to the novice some of the elements of channel codes and will serve as a launching point for subsequent chapters. After the introduction to the (7,4) Hamming code, we present code- and decoder-design criteria and code-performance measures, all of which are used throughout this book.

1.2 Channel-Coding Overview

The large number of coding techniques for error prevention may be partitioned into the set of automatic request-for-repeat (ARQ) schemes and the set of forward-error-correction (FEC) schemes. In ARQ schemes, the role of the code is simply to reliably detect whether or not the received word (e.g., received packet) contains one or more errors. In the event a received word does contain one or more errors, a request for retransmission of the same word is sent out from the receiver back to the transmitter. The codes in this case are said to be *error-detection codes*. In FEC schemes, the code is endowed with characteristics that permit error correction through an appropriately devised decoding algorithm. The codes for this approach are said to be *error-correction codes*, or sometimes *error-control codes*. There also exist *hybrid FEC/ARQ schemes* in which a request for retransmission occurs if the decoder fails to correct the errors incurred over the channel and detects this fact. Note that this is a natural approach for data-storage systems: if the FEC decoder fails, an attempt to re-read the data is made. The codes in this case are said to be *error-detection-and-correction codes*.

The basic ARQ schemes can broadly be subdivided into the following protocols. First is the *stop-and-wait ARQ* scheme in which the transmitter sends a codeword (or encoded packet) and remains idle until the ACK/NAK status signal is returned from the receiver. If a positive acknowledgment (ACK) is returned, a new packet is sent; otherwise, if a negative acknowledgment (NAK) is returned, the current packet, which was stored in a buffer, is retransmitted. The stop-and-wait method is inherently inefficient due to the idle time spent waiting for confirmation.

In *go-back-N ARQ*, the idle time is eliminated by continually sending packets while waiting for confirmations. If a packet is negatively acknowledged, that packet and the $N-1$ subsequent packets sent during the round-trip delay are retransmitted. Note that this preserves the ordering of packets at the receiver.

In *selective-repeat ARQ*, packets are continually transmitted as in go-back-N ARQ, except only the packet corresponding to the NAK message is retransmitted. (The packets have "headers," which effectively number the information block for identification.) Observe that, because only one packet is retransmitted rather than N, the throughput of accepted packets is increased with selective-repeat ARQ relative to go-back-N ARQ. However, there is the added requirement of ample buffer space at the receiver to allow re-ordering of the blocks.

In *incremental-redundancy ARQ*, upon receiving a NAK message for a given packet, the transmitter transmits additional redundancy to the receiver. This additional redundancy is used by the decoder together with the originally received packet in a second attempt to recover the original data. This sequence of steps – NAK, additional redundancy, re-decode – can be repeated a number of times until the data are recovered or the packet is declared lost.

While ARQ schemes are very important, this book deals exclusively with FEC schemes. However, although the emphasis is on FEC, each of the FEC codes introduced can be used in a hybrid FEC/ARQ scheme where the code is used for both correction and detection. There exist many FEC schemes, employing both linear and nonlinear codes, although virtually all codes used in practice can be characterized as linear or linear at their core. Although the concept will be elucidated in Chapter 3, a *linear code* is one for which any sum of codewords is another codeword in the code. Linear codes are traditionally partitioned to the set of block codes and convolutional, or trellis-based, codes, although the turbo codes of Chapter 7 can be seen to be a hybrid of the two. Among the linear block codes are the cyclic and quasi-cyclic codes (defined in Chapter 3), both of which have more algebraic structure than do standard linear block codes. Also, we have been tacitly assuming binary codes, that is, codes whose code symbols are either 0 or 1. However, codes whose symbols are taken from a larger alphabet (e.g., 8-bit ASCII characters or 1000-bit packets) are possible, as described in Chapters 3 and 14.

This book will provide many examples of each of these code types, including nonbinary codes, and their decoders. For now, we introduce the first FEC code, due to Hamming [2], which provides a good introduction to the field of channel codes.

1.3 Channel-Code Archetype: The (7,4) Hamming Code

The (7,4) Hamming code serves as an excellent channel-code prototype since it contains most of the properties of more practical codes. As indicated by the notation (7,4), the codeword length is $n = 7$ and the data word length is $k = 4$, so

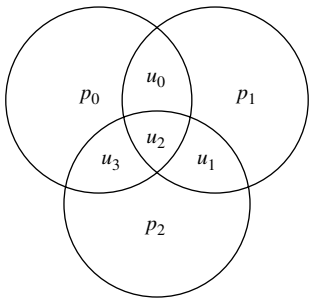


Figure 1.2 Venn-diagram representation of (7,4) Hamming-code encoding and decoding rules.

the code rate is $R = 4/7$. As shown by R. McEliece, the Hamming code is easily described by the simple Venn diagram in Figure 1.2. In the diagram, the information word is represented by the vector $\mathbf{u} = (u_0, u_1, u_2, u_3)$ and the redundant bits (called *parity bits*) are represented by the parity vector $\mathbf{p} = (p_0, p_1, p_2)$. The codeword (also, code vector) is then given by the concatenation of $\mathbf{u}$ and $\mathbf{p}$:

$$\mathbf{v} = (\mathbf{u}\ \mathbf{p}) = (u_0, u_1, u_2, u_3, p_0, p_1, p_2) = (v_0, v_1, v_2, v_3, v_4, v_5, v_6).$$

The encoding rule is trivial: the parity bits are chosen so that each circle has an even number of 1s, i.e., the sum of bits in each circle is 0 modulo 2. From this encoding rule, we may write

$$\begin{aligned} p_0 &= u_0 + u_2 + u_3, \\ p_1 &= u_0 + u_1 + u_2, \\ p_2 &= u_1 + u_2 + u_3, \end{aligned} \tag{1.1}$$

from which the 16 codewords are easily found:

0000 000	1000 110	0010 111
1111 111	0100 011	1001 011
	1010 001	1100 101
	1101 000	1110 010
	0110 100	0111 001
	0011 010	1011 100
	0001 101	0101 110

As an example encoding, consider the third codeword in the middle column, (1010 001), for which the data word is $\mathbf{u} = (u_0, u_1, u_2, u_3) = (1, 0, 1, 0)$. Then,

$$\begin{aligned} p_0 &= 1 + 1 + 0 = 0, \\ p_1 &= 1 + 0 + 1 = 0, \\ p_2 &= 0 + 1 + 0 = 1, \end{aligned}$$

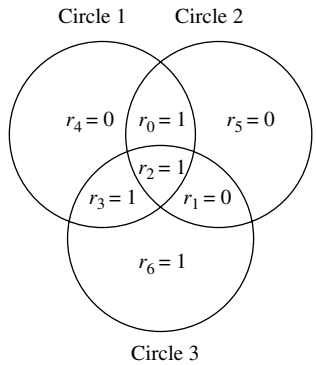


Figure 1.3 Venn-diagram setup for the Hamming decoding example.

yielding $\mathbf{v} = (\mathbf{u}\ \mathbf{p}) = (1010\ 001)$. Observe that this code is linear because the sum of any two codewords yields a codeword. Note also that this code is cyclic: a cyclic shift of any codeword, rightward or leftward, gives another codeword.

Suppose now that $\mathbf{v} = (1010\ 001)$ is transmitted, but $\mathbf{r} = (1011\ 001)$ is received. That is, the channel has converted the 0 in code bit v_3 into a 1. The Venn diagram of Figure 1.3 can be used to decode $\mathbf{r}$ and correct the error. Note that Circle 2 in the figure has an even number of 1s in it, but Circles 1 and 3 do not. Thus, because the code rules are not satisfied by the bits in $\mathbf{r}$, we know that $\mathbf{r}$ contains one or more errors. Because a single error is more likely than two or more errors for most practical channels, we assume that $\mathbf{r}$ contains a single error. Then, the error must be in the intersection of Circles 1 and 3. However, $r_2 = 1$ in the intersection cannot be in error because it is in Circle 2 whose rule is satisfied. Thus, it must be $r_3 = 1$ in the intersection that is in error. Thus, v_3 must be 0 rather than the 1 shown in Figure 1.3 for r_3. In conclusion, the decoded codeword is $\hat{\mathbf{v}} = (1010\ 001)$, from which the decoded data $\hat{\mathbf{u}} = (1010)$ may be recovered.

It can be shown (see Chapter 3) that this particular single error is not special and that, independently of the codeword transmitted, all seven single errors are correctable and no error patterns with more than one error are correctable. The novice might ask what characteristic of these 16 codewords endows them with the ability to correct all single-errors. This is easily explained using the concept of the *Hamming distance* $d_{\mathrm{H}}(\mathbf{x}, \mathbf{x}')$ between two length-n words $\mathbf{x}$ and $\mathbf{x}'$, which is the number of locations in which they disagree. Thus, $d_{\mathrm{H}}(1000\ 110, 0010\ 111) = 3$. It can be shown, either exhaustively or using the principles developed in Chapter 3, that $d_{\mathrm{H}}(\mathbf{v}, \mathbf{v}') \geq 3$ for any two different codewords $\mathbf{v}$ and $\mathbf{v}'$ in the Hamming code. We say that the code's *minimum distance* is therefore $d_{\min} = 3$. Because $d_{\min} = 3$, a single error in some transmitted codeword $\mathbf{v}$ yields a received vector $\mathbf{r}$ that is closer to $\mathbf{v}$, in the sense of Hamming distance, than any other codeword. It is for this reason that all single errors are correctable.

Generalizations of the Venn-diagram code description for the more complex codes used in applications are presented in Chapter 3 and subsequent chapters. In the chapters to follow, we will revisit the Hamming code a number of times, particularly in the problems. We will see how to reformulate encoding so that it employs a so-called generator matrix or, better, a simple shift-register circuit. We will also see how to reformulate decoding so that it employs a so-called parity-check matrix, and we will see many different decoding algorithms. Further, we will see applications of codes to a variety of channels, particularly ones introduced in the next section. Finally, we will see that a "good code" generally has the following properties: it is easy to encode, it is easy to decode, it a has large $d_{\min}$, and/or the number of codewords at the distance $d_{\min}$ from any other codeword is small. We will see many examples of good codes in this book, and of their construction, their encoding, and their decoding.

1.4 Design Criteria and Performance Measures

Although there exist many channel models, it is usual to start with the two most frequently encountered memoryless channels: the binary symmetric channel (BSC) and the binary-input additive white-Gaussian-noise channel (BI-AWGNC). Examination of the BSC and BI-AWGNC illuminates many of the salient features of code and decoder design and code performance. For the sake of uniformity, for both channels, we denote the ith channel input by x_i and the ith channel output by y_i. Given channel input $x_i = v_i \in \{0, 1\}$ and channel output $y_i \in \{0, 1\}$, the BSC is completely characterized by the channel transition probabilities $P(y_i|x_i)$ given by

$$P(y_i = 1|x_i = 0) = P(y_i = 0|x_i = 1) = \varepsilon,$$
$$P(y_i = 1|x_i = 1) = P(y_i = 0|x_i = 0) = 1 - \varepsilon,$$

where ε is called the *crossover probability*. For the BI-AWGNC, the code bits are mapped to the channel inputs as $x_i = (-1)^{v_i} \in \{\pm 1\}$ so that $x_i = +1$ when $v_i = 0$. The BI-AWGNC is then completely characterized by the channel transition probability density function (pdf) $p(y_i|x_i)$ given by

$$p(y_i|x_i) = \frac{1}{\sqrt{2\pi}\sigma} \exp\Big[-(y_i - x_i)^2 / (2\sigma^2)\Big],$$

where σ^2 is the variance of the zero-mean Gaussian noise sample n_i that the channel adds to the transmitted value x_i (so that $y_i = x_i + n_i$). As a consequence of its memorylessness, we have for the BSC

$$P(\mathbf{y}|\mathbf{x}) = \prod_i P(y_i|x_i)\,, \tag{1.2}$$

where $\mathbf{y} = [y_1, y_2, y_3, \ldots]$ and $\mathbf{x} = [x_1, x_2, x_3, \ldots]$. A similar expression exists for the BI-AWGNC with $P(\cdot)$ replaced by $p(\cdot)$.

The most obvious design criterion applicable to the design of a decoder is the minimum-probability-of-error criterion. When the design criterion is to minimize the probability that the decoder fails to decode to the correct codeword, i.e., to minimize the probability of a *codeword error*, it can be shown that this is equivalent to maximizing the *a posteriori* probability $P(\mathbf{x}|\mathbf{y})$ (or $p(\mathbf{x}|\mathbf{y})$ for the BI-AWGNC). The optimal decision for the BSC is then given by

$$\hat{\mathbf{v}} = \arg\max_{\mathbf{v}} P(\mathbf{x}|\mathbf{y}) = \arg\max_{\mathbf{v}} \frac{P(\mathbf{y}|\mathbf{x})P(\mathbf{x})}{P(\mathbf{y})}, \tag{1.3}$$

where $\arg\max_{\mathbf{v}} f(\mathbf{v})$ equals the argument $\mathbf{v}$ that maximizes the function $f(\mathbf{v})$. Frequently, the channel-input words are equally likely and, hence, $P(\mathbf{x})$ is independent of $\mathbf{x}$ (hence, $\mathbf{v}$). Because $P(\mathbf{y})$ is also independent of $\mathbf{v}$, the maximum *a posteriori* (MAP) rule (1.3) can be replaced by the *maximum-likelihood* (ML) rule

$$\hat{\mathbf{v}} = \arg\max_{\mathbf{v}} P(\mathbf{y}|\mathbf{x}).$$

Using (1.2) and the monotonicity of the log function, we have for the BSC

$$\begin{aligned}
\hat{\mathbf{v}} &= \arg\max_{\mathbf{v}} \log \prod_i P(y_i|x_i) \\
&= \arg\max_{\mathbf{v}} \sum_i \log P(y_i|x_i) \\
&= \arg\max_{\mathbf{v}} \left[d_{\mathrm{H}}(\mathbf{y},\mathbf{x})\log(\varepsilon) + (n - d_{\mathrm{H}}(\mathbf{y},\mathbf{x}))\log(1-\varepsilon)\right] \\
&= \arg\max_{\mathbf{v}} \left[d_{\mathrm{H}}(\mathbf{y},\mathbf{x})\log\left(\frac{\varepsilon}{1-\varepsilon}\right) + n\log(1-\varepsilon)\right] \\
&= \arg\min_{\mathbf{v}} d_{\mathrm{H}}(\mathbf{y},\mathbf{x}),
\end{aligned}$$

where n is the codeword length and the last line follows since $\log[\varepsilon/(1-\varepsilon)] < 0$ and $n\log(1-\varepsilon)$ is not a function of $\mathbf{v}$.

For the BI-AWGNC, the ML decision is

$$\hat{\mathbf{v}} = \arg\max_{\mathbf{v}} P(\mathbf{y}|\mathbf{x}),$$

keeping in mind the mapping $\mathbf{x} = (-1)^{\mathbf{v}}$. Following a similar set of steps (and dropping irrelevant terms), we have

$$\begin{aligned}
\hat{\mathbf{v}} &= \arg\max_{\mathbf{v}} \sum_i \log\left(\frac{1}{\sqrt{2\pi}\sigma}\exp\left[-(y_i - x_i)^2/(2\sigma^2)\right]\right) \\
&= \arg\min_{\mathbf{v}} \sum_i (y_i - x_i)^2 \\
&= \arg\min_{\mathbf{v}} d_{\mathrm{E}}(\mathbf{y},\mathbf{x}),
\end{aligned}$$

where

$$d_{\mathrm{E}}(\mathbf{y}, \mathbf{x}) = \sqrt{\sum_i (y_i - x_i)^2}$$

is the *Euclidean distance* between $\mathbf{y}$ and $\mathbf{x}$, and on the last line we replaced $d_{\mathrm{E}}^2(\cdot)$ by $d_{\mathrm{E}}(\cdot)$ due to the monotonicity of the square-root function for non-negative arguments. Note that, once a decision is made on the codeword, the decoded data word $\hat{\mathbf{u}}$ may easily be recovered from $\hat{\mathbf{v}}$, particularly when the codeword is in the form $\mathbf{v} = (\mathbf{u}\ \mathbf{p})$.

In summary, for the BSC, the ML decoder chooses the codeword $\mathbf{v}$ that is closest to the channel output $\mathbf{y}$ in a Hamming-distance sense; for the BI-AWGNC, the ML decoder chooses the code sequence $\mathbf{x} = (-1)^{\mathbf{v}}$ that is closest to the channel output $\mathbf{y}$ in a Euclidean-distance sense. The implication for code design on the BSC is that the code should be designed to maximize the minimum Hamming distance between two codewords (and to minimize the number of codeword pairs at that distance). Similarly, the implication for code design on the BI-AWGNC is that the code should be designed to maximize the minimum Euclidean distance between any two code sequences on the channel (and to minimize the number of code-sequence pairs at that distance).

Finding the codeword $\mathbf{v}$ that minimizes the Hamming (or Euclidean) distance in a brute-force, exhaustive fashion is very complex except for very simple codes such as the (7,4) Hamming code. Thus, ML decoding algorithms have been developed that exploit code structure, vastly reducing complexity. Such algorithms are presented in subsequent chapters. Suboptimal but less complex algorithms, which perform slightly worse than the ML decoder, will also be presented in subsequent chapters. These include so-called bounded-distance decoders, list decoders, and iterative decoders involving component sub-decoders. Often these component decoders are based on the *bit-wise MAP criterion* which minimizes the probability of *bit error* rather than the probability of *codeword* error. This bit-wise MAP criterion is

$$\hat{v}_i = \arg\max_{v_i} P(x_i|\mathbf{y}) = \arg\max_{v_i} \frac{P(\mathbf{y}|x_i)P(x_i)}{P(\mathbf{y})},$$

where the *a priori* probability $P(x_i)$ is constant (and ignored together with $P(\mathbf{y})$) if the decoder is operating in isolation, but is supplied by a companion decoder if the decoder is part of an iterative decoding scheme. This topic will also be discussed in subsequent chapters.

The most commonly used performance measure is the *bit-error probability*, P_b, defined as the probability that the decoder output decision $\hat{u}_i$ does not equal the encoder input bit u_i,

$$P_b \triangleq \Pr\{\hat{u}_i \neq u_i\}.$$

Strictly speaking, we should average over all i to obtain P_b. However, $\Pr\{\hat{u}_i = u_i\}$ is frequently independent of i, although, if it is not, the averaging is understood.

P_b is often called the *bit-error rate*, denoted BER. Another commonly used performance measure is the codeword-error probability, P_{cw}, defined as the probability that the decoder output decision $\hat{\mathbf{v}}$ does not equal the encoder output codeword $\mathbf{v}$,

$$P_{cw} \triangleq \Pr\{\hat{\mathbf{v}} \neq \mathbf{v}\}.$$

In the coding literature, various alternative terms are used for P_{cw}, including *word-error rate* (WER) and *frame-error rate* (FER). A closely related error probability is the probability $P_{uw} \triangleq \Pr\{\hat{\mathbf{u}} \neq \mathbf{u}\}$, which can be useful for some applications, but we shall not emphasize this probability, particularly since $P_{uw} \approx P_{cw}$ for many coding systems. Lastly, for nonbinary codes, the symbol-error probability P_s is pertinent. It is defined as

$$P_s \triangleq \Pr\{\hat{u}_i \neq u_i\},$$

where in this case the encoder input symbols u_i and the decoder output symbols $\hat{u}_i$ are nonbinary. P_s is also called the *symbol-error rate* (SER). We shall use the notation introduced in this paragraph throughout this book.

1.5 Channel-Capacity Formulas for Common Channel Models

From the time of Shannon's seminal work in 1948 until the early 1990s, it was thought that the only codes capable of operating near capacity are long, impractical codes, that is, codes that are impossible to encode and decode in practice. However, the invention of turbo codes and low-density parity-check (LDPC) codes in the 1990s demonstrated that near-capacity performance was possible in practice. (As explained in Chapter 5, LDPC codes were first invented circa 1960 by R. Gallager and later independently re-invented by MacKay and others circa 1995. Their capacity-approaching properties with practical encoders/decoders could not be demonstrated with 1960s technology, so they were mostly ignored for about 35 years.) Because of the advent of these capacity-approaching codes, knowledge of information theory and channel capacity has become increasingly important for both the researcher and the practicing engineer. In this section we catalog capacity formulas for a variety of commonly studied channel models. We point out that these formulas correspond to infinite-length codes. However, we will see numerous examples in this book where finite-length codes operate very close to capacity, although this is possible only with long codes ($n > 5000$, say).

No derivations are given for the various capacity formulas. For such information, see [3–9]. However, it is useful to highlight the general formula for the mutual information between the channel output represented by Y and the channel input represented by X. When the input and output take values from a discrete set, then the *mutual information* may be written as

$$I(X;Y) = H(Y) - H(Y|X), \tag{1.4}$$

where $H(Y)$ is the *entropy* of the channel output,

$$\begin{aligned} H(Y) &= -\mathbb{E}\{\log_2(\Pr(y))\} \\ &= -\sum_y \Pr(y)\log_2(\Pr(y)), \end{aligned}$$

and $H(Y|X)$ is the *conditional entropy* of Y given X,

$$\begin{aligned} H(Y|X) &= -\mathbb{E}\{\log_2(\Pr(y|x))\} \\ &= -\sum_x \sum_y \Pr(x,y)\log_2(\Pr(y|x)) \\ &= -\sum_x \sum_y \Pr(x)\Pr(y|x)\log_2(\Pr(y|x)). \end{aligned}$$

In these expressions, $\mathbb{E}\{\cdot\}$ represents probabilistic expectation. The form (1.4) is most commonly used, although the alternative form $I(X;Y) = H(X) - H(X|Y)$ is sometimes useful. The capacity of a channel is then defined as

$$C = \max_{\{\Pr(x)\}} I(X;Y), \tag{1.5}$$

that is, the capacity is the maximum mutual information, where the maximization is over the channel input probability distribution $\{\Pr(x)\}$. As a practical matter, most channel models are symmetric, in which case the optimal input distribution is uniform so that the capacity is given by

$$I(X;Y)|_{\text{uniform } \{\Pr(x)\}} = [H(Y) - H(Y|X)]_{\text{uniform } \{\Pr(x)\}}. \tag{1.6}$$

For cases in which the channel is not symmetric, the Blahut–Arimoto algorithm [3, 6] can be used to perform the optimization of $I(X;Y)$. Alternatively, the uniform-input information rate of (1.6) can be used as an approximation of capacity, as will be seen below for the Z channel. For a continuous channel output Y, the entropies in (1.4) are replaced by *differential entropies* $h(Y)$ and $h(Y|X)$, which are defined analogously to $H(Y)$ and $H(Y|X)$, with the probability mass functions replaced by probability density functions and the sums replaced by integrals.

Consistently with the code rate defined earlier, C and $I(X;Y)$ are in units of information bits/code bit. Unless indicated otherwise, the capacities presented in the remainder of this chapter have these units, although we will see that it is occasionally useful to convert to alternative units. Also, all code rates R for which $R < C$ are said to be *achievable rates* in the sense that reliable communication is achievable at these rates.

1.5.1 Capacity for Binary-Input Memoryless Channels

1.5.1.1 The BEC and the BSC

The binary erasure channel (BEC) and the binary symmetric channel (BSC) are illustrated in Figures 1.4 and 1.5. For the BEC, p is the probability of a bit erasure, which is represented by the symbol e, or sometimes by ? to indicate the fact that

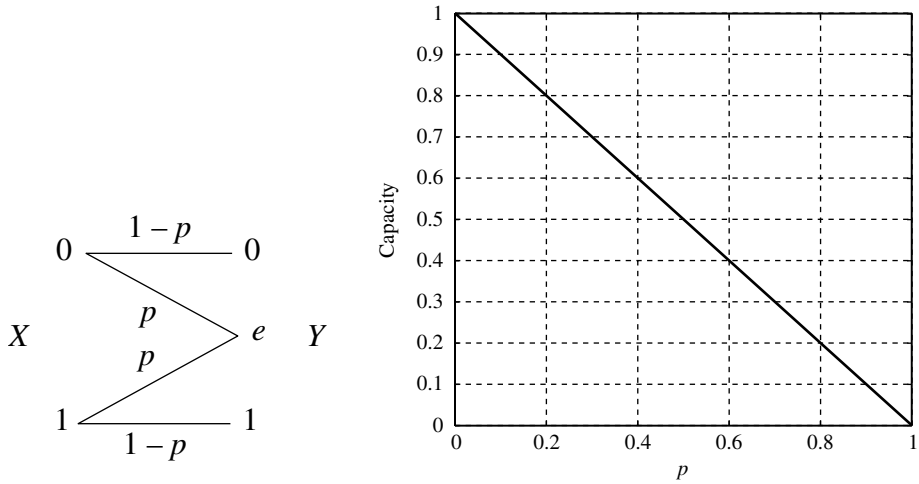


Figure 1.4 The binary erasure channel and a plot of its capacity.

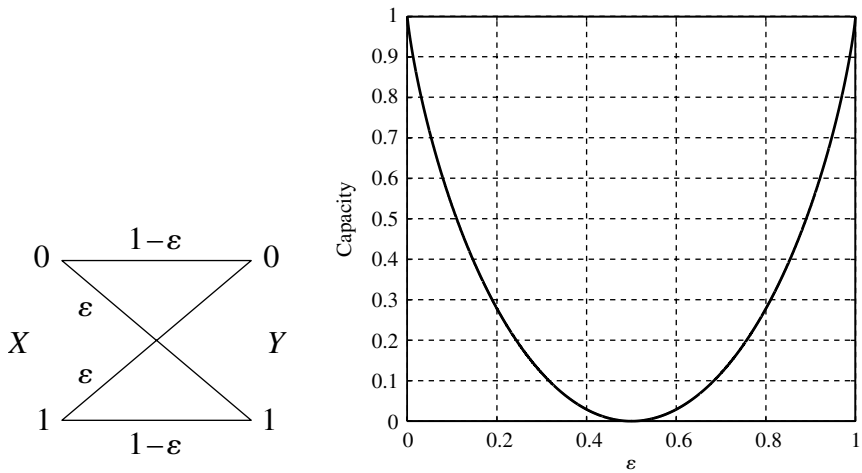


Figure 1.5 The binary symmetric channel and a plot of its capacity.

nothing is known about the bit that was erased. For the BSC, ε is the probability of a bit error. While simple, both models are useful for practical applications and academic research. The capacity of the BEC is easily shown from (1.6) to be

$$C_{\mathrm{BEC}} = 1 - p. \tag{1.7}$$

It can be similarly shown that the capacity of the BSC is given by

$$C_{\mathrm{BSC}} = 1 - \mathcal{H}(\varepsilon), \tag{1.8}$$

where $\mathcal{H}(p)$ is the *binary entropy function* given by

$$\mathcal{H}(\varepsilon) = -\varepsilon \log_2(\varepsilon) - (1-\varepsilon)\log_2(1-\varepsilon).$$

The derivations of these capacity formulas from (1.6) are considered in one of the problems. C_{BEC} is plotted as a function of p in Figure 1.4 and C_{BSC} is plotted as a function of ε in Figure 1.5.

1.5.1.2 *The Z Channel*

The Z channel, depicted in Figure 1.6, is an idealized model of a free-space optical communication channel. It is an extreme case of an asymmetric binary-input/binary-output channel and is sometimes used to model solid-state memories. As indicated in the figure, the probability of error when transmitting a 0 is zero and the probability of error when transmitting a 1 is q. Let u equal the probability of transmitting a 1, $u = \Pr(X = 1)$. Then the capacity is given [10] by

$$\begin{aligned} C_Z &= \max_u \{\mathcal{H}(up) - u\mathcal{H}(q)\} \\ &= \mathcal{H}(u'p) - u'\mathcal{H}(q), \end{aligned} \tag{1.9}$$

where u' is the maximizing value of u, given by

$$u' = \frac{q^{q/(1-q)}}{1 + (1-q)q^{q/(1-q)}}. \tag{1.10}$$

Our intuition tells us that it would be vastly advantageous to design error-correction codes that favor sending 0s, that is, whose codewords have more 0s than

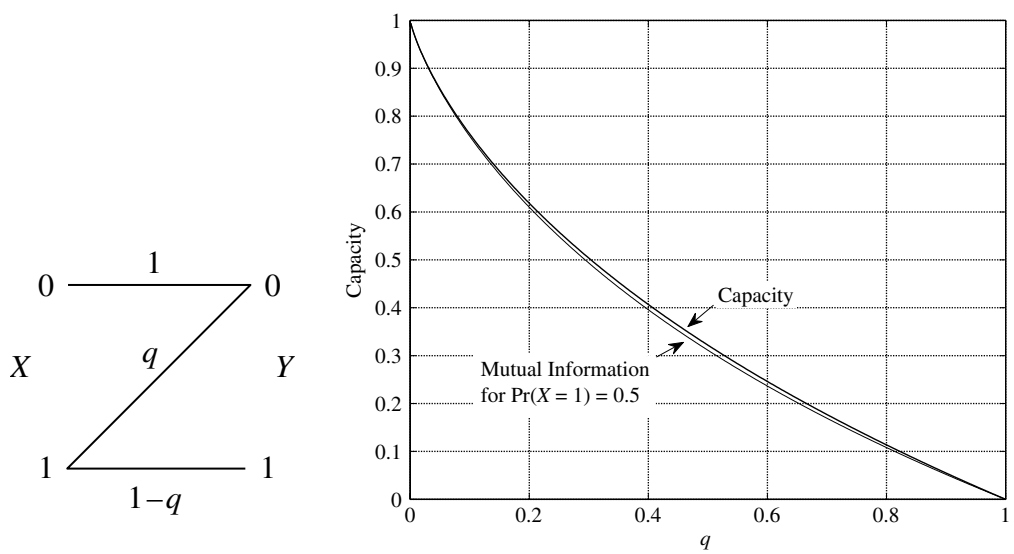


Figure 1.6 The Z channel and a plot of its capacity.

1s so that $u < 0.5$. However, consider the following example from [11]. Suppose that $q = 0.1$. Then $u' = 0.4563$ and $C_Z = 0.7628$. For comparison, suppose that we use a code for which the 0s and 1s are uniformly occurring, that is, $u = 0.5$. In this case, the mutual information $I(X;Y) = \mathcal{H}(up) - u\mathcal{H}(q) = 0.7583$, so that little is lost by using such a code in lieu of an optimal code with $u' = 0.4563$. We have plotted both C_Z and $I(X;Y)$ against q in Figure 1.6, where it is seen that for all $q \in [0,1]$ there is little difference between C_Z and $I(X;Y)$. Thus, it appears that there is little to be gained by trying to invent codes with non-uniform symbols for the Z channel and similar asymmetric channels.

1.5.1.3 *The Binary-Input AWGN Channel*

Consider the discrete-time channel model,

$$y_\ell = x_\ell + z_\ell, \tag{1.11}$$

where $x_\ell \in \{\pm 1\}$ and z_ℓ is a real-valued additive white Gaussian noise (AWGN) sample with variance σ^2, i.e., $z_\ell \sim \mathcal{N}\big(0, \sigma^2\big)$. This channel is called the *binary-input AWGN (BI-AWGN) channel*. The capacity can be shown to be

$$C_{\text{BI-AWGN}} = 0.5 \sum_{x=\pm 1} \int_{-\infty}^{+\infty} p(y|x) \log_2\left(\frac{p(y|x)}{p(y)}\right) dy, \tag{1.12}$$

where

$$p(y|x = \pm 1) = \frac{1}{\sqrt{2\pi}\sigma} \exp\big[-(y \mp 1)^2/(2\sigma^2)\big]$$

and

$$p(y) = \frac{1}{2}[p(y|x = +1) + p(y|x = -1)].$$

An alternative formula, which follows from $C = h(Y) - h(Y|X) = h(Y) - h(Z)$, is

$$C_{\text{BI-AWGN}} = -\int_{-\infty}^{+\infty} p(y) \log_2(p(y))\, dy - 0.5 \log_2(2\pi e \sigma^2), \tag{1.13}$$

where we used $h(Z) = 0.5 \log_2(2\pi e \sigma^2)$, which is shown in one of the problems. Both forms require numerical integration to compute, e.g., Monte Carlo integration. For example, the integral in (1.13) is simply the expectation $\mathbb{E}\{-\log_2(p(y))\}$, which may be estimated as

$$\mathbb{E}\{-\log_2(p(y))\} \simeq -\frac{1}{L} \sum_{\ell=1}^{L} \log_2(p(y_\ell)), \tag{1.14}$$

where $\{y_\ell : \ell = 1, \ldots, L\}$ is a large number of realizations of the random variable Y.

In Figure 1.7, $C_{\text{BI-AWGN}}$ (labeled "soft") is plotted against the commonly used signal-to-noise-ratio (SNR) measure E_b/N_0, where E_b is the average energy per information bit and $N_0/2 = \sigma^2$ is the two-sided power spectral density of the

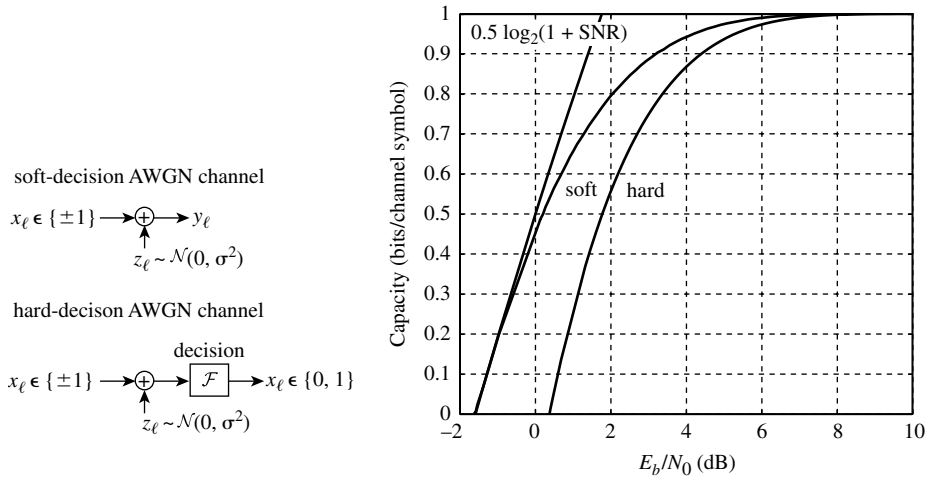


Figure 1.7 Capacity curves for the soft-decision and hard-decision binary-input AWGN channel together with the unconstrained-input AWGN channel-capacity curve.

AWGN process z_ℓ. Because $E_b = \mathbb{E}\left[x_\ell^2\right]/R = 1/R$, E_b/N_0 is related to the alternative SNR definition $E_s/\sigma^2 = \mathbb{E}\{x_\ell^2\}/\sigma^2 = 1/\sigma^2$ by the code rate and, for convenience, a factor of two: $E_b/N_0 = 1/(2R\sigma^2)$. The value of R used in this translation of SNR definitions is $R = C_{\text{BI-AWGN}}$ because R is assumed to be just less than $C_{\text{BI-AWGN}}$ ($R < C_{\text{BI-AWGN}} - \delta$, where $\delta > 0$ is arbitrarily small).

Also shown in Figure 1.7 is the capacity curve for a *hard-decision* BI-AWGN channel (labeled "hard"). For this channel, so-called hard-decisions $\hat{x}_\ell$ are obtained from the *soft-decisions* y_ℓ of (1.11) according to

$$\hat{x}_\ell = \begin{cases} 1 & \text{if } y_\ell \le 0 \\ 0 & \text{if } y_\ell > 0. \end{cases}$$

The soft-decision and hard-decision models are also included in Figure 1.7. Note that the hard-decisions convert the BI-AWGN channel into a BSC with error probability $\varepsilon = Q(1/\sigma) = Q\left(\sqrt{2RE_b/N_0}\right)$, where

$$Q(a) \triangleq \int_a^\infty \frac{1}{\sqrt{2\pi}} \exp\left(-\beta^2/2\right) d\beta.$$

Thus, the hard-decision curve in Figure 1.7 is plotted using $C_{\text{BSC}} = 1 - \mathcal{H}(\varepsilon)$. It is seen in the figure that the conversion to bits, i.e., hard decisions, prior to decoding results in a loss of 1 to 2 dB, depending on the code rate $R = C_{\text{BI-AWGN}}$.

Finally, also included in Figure 1.7 is the capacity of the unconstrained-input AWGN channel discussed in Section 1.5.2.1. This capacity, $C = 0.5\log_2(1 + 2RE_b/N_0)$, often called the *Shannon capacity*, gives the upper limit over all one-dimensional signaling schemes (transmission alphabets) and is discussed in the next section. Observe that the capacity of the soft-decision

Table 1.1. E_b/N_0 limits for various rates and channels

Rate R	$(E_b/N_0)_{\text{Shannon}}$ (dB)	$(E_b/N_0)_{\text{soft}}$ (dB)	$(E_b/N_0)_{\text{hard}}$ (dB)
0.05	−1.440	−1.440	0.480
0.10	−1.284	−1.285	0.596
0.15	−1.133	−1.126	0.713
0.20	−0.976	−0.963	0.839
1/4	−0.817	−0.793	0.972
0.30	−0.657	−0.616	1.112
1/3	−0.550	−0.497	1.211
0.35	−0.495	−0.432	1.261
0.40	−0.333	−0.236	1.420
0.45	−0.166	−0.030	1.590
1/2	0	0.187	1.772
0.55	0.169	0.423	1.971
0.60	0.339	0.682	2.188
0.65	0.511	0.960	2.428
2/3	0.569	1.059	2.514
0.70	0.686	1.275	2.698
3/4	0.860	1.626	3.007
4/5	1.037	2.039	3.370
0.85	1.215	2.545	3.815
9/10	1.396	3.199	4.399
0.95	1.577	4.190	5.295

binary-input AWGN channel is close to that of the unconstrained-input AWGN channel for low code rates. Observe also that reliable communication is not possible for $E_b/N_0 < -1.59$ dB $= 10\log_{10}(\ln(2))$ dB.

Table 1.1 lists the E_b/N_0 values required to achieve selected code rates for these three channels. Modern codes such as turbo and LDPC codes are capable of operating very close to these E_b/N_0 "limits," within 0.5 dB for long codes. The implication of these E_b/N_0 limits is that, for the given code rate R, error-free communication is possible in principle via channel coding if E_b/N_0 just exceeds its limit; otherwise, if the SNR is less than the limit, reliable communication is not possible.

For some applications, such as audio or video transmission, error-free communication is unnecessary and a bit error rate of $P_b = 10^{-3}$, for example, might be sufficient. In this case, one could operate satisfactorily below the error-free E_b/N_0 limits given in Table 1.1, thus easing the requirements on E_b/N_0 (and hence on transmission power). To determine the "P_b-constrained" E_b/N_0 limits [12, 13], one creates a model in which there is a source encoder and decoder in addition to the channel encoder, channel, and channel decoder. Then, the source-coding system (encoder/decoder) is chosen so that it introduces errors at the rate P_b while the

channel-coding system (encoder/channel/decoder) is error-free. Thus, the error rate for the entire model is P_b, and we can examine its theoretical limits.

To this end, suppose that the model's overall code rate of interest is R (information bits/channel bit). Then R is given by the model's channel code rate R_c (source-code bits/channel bit) divided by the model's source-code rate R_s (source-code bits/information bit): $R = R_c/R_s$. It is known [3–6] that the theoretical (lower) limit on R_s with the error rate P_b as the "distortion" is

$$R_s = 1 - \mathcal{H}(P_b).$$

Because $R_c = RR_s$, we have $R_c = R(1 - \mathcal{H}(P_b))$; but, for a given SNR $= 1/\sigma^2$, we have the limit, from (1.12) or (1.13), $R_c = C_{\text{BI-AWGN}}\big(1/\sigma^2\big)$, from which we may write

$$R_c = C_{\text{BI-AWGN}}\big(1/\sigma^2\big) = R(1 - \mathcal{H}(P_b)). \tag{1.15}$$

For a specified R and P_b, we determine from Equation (1.15) that $1/\sigma^2 = C^{-1}_{\text{BI-AWGN}}(R_c)$. Then, we set $E_b/N_0 = 1/(2R\sigma^2)$, which corresponds to the specified R and P_b. In this way, we can produce the rate-1/2 curve displayed in Figure 1.8. Observe that, as P_b decreases, the curve asymptotically approaches the E_b/N_0 value equal to the $(E_b/N_0)_{\text{soft}}$ limit given in Table 1.1 (0.187 dB). The curve can be interpreted as the minimum achievable E_b/N_0 for a given bit error rate P_b for rate-1/2 codes.

Having just discussed theoretical limits for a nonzero bit error rate for infinite-length codes, we note that a number of performance limits exist for finite-length

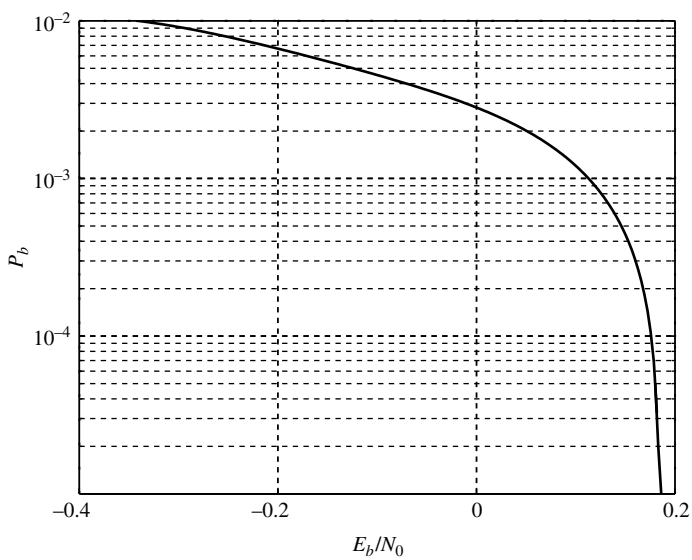


Figure 1.8 The minimum achievable E_b/N_0 for a given bit error rate P_b for rate-1/2 codes.

codes. These so-called sphere-packing bounds are reviewed in [14] and are beyond our scope. We point out, however, a bound due to Gallager [4] (see also [7, 9]), which is useful for code lengths n greater than about 200 bits. The so-called *Gallager random coding bound* is on the codeword error probability P_{cw} instead of the bit error probability P_b and is given by

$$P_{cw} < 2^{-nE(R)}, \tag{1.16}$$

where $E(R)$ is the so-called *Gallager error exponent*, expressed as

$$E(R) = \max_{\{\Pr(x)\}} \max_{0 \le \rho \le 1} [E_0(\rho, \{\Pr(x)\}) - \rho R],$$

with

$$E_0(\rho, \{\Pr(x)\}) = -\log_2 \left[\int_{-\infty}^{\infty} \left[\sum_{x \in \{\pm 1\}} \Pr(x) p(y|x)^{1/(1+\rho)} \right]^{1+\rho} dy \right].$$

Because the BI-AWGN channel is symmetric, the maximizing distribution on the channel input is $\Pr(x = +1) = \Pr(x = -1) = 1/2$. Also, for the BI-AWGN channel, $p(y|x) = [1/(\sqrt{2\pi}\sigma)] \exp\left[-(y-x)/\left(2\sigma^2\right)\right]$ so that $E_0(\rho, \{\Pr(x)\})$ becomes, after some manipulation,

$$E_0(\rho, \{\Pr(x)\}) = -\log_2 \left[\int_{-\infty}^{\infty} \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{y^2+1}{2\sigma^2} \right) \left[\cosh\left(\frac{y}{\sigma^2(1+\rho)} \right) \right]^{1+\rho} dy \right].$$

It can be shown that $E(R) > 0$ if and only if $R < C$, thus proving from (1.16) that, as $n \to \infty$, arbitrarily reliable communication is possible provided that $R < C$.

1.5.2 Coding Limits for M-ary-Input Memoryless Channels

1.5.2.1 The Unconstrained-Input AWGN Channel

Consider the discrete-time channel model,

$$y_\ell = x_\ell + z_\ell, \tag{1.17}$$

where x_ℓ is a real-valued signal whose power is constrained as $\mathbb{E}\left[x_\ell^2\right] \le P$ and z_ℓ is a real-valued additive white-Gaussian-noise sample with variance σ^2, i.e., $z_\ell \sim \mathcal{N}\left(0, \sigma^2\right)$. For this model, since the channel symbol x_ℓ is not binary, the code rate R would be defined in terms of information bits per channel symbol and it is not constrained to be less than unity. It is still assumed that the encoder input is binary, but the encoder output selects a sequence of nonbinary symbols x_ℓ that are transmitted over the AWGN channel. As an example, if x_ℓ draws from an alphabet of 1024 symbols, then code rates of up to 10 bits per channel symbol are possible. The capacity of the channel in (1.17) is given by

$$C_{\text{Shannon}} = \frac{1}{2} \log_2 \left(1 + \frac{P}{\sigma^2} \right) \text{ bits/channel symbol}, \tag{1.18}$$

and this expression is often called the *Shannon capacity*, since it is the upper limit among all signal sets (alphabets) from which x_ℓ may draw its values. As shown in one of the problems, the capacity is achieved when x_ℓ is drawn from a Gaussian distribution, $\mathcal{N}(0, P)$, which has an uncountably infinite alphabet size. In practice, it is possible to closely approach C_{Shannon} with a finite (but large) alphabet that is Gaussian-like.

As before, reliable communication is possible on this channel only if $R < C$. Note that, for large values of SNR, P/σ^2, the capacity grows logarithmically. Thus, each doubling of SNR (increase by 3 dB) corresponds to a capacity increase of about 1 bit/channel symbol. We point out also that, because we assume a real-valued model, the units of this capacity formula might also be bits/channel symbol/dimension. For a complex-valued model, which requires two dimensions, the formula would increase by a factor of two, much as the throughput of quaternary phase-shift keying (QPSK) is double that of binary phase-shift keying (BPSK) for the same channel symbol rate. In general, for d dimensions, the formula in (1.18) would increase by a factor of d.

Frequently, one is interested in a channel capacity in units of bits per second rather than bits per channel symbol. Such a channel capacity is easily obtainable via multiplication of C_{Shannon} in (1.18) by the channel symbol rate R_s (symbols per second):

$$C'_{\text{Shannon}} = R_s C_{\text{Shannon}} = \frac{R_s}{2} \log_2\left(1 + \frac{P}{\sigma^2}\right) \text{ bits/second.}$$

This formula still corresponds to the discrete-time model (1.17), but it leads us to the capacity formula for the continuous-time (baseband) AWGN channel band-limited to W Hz, with power spectral density $N_0/2$. Recall from Nyquist that the maximum distortion-free symbol rate in a bandwidth W is $R_{s,\max} = 2W$. Substitution of this into the above equation gives

$$\begin{aligned} C'_{\text{Shannon}} &= W \log_2\left(1 + \frac{P}{\sigma^2}\right) \text{ bits/second} \\ &= W \log_2\left(1 + \frac{R_b E_b}{W N_0}\right) \text{ bits/second,} \end{aligned} \tag{1.19}$$

where in the second line we have used $P = R_b E_b$ and $\sigma^2 = W N_0$, where R_b is the information bit rate in bits per second and E_b is the average energy per information bit. Note that reliable communication is possible provided that $R_b < C'_{\text{Shannon}}$.

1.5.2.2 *M-ary AWGN Channel*

Consider an M-ary signal set $\{s_m\}_{m=1}^M$ existing in two dimensions so that each s_m signal is representable as a complex number. The capacity formula for two-dimensional M-ary modulation in AWGN can be written as a straightforward generalization of the binary case. (We consider one-dimensional M-ary modulation

in AWGN in the problems.) Thus, we begin with $C_{M\text{-ary}} = h(Y) - h(Y|X) = h(Y) - h(Z)$, from which we may write

$$C_{M\text{-ary}} = -\int_{-\infty}^{+\infty} p(y)\log_2(p(y))dy - \log_2(2\pi e\sigma^2), \tag{1.20}$$

where $h(Z) = \log_2(2\pi e\sigma^2)$ because the noise is now two-dimensional (or complex) with a variance of $2\sigma^2 = N_0$, or a variance of $\sigma^2 = N_0/2$ in each dimension. In this expression, $p(y)$ is determined via

$$p(y) = \frac{1}{M}\sum_{m=1}^{M} p(y|s_m),$$

where $p(y|s_m)$ is the complex Gaussian pdf with mean s_m and variance $2\sigma^2 = N_0$. The first term in (1.20) may be computed as in the binary case using (1.14). Note that an M-ary symbol is transmitted during each symbol epoch and so $C_{M\text{-ary}}$ has the units information bits per channel symbol. Because each M-ary symbol conveys $\log_2(M)$ code bits, $C_{M\text{-ary}}$ may be converted to information bits per code bit by dividing it by $\log_2(M)$.

For QPSK, 8-PSK, 16-PSK, and 16-QAM, $C_{M\text{-ary}}$ is plotted in Figure 1.9 against E_b/N_0, where E_b is related to the average signal energy $E_s = \mathbb{E}[|s_m|^2]$ via the channel rate (channel capacity) as $E_b = E_s/C_{M\text{-ary}}$. Also included in Figure 1.9 is the capacity of the unconstrained-input (two-dimensional) AWGN channel, $C_{\text{Shannon}} = \log_2(1 + E_s/N_0)$, described in the previous subsection. Recall that the Shannon capacity gives the upper limit over all signaling schemes and, hence, its curve lies above all of the other curves. Observe, however, that, at the lower SNR

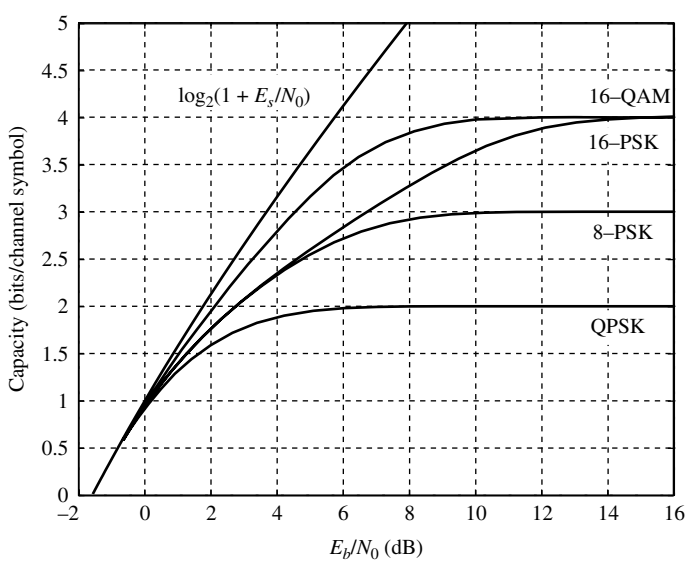


Figure 1.9 Capacity versus E_b/N_0 curves for selected modulation schemes.

values, the capacity curves for the standard modulation schemes are very close to that of the Shannon capacity curve. Next, observe that, again, reliable communication is not possible for $E_b/N_0 < -1.59$ dB. Finally, observe that 16-PSK capacity is inferior to that of 16-QAM over the large range of SNR values that might be used in practice. This is understandable given the smaller minimum inter-signal distance that exists in the 16-PSK signal set (for the same E_s).

1.5.2.3 *M-ary Symmetric Channel*

It was seen in Section 1.5.1.3 that a natural path to the binary-symmetric channel was via hard decisions at the output of a binary-input AWGN channel. Analogously, one may arrive at an M-ary-symmetric channel via hard decisions at the output of an M-ary AWGN channel. The M-ary-symmetric channel is an M-ary-input/M-ary-output channel with transition probabilities $\{P(y|x)\}$ that depend on the geometry of the M-ary signal constellation and the statistics of the noise. Because the channel is symmetric, the capacity is equal to the mutual information of the channel with uniform input probabilities, $\Pr\{x\} = 1/M$. Thus, the capacity of the M-ary-symmetric channel (MSC) is given by

$$C_{\text{MSC}} = \frac{1}{M} \sum_{x=0}^{M-1} \sum_{y=0}^{M-1} P(y|x) P \log_2 \left(\frac{P(y|x)}{P(y)} \right), \tag{1.21}$$

where

$$P(y) = \frac{1}{M} \sum_{x=0}^{M-1} P(y|x).$$

1.5.3 Coding Limits for Channels with Memory

A channel with memory is one whose output depends not only on the input, but also on previous inputs. These previous inputs can typically be encapsulated by a *channel state*, so that the channel output depends on the input and the state. Such a channel with memory is called a *finite-state channel*. There exists a number of important finite-state channel models and we introduce capacity results for two of these, the Gilbert–Elliott channel and the constrained-input intersymbol-interference (ISI) channel. A commonly discussed capacity result is the important water-filling capacity result for unconstrained-input ISI channels [3, 4], but we do not discuss this here.

1.5.3.1 *Gilbert–Elliott Channels*

We define the *Gilbert–Elliott (GE) channel* model in its most general form as follows. A GE channel is a multi-state channel for which a different channel model exists in each state. A common example is a two-state channel that possesses both a "good" state G and a "bad" state B, where a low-error-rate BSC resides in the good state and a high-error-rate BSC resides in the bad state. Such a channel is

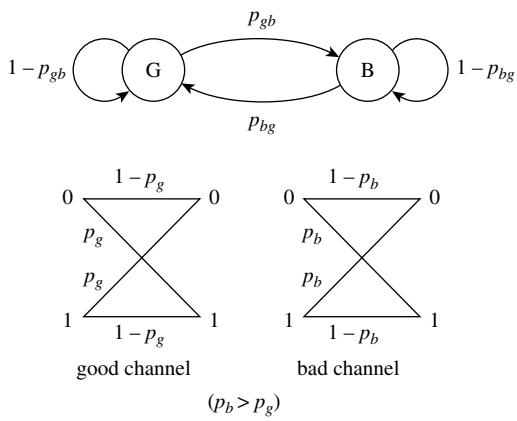


Figure 1.10 A diagram of the two-state Gilbert–Elliott channel with a BSC at each state.

depicted in Figure 1.10. Note that this model is useful for modeling binary channels subject to error bursts for which p_b is close to 0.5 and $p_{gb} << p_{bg}$. Another common GE example is a two-state (good/bad) channel for which a BEC with ε close to unity exists in the bad state. This models the ability to (coarsely) detect errors in the bad state (e.g., low signal-level detection) and assign "erasure flags" to all of the bits transmitted during the high-error-rate (bad) state. The capacity of an S-state GE channel is given by

$$C_{\text{GE}} = \sum_{s=1}^{S} P_s C_s, \tag{1.22}$$

where P_s is the probability of being in state s and C_s is the capacity of the channel in state s.

1.5.3.2 *Computing Achievable Rates for ISI Channels*

The intersymbol-interference channel model we consider has the model

$$y_\ell = f_\ell * x_\ell + z_\ell, \tag{1.23}$$

where z_ℓ is an AWGN noise process with variance $\sigma^2 = N_0/2$, x_ℓ is the channel input process taking values from a finite alphabet, f_ℓ is the discrete-time channel (finite-length) impulse response, and $*$ represents discrete-time convolution. As an example, suppose that the channel has impulse response $[f_0, f_1, f_2]$. Then (1.23) may be written as

$$y_\ell = f_0 x_\ell + f_1 x_{\ell-1} + f_2 x_{\ell-2} + z_\ell.$$

Observe from this that the channel output y_ℓ is a function not only of the "current" input x_ℓ, but also of the previous two inputs, $x_{\ell-1}$ and $x_{\ell-2}$. Hence, this is a finite-state channel. It is useful to represent an ISI channel by a polynomial

that is the "D-transform" of its impulse response, where D is the discrete-time delay operator. Thus, our example ISI channel is representable by the polynomial $f(D) = f_0 + f_1 D + f_2 D^2$. In fact, (1.23) may be rewritten as

$$y(D) = f(D)x(D) + z(D),$$

where the various D-transforms are obviously defined.

Prior to 2001, it was not known how to compute the capacity of ISI channels whose input alphabets were constrained to take values from a finite set. Arnold and Loeliger [15] (and, later, Pfister and Siegel [16]) presented an algorithm for tightly estimating the mutual information of binary-input ISI channels with AWGN assuming equiprobable, independent inputs. Techniques for tightly lower bounding the capacity of binary-input ISI channels are also presented in [15], but capacity is achievable in general only through the use of nonlinear Markovian codes. In this book and in most applications, the interest is in linear codes whose code bitstreams are well approximated by an independent and identically distributed (i.i.d.) equiprobable binary stream. In this case, capacity calculations are unnecessary and, in fact, irrelevant. Thus, we present the algorithm of [15] for estimating the mutual information of an ISI channel with equiprobable, independent, binary inputs.

As before, we let X represent the channel input and Y represent the AWGN channel output. We will also let $\mathbf{X}_L$ and $\mathbf{Y}_L$ represent the sequences of L variables, $(X_1, X_2, \ldots, X_L)$ and $(Y_1, Y_2, \ldots, Y_L)$, respectively. We are then interested in the mutual information $I(X;Y)$ between input and output corresponding to independent and equally likely $x_\ell \in \{\pm 1\}$. This is often called the independent, uniformly distributed (i.u.d.) capacity, denoted by C_{iud}.

As done earlier, we write the mutual information $I(X;Y)$ as

$$I(X;Y) = h(Y) - h(Y|X) = C_{\text{iud}},$$

where $h(Y)$ is the differential entropy of the process Y and $h(Y|X)$ is the conditional entropy of Y given X. It is clear that $h(Y|X) = h(Z)$, where Z represents the noise process, so that $h(Y|X) = \frac{1}{2}\log_2\big(2\pi e\sigma^2\big)$, where $\sigma^2 = N_0/2$ is the variance of the AWGN noise process. Thus, to determine $I(X;Y)$, one need only determine $h(Y)$ under the assumption of i.u.d. channel inputs. To do so, Arnold and Loeliger [15] utilize the Shannon–McMillan–Breiman theorem [3] which asserts that

$$\lim_{L\to\infty}\left(-\frac{1}{L}\log_2(p(\mathbf{Y}_L))\right) = h(Y)$$

with probability one. Thus, one need only compute $p(\mathbf{Y}_L)$ for a large value of L to estimate $h(Y)$. As pointed out in [15], the BCJR algorithm (see Chapter 4) effectively does this since it computes the "forward" metrics $\alpha_L(s) \triangleq p(s_k = s, \mathbf{Y}_L)$ so that $p(\mathbf{Y}_L)$ may be obtained by computing the marginal probability $\sum_s \alpha_L(s) = \sum_s p(s_k = s, \mathbf{Y}_L)$. The channel input $\mathbf{X}_L$ corresponding to the output $\mathbf{Y}_L$ may be any realization of an i.u.d. binary random process. This approach is clearly generalizable to M-ary-input ISI channels.

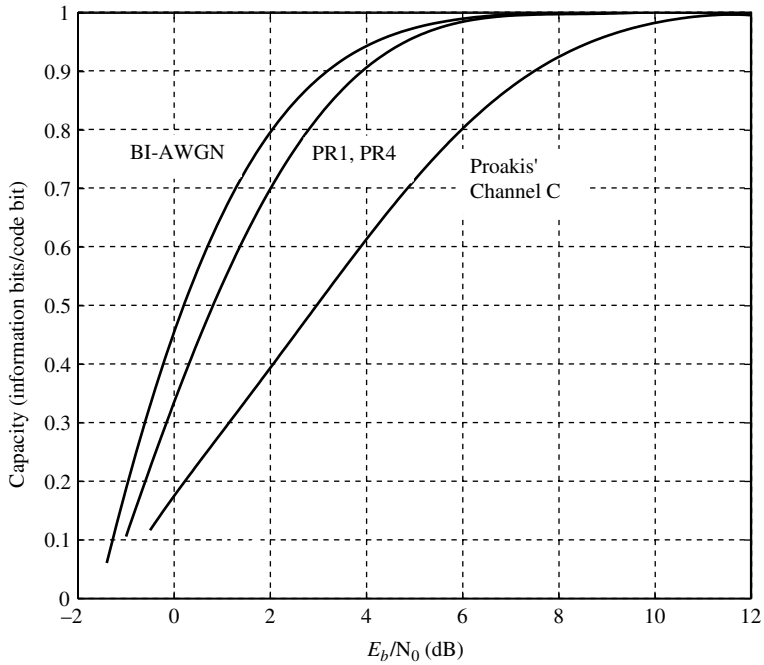


Figure 1.11 Capacity plots for the BI-AWGN channel, the PR1 and PR4 channels, and Proakis' Channel C.

As examples of applications of this approach, Figure 1.11 compares the capacity of the BI-AWGN channel with C_{iud} for (1) partial-response channels of the form $1 \pm D^{\nu}$, $\nu = 1, 2, \ldots$, which includes the Class I partial response (PR1) polynomial $1 + D$ and the Class IV partial response (PR4) polynomial $1 - D^2$; and (2) Proakis' Channel C (see Figure 10.2-5(c) in [8]), which is represented by the polynomial $0.227 + 0.46D + 0.688D^2 + 0.46D^3 + 0.227D^4$. Proakis' Channel C is characterized by a spectral null near the middle of the band, so it is not surprising that its (C_{iud}) capacity is significantly less than that of the other channels. The PR1 channel has a spectral null at DC and the PR4 channel has spectral nulls both at DC and at the Nyquist frequency (half of the channel bit rate). We observe that these two partial-response channels have the same C_{iud} curve (which is also the curve for all $1 \pm D^{\nu}$ PR channels), and that there is a rate loss relative to the BI-AWGN channel.

Problems

1.1 A single error has been added (modulo 2) to a transmitted (7,4) Hamming codeword, resulting in the received word $\mathbf{r} = (1011\ 100)$. Using the decoding algorithm described in the chapter, find the error.

1.2 For linear codes, the list $S_d(\mathbf{c})$ of Hamming distances from codeword $\mathbf{c}$ to all of the other codewords is the same, no matter what the choice of $\mathbf{c}$ is. (See Chapter 3.) $S_d(\mathbf{c})$ is called the *conditional distance spectrum* for the codeword $\mathbf{c}$. Find $S_d(0000\ 000)$ and $S_d(0111\ 001)$ and check that they are the same.

1.3 Using (1.6), derive (1.7) and (1.8). Note that the capacity in each case is given by (1.6) because these channels are symmetric.

1.4 Consider a hybrid of the BSC and the BEC in which there are two input symbols (0 and 1) and three output symbols (0, e, and 1). Further, the probability of an error is ε and the probability of an erasure is p. Derive a formula for the capacity of this channel. Because the channel is symmetric, C is given by (1.6).

1.5 Derive Equation (1.10). You may assume that the expression being maximized in (1.9) is "convex cap." Now reproduce Figure 1.6.

1.6 Consider a binary-input AWGN channel in which the channel SNR is $E_s/N_0 = -2.823$ dB when the channel bit rate is $R_s = 10$ Mbits/second. What is the maximum information rate R_b possible for reliable communication? HINT: $R_b = RR_s$ and $E_s = RE_b$, where R is the code rate in information bits per channel bit. The R of interest is one of the rates listed in Table 1.1.

1.7 Consider binary signaling over an AWGN channel bandlimited to $W = 5000$ Hz. Suppose that the power at the input to the receiver is $P = 1\,\mu$W and the power spectral density of the AWGN is $N_0/2 = 5 \times 10^{-11}$ W/Hz. Assuming we use the maximum possible Nyquist signaling rate R_s over the channel, find the maximum information rate R_b possible for reliable communication through this channel. HINT: To start, you will have to find E_s/N_0 and then compute $C_{\text{BI-AWGN}}$ for the E_s/N_0 that you find. You will also need $R_b = RR_s$, $E_s = RE_b$, and $E_s = P/R_s$, where R is the code rate in information bits per channel bit.

1.8 Write a computer program that estimates the capacity of the BI-AWGN channel and use it to reproduce Figure 1.7.

1.9 Write a computer program that reproduces Figure 1.8.

1.10 Consider a bandlimited AWGN channel with bandwidth $W = 10\,000$ Hz and SNR $E_b/N_0 = 10$ dB. Assuming an unconstrained signaling alphabet, is reliable communication possible at $R_b = 50\,000$ bits/second? What about at $60\,000$ bits/second?

1.11 From Equation (1.19), we may write $\xi < \log_2(1 + \xi E_b/N_0)$, where $\xi = R_b/W$ is the spectral efficiency, a measure of the information rate per unit bandwidth in units of bps/Hz. Show that, when $\xi = 1$ bps/Hz, we must have $E_b/N_0 > 0$ dB; when $\xi = 4$ bps/Hz, we must have $E_b/N_0 > 5.74$ dB; and as $\xi \to 0$ bps/Hz, we must have $E_b/N_0 \to -1.6$ dB.

1.12 (a) Let G be a Gaussian random variable, $G \sim \mathcal{N}(\mu, \sigma^2)$. Show that its differential entropy is given by $h(G) \triangleq -E[\log_2(p(G))] = \frac{1}{2}\log_2\big(2\pi e\sigma^2\big)$ by performing the integral implied by $-E[\log_2(p(G))]$. (b) We are given the channel model $Y = X + Z$, where $X \sim \mathcal{N}(0, P)$ and $Z \sim \mathcal{N}(0, \sigma^2)$ are independent. Show that the mutual information is given by $I(X;Y) = h(Y) - h(Y|X) = h(Y) - h(N) = \frac{1}{2}\log_2\big(1 + P/\sigma^2\big) = C$. Thus, a Gaussian-distributed signal achieves capacity on the arbitrary-input AWGN channel.

1.13 Let G be a zero-mean complex Gaussian random variable with variance σ^2. Show that its differential entropy is given by $h(G) \triangleq -E[\log_2(p(G))] = \log_2\big(2\pi e\sigma^2\big)$ by performing the integral implied by $-E[\log_2(p(G))]$.

1.14 Write a computer program to reproduce Figure 1.9.

1.15 Derive capacity formulas for one-dimensional M-ary modulation and plot the capacity of M-ary pulse amplitude modulation versus E_b/N_0 for $M = 2, 4, 8$, and 16. Add the appropriate Shannon capacity curve to your plot for comparison.

1.16 As mentioned in the chapter, the mutual information may alternatively be written as $I(X;Y) = H(X) - H(X|Y)$. Use this expression to obtain a formula for the capacity of the M-ary symmetric channel that is an alternative to Equation (1.21).

1.17 Consider an independent *Rayleigh fading channel* with channel output $y_k = a_k x_k + n_k$, where $n_k \sim \mathcal{N}(0, N_0/2)$ is an AWGN sample, $x_k \in \{\pm 1\}$ equally likely, and a_k is a Rayleigh random variable with expectation $\mathbb{E}\big\{a_k^2\big\} = 1$. By generalizing the notion presented in (1.22), derive a capacity formula for this channel and plot its capacity against $\bar{E}_b/N_0$, where $\bar{E}_b$ is the average information bit energy.

1.18 Write a computer program to reproduce Figure 1.11.

References

[1] C. Shannon, "A mathematical theory of communication," *Bell System Tech. J.*, pp. 379–423 (Part I), pp. 623–656 (Part II), July 1948.

[2] R. Hamming, "Error detecting and correcting codes," *Bell System Tech. J.*, pp. 147–160, April 1950.

[3] T. Cover and J. Thomas, *Elements of Information Theory*, 2nd edn., New York, Wiley, 2006.

[4] R. Gallager, *Information Theory and Reliable Communication*, New York, Wiley, 1968.

[5] R. McEliece, *The Theory of Information and Coding: Student Edition*, Cambridge, Cambridge University Press, 2004.

[6] R. Blahut, *Principles and Practice of Information Theory*, Reading, MA, Addison-Wesley, 1987.

[7] A. Viterbi and J. Omura, *Principles of Digital Communication and Coding*, New York, McGraw-Hill, 1979.

[8] J. Proakis, *Digital Communications*, 4th edn., New York, McGraw-Hill, 2000.

[9] S. Wilson, *Digital Modulation and Coding*, Englewood Cliffs, NJ, Prentice-Hall, 1996.

[10] S. Golomb, "The limiting behavior of the Z-channel," *IEEE Trans. Information Theory*, vol. 26, no. 5, p. 372, May 1980.

[11] R. J. McEliece, "Are turbo-like codes effective on non-standard channels?," *IEEE Information Theory Soc. Newslett.* vol. 51, no. 4, pp. 1–8, December 2001.

[12] H. Nickl, J. Hagenauer, and F. Burkert, "Approaching Shannon's capacity limit by 0.27 dB using simple Hamming codes," *IEEE Communications Lett.*, vol. 2, no. 7, pp. 130–132, September 1997.

[13] B. Frey, *Graphical Models for Machine Learning and Digital Communication*, Cambridge, MA, MIT Press, 1998.

[14] G. Wiechman and I. Sason, "An improved sphere-packing bound for finite-length codes on symmetric memoryless channels," *IEEE Trans. Information Theory*, vol. 54, no. 5, pp. 1962–1990, May 2008.

[15] D. Arnold and H.-A. Loeliger, "On the information rate of binary-input channels with memory," *2001 IEEE Int. Conf. on Communications*, 2001, pp. 2692–2695.

[16] H. D. Pfister and P. H. Siegel, "On the achievable information rates of finite-state ISI channels," *2001 IEEE GlobeCom Conf.*, 2001, pp. 2992–2996.

[17] A. Dholakia, E. Eleftheriou, and T. Mittelholzer, "On iterative decoding for magnetic recording channels," *Proc. 2nd Int. Symp. on Turbo Codes*, Brest, September 2000, pp. 219–226.

2 Finite Fields, Vector Spaces, Finite Geometries, and Graphs

This chapter presents some important elements of modern algebra and combinatorial mathematics, namely, finite fields, vector spaces, finite geometries, and graphs, that are needed in the presentation of the fundamentals of classical channel codes and various constructions of modern channel codes in the following chapters. The treatment of these mathematical elements is by no means rigorous and coverage is kept at an elementary level. There are many good text books on modern algebra, combinatorial mathematics, and graph theory that provide rigorous treatment and in-depth coverage of finite fields, vector spaces, finite geometries, and graphs. Some of these texts are listed at the end of this chapter.

2.1 Sets and Binary Operations

A set is a *collection* of certain objects, commonly called the *elements* of the set. A set and its elements will often be denoted by letters of an alphabet. Commonly, a set is represented by a capital letter and its elements are represented by lower-case letters (with or without subscripts). For example, $X = \{x_1, x_2, x_3, x_4, x_5\}$ is a set with five elements, x_1, x_2, x_3, x_4, and x_5. A set S with a finite number of elements is called a *finite set*; otherwise, it is called an *infinite set*. In error-control coding theory, we mostly deal with finite sets. The number of elements in a set S is called the *cardinality* of the set and is denoted by $|S|$. The cardinality of the above set X is five. A part of a set S is of course itself a set. Such a part of S is called a *subset* of S. A subset of S with cardinality smaller than that of S is called a *proper subset* of S. For example $Y = \{x_1, x_3, x_4\}$ is a proper subset of the above set X.

A *binary operation* on a set S is a *rule* that assigns to any pair of elements in S, taken in a *definite order*, another element in the same set. For the present, we denote such a binary operation by $*$. Thus, if a and b are two elements of the set S, then the $*$ operation assigns to the pair (a, b) an element of S, namely $a * b$. The operation also assigns to the pair (b, a) an element in S, namely $b * a$. Note that $a * b$ and $b * a$ do not necessarily give the same element in S. When a binary operation $*$ is defined on S, we say that S is *closed* under $*$. For example, let S be the set of all integers and let the binary operation on S be real (or ordinary) addition $+$. We all know that, for any two integers i and j in S, $i + j$ is another integer in S. Hence, the set of integers is closed under real

(or ordinary) addition. Another well-known binary operation on the set of integers is real (or ordinary) multiplication, denoted by "$\cdot$."

The adjective *binary* is used because the operation produces from a pair of elements in S (taken in a *specific order*) an element in S. If we take elements a, b, and c of S (not necessarily different), there are various ways in which we can combine them by the operation $*$. For example, we first combine a and b with $*$ to obtain an element $a * b$ of S and then combine $a * b$ and c with $*$ to produce an element $(a * b) * c$ of S. We can also combine the pair a and $b * c$ to obtain an element $a * (b * c)$ in S. The parentheses are necessary to indicate just how the elements are to be grouped into pairs for combination by the operation $*$.

Definition 2.1. Let S be a set with a binary operation $*$. The operation is said to be *associative* if, given any elements a, b, and c in S, the following equality holds:

$$(a * b) * c = a * (b * c). \tag{2.1}$$

The equality sign "=" is taken to mean "the same as." We also say that the binary operation $*$ satisfies the *associative law*, if the equality (2.1) holds.

The associative law simply implies that, when we combine elements of S with repeated applications of the binary operation $*$, the result does not depend upon the *grouping* of the elements involved. Hence, we can write $(a * b) * c = a * (b * c) = a * b * c$. The associative law is a very important law, and most of the binary operations which we shall encounter will satisfy it. Addition $+$ and multiplication $\cdot$ on the set of integers (or real numbers) are associative binary operations.

Definition 2.2. Let S be a set with a binary operation $*$. The operation is said to be *commutative* if, given any two elements, a and b, in S, the following equality holds:

$$a * b = b * a. \tag{2.2}$$

We also say that $*$ satisfies the *commutative law*, if (2.2) holds.

Note that the definition of the commutative law is independent of the associative law. The commutative law simply implies that, when we combine two elements of S by the $*$ operation, the result does not depend on the order of combination of the two elements. Addition $+$ and multiplication $\cdot$ on the set of integers (or real numbers) are commutative binary operations. Therefore, addition and multiplication on the set of integers (or real numbers) are both associative and commutative.

Let S be a set with an associative and commutative binary operation $*$. Then, if any combination of elements of S is formed by means of repeated applications of $*$, the result depends neither on the grouping of the elements involved nor on their order. For example, let a, b, and c be elements of S. Then

$$\begin{aligned}(a * b) * c &= a * (b * c) = a * (c * b) = (a * c) * b \\ &= c * (a * b) = (c * a) * b = (c * b) * a.\end{aligned}$$

Given two sets, X and Y, we can form a set $X \cup Y$ that consists of the elements in either X or Y, or both. This set $X \cup Y$ is called the *union* of X and Y. We can also form a set $X \cap Y$ that consists of the set of all elements in both X and Y. The set $X \cap Y$ is called the *intersection* of X and Y. A set with no element is called an *empty set*, denoted by $\emptyset$. Two sets are said to be *disjoint* if their intersection is empty, i.e., they have no element in common. If Y is a subset of X, the set that contains the elements in X but not in Y is called the *difference* between X and Y and is denoted by $X \backslash Y$.

2.2 Groups

In this section, we introduce a simple algebraic system. By an algebraic system, we mean a set with one or more binary operations defined on it.

2.2.1 Basic Concepts of Groups

A group is an algebraic system with one binary operation defined on it.

Definition 2.3. A *group* is a set G together with a binary operation $*$ defined on G such that the following axioms (conditions) are satisfied.

1. The binary operation $*$ is associative.
2. G contains an element e such that, for any element a of G,

$$a * e = e * a = a.$$

   This element e is called an *identity element* of G with respect to the operation $*$.
3. For any element a in G, there exists an element a' in G such that

$$a * a' = a' * a = e.$$

   The element a' is called an *inverse* of a, and vice versa, with respect to the operation $*$.

A group G is said to be commutative if the binary operation $*$ defined on it is also commutative. A commutative group is also called an *abelian* group.

Theorem 2.1. The identity element e of a group G is *unique*.

Proof. Suppose both e and e' are identity elements of G. Then $e * e' = e$ and $e * e' = e'$. This implies that e and e' are identical. Therefore, the identity of a group is unique. □

Theorem 2.2. The inverse of any element in a group G is *unique*.

Proof. Let a be an element of G. Suppose a has two inverses, say a' and a''. Then

$$a'' = e * a'' = (a' * a) * a'' = a' * (a * a'') = a' * e = a'.$$

This implies that a' and a'' are identical and hence a has a unique inverse. $\square$

The set of all rational numbers with real addition $+$ forms a commutative group. We all know that real addition $+$ is both associative and commutative. The integer 0 is the identity element of the group. The rational number $-a/b$ (b is a nonzero integer) is the additive inverse of the rational number a/b and vice versa. The set of all rational numbers, *excluding* zero, forms a commutative group under real multiplication $\cdot$. The integer 1 is the identity element with respect to real multiplication, and the rational number b/a (both a and b are nonzero integers) is the inverse of the rational number a/b.

The two groups given above have infinite numbers of elements. However, groups with a finite number of elements do exist, as will be seen later in this section.

Definition 2.4. A group G is called *finite* (or *infinite*) if it contains a finite (or an infinite) number of elements. The number of elements in a finite group is called the *order* of the group.

It is clear that the order of a finite group G is simply its cardinality $|G|$. There is a convenient way of presenting a finite group G with operation $*$. A table displaying the group operation $*$ is constructed by labeling the rows and the columns of the table by the group elements. The element appearing in the row labeled by a and the column labeled by b is then taken to be $a * b$. Such a table is called a *Cayley table*.

Example 2.1. This example gives a simple group with two elements. We start with the set of two integers, $G = \{0, 1\}$. Define a binary operation, denoted by $\oplus$, on G as shown by the Cayley table below:

$\oplus$	0	1
0	0	1
1	1	0

From the table, we readily see that

$$0 \oplus 0 = 0, \quad 0 \oplus 1 = 1, \quad 1 \oplus 0 = 1, \quad 1 \oplus 1 = 0.$$

It is clear that the set $G = \{0, 1\}$ is closed under the operation $\oplus$ and $\oplus$ is commutative. To prove that $\oplus$ is associative, we simply determine $(a \oplus b) \oplus c$ and $a \oplus (b \oplus c)$ for eight possible combinations of a, b, and c with a, b, and c in $G = \{0, 1\}$, and show that

$$(a \oplus b) \oplus c = a \oplus (b \oplus c),$$

for each combination of a, b, and c. This kind of proof is known as *perfect induction*. From the table, we see that 0 is the identity element, the inverse of 0 is itself, and the inverse of 1 is also itself. Thus, $G = \{0, 1\}$ with operation $\oplus$ is a commutative group of order 2. The binary operation $\oplus$ defined on the set $G = \{0, 1\}$ by the above Cayley table is called *modulo-2 addition*.

2.2.2 Finite Groups

Finite groups are of practical importance in many areas, especially in the area of coding theory. In the following, we will present two classes of finite groups, one with a binary operation very similar to real addition and the other with a binary operation very similar to real multiplication.

Let m be a positive integer. Consider the set of integers $G = \{0, 1, \ldots, m-1\}$. Let $+$ denote real addition. Define a binary operation, denoted by $\boxplus$. For any two integers i and j in G,

$$i \boxplus j = r, \tag{2.3}$$

where r is the remainder resulting from dividing the sum $i + j$ by m. By Euclid's division algorithm, r is a non-negative integer between 0 and $m - 1$, and is therefore an element in G. Thus, G is closed under the binary operation $\boxplus$, which is called *modulo-m addition*.

The set $G = \{0, 1, \ldots, m-1\}$ with modulo-m addition forms a commutative group. To prove this, we need to show that G satisfies all the three axioms given in Definition 2.3. For any integer i in G, since $0 \leq i < m$, it follows from the definition of $\boxplus$ that $0 \boxplus i = i$ and $i \boxplus 0 = i$. Hence 0 is the identity element. For any nonzero integer i in G, $m - i$ is also in G. Since $i + (m - i) = m$ and $(m - i) + i = m$, it follows from the definition of $\boxplus$ that $i \boxplus (m - i) = 0$ and $(m - i) \boxplus i = 0$. Therefore, $m - i$ is the inverse of i with respect to $\boxplus$, and vice versa. Note that the inverse of 0 is itself. So every element in G has an inverse with respect to $\boxplus$. For any two integers i and j in G, since real addition $+$ is commutative, $i + j = j + i$. On dividing $i + j$ and $j + i$ by m, both give the same remainder r with r in G. Hence $i \boxplus j = j \boxplus i$ and $\boxplus$ is commutative. Next, we prove that $\boxplus$ is also associative. The proof is based on the fact that real addition is associative. For i, j, and k in G,

$$i + j + k = (i + j) + k = i + (j + k).$$

On dividing the sum $i + j + k$ by m, we obtain

$$i + j + k = q \cdot m + r,$$

where q and r are the quotient and remainder, respectively, and $0 \leq r < m$. On dividing $i + j$ by m, we have

$$i + j = q_1 \cdot m + r_1, \tag{2.4}$$

where $0 \leq r_1 < m$. It follows from the definition of $\boxplus$ that $i \boxplus j = r_1$. Dividing $r_1 + k$ by m results in

$$r_1 + k = q_2 \cdot m + r_2, \tag{2.5}$$

where $0 \leq r_2 < m$. Hence, $r_1 \boxplus k = r_2$ and

$$(i \boxplus j) \boxplus k = r_2.$$

Table 2.1. The additive group with modulo-7 addition

$\boxplus$	0	1	2	3	4	5	6
0	0	1	2	3	4	5	6
1	1	2	3	4	5	6	0
2	2	3	4	5	6	0	1
3	3	4	5	6	0	1	2
4	4	5	6	0	1	2	3
5	5	6	0	1	2	3	4
6	6	0	1	2	3	4	5

On combining (2.4) and (2.5), we have $i+j+k=(q_1+q_2)\cdot m+r_2$. This implies that r_2 is also the remainder when we divide $i+j+k$ by m. Since the remainder is unique when we divide an integer by another integer, we must have $r_2=r$. Consequently, we have

$$(i \boxplus j) \boxplus k = r.$$

Similarly, we can prove that

$$i \boxplus (j \boxplus k) = r.$$

Therefore, $(i \boxplus j) \boxplus k = i \boxplus (j \boxplus k)$ and the binary operation $\boxplus$ is associative. This concludes our proof that the set $G=\{0,1,\ldots,m-1\}$ with modulo-m addition $\boxplus$ forms a commutative group of order m. For $m=2$, $G=\{0,1\}$ with modulo-2 addition gives the binary group constructed in Example 2.1. The above construction gives an infinite class of finite groups under modulo-m addition with $m=2,3,4,\ldots$. The groups in this class are called *additive groups*.

Example 2.2. Let $m=7$. Table 2.1 displays the additive group $G=\{0,1,2,3,4,5,6\}$ under modulo-7 addition.

Next we present a class of finite groups under a binary operation similar to real multiplication. Let p be a prime. Consider the set of $p-1$ positive integers less than p, $G=\{1,2,\ldots,p-1\}$. Let "$\cdot$" denote real multiplication. Every integer i in G is relatively prime to p. Define a binary operation, denoted by $\boxdot$, on G as follows. For any two integers, i and j, in G,

$$i \boxdot j = r, \tag{2.6}$$

where r is the remainder resulting from dividing $i\cdot j$ by p. Since i and j are both relatively prime to p, $i\cdot j$ is also relatively prime to p. Hence, $i\cdot j$ is not divisible by p and r cannot be zero. As a result, $1\le r<p$ and r is an element in G. Therefore, the set $G=\{1,2,\ldots,p-1\}$ is closed under the operation $\boxdot$, which is called *modulo-p multiplication*. G with modulo-p multiplication forms a commutative group of order $p-1$. It is easy to prove that modulo-p multiplication

Table 2.2. The multiplicative group with modulo-7 multiplication

$\boxdot$	1	2	3	4	5	6
1	1	2	3	4	5	6
2	2	4	6	1	3	5
3	3	6	2	5	1	4
4	4	1	5	2	6	3
5	5	3	1	6	4	2
6	6	5	4	3	2	1

$\boxdot$ is both associative and commutative, and that 1 is the identity element of G. The only thing left to be proved is that every element in G has an inverse with respect to $\boxdot$. Let i be an integer in G. Since i and p are relatively prime, there exist two integers a and b such that

$$a \cdot i + b \cdot p = 1, \tag{2.7}$$

and a and p are relatively prime (Euclid's theorem). Rearrange (2.7) as follows:

$$a \cdot i = -b \cdot p + 1. \tag{2.8}$$

This says that, when we divide $a \cdot i$ (or $i \cdot a$) by p, the remainder is 1. If $0 < a < p$, a is an integer in G. Then, it follows from the definition of $\boxdot$ and (2.8) that

$$a \boxdot i = i \boxdot a = 1.$$

Therefore, a is the inverse of i with respect to operation $\boxdot$, and vice versa. If a is not an integer in G, we divide a by p, which gives

$$a = q \cdot p + r. \tag{2.9}$$

Since a and p are relatively prime, r must be an integer between 1 and $p - 1$ and hence is an element in G. On combining (2.8) and (2.9), we obtain

$$r \cdot i = -(b + q \cdot i) \cdot p + 1. \tag{2.10}$$

Since $r \cdot i = i \cdot r$, it follows from the definition of modulo-p multiplication $\boxdot$ and (2.10) that $r \boxdot i = i \boxdot r = 1$ and r is the inverse of i with respect to $\boxdot$. Hence, every element i in G has an inverse with respect to modulo-p multiplication $\boxdot$. This completes the proof that the set $G = \{1, 2, \ldots, p - 1\}$ under modulo-p multiplication $\boxdot$ is a commutative group. The above construction gives a class of finite groups under modulo-p multiplication. The groups in this class are called *multiplicative groups*. Note that if p is not a prime, the set $G = \{1, 2, \ldots, p - 1\}$ does not form a group under modulo-p multiplication (proof of this is left as an exercise).

Example 2.3. Let $p = 7$, which is a prime. Table 2.2 gives the multiplicative group $G = \{1, 2, 3, 4, 5, 6\}$ under modulo-7 multiplication.

Table 2.3. The additive group under modulo-8 addition

$\boxplus$	0	1	2	3	4	5	6	7
0	0	1	2	3	4	5	6	7
1	1	2	3	4	5	6	7	0
2	2	3	4	5	6	7	0	1
3	3	4	5	6	7	0	1	2
4	4	5	6	7	0	1	2	3
5	5	6	7	0	1	2	3	4
6	6	7	0	1	2	3	4	5
7	7	0	1	2	3	4	5	6

Consider the multiplicative group $G = \{1, 2, \ldots, p-1\}$ under the modulo-p multiplication. Let a be an element in G. We define powers of a as follows:

$$a^1 = a, \quad a^2 = a \boxdot a, \quad a^3 = a \boxdot a \boxdot a, \quad \ldots, \quad a^i = \underbrace{a \boxdot a \boxdot \cdots \boxdot a}_{i\,\text{factors}}, \ldots$$

Clearly, these powers of a are elements of G.

Definition 2.5. A multiplicative group G is said to be *cyclic* if there exists an element a in G such that, for any b in G, there is some integer i with $b = a^i$. Such an element a is called a *generator* of the cyclic group and we write $G = \langle a \rangle$.

A cyclic group may have more than one generator. This is illustrated by the next example.

Example 2.4. Consider the multiplicative group $G = \{1, 2, 3, 4, 5, 6\}$ under modulo-7 multiplication given in Example 2.2. Taking the powers of 3 and using Table 2.3, we obtain

$$3^1 = 3, \;\; 3^2 = 3 \boxdot 3 = 2, \;\; 3^3 = 3^2 \boxdot 3 = 2 \boxdot 3 = 6, \;\; 3^4 = 3^3 \boxdot 3 = 6 \boxdot 3 = 4,$$
$$3^5 = 3^4 \boxdot 3 = 4 \boxdot 3 = 5, \;\; 3^6 = 3^5 \boxdot 3 = 5 \boxdot 3 = 1.$$

We see that the above six powers of 3 give all the six elements of G. Hence, G is a cyclic group and 3 is a generator of G. The element of 5 is also a generator of G as shown below,

$$5^1 = 5, \;\; 5 \boxdot 5 = 4, \;\; 5^3 = 5^2 \boxdot 5 = 4 \boxdot 5 = 6, \;\; 5^4 = 5^3 \boxdot 5 = 6 \boxdot 5 = 2,$$
$$5^5 = 5^4 \boxdot 5 = 2 \boxdot 5 = 3, \;\; 5^6 = 5^5 \boxdot 5 = 3 \boxdot 5 = 1.$$

It can be proved that, for every prime p, the multiplicative group $G = \{1, 2, \ldots, p-1\}$ under modulo-p multiplication is cyclic.

2.2.3 Subgroups and Cosets

A group G contains certain subsets that form groups in their own right under the operation of G. Such subsets are called *subgroups* of G.

Table 2.4. A subgroup of the additive group under modulo-8 addition

$\boxplus$	0	2	4	6
0	0	2	4	6
2	2	4	6	0
4	4	6	0	2
6	6	0	2	4

Definition 2.6. An non-empty subset H of a group G with an operation $*$ is called a *subgroup* of G if H is itself a group with respect to the operation $*$ of G, that is, H satisfies all the axioms of a group under the same operation $*$ of G.

To determine whether a subset H of a group G with an operation $*$ is a subgroup, we need not verify all the axioms. The following axioms are sufficient.

(S1) For any two elements a and b in H, $a * b$ is also an element of H.
(S2) For any element a in H, its inverse is also in H.

The axiom (S1) implies that H is closed under the operation $*$, i.e., $*$ is a binary operation on H. Axioms (S1) and (S2) together imply that H contains the identity element e of G. H satisfies the associative law by virtue of the fact that every element of H is also in G and $*$ is an operation on H. Hence, H is a group and a subgroup of G. The subset $\{e\}$ that contains the identity element e of G alone forms a subgroup of G. G may be considered as a subgroup of itself. These two subgroups are trivial subgroups of G.

The set of all integers forms a subgroup of the group of all rational numbers under real addition $+$.

Example 2.5. Table 2.3 gives an additive group $G = \{0, 1, 2, 3, 4, 5, 6, 7\}$ under modulo-8 addition. The subset $H = \{0, 2, 4, 6\}$ forms a subgroup of G under modulo-8 addition. Table 2.4 shows the group structure of H.

Suppose that m is not a prime. Let k be a factor of m and $m = ck$. Let $G = \{0, 1, \ldots, m-1\}$ be the additive group under modulo-m addition. Then the subset $H = \{0, c, 2c, \ldots, (k-1)c\}$ of $G = \{0, 1, \ldots, m-1\}$ forms a subgroup of G under modulo-m addition. It can be easily checked that, for $1 \leq i < k$, $(k-i)c$ is the additive inverse of ic, and vice versa.

Example 2.6. Consider the multiplicative group $G = \{1, 2, 3, 4, 5, 6\}$ under modulo-7 multiplication given by Table 2.2 (Example 2.3). Table 2.5 shows that the subset $H = \{1, 2, 4\}$ of G forms a subgroup of G under modulo-7 multiplication.

Table 2.5. A subgroup of the multiplicative group $G = \{1, 2, 3, 4, 5, 6\}$ under modulo-7 multiplication

$\boxdot$	1	2	4
1	1	2	4
2	2	4	1
4	4	1	2

Definition 2.7. Let H be a subgroup of a group G with binary operation $*$. Let a be an element of G. Define two subsets of elements of G as follows: $a * H = \{a * h : h \in H\}$ and $H * a = \{h * a : h \in H\}$. These two subsets, $a * H$ and $H * a$, are called *left* and *right cosets* of H.

It is clear that, if $a = e$ is the identity element of G, then $e * H = H * e = H$, and so H is also considered as a coset of itself. If G is a commutative group, then the left coset $a * H$ and the right coset $H * a$ are identical, i.e., $a * H = H * a$ for any $a \in G$. In this text, we deal only with commutative groups. In this case, we do not differentiate between left and right cosets, and we simply call them cosets.

Example 2.7. Consider the additive group $G = \{0, 1, 2, 3, 4, 5, 6, 7\}$ under modulo-8 addition given in Example 2.5. We have shown that $H = \{0, 2, 4, 6\}$ is a subgroup of G under modulo-8 addition. The coset $3 \boxplus H$ consists of the following elements:

$$\begin{aligned} 3 \boxplus H &= \{3 \boxplus 0, 3 \boxplus 2, 3 \boxplus 4, 3 \boxplus 6\} \\ &= \{3, 5, 7, 1\}. \end{aligned}$$

We see that H and the coset $3 \boxplus H$ are disjoint, and that they together form the group $G = \{0, 1, 2, 3, 4, 5, 6, 7\}$.

Next we consider the multiplicative group $G = \{1, 2, 3, 4, 5, 6\}$ under modulo-7 multiplication given in Example 2.3 (see Table 2.2). We have shown in Example 2.6 that $H = \{1, 2, 4\}$ forms a subgroup of G under modulo-7 multiplication. The coset $3 \boxdot H$ consists of the following elements:

$$\begin{aligned} 3 \boxdot H &= \{3 \boxdot 1, 3 \boxdot 2, 3 \boxdot 4\} \\ &= \{3, 6, 5\}. \end{aligned}$$

We see that H and $3 \boxdot H$ are disjoint, and that they together form the group $G = \{1, 2, 3, 4, 5, 6\}$.

The cosets of a subgroup H of a group G have the following structural properties.

1. No two elements of a coset of H are identical.
2. No two elements from two different cosets of H are identical.
3. Every element of G appears in one and only one coset of H.

4. All distinct cosets of H are disjoint.
5. The union of all the distinct cosets of H forms the group G.

The proofs of the above structural properties are not given here, but are left as an exercise. All the distinct cosets of a subgroup H of a group G are said to form a *partition* of G, denoted by G/H. It follows from the above structural properties of cosets of a subgroup H of a group G that we have Theorem 2.3.

Theorem 2.3 (Lagrange's theorem). Let G be a group of order n, and let H be a subgroup of order m. Then m divides n, and the partition G/H consists of n/m cosets of H.

2.3 Fields

In this section, we introduce an algebraic system with two binary operations, called a *field*. Fields with finite numbers of elements, called *finite fields*, play an important role in developing algebraic coding theory and constructing error-correction codes that can be efficiently encoded and decoded. Well-known and widely used error-correction codes in communication and storage systems based on finite fields are *BCH* (Bose–Chaudhuri–Hocquenghem) codes and *RS* (Reed–Solomon) codes, which were introduced at the very beginning of the 1960s. Most recently, finite fields have successfully been used for constructing *low-density parity-check* (LDPC) codes that perform close to the Shannon limit with iterative decoding. In this section, we first introduce the basic concepts and fundamental properties of a field, and then give the construction of a class of finite fields that serve as *ground fields* from which *extension fields* are constructed in a later section. Some important properties of finite fields are also presented in this section.

2.3.1 Definitions and Basic Concepts

Definition 2.8. Let F be a set of elements on which two binary operations, called *addition* "+" and *multiplication* "·," are defined. F is a *field* under these two operations, addition and multiplication, if the following conditions (or axioms) are satisfied.

1. F is a commutative group under addition $+$. The identity element with respect to addition $+$ is called the *zero element* (or *additive identity*) of F and is denoted by 0.
2. The set $F\backslash\{0\}$ of nonzero elements of F forms a commutative group under multiplication $\cdot$. The identity element with respect to multiplication $\cdot$ is called the *unit element* (or *multiplicative identity*) of F and is denoted by 1.
3. For any three elements a, b, and c in F,

$$a \cdot (b + c) = a \cdot b + a \cdot c. \tag{2.11}$$

The equality of (2.11) is called the *distributive law*, i.e., multiplication is distributive over addition.

From the definition, we see that a field consists of two groups with respect to two operations, addition and multiplication. The group with respect to addition is called the *additive group* of the field, and the group with respect to multiplication is called the *multiplicative group* of the field. Since each group must contain an identity element, a field must contain at least two elements. Later, we will show that a field with two elements does exist. In a field, the additive inverse of an element a is denoted by "$-a$," and the multiplicative inverse of a is denoted by "a^{-1}," provided that $a \neq 0$. From the additive and multiplicative inverses of elements of a field, two other operations, namely *subtraction* "$-$" and *division* "$\div$," can be defined. Subtracting a field element b from a field element a is defined as adding the additive inverse, $-b$, of b to a, i.e.,

$$a - b \triangleq a + (-b).$$

It is clear that $a - a = 0$. Dividing a field element a by a nonzero element b is defined as multiplying a by the multiplicative inverse, b^{-1}, of b, i.e.,

$$a \div b \triangleq a \cdot (b^{-1}).$$

It is clear that $a \div a = 1$ provided that $a \neq 0$.

A field is simply an algebraic system in which we can perform addition, subtraction, multiplication, and division without leaving the field.

Definition 2.9. The number of elements in a field is called the *order of the field*. A field with finite order is called a *finite field*.

Rational numbers under real addition and multiplication form a field with an infinite number of elements. All of the rational numbers under real addition form the additive group of the field, and all the nonzero rational numbers under real multiplication form the multiplicative group of the field. It is known that multiplication is distributive over addition in the system of rational numbers. Another commonly known field is the field of all real numbers under real addition and multiplication. An *extension* of the real-number field is the field of complex numbers under complex-number addition and multiplication. The complex-number field is constructed from the real-number field and a *root*, denoted by $\sqrt{-1}$, of the irreducible polynomial $X^2 + 1$ over the real-number field.

Definition 2.10. Let F be a field. A subset K of F that is itself a field under the operations of F is called a *subfield* of F. In this context, F is called an *extension field* of K. If $K \neq F$, we say that K is a *proper subfield* of F. A field containing no proper subfields is called a *prime field*.

The rational-number field is a proper subfield of the real-number field, and the real-number field is a proper subfield of the complex-number field. The

rational-number field is a prime field. The real- and complex-number fields are extension fields of the rational-number field.

Since the unit element 1 is an element of a field F, it follows from the closure property of the addition operation of F that the sums

$$1, \quad 1+1, \quad 1+1+1, \quad \ldots, \quad \underbrace{1+1+\cdots+1}_{\text{sum of } k \text{ unit elements}}, \quad \ldots$$

are elements of F.

Definition 2.11. Let F be a field and 1 be its unit element (or multiplicative identity). The *characteristic* of F is defined as the smallest positive integer λ such that

$$\sum_{i=1}^{\lambda} 1 = \underbrace{1+1+\cdots+1}_{\lambda} = 0,$$

where the summation represents repeated applications of addition $+$ of the field. If no such λ exists, F is said to have *zero characteristic*, i.e., $\lambda = 0$, and F is an infinite field.

The rational-, real-, and complex-number fields all have zero characteristics.

In the following, we state some fundamental properties of a field F without proofs (left as exercises).

1. For every element a in F, $a \cdot 0 = 0 \cdot a = 0$.
2. For any two nonzero elements a and b in F, $a \cdot b \neq 0$.
3. For two elements a and b in F, $a \cdot b = 0$ implies that either $a = 0$ or $b = 0$.
4. For any two elements a and b in F,
$$-(a \cdot b) = (-a) \cdot b = a \cdot (-b).$$
5. For $a \neq 0$, $a \cdot b = a \cdot c$ implies that $b = c$ (called the *cancelation law*).

Theorem 2.4. The characteristic λ of a finite field is a prime.

Proof. Suppose that λ is not a prime. Then, λ can be factored as a product of two smaller positive integers, say k and l, with $1 < k, l < \lambda$. Then $\lambda = kl$. It follows from the distributive law that

$$\sum_{i=1}^{\lambda} 1 = \left(\sum_{i=1}^{k} 1\right)\left(\cdot \sum_{i=1}^{l} 1\right) = 0.$$

It follows from property 3 given above that either $\sum_{i=1}^{k} 1 = 0$ or $\sum_{i=1}^{l} 1 = 0$. Since $1 < k, l < \lambda$, either $\sum_{i=1}^{k} 1 = 0$ or $\sum_{i=1}^{l} 1 = 0$ contradicts the definition that λ is the smallest positive integer such that $\sum_{i=1}^{\lambda} 1 = 0$. Therefore, λ must be a prime. $\square$

2.3.2 Finite Fields

The rational-, real- and complex-number fields are infinite fields. In error-control coding, both theory and practice, we mainly use finite fields. In the following, we first present a class of finite fields that are constructed from prime numbers and then introduce some fundamental properties of a finite field. Finite fields are commonly called *Galois fields*, in honor of their discoverer, Évariste Galois.

Let p be a prime number. In Section 2.2.2, we have shown that the set of integers, $\{0, 1, \ldots, p-1\}$, forms a commutative group under modulo-p addition $\boxplus$. We have also shown that the set of nonzero elements, $\{1, 2, \ldots, p-1\}$, forms a commutative group under modulo-p multiplication $\boxdot$. Following the definitions of modulo-p addition and multiplication and the fact that real-number multiplication is distributive over real number addition, we can readily show that the modulo-p multiplication is distributive over modulo-p addition. Therefore, the set of integers, $\{0, 1, \ldots, p-1\}$, forms a finite field GF(p) of order p under modulo-p addition and multiplication, where 0 and 1 are the zero and unit elements of the field, respectively. The following sums of the unit element 1 give all the elements of the field:

$$1, \quad \sum_{i=1}^{2} 1 = 2, \quad \sum_{i=1}^{3} 1 = 1 \boxplus 1 \boxplus 1 = 3, \quad \ldots, \quad \sum_{i=1}^{p-1} 1 = p-1, \quad \sum_{i=1}^{p} 1 = 0.$$

Since p is the smallest positive integer such that $\sum_{i=1}^{p} 1 = 0$, the characteristic of the field is p. The field contains no proper subfield and hence is a prime field.

Example 2.8. Consider the special case for which $p = 2$. For this case, the set $\{0, 1\}$ under modulo-2 addition and multiplication as given by Tables 2.6 and 2.7 forms a field of two elements, denoted by GF(2).

Note that $\{1\}$ forms the multiplicative group of GF(2), a group with only one element. The two elements of GF(2) are simply the additive and multiplicative identities of the two groups of GF(2). The additive inverses of 0 and 1 are themselves. This implies that $1 - 1 = 1 + 1$. Hence, over GF(2), subtraction and addition are the same. The multiplicative inverse of 1 is itself. GF(2) is commonly called a *binary field*, the simplest finite field. GF(2) plays an important role in coding theory and is most commonly used as the alphabet of code symbols for error-correction codes.

Example 2.9. Let $p = 5$. The set $\{0, 1, 2, 3, 4\}$ of integers under modulo-5 addition and multiplication given by Tables 2.8 and 2.9 form a field GF(5) of five elements.

The addition table is also used for subtraction, denoted $\boxminus$. For example, suppose that 4 is subtracted from 2. First we use the addition table to find the additive inverse of 4, which is 1. Then we add 1 to 2 with modulo-5 addition. This gives the element 3. The entire subtraction process is expressed as follows:

$$2 \boxminus 4 = 2 \boxplus (-4) = 2 \boxplus 1 = 3.$$

Table 2.6. Modulo-2 addition

$\boxplus$	0	1
0	0	1
1	1	0

Table 2.7. Modulo-2 multiplication

$\boxdot$	0	1
0	0	0
1	0	1

Table 2.8. Modulo-5 addition

$\boxplus$	0	1	2	3	4
0	0	1	2	3	4
1	1	2	3	4	0
2	2	3	4	0	1
3	3	4	0	1	2
4	4	0	1	2	3

Table 2.9. Modulo-5 multiplication

$\boxplus$	0	1	2	3	4
0	0	0	0	0	0
1	0	1	2	3	4
2	0	2	4	1	3
3	0	3	1	4	2
4	0	4	3	2	1

For division, denoted by $\div$, we use the multiplication table. Suppose that we divide 3 by 2. From the multiplication table, we find the multiplicative inverse of 2, which is 3. We then multipy 3 by 3 with modulo-5 multiplication. The result is 4. The entire division process is expressed as follows:

$$3 \div 2 = 3 \boxdot (2^{-1}) = 3 \boxdot 3 = 4.$$

Let $p = 7$. With the modulo-7 addition and multiplication given by Tables 2.1 and 2.2, the set $\{0, 1, 2, 3, 4, 5, 6\}$ of integers forms a field, GF(7), of seven elements.

We have shown that, for every prime number p, there exists a prime field GF(p). For any positive integer m, it is possible to construct a Galois field GF(p^m) with

p^m elements based on the prime field GF(p) and a root of a *special irreducible polynomial* with coefficients from GF(p). This will be shown in a later section. The Galois field GF(p^m) contains GF(p) as a subfield and is an extension field of GF(p). A very important feature of a finite field is that its order must be a power of a prime. That is to say that the order of any finite field is a power of a prime.

In algebraic coding theory, Galois fields are used for code construction as well as decoding, such as BCH and RS codes and their decodings. In later chapters of this book (Chapters 10 to 14), we will use Galois fields to construct structured LDPC codes.

Hereafter, we use GF(q) to mean a Galois (or finite) field with q elements, where q is a power of a prime. Before we consider construction of extension fields of prime fields, we continue to develop some important properties of a finite field that will be used in the later chapters for code construction.

Let a be a nonzero element of GF(q). The powers

$$a^1 = a, \quad a^2 = a \cdot a, \quad a^3 = a \cdot a \cdot a, \quad \ldots$$

are also elements of GF(q). Since GF(q) has only a finite number of elements, the above powers of a cannot be all distinct. Hence, at some point of the sequence of powers of a, there must be a repetition, say $a^m = a^k$, with $m > k$. We express $a^m = a^k$ as $a^k \cdot a^{m-k} = a^k \cdot 1$. Using the cancelation law, we obtain the equality

$$a^{m-k} = 1.$$

This implies that, for any nonzero element a in GF(q), there exists at least one positive integer n such that $a^n = 1$. This gives us the following definition.

Definition 2.12. Let a be a nonzero element of a finite field GF(q). The smallest positive integer n such that $a^n = 1$ is called the *order* of the nonzero field element a.

Theorem 2.5. Let a be a nonzero element of order n in a finite field GF(q). Then the powers of a,

$$a^n = 1, \quad a, \quad a^2, \quad \ldots, \quad a^{n-1},$$

form a cyclic subgroup of the multiplicative group of GF(q).

Proof. Let $G = \{a^n = 1, a, \ldots, a^{n-1}\}$. Since n is the smallest positive integer such that $a^n = 1$, all the elements in G are distinct nonzero elements in the multiplicative group of GF(q), denoted by GF(q)\{0\}. Hence G is a subset of GF(q)\{0\}. For $1 \le i, j \le n$, consider $a^i \cdot a^j$. If $i + j \le n$, $a^i \cdot a^j = a^{i+j}$, which is an element in G. If $i + j > n$, then $i + j = n + k$ with $1 \le k < n$. In this case, $a^i \cdot a^j = a^k$, which is an element in G. Therefore, G is closed under the multiplication operation of the field. G contains the identify element 1 of the multiplicative group GF(q)\{0\} of GF(q). For $1 \le i < n$, a^{n-i} is the multiplicative inverse of the element a^i. The multiplicative inverse of the identity element 1 is itself. So every element in G has a multiplicative inverse. Therefore, G is a subgroup of GF(q). It follows from Definition 2.5 that G is a cyclic subgroup of GF(q)\{0\} and a is a generator of G. □

Theorem 2.6. Let a be a nonzero element of a finite field GF(q). Then $a^{q-1} = 1$.

Proof. Let $b_1, b_2, \ldots, b_{q-1}$ be the $q-1$ nonzero elements of GF(q). Consider the $q-1$ elements, $a \cdot b_1, a \cdot b_2, \ldots, a \cdot b_{q-1}$, which are nonzero. These $q-1$ elements must be distinct, otherwise, for some $i \neq j$, $a \cdot b_i = a \cdot b_j$, which implies that $b_i = b_j$ (use the cancelation law). Therefore, $a \cdot b_1, a \cdot b_2, \ldots, a \cdot b_{q-1}$ also form the $q-1$ nonzero elements of GF(q). Hence,

$$(a \cdot b_1) \cdot (a \cdot b_2) \cdot \cdots \cdot (a \cdot b_{q-1}) = b_1 \cdot b_2 \cdot \cdots \cdot b_{q-1}.$$

Since the multiplication operation in GF(q) is associative and commutative, the above equality can be put in the following form:

$$a^{q-1}(b_1 \cdot b_2 \cdot \cdots \cdot b_{q-1}) = b_1 \cdot b_2 \cdot \cdots \cdot b_{q-1}.$$

Since the product of nonzero elements is nonzero, it follows from the cancelation law that $a^{q-1} = 1$. This proves the theorem. □

Theorem 2.7. Let n be the order of a nonzero element a in GF(q). Then n divides $q-1$.

Proof. Suppose $q-1$ is not divisible by n. On dividing $q-1$ by n, we have $q-1 = kn + r$, where $1 \leq r < n$. Then $a^{q-1} = a^{kn+r} = a^{kn} \cdot a^r$. Since $a^{q-1} = 1$ and $a^n = 1$, we must have $a^r = 1$. This contradicts the fact that n is the smallest positive integer such that $a^n = 1$. Therefore, n must divide $q-1$. □

Definition 2.13. A nonzero element a of a finite field GF(q) is called a *primitive element* if its order is $q-1$. That is, $a^{q-1} = 1, a, a^2, \ldots, a^{q-2}$ form all the nonzero elements of GF(q).

Every finite field has at least one primitive element (see Problem 2.10). Consider the prime field GF(5) under modulo-5 addition and multiplication given in Example 2.9. If we take powers of 2 in GF(5) using the multiplication given by Table 2.9, we obtain all the nonzero elements of GF(5),

$$2^1 = 2, \quad 2^2 = 2 \boxdot 2 = 4, \quad 2^3 = 2^2 \boxdot 2 = 4 \boxdot 2 = 3,$$

$$2^4 = 2^3 \boxdot 2 = 3 \boxdot 2 = 1.$$

Hence 2 is a primitive element of GF(5). The integer 3 is also a primitive element of GF(5) as shown below,

$$3^1 = 3, \quad 3^2 = 3 \boxdot 3 = 4, \quad 3^3 = 3^2 \boxdot 3 = 4 \boxdot 3 = 2,$$

$$3^4 = 3^3 \boxdot 3 = 2 \boxdot 3 = 1.$$

However, the integer 4 is not a primitive element and its order is 2,

$$4^1 = 4, \quad 4^2 = 4 \boxdot 4 = 1.$$

2.4 Vector Spaces

In this section, we present another algebraic system, called a *vector space*. A special type of vector space plays an important role in error-control coding. The main ingredients of a vector space are a field F, a group V that is commutative under an addition, and a multiplication operation that combines the elements of F and the elements of V. Therefore, a vector space may be considered as a super-algebraic system with a total of four binary operations: addition and multiplications on F, addition on V, and the multiplication operation that combines the elements of F and the elements of V.

2.4.1 Basic Definitions and Properties

Definition 2.14. Let F be a field. Let V be a set of elements on which a binary operation called addition $+$ is defined. A multiplication operation, denoted by "$\cdot$," is defined between the elements of F and the elements of V. The set V is called a *vector space* over the field F if it satisfies the following axioms.

1. V is a commutative group under addition $+$ defined on V.
2. For any element a in F and any element $\mathbf{v}$ in V, $a \cdot \mathbf{v}$ is an element in V.
3. For any elements a and b in F and any element $\mathbf{v}$ in V, the following associative law is satisfied:
$$(a \cdot b) \cdot \mathbf{v} = a \cdot (b \cdot \mathbf{v}).$$
4. For any element a in F and any elements $\mathbf{u}$ and $\mathbf{v}$ in V, the following distributive law is satisfied:
$$a \cdot (\mathbf{u} + \mathbf{v}) = a \cdot \mathbf{u} + a \cdot \mathbf{v}.$$
5. For any two elements a and b in F and any element $\mathbf{v}$ in V, the following distributive law is satisfied:
$$(a + b) \cdot \mathbf{v} = a \cdot \mathbf{v} + b \cdot \mathbf{v}.$$
6. Let 1 be the unit element of F. Then, for any element $\mathbf{v}$ in V, $1 \cdot \mathbf{v} = \mathbf{v}$.

The elements of V are called *vectors* and denoted by boldface lower-case letters, $\mathbf{u}$, $\mathbf{v}$, $\mathbf{w}$, etc. The elements of F are called *scalars* and denoted by lower-case italic letters, a, b, c, etc. The addition $+$ on V is called a *vector addition*, the multiplication $\cdot$ that combines a scalar a in F and a vector $\mathbf{v}$ in V into a vector $a \cdot \mathbf{v}$ in V is referred to as *scalar multiplication*, and the vector $a \cdot \mathbf{v}$ is called the *product* of a and $\mathbf{v}$. The additive identity of V is denoted by boldface $\mathbf{0}$, called the *zero vector* of V. Note that we use $+$ for additions on both V and F. It should be clear that, when we combine two vectors in V, $+$ means the vector addition; and when we combine two scalars in F, $+$ means the addition defined on F. We also use $\cdot$ for both scalar multiplication and multiplication defined on F. When a

scalar in F and a vector in V are combined, "$\cdot$" means scalar multiplication; and when two scalars in F are combined, "$\cdot$" means multiplication on F.

Some basic properties of a vector space V over a field F are given by the propositions below without proofs. The proofs are left as exercises.

Proposition 2.1. Let 0 be the zero element of the field F. For any vector $\mathbf{v}$ in V, $0 \cdot \mathbf{v} = \mathbf{0}$.

Proposition 2.2. For any element a in F, $a \cdot \mathbf{0} = \mathbf{0}$.

Proposition 2.3. For any element a in F and any vector $\mathbf{v}$ in V, $(-a) \cdot \mathbf{v} = a \cdot (-\mathbf{v}) = -(a \cdot \mathbf{v})$, i.e., either $(-a) \cdot \mathbf{v}$ or $a \cdot (-\mathbf{v})$ is the additive inverse of the vector $a \cdot \mathbf{v}$.

Let R and C be the real- and complex-number fields, respectively. C, with its usual operation of addition (but with multiplication ignored) is a vector space over R if scalar multiplication is defined by the rule $c \cdot (a + bi) = (c \cdot a) + (c \cdot b)i$, where $a + bi$ is any complex number and c is any real number.

A vector space V over a field F contains certain subsets that form vector spaces over F in their own right. Such subsets are called *subspaces* of V.

Definition 2.15. Let S be a non-empty subset of a vector space V over a field F. S is called a *subspace* of V if it satisfies the axioms for a vector space given by Definition 2.14.

To determine whether a subset S of a vector space V over a field F is a subspace, it is not necessary to verify all the axioms given by Definition 2.14. The following axioms are sufficient.

(S1) For any two vectors $\mathbf{u}$ and $\mathbf{v}$ in S, $\mathbf{u} + \mathbf{v}$ is also a vector in S.
(S2) For any element a in F and any vector $\mathbf{u}$ in S, $a \cdot \mathbf{u}$ is also in S.

Axioms (S1) and (S2) simply say that S is closed under vector addition and scalar multiplication of V. Axiom (S2) ensures that, for any vector $\mathbf{u}$ in S, its additive inverse $(-1) \cdot \mathbf{u}$ is also a vector in S. Consequently, the zero vector $\mathbf{0} = \mathbf{u} + (-1) \cdot \mathbf{u}$ is also in S. Therefore, S is a subgroup of V under vector addition. Since the vectors of S are also vectors of V, the associative and distributive laws must hold for S. Furthermore, axiom (6) given by Definition 2.14 is obviously satisfied. Hence, S is a vector space over F, a subspace of V.

2.4.2 Linear Independence and Dimension

Hereafter, when we take the product $a \cdot \mathbf{v}$ of a scalar a and a vector $\mathbf{v}$ of a vector space V over a field F, we drop the scalar multiplication and simply write the product $a \cdot \mathbf{v}$ as $a\mathbf{v}$.

Let $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_k$ be k vectors in V. Let $a_1, a_2, \ldots, a_k$ be k arbitrary elements of F. Then $a_1\mathbf{v}_1, a_2\mathbf{v}_2, \ldots, a_k\mathbf{v}_k$ and the sum

$$a_1\mathbf{v}_1 + a_2\mathbf{v}_2 + \cdots + a_k\mathbf{v}_k$$

are vectors in V. The sum is called a *linear combination* of $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_k$. It is clear that the sum of two linear combinations of $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_k$, namely

$$\begin{aligned}&(a_1\mathbf{v}_1 + a_2\mathbf{v}_2 + \cdots + a_k\mathbf{v}_k) + (b_1\mathbf{v}_1 + b_2\mathbf{v}_2 + \cdots + b_k\mathbf{v}_k)\\ &\quad = (a_1 + b_1)\mathbf{v}_1 + (a_2 + b_2)\mathbf{v}_2 + \cdots + (a_k + b_k)\mathbf{v}_k,\end{aligned}$$

is also a linear combination of $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_k$, where $a_i + b_i$ is carried out with the addition of the field F. It is also clear that the product of an element c in F and a linear combination of $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_k$,

$$c(a_1\mathbf{v}_1 + a_2\mathbf{v}_2 + \cdots + a_k\mathbf{v}_k) = (c \cdot a_1)\mathbf{v}_1 + (c \cdot a_2)\mathbf{v}_2 + \cdots + (c \cdot a_k)\mathbf{v}_k,$$

is a linear combination of $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_k$, where $c \cdot a_i$ is carried out with the multiplication of the field F. Let S be the set of all linear combinations of $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_k$. Then S satisfies axioms (S1) and (S2) for a subspace of a vector space. Hence, S is a subspace of the vector space V over F.

Definition 2.16. A set of vectors $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_k$ in a vector space V over a field F is said to be *linearly independent* over F if and only if, for all k scalars a_i in F,

$$a_1\mathbf{v}_1 + a_2\mathbf{v}_2 + \cdots + a_k\mathbf{v}_k = \mathbf{0}$$

implies

$$a_1 = a_2 = \cdots = a_k = 0.$$

Vectors that are not linearly independent are said to be *linearly dependent.*

It follows from Definition 2.16 that, if $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_k$ are linearly dependent, then there exist k scalars $a_1, a_2, \ldots, a_k$ in F, *not all zero*, such that

$$a_1\mathbf{v}_1 + a_2\mathbf{v}_2 + \cdots + a_k\mathbf{v}_k = \mathbf{0}.$$

Since there is at least one scalar, say a_k, not equal to zero, the above equality can be written as

$$\begin{aligned}\mathbf{v}_k &= -a_k^{-1}(a_1\mathbf{v}_1 + a_2\mathbf{v}_2 + \cdots + a_{k-1}\mathbf{v}_{k-1})\\ &= c_1\mathbf{v}_1 + c_2\mathbf{v}_2 + \cdots + c_{k-1}\mathbf{v}_{k-1},\end{aligned}$$

where $c_i = -a_k^{-1}a_i$ for $1 \le i < k$ and a_k^{-1} is the multiplicative inverse of the field element a_k. In this case, we say that $\mathbf{v}_k$ is linearly dependent on $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_{k-1}$.

Definition 2.17. Let $\mathbf{u}, \mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_k$ be vectors in a vector space V over a field F. Then $\mathbf{u}$ is said to be *linearly dependent* on $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_k$ if it can be expressed as a linear combination of $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_k$ as follows:

$$\mathbf{u} = a_1\mathbf{v}_1 + a_2\mathbf{v}_2 + \cdots + a_k\mathbf{v}_k,$$

with a_i in F for $1 \le i \le k$.

Definition 2.18. A set of vectors is said to *span* a vector space V over a field F if every vector in the vector space V is equal to a linear combination of the vectors in the set.

In any vector space (or subspace), there exists at least one set B of linearly independent vectors that spans the space. This set B is called a *basis* (or *base*) of the vector space. Two different bases of a vector space have the *same number* of linearly independent vectors (the proof of this fact is left as an exercise). The number of linearly independent vectors in a basis of a vector space is called the *dimension* of the vector space. In the case that a basis of a vector space has a finite number, say n, of linearly independent vectors, we write $\dim(V) = n$. If a basis contains infinitely many linearly independent vectors, then we write $\dim(V) = \infty$. Let V be a vector space with dimension n. For $0 \leq k \leq n$, a set of k linearly independent vectors in V spans a *k-dimensional* subspace of V.

2.4.3 Finite Vector Spaces over Finite Fields

In this subsection, we present a very useful vector space over a finite field $\mathrm{GF}(q)$, which plays a central role in error-control coding theory. Let n be a positive integer. Consider an *ordered sequence* of n components,

$$\mathbf{v} = (v_0, v_1, \ldots, v_{n-1}),$$

where each component v_i, $0 \leq i < n$, is an element of the finite field $\mathrm{GF}(q)$. This ordered sequence is called an *n-tuple* over $\mathrm{GF}(q)$. Since each component v_i can be any of the q elements in $\mathrm{GF}(q)$, there are q^n distinct n-tuples over $\mathrm{GF}(q)$. Let V_n denote this set of q^n distinct n-tuples. We define addition $+$ of two n-tuples over $\mathrm{GF}(q)$, $\mathbf{u} = (u_0, u_1, \ldots, u_{n-1})$ and $\mathbf{v} = (v_0, v_1, \ldots, v_{n-1})$, as follows:

$$\begin{aligned} \mathbf{u} + \mathbf{v} &= (u_0, u_1, \ldots, u_{n-1}) + (v_0, v_1, \ldots, v_{n-1}) \\ &= (u_0 + v_0, u_1 + v_1, \ldots, u_{n-1} + v_{n-1}), \end{aligned} \tag{2.12}$$

where $u_i + v_i$ is carried out with the addition of $\mathrm{GF}(q)$ and is hence an element of $\mathrm{GF}(q)$. Therefore, addition of two n-tuples over $\mathrm{GF}(q)$ is also an n-tuple over $\mathrm{GF}(q)$. Hence, V_n is closed under the addition $+$ defined by (2.12). We can readily verify that V_n under the addition $+$ defined by (2.12) is a commutative group. Since the addition operation on $\mathrm{GF}(q)$ is associative and commutative, we can easily check that the addition defined by (2.12) is also associative and commutative. Let 0 be the zero element of $\mathrm{GF}(q)$. The *all-zero* n-tuple, $\mathbf{0} = (0, 0, \ldots, 0)$ is the additive identity of V_n, since

$$\begin{aligned} (0, 0, \ldots, 0) + (v_0, v_1, \ldots, v_{n-1}) &= (0 + v_0, 0 + v_1, \ldots, 0 + v_{n-1}) \\ &= (v_0, v_1, \ldots, v_{n-1}), \end{aligned}$$

and

$$\begin{aligned} (v_0, v_1, \ldots, v_{n-1}) + (0, 0, \ldots, 0) &= (v_0 + 0, v_1 + 0, \ldots, v_{n-1} + 0) \\ &= (v_0, v_1, \ldots, v_{n-1}). \end{aligned}$$

Consider an n-tuple, $\mathbf{v} = (v_0, v_1, \ldots, v_{n-1})$ in V_n. For $0 \le i < n$, let $-v_i$ be the additive inverse of the field element v_i of GF(q). Then, the n-tuple $-\mathbf{v} = (-v_0, -v_1, \ldots, -v_{n-1})$ in V_n is the additive inverse of the n-tuple $\mathbf{v} = (v_0, v_1, \ldots, v_{n-1})$, since $\mathbf{v} + (-\mathbf{v}) = (-\mathbf{v}) + \mathbf{v} = \mathbf{0}$. Hence, every n-tuple in V_n has an additive inverse with respect to the addition of two n-tuples over GF(q). Therefore, V_n under the addition defined by (2.12) satisfies all the axioms of a commutative group and is thus a commutative group.

Next, we define scalar multiplication of an n-tuple $\mathbf{v} = (v_0, v_1, \ldots, v_{n-1})$ by a field element c as follows:

$$c \cdot \mathbf{v} = c \cdot (v_0, v_1, \ldots, v_{n-1}) = (c \cdot v_0, c \cdot v_1, \ldots, c \cdot v_{n-1}), \tag{2.13}$$

where $c \cdot v_i$, $0 \le i < n$, is carried out with the multiplication of GF(q). Since every component $c \cdot v_i$ of the ordered sequence $c \cdot \mathbf{v} = (c \cdot v_0, c \cdot v_1, \ldots, c \cdot v_{n-1})$ given by (2.13) is an element of GF(q), $c \cdot \mathbf{v} = (c \cdot v_0, c \cdot v_1, \ldots, c \cdot v_{n-1})$ is an n-tuple in V_n. If $c = 1$, then $1 \cdot \mathbf{v} = \mathbf{v}$. We can easily verify that the addition of two n-tuples over GF(q) and the scalar multiplication of an n-tuple by an element in GF(q) defined by (2.12) and (2.13), respectively, satisfy the associative and distributive laws. Therefore, V_n is a vector space over GF(q) with all the n-tuples over GF(q) as vectors.

Consider the following n n-tuples over GF(q):

$$\begin{aligned} \mathbf{e}_0 &= (1, 0, 0, \ldots, 0), \\ \mathbf{e}_1 &= (0, 1, 0, \ldots, 0), \\ &\vdots \\ \mathbf{e}_{n-1} &= (0, 0, 0, \ldots, 1), \end{aligned} \tag{2.14}$$

where the n-tuple $\mathbf{e}_i$ has a single nonzero component at position i, which is the unit element of GF(q). Every n-tuple $\mathbf{v} = (v_0, v_1, \ldots, v_{n-1})$ over GF(q) can be expressed as a linear combination of $\mathbf{e}_0, \mathbf{e}_1, \ldots, \mathbf{e}_{n-1}$ as follows:

$$\begin{aligned} \mathbf{v} &= (v_0, v_1, \ldots, v_{n-1}) \\ &= v_0\mathbf{e}_0 + v_1\mathbf{e}_1 + \cdots + v_{n-1}\mathbf{e}_{n-1}. \end{aligned} \tag{2.15}$$

Therefore, $\mathbf{e}_0, \mathbf{e}_1, \ldots, \mathbf{e}_{n-1}$ span the vector space V_n of all the q^n n-tuples over GF(q). From (2.15), we readily see that the linear combination given by (2.15) is equal to the $\mathbf{0}$ vector if and only if $v_0, v_1, \ldots, v_{n-1}$ are all equal to the zero element of GF(q). Hence, $\mathbf{e}_0, \mathbf{e}_1, \ldots, \mathbf{e}_{n-1}$ are linearly independent and they form a basis of V_n. The dimension of V_n is then equal to n. Any basis of V_n consists of n linearly independent n-tuples over GF(q).

For $0 \le k \le n$, let $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_k$ be k linearly independent n-tuples in V_n. Then, the set S of q^k linear combinations of $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_k$ of the form

$$\mathbf{w} = c_1\mathbf{v}_1 + c_2\mathbf{v}_2 + \cdots + c_k\mathbf{v}_k$$

forms a *k-dimensional subspace* of V_n.

The most commonly used vector space in error-control coding theory is the vector space V_n of all the 2^n tuples over the binary field GF(2). In this case,

the n-tuples over GF(2) are commonly called binary n-tuples. Adding two binary n-tuples requires adding their corresponding binary components using modulo-2 addition. In GF(2), the unit element 1 is its own inverse, i.e., $1 + 1 = 0$. Therefore, the additive inverse of a binary n-tuple $\mathbf{v} = (v_0, v_1, \ldots, v_{n-1})$ is simply itself, since

$$\mathbf{v} + \mathbf{v} = (v_0 + v_0, v_1 + v_1, \ldots, v_{n-1} + v_{n-1}) = (0, 0, \ldots, 0).$$

Example 2.10. Let $n = 4$. The vector space V_4 of all the 4-tuples over GF(2) consists of the following 16 binary 4-tuples:

$$\begin{gathered}(0000), (0001), (0010), (0011), (0100), (0101), (0110), (0111),\\ (1000), (1001), (1010), (1011), (1100), (1101), (1110), (1111).\end{gathered}$$

The vector sum of (0101) and (1011) is

$$(0111) + (1011) = (0 + 1, 1 + 0, 1 + 1, 1 + 1) = (1100).$$

The four vectors (1000), (1100), (1110), and (1111) are linearly independent and they form a basis of V_4. The four vectors (0001), (0010), (0111), and (1110) are also linearly independent and they form another basis of V_4. The linear combinations of (1000), (1110), and (1111) give a three-dimensional subspace of V_4 with the following eight vectors.

$$\begin{gathered}(0000), (1000), (1110), (1111),\\ (0110), (0111), (0001), (1001).\end{gathered}$$

2.4.4 Inner Products and Dual Spaces

Let $\mathbf{u} = (u_0, u_1, \ldots, u_{n-1})$ and $\mathbf{v} = (v_0, v_1, \ldots, v_{n-1})$ be two n-tuples over GF(q). We define the *inner product* of $\mathbf{u}$ and $\mathbf{v}$ as the following sum:

$$\mathbf{u} \cdot \mathbf{v} = u_0 \cdot v_0 + u_1 \cdot v_1 + \cdots + u_{n-1} \cdot v_{n-1},$$

where the the multiplications and additions in the sum are carried out with multiplication and addition of GF(q). So the inner product of two n-tuples over GF(q) is an element of GF(q), i.e., a scalar. If $\mathbf{u} \cdot \mathbf{v} = 0$, we say that $\mathbf{u}$ and $\mathbf{v}$ are *orthogonal* to each other. Often, the notation $\langle \mathbf{u}, \mathbf{v} \rangle$ is also used to denote the inner product of $\mathbf{u}$ and $\mathbf{v}$. The inner product has the following properties.

1. $\mathbf{u} \cdot \mathbf{v} = \mathbf{v} \cdot \mathbf{u}$ (commutative law).
2. $\mathbf{u} \cdot (\mathbf{v} + \mathbf{w}) = \mathbf{u} \cdot \mathbf{v} + \mathbf{u} \cdot \mathbf{w}$ (distributive law).
3. For any element c in GF(q), $(c\mathbf{u}) \cdot \mathbf{v} = c(\mathbf{u} \cdot \mathbf{v})$ (associative law).
4. $\mathbf{0} \cdot \mathbf{u} = 0$.

Example 2.11. Consider the two 4-tuples, (1011) and (1101), given in Example 2.10. The inner product of these two binary 4-tuples is

$$(1011) \cdot (1101) = 1 \cdot 1 + 0 \cdot 1 + 1 \cdot 0 + 1 \cdot 1 = 1 + 0 + 0 + 1 = 0.$$

Since their inner product is 0, they are orthogonal to each other.

For $0 \leq k < n$, let S be a k-dimensional subspace of the vector space V_n of all the n-tuples over $\mathrm{GF}(q)$. Let S_d be the set of n-tuples in V_n such that, for any $\mathbf{u}$ in S and $\mathbf{v}$ in S_d,

$$\mathbf{u} \cdot \mathbf{v} = 0,$$

i.e.,

$$S_d = \{\mathbf{v} \in V_n \colon \mathbf{u} \cdot \mathbf{v} = 0, \mathbf{u} \in S\}. \tag{2.16}$$

Since $\mathbf{0} \cdot \mathbf{u} = 0$ for any $\mathbf{u} \in S$, S_d contains at least the all-zero n-tuple of V_n and hence is non-empty. Let $\mathbf{v}$ and $\mathbf{w}$ be any two tuples in S_d and $\mathbf{u}$ be any n-tuple in S. It follow from the distributive law of the inner product that

$$\mathbf{u} \cdot (\mathbf{v} + \mathbf{w}) = \mathbf{u} \cdot \mathbf{v} + \mathbf{u} \cdot \mathbf{w} = 0 + 0 = 0.$$

Hence, $\mathbf{u} + \mathbf{w}$ is also in S_d. It follows from the associative law of the inner product that, for any scalar a in $\mathrm{GF}(q)$, any n-tuple $\mathbf{v}$ in S_d, and any n-tuple $\mathbf{u}$ in S,

$$(a \cdot \mathbf{v}) \cdot \mathbf{u} = a \cdot (\mathbf{u} \cdot \mathbf{v}) = a \cdot 0 = 0.$$

Thus $a \cdot \mathbf{v}$ is also an n-tuple in S_d. Hence, S_d satisfies the two axioms for a subspace of a vector space over a finite field. Consequently, S_d is a subspace of the vector space V_n of all the n-tuples over $\mathrm{GF}(q)$. S_d is called the *dual* (or *null*) space of S and vice versa. The dimension of S_d is given by the following theorem, whose proof is omitted here.

Theorem 2.8. For $0 \leq k \leq n$, let S be a k-dimensional subspace of the vector space V_n of all n-tuples over $\mathrm{GF}(q)$. The dimension of its dual space S_d is $n - k$. In other words, $\dim(S) + \dim(S_d) = n$.

2.5 Polynomials over Finite Fields

In elementary algebra, one regards a polynomial as an expression of the following form: $f(X) = f_0 + f_1 X + \cdots + f_n X^n$. The f_is are called *coefficients* and are usually *real* or *complex* numbers, and X is viewed as a *variable* such that, on substituting an arbitrary number c for X, a well-defined number $f(c) = f_0 + f_1 c + \cdots + f_n c^n$ is obtained. The arithmetic of polynomials is governed by familiar rules, such as addition, subtraction, multiplication, and division.

In this section, we consider polynomials with coefficients from a finite field GF(q), which are used in error-control coding theory. A polynomial with one variable (or *indeterminate*) X over GF(q) is an expression of the following form:

$$a(X) = a_0 + a_1X + \cdots + a_nX^n,$$

where n is a non-negative integer and the coefficients a_i, $0 \le i \le n$, are elements of GF(q). The degree of a polynomial is defined as the *largest power* of X with a nonzero coefficient. For the polynomial $a(X)$ above, if $a_n \neq 0$, its degree is n; if $a_n = 0$, its degree is less than n. The degree of a polynomial $a(X) = a_0$ with only the constant term is zero. We use the notation $\deg(a(X))$ to denote the degree of polynomial $a(X)$. A polynomial is called *monic* if the coefficient of the highest power of X is 1 (the unit element of GF(q)). The polynomial whose coefficients are all equal to zero is called the *zero polynomial* and denoted by 0. We adopt the convention that a term a_iX^i with $a_i = 0$ need not be written down. The above polynomial $a(X)$ can also be written in the following *equivalent* form:

$$a(X) = a_0 + a_1X + \cdots + a_nX^n + 0X^{n+1} + \cdots + 0X^{n+k},$$

where k is any non-negative integer. With this equivalent form, when comparing two polynomials over GF(q), we can assume that they both involve the same powers of X.

Two polynomials over GF(q) can be added and multiplied in the usual way. Let

$$a(X) = a_0 + a_1X + \cdots + a_nX^n, \qquad b(X) = b_0 + b_1X + \cdots + b_mX^m$$

be two polynomials over GF(q) with degrees n and m, respectively. Without loss of generality, we assume that $m \le n$. To add $a(X)$ and $b(X)$, we simply add the coefficients of the same power of X in $a(X)$ and $b(X)$ as follows:

$$\begin{aligned} a(X) + b(X) = (a_0 + b_0) + (a_1 + b_1)X + \cdots + (a_m + b_m)X^m \\ + (a_{m+1} + 0)X^{m+1} + \cdots + (a_n + 0)X^n, \end{aligned} \tag{2.17}$$

where $a_i + b_i$ is carried out with the addition of GF(q). Multiplication (or the product) of $a(X)$ and $b(X)$ is defined as follows:

$$a(X) \cdot b(X) = c_0 + c_1X + \cdots + c_{n+m}X^{n+m}, \tag{2.18}$$

where, for $0 \le k \le n + m$,

$$c_k = \sum_{\substack{i+j=k, \\ 0 \le i \le n, 0 \le j \le m}} a_i \cdot b_j. \tag{2.19}$$

It is clear from (2.18) and (2.19) that, if $b(X) = 0$ (the zero polynomial), then

$$a(X) \cdot 0 = 0. \tag{2.20}$$

On the basis of the definitions of addition and multiplication of two polynomials and the facts that addition and multiplication of the field GF(q) satisfy the

associative, commutative, and distributive laws, we can readily verify that the polynomials over GF(q) satisfy the following conditions.

1. Associative laws. For any three polynomials $a(X)$, $b(X)$, and $c(X)$ over GF(q),

$$a(X) + [b(X) + c(X)] = [a(X) + b(X)] + c(X),$$
$$a(X) \cdot [b(X) \cdot c(X)] = [a(X) \cdot b(X)] \cdot c(X).$$

2. Commutative laws. For any two polynomials $a(X)$ and $b(X)$ over GF(q),

$$a(X) + b(X) = b(X) + a(X),$$
$$a(X) \cdot b(X) = b(X) \cdot a(X).$$

3. Distributive law. For any three polynomials $a(X)$, $b(X)$, and $c(X)$ over GF(q),

$$a(X) \cdot [b(X) + c(X)] = a(X) \cdot b(X) + a(X) \cdot c(X).$$

In algebra, the set of polynomials over GF(q) under polynomial addition and multiplication defined above is called a *polynomial ring* over GF(q).

Subtraction of $b(X)$ from $a(X)$ is carried out as follows:

$$\begin{aligned} a(X) - b(X) = (a_0 - b_0) + (a_1 - b_1)X + \cdots + (a_m - b_m)X^m \\ +(a_{m+1} - 0)X^{m+1} + \cdots + (a_n - 0)X^n, \end{aligned} \tag{2.21}$$

where $a_i - b_i = a_i + (-b_i)$ is carried out in GF(q) and $-b_i$ is the additive inverse of b_i.

Example 2.12. Consider the ring of polynomials over the prime field GF(5) given by Example 2.9 (see Tables 2.8 and 2.9). Let $a(X) = 2 + X + 3X^2 + 4X^4$ and $b(X) = 1 + 3X + 2X^2 + 4X^3 + X^4$. On subtracting $b(X)$ from $a(X)$, we obtain

$$\begin{aligned} a(X) - b(X) &= (2-1) + (1-3)X + (3-2)X^2 + (0-4)X^3 + (4-1)X^4 \\ &= (2 + (-1)) + (1 + (-3))X + (3 + (-2))X^2 + (0 + (-4))X^3 \\ &\quad + (4 + (-1))X^4. \end{aligned}$$

Using Table 2.8, we find that $-1 = 4$, $-3 = 2$, $-2 = 3$, and $-4 = 1$. Then

$$a(X) - b(X) = (2+4) + (1+2)X + (3+3)X^2 + (0+1)X^3 + (4+4)X^4.$$

Using Table 2.8 for addition of the coefficients in the above expression, we have the following polynomial:

$$a(X) - b(X) = 1 + 3X + X^2 + X^3 + 3X^4,$$

which is a polynomial of degree 4 over GF(5).

Definition 2.19. A set $\mathcal{R}$ with two binary operations, called addition $+$ and multiplication $\cdot$, is called a *ring*, if the following axioms are satisfied.

1. $\mathcal{R}$ is a commutative group with respect to $+$.
2. Multiplication "$\cdot$" is associative, i.e., $(a \cdot b) \cdot c = a \cdot (b \cdot c)$ for all $a, b, c \in \mathcal{R}$.

3. The distributive laws hold: for $a, b, c \in \mathcal{R}$, $a \cdot (b + c) = a \cdot b + a \cdot c$ and $(b + c) \cdot a = b \cdot a + b \cdot a$.

The set of all polynomials over GF(q) defined above satisfies all three axioms of a ring. Therefore, it is a ring of polynomials. The set of all integers under real addition and multiplication is a ring. For any positive integer $m \geq 2$, the set $\mathcal{R} = \{0, 1, \ldots, m-1\}$ under modulo-m addition and multiplication forms a ring (the proof is left as an exercise). In Section 2.3, it was shown that, if m is a prime, $\mathcal{R} = \{0, 1, \ldots, m-1\}$ is a prime field under modulo-m addition and multiplication.

Let $a(X)$ and $b(X)$ be two polynomials over GF(q). Suppose that $b(X)$ is not the zero polynomial, i.e., $b(X) \neq 0$. When $a(X)$ is divided by $b(X)$, we obtain a *unique pair* of polynomials over GF(q), $q(X)$ and $r(X)$, such that

$$a(X) = q(X) \cdot b(X) + r(X), \tag{2.22}$$

where $0 \leq \deg(r(X)) < \deg(b(X))$ and the polynomials $q(X)$ and $r(X)$ are called the *quotient* and *remainder*, respectively. This is known as *Euclid's division algorithm*. The quotient $q(x)$ and remainder $r(X)$ can be obtained by ordinary *long division* of polynomials. If $r(X) = 0$, we say that $a(X)$ is divisible by $b(X)$ (or $b(X)$ divides $a(X)$).

Example 2.13. Let $a(x) = 3 + 4X + X^4 + 2X^5$ and $b(X) = 1 + 3X^2$ be two polynomials over the prime field GF(5) given by Example 2.9 (see Tables 2.8 and 2.9). Suppose we divide $a(X)$ by $b(X)$. Using long division, and Tables 2.8 and 2.9, we obtain

$$\begin{array}{rrrrrrrrrr}
 & & & & & & 4X^3 & +2X^2 & +2X & +1 \\
3X^2 & & & & +1 & \big) & 2X^5 & +X^4 & & +4X+3 \\
 & & & & & & -2X^5 & & -4X^3 & \\ \cline{7-10}
 & & & & & & & X^4 & +X^3 & +4X+3 \\
 & & & & & & & -X^4 & & -2X^2 \\ \cline{8-10}
 & & & & & & & & X^3 & +3X^2+4X+3 \\
 & & & & & & & & -X^3 & -2X \\ \cline{9-10}
 & & & & & & & & & 3X^2+2X+3 \\
 & & & & & & & & & -3X^2 \quad -1 \\ \cline{10-10}
 & & & & & & & & & 2X+2
\end{array}$$

Thus the quotient $q(X) = 4X^3 + 2X^2 + 2X + 1$ and the remainder $r(X) = 2X + 2$.

If a polynomial $c(X)$ is equal to a product of two polynomials $a(X)$ and $b(X)$, i.e., $c(X) = a(X) \cdot b(X)$, then we say that $c(X)$ is divisible by $a(X)$ (or $b(X)$) or that $a(X)$ (or $b(X)$) divides $c(X)$, and that $a(X)$ and $b(X)$ are factors of $c(X)$.

Definition 2.20. A polynomial $p(X)$ of degree m over GF(q) is said to be *irreducible* over GF(q) if it is not divisible by any polynomial over GF(q) that has a degree less than m but greater than zero.

A polynomial over GF(q) of positive degree that is not irreducible over GF(q) is said to be reducible over GF(q). The irreducibility of a given polynomial depends heavily on the field under consideration. An irreducible polynomial over GF(q) may become reducible over a larger field that contains GF(q) as a subfield. An important property of irreducible polynomials is stated in the following theorem without a proof.

Theorem 2.9. Any irreducible polynomial $p(X)$ over GF(q) of degree m divides $X^{q^m-1} - 1$.

Definition 2.21. A monic irreducible polynomial $p(X)$ over GF(q) of degree m is said to be *primitive* if the smallest positive integer n for which $p(X)$ divides $X^n - 1$ is $n = q^m - 1$.

Primitive polynomials are of fundamental importance for constructing Galois fields, as will be seen in the next section. Lists of primitive polynomials over various prime fields are given in Appendix A at the end of this chapter. Long lists of irreducible and primitive polynomials over various prime fields can be found in [1].

Let n be a positive integer. Let $\mathcal{A}_n$ be the set of q^n polynomials over GF(q) with degree $n-1$ or less. Let $a(X)$ and $b(X)$ be two polynomials in $\mathcal{A}_n$. Take the product of $a(X)$ and $b(X)$ using the rules of (2.18) and (2.19). Dividing the product $a(X) \cdot b(X)$ by $X^n - 1$ gives a remainder $r(X)$ whose degree is $n-1$ or less. It is clear that $r(X)$ is also a polynomial in $\mathcal{A}_n$. Now we define a multiplication "$\cdot$" on $\mathcal{A}_n$ as follows. For any two polynomials $a(X)$ and $b(X)$ in $\mathcal{A}_n$, assign $a(X) \cdot b(X)$ to $r(X)$, which is the remainder obtained from dividing $a(X) \cdot b(X)$ by $X^n - 1$. This multiplication is called *multiplication modulo*-($X^n - 1$) (or modulo-($X^n - 1$) multiplication). Mathematically, this multiplication is written as follows:

$$a(X) \cdot b(X) = r(X) \text{ modulo-}(X^n - 1). \tag{2.23}$$

$\mathcal{A}_n$ is closed under modulo-($X^n - 1$) multiplication. This multiplication satisfies the associative, commutative, and distributive laws. Note that addition of two polynomials in $\mathcal{A}_n$ is carried out as defined by (2.17). The sum of two polynomials in $\mathcal{A}_n$ is a polynomial with degree $n-1$ or less and is hence a polynomial in $\mathcal{A}_n$. The polynomials in $\mathcal{A}_n$ under addition and modulo-($X^n - 1$) multiplication form an *algebra* over GF(q). Let

$$a(X) = a_0 + a_1 X + \cdots + a_{n-1} X^{n-1}$$

be a polynomial in $\mathcal{A}_n$. On cyclically shifting the coefficients of $a(X)$ one place to the right, we obtain the following polynomial in $\mathcal{A}_n$:

$$a^{(1)}(X) = a_{n-1} + a_0 X + \cdots + a_{n-2} X^{n-1}.$$

Polynomial $a^{(1)}(X)$ is called a *right cyclic-shift* of $a(X)$. Actually, $\mathcal{A}_n$ is also a vector space over GF(q). Each polynomial $a(X) = a_0 + a_1 X + \cdots + a_{n-1} X^{n-1}$ in $\mathcal{A}_n$ can be represented by an n-tuple over GF(q) using its n coefficients as the

components as follows:

$$\mathbf{a} = (a_0, a_1, \ldots, a_{n-1}).$$

With this representation, $\mathcal{A}_n$ is simply equivalent to the vector space of all the n-tuples over GF(q).

2.6 Construction and Properties of Galois Fields

In Section 2.3, we have shown that, for any prime number p, there exists a prime field with p elements under modulo-p addition and multiplication. Construction of prime fields is very simple. In this section, we consider construction of *extension fields* of prime fields and give some important structural properties of these extension fields.

2.6.1 Construction of Galois Fields

Construction of an extension field of a prime field GF(p)= $\{0, 1, \ldots, p-1\}$ begins with a primitive polynomial of degree m over GF(p),

$$p(X) = p_0 + p_1 X + \cdots + p_{m-1} X^{m-1} + X^m. \tag{2.24}$$

Since the degree of $p(X)$ is m, it has m roots. Since $p(X)$ is irreducible over GF(p), its roots cannot be in GF(p) and they must be in a larger field that contains GF(p) as a *subfield*.

Let α be a root of $p(X)$. Let 0 and 1 be the zero and unit elements of GF(p). We define a multiplication "$\cdot$" and introduce a sequence of powers of α as follows:

$$\begin{aligned}
&0 \cdot 0 = 0,\\
&0 \cdot 1 = 1 \cdot 0 = 0,\\
&0 \cdot \alpha = \alpha \cdot 0 = 0,\\
&1 \cdot 1 = 1,\\
&1 \cdot \alpha = \alpha \cdot 1 = \alpha,\\
&\alpha^2 = \alpha \cdot \alpha,\\
&\alpha^3 = \alpha \cdot \alpha \cdot \alpha,\\
&\vdots\\
&\alpha^j = \alpha \cdot \alpha \cdot \cdots \cdot \alpha \ (j \text{ times}),\\
&\vdots
\end{aligned} \tag{2.25}$$

It follows from the definition of multiplication above that

$$\begin{aligned}
&0 \cdot \alpha^j = \alpha^j \cdot 0 = 0,\\
&1 \cdot \alpha^j = \alpha^j \cdot 1 = \alpha^j,\\
&\alpha^i \cdot \alpha^j = \alpha^j \cdot \alpha^i = \alpha^{i+j}.
\end{aligned}$$

Since α is a root of $p(X)$,

$$p(\alpha) = p_0 + p_1\alpha + \cdots + p_{m-1}\alpha^{m-1} + \alpha^m = 0. \tag{2.26}$$

Since $p(X)$ divides $X^{p^m-1} - 1$ (Theorem 2.9), we have

$$X^{p^m-1} - 1 = q(X) \cdot p(X). \tag{2.27}$$

On replacing X by α in (2.27), we obtain

$$\alpha^{p^m-1} - 1 = q(\alpha) \cdot p(\alpha) = q(\alpha) \cdot 0. \tag{2.28}$$

If we regard $q(\alpha)$ as a polynomial of α over GF(p), it follows from (2.20) that $q(\alpha) \cdot 0 = 0$. Consequently,

$$\alpha^{p^m-1} - 1 = 0. \tag{2.29}$$

This says that α is also a root of $X^{p^m-1} - 1$. On adding the unit element 1 of GF(p) to both sides of (2.29) (use modulo-p addition), we have the following identity:

$$\alpha^{p^m-1} = 1. \tag{2.30}$$

From equation (2.30), we see that the sequence of powers of α given by (2.25) (excluding the 0 element) repeats itself at the $\alpha^{k(p^m-1)}$ for $k = 1, 2, \ldots$. Therefore, the sequence contains at most p^m distinct elements including the 0 element, i.e., $0, 1, \alpha, \ldots, \alpha^{p^m-2}$. Let

$$\mathcal{F} = \{0, 1, \alpha, \ldots, \alpha^{p^m-2}\}. \tag{2.31}$$

Later, we will show that the p^m elements of $\mathcal{F}$ are distinct. The nonzero elements of $\mathcal{F}$ are closed under the multiplication operation "$\cdot$" defined by (2.25). To see this, let i and j be two integers such that $0 \le i, j < p^m - 1$. If $i + j < p^m - 1$, then $\alpha^i \cdot \alpha^j = \alpha^{i+j}$ is a nonzero element in $\mathcal{F}$. If $i + j \ge p^m - 1$, we can express $i + j$ as follows:

$$i + j = (p^m - 1) + r \tag{2.32}$$

with $0 \le r < p^m - 1$. Then

$$\alpha^i \cdot \alpha^j = \alpha^{i+j} = \alpha^{(p^m-1)+r} = \alpha^{p^m-1} \cdot \alpha^r = 1 \cdot \alpha^r = \alpha^r, \tag{2.33}$$

which is also a nonzero element of $\mathcal{F}$. Hence, the nonzero elements of $\mathcal{F}$ are closed under the multiplication "$\cdot$" defined by (2.25). We readily see that this multiplication is associative and commutative. Since $1 \cdot \alpha^j = \alpha^j \cdot 1 = \alpha^j$, 1 is the identity element with respect to this multiplication. For $0 \le j < p^m - 1$, α^{p^m-1-j} is the multiplicative inverse of α^j. Note that, for $j = 0$, the unit element 1 is its own multiplicative inverse. Therefore, the nonzero elements of $\mathcal{F}$ form a commutative group of order $p^m - 1$ under the multiplication operation "$\cdot$" defined by (2.25).

For $0 \le i < p^m - 1$, on dividing X^i by $p(X)$ we obtain

$$X^i = q_i(X) \cdot p(X) + a_i(X), \tag{2.34}$$

where $q_i(X)$ and $a_i(X)$ are the quotient and remainder, respectively. The remainder $a_i(X)$ is a polynomial over GF(p) with degree $m-1$ or less and is of the following form:

$$a_i(X) = a_{i,0} + a_{i,1}X + \cdots + a_{i,m-1}X^{m-1}, \tag{2.35}$$

where the $a_{i,j}$s are elements of the prime field GF(p). Since $p(X)$ is an irreducible polynomial over GF(p) and X and $p(X)$ are relatively prime, X^i is not divisible by $p(X)$. Hence, for $0 \le i < p^m - 1$,

$$a_i(X) \neq 0. \tag{2.36}$$

To prove that all the nonzero elements of $\mathcal{F}$ are distinct, we need the following theorem.

Theorem 2.10. For $0 \le i, j < p^m - 1$, and $i \neq j$,

$$a_i(X) \neq a_j(X). \tag{2.37}$$

Proof. Assume that $j > i$. Suppose $a_i(X) = a_j(X)$. Then, it follows from (2.34) that

$$X^j - X^i = (q_j(X) - q_i(X)) \cdot p(X).$$

This says that $X^i(X^{j-i} - 1)$ is divisible by $p(X)$. Since X^i and $p(X)$ are relatively prime, $p(X)$ must divide $X^{j-i} - 1$, where $j - i < p^m - 1$. However, this is not possible since $p(X)$ is a primitive polynomial of degree m and the smallest n for which $p(X)$ divides $X^n - 1$ is $n = p^m - 1$ (see Definition 2.21). Therefore, the hypothesis that $a_i(X) = a_j(X)$ is invalid. Consequently, $a_i(X) \neq a_j(X)$ for $i \neq j$. □

It follows from Theorem 2.10 that dividing X^i by $p(X)$ with $i = 0, 1, \ldots, p^m - 2$ gives $p^m - 1$ distinct nonzero-remainder polynomials over GF(p), $a_0(X), a_1(X), \ldots, a_{p^m-2}(X)$. On replacing X by α in (2.34) and (2.35), we obtain

$$\alpha^i = q(\alpha) \cdot p(\alpha) + a_{i,0} + a_{i,1}\alpha + \cdots + a_{i,m-1}\alpha^{m-1}. \tag{2.38}$$

Since α is a root of $p(X)$, $p(\alpha) = 0$ and $q(\alpha) \cdot 0 = 0$. Consequently, we have

$$\alpha^i = a_{i,0} + a_{i,1}\alpha + \cdots + a_{i,m-1}\alpha^{m-1}, \tag{2.39}$$

where the $a_{i,j}$s are elements of GF(p). It follows from Theorem 2.10 and (2.39) that the $p^m - 1$ nonzero elements in $\mathcal{F}$ are distinct and represented by $p^m - 1$ distinct nonzero polynomials of α over GF(p) with degree $m-1$ or less. The 0-element in $\mathcal{F}$ may be represented by the zero polynomial. Note that there are exactly p^m polynomials of α over GF(p) with degree $m-1$ or less. Each of the p^m elements in $\mathcal{F}$ is represented by one and only one of these polynomials, and each of these polynomials represents one and only one element in $\mathcal{F}$. Among these polynomials of α over GF(p) with degree $m-1$ or less, there are p polynomials of zero degree, which are $0, 1, 2, \ldots, p-1$, the elements of GF(p). Therefore, $\mathcal{F}$ contains the elements of GF(p) as a subset.

Next, we define an addition "+" on $\mathcal{F}$ using polynomial representations of the elements in F as follows. For any two elements α^i and α^j,

$$\begin{aligned}\alpha^i + \alpha^j &= (a_{i,0} + a_{i,1}\alpha + \cdots + a_{i,m-1}\alpha^{m-1}) + (a_{j,0} + a_{j,1}\alpha + \cdots + a_{j,m-1}\alpha^{m-1}) \\ &= (a_{i,0} + a_{j,0}) + (a_{i,1} + a_{j,1})\alpha + \cdots + (a_{i,m-1} + a_{j,m-1})\alpha^{m-1}, \qquad (2.40)\end{aligned}$$

where $a_{i,l} + a_{j,l}$ is carried out over the prime field GF(p) using modulo-p addition. Hence, the polynomial given by (2.40) is a polynomial of α over GF(p) and thus represents an element α^k in $\mathcal{F}$. Therefore, $\mathcal{F}$ is closed under the addition defined by (2.40). Since the addition of GF(p) is associative and commutative, we can readily verify that the addition defined by (2.40) is also associative and commutative. From (2.40), we can easily verify that $0 + \alpha^j = \alpha^j + 0$. Hence, 0 is the additive identity with respect to the addition defined by (2.40). The additive inverse of $\alpha^i = a_{i,0} + a_{i,1}\alpha + \cdots + a_{i,m-1}\alpha^{m-1}$ is

$$-\alpha^i = (-a_{i,0}) + (-a_{i,1})\alpha + \cdots + (-a_{i,m-1})\alpha^{m-1},$$

where $-a_{i,l}$ is the additive inverse of the field element $a_{i,l}$ of GF(p). Therefore, $\mathcal{F}$ satisfies all the axioms of a commutative group under the addition defined by (2.40).

So far, we have shown that all the elements of $\mathcal{F}$ form a commutative group under addition defined by (2.40) and all the nonzero elements of $\mathcal{F}$ form a commutative group under the multiplication defined by (2.33). Using the polynomial representations for the elements of $\mathcal{F}$ and the fact that multiplication of polynomials over a field satisfies the distributive law, we can prove that the multiplication defined by (2.33) satisfies the distributive law over the addition defined by (2.40). Hence, the set $\mathcal{F} = \{0, 1, \alpha, \alpha^2, \ldots, \alpha^{p^m-2}\}$ forms a field with p^m elements, denoted by GF(p^m), under the addition defined by (2.40) and the multiplication defined by (2.33). GF(p^m) contains GF(p) as a subfield. Since GF(p^m) is constructed from GF(p) and a monic primitive polynomial over GF(p), we call GF(p^m) an *extension field* of GF(p). The prime field GF(p) is called the *ground field* of GF(p^m). Since the characteristic of GF(p) is p and GF(p) is a subfield of GF(p^m), the characteristic of GF(p^m) is also p.

In the process of constructing the Galois field GF(p^m) from the ground field GF(p), we have developed two representations for each element of GF(p^m), namely the power and polynomial representations. However, there is another useful representation for a field element of GF(p^m). Let $a_{i,0} + a_{i,1}\alpha + \cdots + a_{i,m-1}\alpha^{m-1}$ be the polynomial representation of element α^i. We can represent α^i in vector form by the following m-tuple:

$$(a_{i,0}, a_{i,1}, \ldots, a_{i,m-1}),$$

where the m components are simply the coefficients of the polynomial representation of α^i.

With vector representation, to add two elements α^i and α^j, we simply add their corresponding vector representations, $(a_{i,0}, a_{i,1}, \ldots, a_{i,m-1})$ and

$(a_{j,0}, a_{j,1}, \ldots, a_{j,m-1})$, component-wise as follows:

$$\begin{aligned}&(a_{i,0}, a_{i,1}, \ldots, a_{i,m-1}) + (a_{j,0}, a_{j,1}, \ldots, a_{j,m-1})\\ &\quad = (a_{i,0} + a_{j,0}, a_{i,1} + a_{j,1}, \ldots, a_{i,m-1} + a_{j,m-1}),\end{aligned}$$

where addition $a_{i,l} + a_{j,l}$ is carried out over the ground field GF(p). The vector sum above is an m-tuple over GF(p) and represents an element α^k in the extension field GF(p^m). Now we have three representations for each element in GF(p^m). The power representation is convenient for multiplication, the polynomial and vector representations are convenient for addition. With the power representation, the 0-element is represented by $\alpha^{-\infty}$.

The extension field GF(p^m) may be viewed as an m-dimensional vector space over the ground field GF(p). The elements of GF(p^m) are regarded as the "vectors" and the elements of the ground field GF(p) are regarded as the "scalars." To see this, we first note that all the elements of GF(p^m) form a commutative group under the addition defined by (2.40). Since GF(p) is a subfield of GF(p^m), each element $\alpha^i \in$ GF(p^m), $i = -\infty, 0, 1, \ldots, p^{m-2}$, can be multiplied by an element $\beta \in$ GF(p) such that $\beta\alpha^i$ is also an element (or vector) in GF(p^m). This multiplication is viewed as a "scalar" multiplication with β as a scalar and α^i as a vector. It is obvious that this scalar multiplication satisfies the associative and commutative laws given in Definition 2.14. Since every element α^i in GF(p^m) is a linear combination of $\alpha^0 = 1, \alpha, \ldots, \alpha^{m-1}$,

$$\alpha^i = a_{i,0}\alpha^0 + a_{i,1}\alpha + \cdots + a_{i,m-1}\alpha^{m-1},$$

and $\alpha^i = 0$ if and only if all the coefficients of the $a_{i,j}$s are zero, the m elements (or vectors), $\alpha^0, \alpha, \ldots, \alpha^{m-1}$ are linearly independent and they form a basis of GF(p^m), viewed as an m-dimensional vector space over GF(p). This basis is commonly referred to as the *polynomial basis* of GF(p^m). GF(p^m) is also called an *extension* of GF(p) of degree m.

Example 2.14. Let the binary field GF(2) be the ground field. Suppose we want to construct an extension field GF(2^5) of GF(2) of degree $m = 5$. First, we choose a primitive polynomial of degree 5 over GF(2), say $p(X) = 1 + X^2 + X^5$ from the table given in Appendix A. Let α be a root of $p(X) = 1 + X^2 + X^5$. Then $p(\alpha) = 1 + \alpha^2 + \alpha^5 = 0$ and $\alpha^5 = 1 + \alpha^2$. Furthermore, it follows from (2.30) that $\alpha^{2^5-1} = \alpha^{31} = 1$. Using the equalities $\alpha^5 = 1 + \alpha^2$ and $\alpha^{31} = 1$, we can construct GF(2^5). The elements of GF(2^5) with three different representations are given in Table 2.10. In the construction of Table 2.10, we use the identity $\alpha^5 = 1 + \alpha^2$ repeatedly to form the polynomial representations for the elements of GF(2^5). For example,

$$\begin{aligned}&\alpha^6 = \alpha \cdot \alpha^5 = \alpha \cdot (1 + \alpha^2) = \alpha + \alpha^3,\\ &\alpha^7 = \alpha \cdot \alpha^6 = \alpha \cdot (\alpha + \alpha^3) = \alpha^2 + \alpha^4,\\ &\alpha^8 = \alpha \cdot \alpha^7 = \alpha \cdot (\alpha^2 + \alpha^4) = \alpha^3 + \alpha^5 = \alpha^3 + 1 + \alpha^2 = 1 + \alpha^2 + \alpha^3.\end{aligned}$$

To multiply two elements α^i and α^j, we simply add their exponents and use the identity $\alpha^{31} = 1$. For example, $\alpha^{13} \cdot \alpha^{25} = \alpha^{38} = \alpha^{31} \cdot \alpha^7 = 1 \cdot \alpha^7 = \alpha^7$. The multiplicative inverse of α^i is α^{31-i}. For example, the multiplicative inverse of α^9 is $\alpha^{31-9} = \alpha^{22}$. To

Table 2.10. GF(2^5) generated by the primitive polynomial $p(X) = 1 + X^2 + X^5$ over GF(2)

Power representation	Polynomial representation	Vector representation
0	0	(00000)
1	1	(10000)
α	α	(01000)
α^2	α^2	00100)
α^3	α^3	(00010)
α^4	α^4	(00001)
α^5	$1+\alpha^2$	(10100)
α^6	$\alpha+\alpha^3$	(01010)
α^7	$\alpha^2+\alpha^4$	(00101)
α^8	$1+\alpha^2+\alpha^3$	(10110)
α^9	$\alpha+\alpha^3+\alpha^4$	(01011)
α^{10}	$1+\alpha^4$	(10001)
α^{11}	$1+\alpha+\alpha^2$	(11100)
α^{12}	$\alpha+\alpha^2+\alpha^3$	(01110)
α^{13}	$\alpha^2+\alpha^3+\alpha^4$	(00111)
α^{14}	$1+\alpha^2+\alpha^3+\alpha^4$	(10111)
α^{15}	$1+\alpha+\alpha^2+\alpha^3+\alpha^4$	(11111)
α^{16}	$1+\alpha+\alpha^3+\alpha^4$	(11011)
α^{17}	$1+\alpha+\alpha^4$	(11001)
α^{18}	$1+\alpha$	(11000)
α^{19}	$\alpha+\alpha^2$	(01100)
α^{20}	$\alpha^2+\alpha^3$	(00110)
α^{21}	$\alpha^3+\alpha^4$	(00011)
α^{22}	$1+\alpha^2+\alpha^4$	(10101)
α^{23}	$1+\alpha+\alpha^2+\alpha^3$	(11110)
α^{24}	$\alpha+\alpha^2+\alpha^3+\alpha^4$	(01111)
α^{25}	$1+\alpha^3+\alpha^4$	(10011)
α^{26}	$1+\alpha+\alpha^2+\alpha^4$	(11101)
α^{27}	$1+\alpha+\alpha^3$	(11010)
α^{28}	$\alpha+\alpha^2+\alpha^4$	(01101)
α^{29}	$1+\alpha^3$	(10010)
α^{30}	$\alpha+\alpha^4$	(01001)
$\alpha^{31}=1$		

divide α^j by α^i, we simply multiply α^j by the multiplicative inverse α^{31-i} of α^i. For example, $\alpha^7 \div \alpha^9 = \alpha^7 \cdot \alpha^{31-9} = \alpha^7 \cdot \alpha^{22} = \alpha^{29}$. To add two elements, α^i and α^j, we may use either their polynomial representations or vector representations. Suppose we want to add α^{11} and α^{17} using their vector representations. From Table 2.10, we find that the vector representations of α^{11} and α^{17} are (11100) and (11001), respectively. On adding these two vectors, we have

$$(11100) + (11001) = (00101).$$

From Table 2.10, we find that the vector (00101) represents the element α^7. In polynomial form, it is $\alpha^2 + \alpha^4$.

Besides α, $p(X)$ has four other roots. These four other roots can be found by substituting X of $p(X) = 1 + X^2 + X^5$ by the elements of $\mathrm{GF}(2^5)$ in turn. By doing so, we find that the other four roots are α^2, α^4, α^8, and α^{16}. For example, on replacing X of $p(X) = 1 + X^2 + X^5$ by α^8, we have

$$p(\alpha^8) = 1 + \alpha^{16} + \alpha^{40} = 1 + \alpha^{16} + \alpha^9.$$

Using polynomial representations of the elements in the above sum, we have

$$\begin{aligned} p(\alpha^8) &= 1 + (1 + \alpha + \alpha^3 + \alpha^4) + (\alpha + \alpha^3 + \alpha^4) \\ &= (1+1) + (1+1)\alpha + (1+1)\alpha^3 + (1+1)\alpha^4 \\ &= 0 + 0 \cdot \alpha + 0 \cdot \alpha^3 + 0 \cdot \alpha^4 \\ &= 0, \end{aligned}$$

where addition of two coefficients is modulo-2 addition. Hence, α^8 is a root of $p(X) = 1 + X^2 + X^5$. Similarly, we can prove that α^2, α^4, and α^{16} are also roots of $p(X) = 1 + X^2 + X^5$. Later we will show how to find the roots of $p(X)$ in a much easier way.

Example 2.15. In this example, we construct an extension field $\mathrm{GF}(3^2)$ of the prime field $\mathrm{GF}(3)$ of degree 2 using the primitive polynomial $p(X) = 2 + X + X^2$ over $\mathrm{GF}(3)$ (taken from the table given in Appendix A). The modulo-3 addition and multiplication for $\mathrm{GF}(3)$ are given by Tables 2.11 and 2.12. Let α be a root of $p(X)$. Then $p(\alpha) = 2 + \alpha + \alpha^2 = 0$ and $\alpha^2 = -2 - \alpha$. From Table 2.11, we see that the additive inverse -2 of 2 is 1 and the additive inverse -1 of 1 is 2. Consequently, $\alpha^2 = -2 - \alpha$ can be put as $\alpha^2 = 1 + 2\alpha$. Using the identities $\alpha^2 = 1 + 2\alpha$ and $\alpha^8 = 1$, we can construct the extension field $\mathrm{GF}(3^2)$. The elements of $\mathrm{GF}(3^2)$ in three forms are given in Table 2.13. Since $p(X)$ has degree 2, it has two roots. Besides α, the other root of $p(X)$ is α^3. To see this, we replace X by α^3. This results in $p(\alpha^3) = 2 + \alpha^3 + \alpha^6$. On replacing α^3 and α^6 by their polynomial representations, we have $p(\alpha^3) = 2 + 2 + 2\alpha + 2 + \alpha = 0$. From Table 2.13, we also see that $\mathrm{GF}(3)$ is a subfield of $\mathrm{GF}(3^2)$.

As we have shown, to construct an extension field $\mathrm{GF}(p^m)$ of degree m of the prime field $\mathrm{GF}(p)$, we need a primitive polynomial $p(X)$ of degree m over $\mathrm{GF}(p)$. If a different primitive polynomial $p^*(X)$ of degree m over $\mathrm{GF}(p)$ is used, then the construction results in an extension field $\mathrm{GF}^*(p^m)$ of degree m of the prime field $\mathrm{GF}(p)$ that has the *same set of elements* as that of $\mathrm{GF}(p^m)$. There is a *one-to-one correspondence* between $\mathrm{GF}(p^m)$ and $\mathrm{GF}^*(p^m)$ such that, if $a \leftrightarrow a^*$, $b \leftrightarrow b^*$, and $c \leftrightarrow c^*$, then

$$\begin{gathered} a + b \leftrightarrow a^* + b^*, \\ a \cdot b \leftrightarrow a^* \cdot b^*, \\ (a + b) + c \leftrightarrow a^* + (b^* + c^*), \\ (a \cdot b) \cdot c \leftrightarrow a^* \cdot (b^* \cdot c^*), \\ a \cdot (b + c) \leftrightarrow a^* \cdot (b^* + c^*). \end{gathered}$$

Table 2.11. Modulo-3 addition

+	0	1	2
0	0	1	2
1	1	2	0
2	2	0	1

Table 2.12. Modulo-3 multiplication

·	0	1	2
0	0	0	0
1	0	1	2
2	0	2	1

Table 2.13. GF(3^2) generated by the primitive polynomial $p(X) = 2 + X + X^2$ over GF(3)

Power representation	Polynomial representation	Vector representation
0	0	(00)
1	1	(10)
α	α	(01)
α^2	$1 + 2\alpha$	(21)
α^3	$2 + 2\alpha$	(22)
α^4	2	(20)
α^5	2α	(02)
α^6	$2 + \alpha$	(21)
α^7	$1 + \alpha$	(11)
$\alpha^8 = 1$		

GF(p^m) and GF*(p^m) are said to be *isomorphic*. That is, GF(p^m) and GF*(p^m) are structurally identical. In this sense, we may say that any primitive polynomial $p(X)$ of degree m over the prime field GF(p) gives the same extension field GF(p^m). This implies uniqueness of the extension field GF(p^m).

The above construction of an extension field GF(p^m) of a prime field GF(p) can be generalized. Let GF(q) be a finite field with q elements, where q is a power of a prime, say $q = p^s$. It is known in modern algebra that, *for any positive integer m, there exists a primitive polynomial $p(X)$ of degree m over GF(q)*. Let α be a root of $p(X)$. Then the construction of the extension field GF(q^m) of GF(q) of degree m is exactly the same as the construction of the extension field GF(p^m) of the prime field GF(p) of degree m as given above.

2.6.2 Some Fundamental Properties of Finite Fields

As shown in the previous subsection, for any prime field GF(p) and any positive integer m, it is possible to construct an extension field GF(p^m) of degree m with GF(p) as the ground field. In fact, for any finite field GF(q) with q elements, q must be a power of a prime p and GF(q) contains the prime field GF(p) as a subfield. Proof of this can be found in any text on modern algebra and the theory of finite fields. The extension field GF(p^m) has the same characteristic as that of the prime ground field GF(p), i.e., p. It has also been shown at the end of the last subsection that, for any positive integer m, it is possible to construct an extension field GF(q^m) of degree m with GF(q) as the ground field. Since GF(q) contains GF(p) as a subfield, GF(q^m) contains both GF(q) and GF(p) as subfields. GF(q^m) may be viewed as an extension field of the prime field GF(p) and has characteristic p.

In Section 2.3, some fundamental properties of finite fields have been presented. In the following, we will present some additional fundamental properties of finite fields that are important for constructing error-correcting codes. First, we start with an important property of the ring of polynomials over GF(q) without proof. Let $f(X) = f_0 + f_1X + \cdots + f_kX^k$ be a polynomial over GF(q). Then, for any non-negative integer t, the following equality holds:

$$[f(X)]^{q^t} = f(X^{q^t}). \tag{2.41}$$

Theorem 2.11. Let $f(X)$ be a polynomial over GF(q). Let β be an element of the extension field GF(q^m) of GF(q). If β is a root of $f(X)$, then, for any non-negative integer t, β^{q^t} is also a root of $f(X)$.

Proof. The proof of this theorem follows directly from the equality of (2.41). On substituting X by β in (2.41), we have

$$[f(\beta)]^{q^t} = f(\beta^{q^t}).$$

Since β is a root of $f(X)$, $f(\beta) = 0$. Then, it follows from the equality above that $f(\beta^{q^t}) = 0$. Hence β^{q^t} is a root of $f(X)$. □

The element β^{q^t} is called a *conjugate* of β. Theorem 2.11 says that if β, an element of GF(q^m), is a root of a polynomial $f(X)$ over GF(q) then all its conjugates, $\beta, \beta^q, \beta^{q^2}, \ldots$, are also roots of $f(X)$.

Theorem 2.12. If β is an element of order n in GF(q^m), then all its conjugates have the same order n. If β is a primitive element of GF(q^m), then all its conjugates are primitive elements. (The proof is left as an exercise.)

Example 2.16. Consider the polynomial $f(X) = 1 + X^3 + X^5$ over GF(2). This polynomial has α^{15}, an element of GF(2^5) given by Table 2.10, as a root. To verify this, we use Table 2.10 and the identity $\alpha^{31} = 1$. On substituting the variable X of $f(X)$ by α^{15}, we

obtain

$$\begin{aligned} f(\alpha^{15}) &= 1 + (\alpha^{15})^3 + (\alpha^{15})^5 = 1 + \alpha^{45} + \alpha^{75} \\ &= 1 + \alpha^{14} + \alpha^{13} \\ &= 1 + (1 + \alpha^2 + \alpha^3 + \alpha^4) + (\alpha^2 + \alpha^3 + \alpha^4) \\ &= (1+1) + (1+1)\cdot\alpha^2 + (1+1)\cdot\alpha^3 + (1+1)\cdot\alpha^4 \\ &= 0 + 0\cdot\alpha^2 + 0\cdot\alpha^3 + 0\cdot\alpha^4 = 0+0+0+0 = 0. \end{aligned}$$

The conjugates of α^{15} are

$$(\alpha^{15})^2 = \alpha^{30}, \quad (\alpha^{15})^{2^2} = \alpha^{60} = \alpha^{29},$$

$$(\alpha^{15})^{2^3} = \alpha^{120} = \alpha^{27}, \quad (\alpha^{15})^{2^4} = \alpha^{240} = \alpha^{23}.$$

Note that $(\alpha^{15})^{2^5} = \alpha^{480} = \alpha^{15}$. It follows from Theorem 2.11 that α^{30}, α^{29}, α^{27}, and α^{23} are also roots of $f(X) = 1 + X^3 + X^5$. Consider the conjugate α^{23}. On replacing X by α^{23} in $f(X)$, we have

$$\begin{aligned} f(\alpha^{23}) &= 1 + (\alpha^{23})^3 + (\alpha^{23})^5 = 1 + \alpha^{69} + \alpha^{115} \\ &= 1 + \alpha^7 + \alpha^{22} = 1 + (\alpha^2 + \alpha^4) + (1 + \alpha^2 + \alpha^4) \\ &= 0. \end{aligned}$$

Hence α^{23} is a root of $f(X)$. Since the degree of $f(X)$ is 5, it must have five roots. The elements α^{15}, α^{23}, α^{27}, α^{29}, and α^{30} give all five roots of $f(X) = 1 + X^3 + X^5$. So all the roots of $f(X)$ are in GF(2^5). With some computational effort, we can check that α^{15} and its conjugates have order 31 and that they are all primitive.

Example 2.17. Consider the polynomial $f(X) = 2 + 2X + X^2$ over GF(3) given by Tables 2.11 and 2.12. Using modulo-3 addition and multiplication, the extension field GF(3^2) of GF(3) given by Table 2.13, and the fact $\alpha^8 = 1$, we find that α^5 and its conjugate $(\alpha^5)^3 = \alpha^{15} = \alpha^7$ are roots of $f(X) = 2 + 2X + X^2$. Suppose we take the powers of α^5,

$$\alpha^5, \quad (\alpha^5)^2 = \alpha^{10} = \alpha^2, \quad (\alpha^5)^3 = \alpha^{15} = \alpha^7, \quad (\alpha^5)^4 = \alpha^{20} = a^4,$$

$$(\alpha^5)^5 = \alpha^{25} = \alpha, \quad (\alpha^5)^6 = \alpha^{30} = \alpha^6, \quad (\alpha^5)^7 = \alpha^{35} = a^3,$$

$$(\alpha^5)^8 = \alpha^{40} = 1.$$

We see that the powers of α^5 give all the elements of GF(3^2). Therefore, α^5 is a primitive element. Similarly, we can show that the powers of the conjugate α^7 of α^5 also give all the elements of GF(3^2). Hence α^7 is also a primitive element of GF(3^2).

Theorem 2.13. Let $\mathrm{GF}(q^m)$ be an extension field of $\mathrm{GF}(q)$. The $q^m - 1$ nonzero elements of $\mathrm{GF}(q^m)$ form all the roots of the polynomial $X^{q^m-1} - 1$ over $\mathrm{GF}(q)$.

Proof. Let β be a nonzero element of $\mathrm{GF}(q^m)$. On substituting X of $X^{q^m-1} - 1$ by β, we have $\beta^{q^m-1} - 1$. However, it follows from Theorem 2.6 that $\beta^{q^m-1} = 1$. Hence, $\beta^{q^m-1} - 1 = 1 - 1 = 0$. This says that β is a root of $X^{q^m-1} - 1$. Therefore, every nonzero element of $\mathrm{GF}(q^m)$ is a root of $X^{q^m-1} - 1$. Since the degree of $X^{q^m-1} - 1$ is $q^m - 1$, the $q^m - 1$ nonzero elements of $\mathrm{GF}(q^m)$ form all the roots of X^{q^m-1}. □

Since the zero element 0 of $\mathrm{GF}(q^m)$ is the root of X, Theorem 2.13 has the following corollary.

Corollary 2.1. The q^m elements of $\mathrm{GF}(q^m)$ form all the roots of $X^{q^m} - X$.

It follows from Corollary 2.1 that

$$X^{q^m} - X = \prod_{\beta \in \mathrm{GF}(q^m)} (X - \beta).$$

Since any element β in $\mathrm{GF}(q^m)$ is a root of the polynomial $X^{q^m} - X$ over $\mathrm{GF}(q)$, β may be a root of a polynomial over $\mathrm{GF}(q)$ with a degree less than q^m.

Definition 2.22. Let β be an element of $\mathrm{GF}(q^m)$, an extension field of $\mathrm{GF}(q)$. Let $\phi(X)$ be the monic polynomial of smallest degree over $\mathrm{GF}(q)$ that has β as a root, i.e., $\phi(\beta) = 0$. This polynomial $\phi(X)$ is called the *minimal polynomial* of β.

It is clear that the minimal polynomial of the 0-element of $\mathrm{GF}(q^m)$ is X. In the following, we present a number of properties of minimal polynomials of elements of $\mathrm{GF}(q^m)$.

Theorem 2.14. The minimal polynomial $\phi(X)$ of a field element β is irreducible.

Proof. Suppose that $\phi(X)$ is not irreducible. Then $\phi(X)$ can be expressed as a product of two factors, $\phi(X) = \phi_1(X)\phi_2(X)$, where both $\phi_1(X)$ and $\phi_2(X)$ are monic and have degrees greater than 0 but less than that of $\phi(X)$. Since $\phi(\beta) = \phi_1(\beta)\phi_2(\beta) = 0$, either $\phi_1(\beta) = 0$ or $\phi_2(\beta) = 0$. This contradicts the hypothesis that $\phi(X)$ is a polynomial of smallest degree over $\mathrm{GF}(q)$ that has β as a root. Hence, $\phi(X)$ must be irreducible. □

Theorem 2.15. The minimal polynomial $\phi(X)$ of a field element β is unique.

Proof. Suppose that β has two different minimal polynomials, $\phi(X)$ and $\phi'(X)$. Then these two minimal polynomials are monic and have the same degree, say k. Then $m(X) = \phi(X) - \phi'(X)$ is a polynomial of degree $k - 1$ or less over $\mathrm{GF}(q)$. If $m(X)$ is not monic, it can be made monic by multiplying it by the inverse of the

coefficient of the highest degree of $m(X)$. Since both $\phi(X)$ and $\phi'(X)$ have β as a root, $m(\beta) = \phi(\beta) - \phi'(\beta) = 0 - 0 = 0$. If $m(X) \neq 0$, it is a polynomial over GF(q) with degree less than that of $\phi(X)$ and $\phi'(X)$ that has β as a root. This contradicts the hypothesis that $\phi(X)$ and $\phi'(X)$ are the smallest-degree polynomials over GF(q) with β as a root. Therefore, $m(X)$ must be the zero polynomial and $\phi(X) = \phi'(X)$. This proves the theorem. □

Theorem 2.16. Let $f(X)$ be a polynomial over GF(q). Let $\phi(X)$ be the minimal polynomial of an element β in GF(q^m). If β is a root of $f(X)$, then $f(X)$ is divisible by $\phi(X)$.

Proof. On dividing $f(X)$ by $\phi(X)$ we obtain

$$f(X) = a(X)\phi(X) + r(X),$$

where $r(X)$ is the remainder with degree less than that of $\phi(X)$. On substituting X of the equation above with β, we have $f(\beta) = a(\beta)\phi(\beta) + r(\beta)$. Since both $f(X)$ and $\phi(X)$ have β as a root, we have $r(\beta) = 0$. This says that β is also a root of $r(X)$. If $r(X) \neq 0$, then $r(X)$ is a polynomial over GF(q) with degree lower than the degree of $\phi(X)$, that has β as a root. This contradicts the fact that $\phi(X)$ is the polynomial over GF(q) of smallest degree that has β as a root. Hence, we must have $r(X) = 0$ and $f(X)$ must be divisible by $\phi(X)$. □

A direct consequence of the above theorem is the following corollary.

Corollary 2.2. Let $p(X)$ be a monic irreducible polynomial over GF(q). Let β be an element of GF(q^m) and $\phi(X)$ be its minimal polynomial. If β is a root of $p(X)$, then $p(X) = \phi(X)$.

Corollary 2.2 simply says that, if a monic irreducible polynomial has β as a root, it is the minimal polynomial $\phi(X)$ of β. It follows form Theorem 2.11 that β and its conjugates, $\beta^q, \ldots, \beta^{q^t}, \ldots$ are roots of $\phi(X)$. Let e be the smallest positive integer such that $\beta^{q^e} = \beta$. Then $\beta, \beta^q, \ldots, \beta^{q^{e-1}}$ are all the distinct conjugates of β. Since $\beta^{q^m} = \beta$ and e is the smallest positive integer such that $\beta^{q^e} = \beta$, we must have $e \leq m$. In fact, e divides m (proof is left as an exercise).

Theorem 2.17. Let $\phi(X)$ be the minimal polynomial of an element β of GF(q^m). Then $\phi(X)$ divides $X^{q^m} - X$.

Proof. The proof of this theorem follows from Corollary 2.1 and Theorem 2.16.□

It follows from Theorem 2.11, Theorem 2.14, and Corollary 2.2 that we have the following theorem.

Theorem 2.18. Let β be an element of GF(q^m). Then, all its conjugates, $\beta, \beta^q, \ldots, \beta^{q^{e-1}}$, have the same minimal polynomial.

A direct consequence of Theorems 2.17 and 2.18 is that $X^{q^m} - X$ is equal to the product of the distinct minimal polynomials of the elements of $\mathrm{GF}(q^m)$.

The next theorem tells us how to determine the minimal polynomials of elements of the extension field $\mathrm{GF}(q^m)$ of degree m of $\mathrm{GF}(q)$. The proof of this theorem is left as an exercise.

Theorem 2.19. Let $\phi(X)$ be the minimal polynomial of an element β of $\mathrm{GF}(q^m)$. Let e be the smallest positive integer such that $\beta^{q^e} = \beta$. Then

$$\phi(X) = \prod_{i=0}^{e-1} (X - \beta^{q^i}). \tag{2.42}$$

Since $e \leq m$, the degree of the minimal polynomial of any element β of $\mathrm{GF}(q^m)$ is m or less.

Example 2.18. Consider the extension field $\mathrm{GF}(2^5)$ of the binary field $\mathrm{GF}(2)$ given by Table 2.10. Let $\beta = \alpha^3$. The conjugates of β are

$$\beta^2 = (\alpha^3)^2 = \alpha^6, \quad \beta^{2^2} = (\alpha^3)^{2^2} = \alpha^{12},$$
$$\beta^{2^3} = (\alpha^3)^{2^3} = \alpha^{24}, \quad \beta^{2^4} = (\alpha^3)^{2^4} = \alpha^{48} = \alpha^{17}.$$

Note that $\beta^{2^5} = (\alpha^3)^{2^5} = \alpha^{96} = \alpha^3$. Hence, $e = 5$. The minimal polynomial of α^3 is then

$$\phi(X) = (X - \alpha^3)(X - \alpha^6)(X - \alpha^{12})(X - \alpha^{17})(X - \alpha^{24}).$$

On multiplying out the right-hand side of the equation above with the aid of Table 2.10 and the modulo-2 addition of coefficients, we obtain

$$\phi(X) = 1 + X^2 + X^3 + X^4 + X^5.$$

Example 2.19. Consider the extension field $\mathrm{GF}(3^2)$ of degree 2 of the prime field $\mathrm{GF}(3)$ given by Table 2.13. Let $\beta = \alpha^5$. The only conjugate of β is α^7. Then the minimal polynomial of α^3 and α^7 is given by

$$\begin{aligned}\phi(X) &= (X - \alpha^5)(X - \alpha^7) \\ &= \alpha^{12} - (\alpha^5 + \alpha^7)X + X^2.\end{aligned}$$

With the aid of Table 2.13, we find that

$$\begin{aligned}\phi(X) &= \alpha^4 - (2\alpha + 1 + \alpha)X + X^2 \\ &= 2 - X + X^2 = 2 + 2X + X^2.\end{aligned}$$

In the construction of the extension field $\mathrm{GF}(q^m)$ of degree m of $\mathrm{GF}(q)$, we use a primitive polynomial $p(X)$ of degree m over $\mathrm{GF}(q)$ and require that the element α be a root of $p(X)$. Since the powers of α generate all the nonzero elements of $\mathrm{GF}(q^m)$, α is a primitive element. It follows from Theorem 2.12 that all its conjugates are primitive elements. There is a simple rule to determine whether a nonzero element of $\mathrm{GF}(q^m)$ is primitive. For $0 < j < q^m - 1$, a nonzero element α^j of $\mathrm{GF}(q^m)$ is a primitive element if and only if j and $q^m - 1$ are relatively prime. Consequently, the number of primitive elements in $\mathrm{GF}(q^m)$ is given by $K_p = \phi(q^m - 1)$, which is known as *Euler's formula*. To determine K_p, we first factor $q^m - 1$ as a product of powers of primes,

$$q^m - 1 = p_1^{k_1} p_2^{k_2} \cdots p_t^{k_t}.$$

Then

$$K_p = (q^m - 1) \prod_{i=1}^{t} (1 - 1/p_i). \tag{2.43}$$

If $q^m - 1$ is a prime, then $K_p = q^m - 2$. In this case, except for the unit element 1, every nonzero element of $\mathrm{GF}(q^m)$ is a primitive element.

Example 2.20. Consider the extension field $\mathrm{GF}(2^6)$ of degree 6 of the binary field $\mathrm{GF}(2)$. The number $2^6 - 1 = 63$ can be factored as the product $63 = 3^2 \times 7$. It follows from (2.43) that $\mathrm{GF}(2^6)$ consists of

$$K_p = 63(1 - 1/3)(1 - 1/7) = 63 \times (2/3)(6/7) = 36$$

primitive elements.

2.6.3 Additive and Cyclic Subgroups

Consider the extension field $\mathrm{GF}(q^m)$ of degree m with $\mathrm{GF}(q)$ as the ground field. Let α be a primitive element of $\mathrm{GF}(q^m)$. Then $\alpha^0 = 1, \alpha, \ldots, \alpha^{m-1}$ form a polynomial basis of $\mathrm{GF}(q^m)$ over $\mathrm{GF}(q)$. For $i = -\infty, 0, 1, \ldots, q^m - 2$, every element α^i can be expressed as a linear sum (or combination) of $\alpha^0, \alpha, \ldots, \alpha^{m-1}$ as follows:

$$\alpha^i = a_{i,0}\alpha^0 + a_{i,1}\alpha + \cdots + a_{i,m-1}\alpha^{m-1}, \tag{2.44}$$

with $a_{i,j} \in \mathrm{GF}(q)$. For $0 \le t < m - 1$, Let $j_0, j_1, \ldots, j_{t-1}$ be t integers such that $0 \le j_0 < j_1 < \cdots < j_{t-1} \le m - 1$. Consider the q^t elements of $\mathrm{GF}(q^m)$ of the following form:

$$\alpha^k = c_{k,0}\alpha^{j_0} + c_{k,1}\alpha^{j_1} + \cdots + c_{k,t-1}\alpha^{j_{t-1}}, \tag{2.45}$$

where $c_{k,j} \in \mathrm{GF}(q)$. Let S denote the set of these q^t elements. S forms a subgroup of the additive group of $\mathrm{GF}(q^m)$ and is called an additive subgroup of $\mathrm{GF}(q^m)$.

Example 2.21. Consider the extension field $\mathrm{GF}(2^5)$ of the binary field $\mathrm{GF}(2)$ given by Table 2.10 (see Example 2.14). The elements, α^0, α, α^2, α^3, α^4, form a polynomial basis. Let $j_0 = 1$, $j_1 = 3$, and $j_2 = 4$. Consider the following eight linear sums of α, α^3, and α^4:

$$\alpha^k = c_{k,0}\alpha + c_{k,1}\alpha^3 + c_{k,2}\alpha^4,$$

with $c_{k,j} \in \mathrm{GF}(2)$. Using Table 2.10, we find that the linear sums of α, α^3, and α^4 give the following eight elements:

$$\alpha^{-\infty} = 0, \ \ \alpha, \ \ \alpha^3, \ \ \alpha^4, \ \ \alpha^6, \ \ \alpha^9, \ \ \alpha^{21}, \ \ \alpha^{30}.$$

These eight elements give a subgroup of the additive group of $\mathrm{GF}(2^5)$.

Suppose $q^m - 1$ is not a prime. It can be factored as a product of two positive integers greater than 1, say $q^m - 1 = kn$. Let α be a primitive element of $\mathrm{GF}(q^m)$ and $\beta = \alpha^k$. Then β is a nonzero element of order n, i.e., n is the smallest positive integer such that $\beta^n = 1$ (see Section 2.3). The powers $\beta^0 = 1, \beta, \ldots, \beta^{n-1}$ form a subgroup of the multiplicative group of $\mathrm{GF}(q^m)$. Such a subgroup is called a *cyclic subgroup* of $\mathrm{GF}(q^m)$ with $\beta = \alpha^k$ as the generator.

Example 2.22. Consider the extension field $\mathrm{GF}(3^2)$ of the prime field $\mathrm{GF}(3)$ given by Table 2.13. The integer $3^2 - 1 = 8$ can be factored as a product of $k = 2$ and $n = 4$. Let $\beta = \alpha^2$. The order of β is 4. The four distinct powers of β are $\beta^0 = 1, \beta = \alpha^2, \beta^2 = \alpha^4, \beta^3 = \alpha^6$. These four elements form a cyclic subgroup of $\mathrm{GF}(3^2)$.

2.7 Finite Geometries

In contrast to ordinary geometry, a finite geometry has finite numbers of points, lines, and flats. However, finite and ordinary geometries also have some fundamental structures in common, such as the following: (1) two points are connected by a line; (2) two lines are either disjoint (i.e., have no point in common) or they intersect at one and only one point (i.e., they have one point in common); and (3) if two lines have two points in common, they are the same line. In this section, we present two families of finite geometries over finite fields, namely Euclidean and projective geometries. These geometries will be used to construct low-density parity-check codes for iterative decoding in Chapters 10 and 11.

2.7.1 Euclidean Geometries

Let m be a positive integer greater than unity and $\mathrm{GF}(q)$ be a finite field with q elements, where q is a power of prime p, say p^s, with $s \geq 1$. The *m-dimensional Euclidean geometry over* $\mathrm{GF}(q)$, denoted by $\mathrm{EG}(m,q)$, consists of points, lines,

and flats [1–4]. Each point is an m-tuple over GF(q). Therefore, there are

$$n = q^m \tag{2.46}$$

points in EG(m,q), which actually form the vector space of all the m-tuples over GF(q), denoted by V_m. The all-zero m-tuple, $(0, 0, \ldots, 0)$, is referred to as the *origin* of the geometry.

A *line* in EG(m,q) is either a one-dimensional subspace or a coset of a one-dimensional subspace of the vector space V_m of all the m-tuples over GF(q). There are

$$J_{\mathrm{EG}}(m, 1) = q^{m-1}(q^m - 1)/(q - 1) \tag{2.47}$$

lines in EG(m,q) and each line consists of q points. Any two points are connected by one and only one line. Any two lines are either disjoint (i.e., have no point in common) or they intersect at one and only one point (i.e., have one point in common). Lines that correspond to two cosets of the same one-dimensional subspace of V_m do not have any point in common and they are said to be *parallel*. Each one-dimensional subspace of V_m has q^{m-1} cosets. Lines in EG(m,q) that correspond to these q^{m-1} cosets are parallel to each other and they form a *parallel bundle*. A parallel bundle of lines contains all the points of EG(m,q), each point appears once and only once on only one line in the parallel bundle. It follows from (2.47) and the fact that each parallel bundle of lines consists of q^{m-1} parallel lines that the lines of EG(m,q) can be partitioned into

$$K_{\mathrm{EG}}(m, 1) = (q^m - 1)/(q - 1) \tag{2.48}$$

parallel bundles.

For any point $\mathbf{a}$ in EG(m,q), there are exactly

$$g_{\mathrm{EG}}(m, 1, 0) = (q^m - 1)/(q - 1) \tag{2.49}$$

lines in EG(m,q) that intersect at $\mathbf{a}$ (this will be shown later). Note that $K_{\mathrm{EG}}(m, 1) = g_{\mathrm{EG}}(m, 1, 0)$. The $g_{\mathrm{EG}}(m, 1, 0)$ lines in EG(m,q) that intersect at a given point $\mathbf{a}$ are said to form an *intersecting bundle of lines* at $\mathbf{a}$. A line that contains a point $\mathbf{a}$ is said to pass through $\mathbf{a}$.

For $0 \leq \mu \leq m$, a *μ-flat* in EG(m,q) is either a μ-dimensional subspace or a coset of a μ-dimensional subspace of the vector space V_m over GF(q). For $\mu = 0$, a 0-flat is simply a point of EG(m,q). For $\mu = 1$, a 1-flat is simply a line of EG(m,q). Two flats that correspond to two cosets of the same μ-dimensional subspace of V_m are said to be parallel, and they do not have any point in common. For each μ-dimensional subspace of V_m, there are $q^{m-\mu}$ parallel μ-flats in EG(m,q) and they form a parallel bundle of μ-flats. These parallel μ-flats are mutually disjoint and they contain all the points of EG(m,q), each point appearing once and only once on only one of these parallel flats. In EG(m,q), there are

$$J_{\mathrm{EG}}(m, \mu) = q^{m-\mu} \prod_{i=1}^{\mu} \frac{q^{m-i+1} - 1}{q^{\mu-i+1} - 1} \tag{2.50}$$

μ-flats. These μ-flats can be partitioned into

$$K_{\mathrm{EG}}(m,\mu) = \prod_{i=1}^{\mu} \frac{q^{m-i+1}-1}{q^{\mu-i+1}-1} \tag{2.51}$$

parallel bundles of μ-flats.

For $1 \le \mu < m$, if two μ-flats are not disjoint, they intersect on a flat of smaller dimension. The largest flat that two μ-flats can intersect on is a $(\mu-1)$-flat. If two μ-flats in EG(m,q) intersect on a $(\mu-1)$-flat, then they can intersect on one and only one $(\mu-1)$-flat. The number of μ-flats in EG(m,q) that intersect on a given $(\mu-1)$-flat is given by

$$g_{\mathrm{EG}}(m,\mu,\mu-1) = \frac{q^{m-u+1}-1}{q-1}. \tag{2.52}$$

As shown in Section 2.6.1, GF(q^m) as an extension field of GF(q) can be viewed as an m-dimensional vector space V_m of all m-tuples over GF(q); each element in GF(q^m) can be represented by an m-tuple over GF(q). Therefore, GF(q^m) as an extension field of GF(q) is a realization of EG(m,q). Let α be a primitive element of GF(q^m). Then the elements, $\alpha^{-\infty}=0, \alpha^0=1, \alpha, \ldots, \alpha^{q^m-2}$, of GF($q^m$) represent the $n=q^m$ points of EG(m,q), and the 0-element represents the origin of the geometry.

Using GF(q^m) as a realization of the m-dimensional Euclidean geometry EG(m,q) over GF(q), it is much easier to develop the fundamental properties of the geometry. Hereafter, we refer to the powers of α as points in EG(m,q). Let α^{j_1} be a non-origin point in EG(m,q). The set of q points

$$\{\beta\alpha^{j_1} : \beta \in \mathrm{GF}(q)\} \tag{2.53}$$

constitutes a line (or 1-flat) of EG(m,q). For $\beta = 0$, $0 \cdot \alpha^{j_1} = 0$ is the origin point of EG(m,q). Therefore, this line passes through the origin of EG(m,q). Viewing the q points on this line as vectors in the vector space V_m of all the m-tuples over GF(q), this line forms a one-dimensional subspace of V_m. For convenience, we use the notation $\{\beta\alpha^{j_1}\}$ to denote this line. Let α^{j_0} and α^{j_1} be two independent points in EG(m,q). Then the collection of q points

$$\{\alpha^{j_0} + \beta\alpha^{j_1}\}, \tag{2.54}$$

with $\beta \in \mathrm{GF}(q)$, forms a line passing through the point α^{j_0}. We may view this as a line connecting the points α^{j_0} and α^{j_1}. This line is simply a coset of the one-dimensional subspace $\{\beta\alpha^{j_1}\}$. The q^{m-1} cosets of $\{\beta\alpha^{j_1}\}$ form a parallel bundle of lines in EG(m,q). Since there are $(q^m-1)/(q-1)$ different one-dimensional subspaces in GF(q^m) (viewed as an m-dimensional vector space V_m over GF(q)) and each one-dimensional subspace of GF(q^m) has q^{m-1} cosets, we readily see that the number of lines in EG(m,q) is given by (2.47).

Let α^{j_0}, α^{j_1}, and α^{j_2} be three points in EG(m,q) that are pair-wise linearly independent. Then $\{\alpha^{j_0}+\beta\alpha^{j_1}\}$ and $\{\alpha^{j_0}+\beta\alpha^{j_2}\}$ form two different lines in EG(m,q)

that intersect at the point α^{j_0}. On the basis of this fact, we readily determine that the number of lines in EG(m,q) that intersect at any point in EG(m,q) is given by (2.49).

Example 2.23. Consider the two-dimensional Euclidean geometry EG(2,2^2) over GF(2^2). This geometry consists of $2^{2\times 2} = 16$ points, each point is a 2-tuple over GF(2^2). Each line in EG(2,2^2) consists of four points. Consider the Galois field GF(2^4) generated by the primitive polynomial $1 + X + X^4$ over GF(2) given by Table 2.14. Let α be a primitive element of GF(2^4) and $\beta = \alpha^5$. The order of β is 3. The set

$$\mathrm{GF}(2^2) = \{0, \beta^0 = 1, \beta = \alpha^5, \beta^2 = \alpha^{10}\}$$

forms the subfield GF(2^2) of GF(2^4). Every element α^i in GF(2^4) can be expressed as the following polynomial form:

$$\alpha^i = \beta_0\alpha^0 + \beta_1\alpha,$$

with $\beta_0, \beta_1 \in \mathrm{GF}(2^2)$. In vector form, α^i is represented by the 2-tuple (β_0, β_1) over GF(2^2). GF(2^4) as an extension field of GF(2^2) is given by Table 2.15. Regard GF(2^4) as a realization of the two-dimensional Euclidean geometry EG(2,2^2) over GF(2^2). Then

$$\mathcal{L}_0 = \{\beta_1\alpha\} = \{0, \alpha, \alpha^6, \alpha^{11}\}$$

with $\beta_1 \in \mathrm{GF}(2^2)$ forms a line passing through the origin of GF(2^4). The three lines parallel to $\mathcal{L}_0$ are (using Table 2.14 or Table 2.15)

$$\begin{aligned}
\mathcal{L}_1 &= \{1 + \beta_1\alpha\} = \{1, \alpha^4, \alpha^{12}, \alpha^{13}\},\\
\mathcal{L}_2 &= \{\alpha^5 + \beta_1\alpha\} = \{\alpha^2, \alpha^3, \alpha^5, \alpha^9\},\\
\mathcal{L}_3 &= \{\alpha^{10} + \beta_1\alpha\} = \{\alpha^8, \alpha^7, \alpha^{10}, \alpha^{14}\}.
\end{aligned}$$

$\mathcal{L}_0$, $\mathcal{L}_1$, $\mathcal{L}_2$, and $\mathcal{L}_3$ together form a parallel bundle of lines in EG(2,2^2).

For every point in EG(2,2^4), there are $(2^{2\times 2} - 1)/(2^2 - 1) = 5$ lines intersecting at it. For example, consider the point α. The lines that intersect at α are

$$\begin{aligned}
\mathcal{L}'_0 &= \{\alpha + \beta_1\alpha^2\} = \{\alpha, \alpha^5, \alpha^{13}, \alpha^{14}\},\\
\mathcal{L}'_1 &= \{\alpha + \beta_1\alpha^3\} = \{\alpha, \alpha^9, \alpha^{10}, \alpha^{12}\},\\
\mathcal{L}'_2 &= \{\alpha + \beta_1\alpha^4\} = \{1, \alpha, \alpha^3, \alpha^7\},\\
\mathcal{L}'_3 &= \{\alpha + \beta_1\alpha^5\} = \{\alpha, \alpha^2, \alpha^4, \alpha^8\},\\
\mathcal{L}'_4 &= \{\alpha + \beta_1\alpha^6\} = \{0, \alpha, \alpha^{11}, \alpha^6\}.
\end{aligned}$$

These five lines form an intersecting bundle of lines at point α.

Besides the parallel and intersecting structures, the lines of EG(m,q) also have *cyclic structure*. Let EG*(m,q) be the subgeometry obtained from EG(m,q) by

Table 2.14. GF(2^4) generated by the primitive polynomial $p(X) = 1 + X + X^4$

Power representation	Polynomial representation							Vector representation
0	0							$(0\ 0\ 0\ 0)$
1	1							$(1\ 0\ 0\ 0)$
α			α					$(0\ 1\ 0\ 0)$
α^2					α^2			$(0\ 0\ 1\ 0)$
α^3							α^3	$(0\ 0\ 0\ 1)$
α^4	1	$+$	α					$(1\ 1\ 0\ 0)$
α^5			α	$+$	α^2			$(0\ 1\ 1\ 0)$
α^6					α^2	$+$	α^3	$(0\ 0\ 1\ 1)$
α^7	1	$+$	α			$+$	α^3	$(1\ 1\ 0\ 1)$
α^8	1			$+$	α^2			$(1\ 0\ 1\ 0)$
α^9			α			$+$	α^3	$(0\ 1\ 0\ 1)$
α^{10}	1	$+$	α	$+$	α^2			$(1\ 1\ 1\ 0)$
α^{11}			α	$+$	α^2	$+$	α^3	$(0\ 1\ 1\ 1)$
α^{12}	1	$+$	α	$+$	α^2	$+$	α^3	$(1\ 1\ 1\ 1)$
α^{13}	1			$+$	α^2	$+$	α^3	$(1\ 0\ 1\ 1)$
α^{14}	1					$+$	α^3	$(1\ 0\ 0\ 1)$

Table 2.15. GF(2^4) as an extension field GF(2^2)$= \{0, 1, \beta, \beta^2\}$ with $\beta = \alpha^5$

Power representation	Polynomial representation			Vector representation
0	0			$(0, 0)$
1	1			$(1, 0)$
α			α	$(0, 1)$
α^2	β	$+$	α	$(\beta, 1)$
α^3	β	$+$	$\beta^2\alpha$	(β, β^2)
α^4	1	$+$	α	$(1, 1)$
α^5	β			$(\beta, 0)$
α^6			$\beta\alpha$	$(0, \beta)$
α^7	β^2	$+$	$\beta\alpha$	(β^2, β)
α^8	β^2	$+$	α	$(\beta^2, 1)$
α^9	β	$+$	$\beta\alpha$	(β, β)
α^{10}	β^2			$(\beta^2, 0)$
α^{11}			$\beta^2\alpha$	$(0, \beta^2)$
α^{12}	1	$+$	$\beta^2\alpha$	$(1, \beta^2)$
α^{13}	1	$+$	$\beta\alpha$	$(1, \beta)$
α^{14}	β^2	$+$	$\beta^2\alpha$	(β^2, β^2)

removing the origin and all the lines passing through the origin. This subgeometry consists of $q^m - 1$ non-origin points and

$$J_{0,\mathrm{EG}}(m, 1) = (q^{m-1} - 1)(q^m - 1)/(q - 1) \tag{2.55}$$

lines not passing through the origin. Let $\mathcal{L} = \{\alpha^{j_0} + \beta\alpha^{j_1}\}$ be a line in $\mathrm{EG}^*(m,q)$. For $0 \le i < q^m - 1$, let

$$\alpha^i\mathcal{L} \triangleq \{\alpha^{j_0+i} + \beta\alpha^{j_1+i}\}.$$

Since α^{j_0} and α^{j_1} are linearly independent points, it is clear that α^{j_0+i} and α^{j_1+i} are also linearly independent points. Therefore, $\alpha^i\mathcal{L}$ is also a line in $\mathrm{EG}^*(m,q)$ that passes through the non-origin point α^{j_0+i}. It can be proved for $0 \le i, k < q^m - 2$, and $i \ne k$ that $\alpha^i\mathcal{L} \ne \alpha^k\mathcal{L}$ (this is left as an exercise). A line with this property is said to be *primitive*. Every line in $\mathrm{EG}(m,q)$ is primitive [5]. Then $\mathcal{L}, \alpha\mathcal{L}, \ldots, \alpha^{q^m-2}\mathcal{L}$ form $q^m - 1$ different lines in $\mathrm{EG}^*(m,q)$. Since $\alpha^{q^m-1} = 1$, $\alpha^{q^m-1}\mathcal{L} = \mathcal{L}$. The line $\alpha^i\mathcal{L}$ is called the ith *cyclic-shift* of $\mathcal{L}$. The set $\{\mathcal{L}, \alpha\mathcal{L}, \ldots, \alpha^{q^m-2}\mathcal{L}\}$ of $q^m - 1$ lines is said to form a *cyclic class* of lines in $\mathrm{EG}^*(m,q)$ of size $q^m - 1$. It follows from (2.55) that the lines in $\mathrm{EG}^*(m,q)$ can be partitioned into

$$K_{\mathrm{c,EG}}(m,1) = (q^{m-1} - 1)/(q - 1) \tag{2.56}$$

cyclic classes of size $q^m - 1$.

The mathematical formulation of lines in the m-dimensional Euclidean geometry $\mathrm{EG}(m,q)$ over $\mathrm{GF}(q)$ in terms of the extension field $\mathrm{GF}(q^m)$ of $\mathrm{GF}(q)$ can be extended to flats of $\mathrm{EG}(m,q)$. For $1 \le \mu < m$, let $\alpha^{j_1}, \alpha^{j_2}, \ldots, \alpha^{j_\mu}$ be μ linearly independent points of $\mathrm{EG}(m,q)$. The set of q^μ points

$$\mathcal{F}^{(\mu)} = \{\beta_1\alpha^{j_1} + \beta_2\alpha^{j_2} + \cdots + \beta_\mu\alpha^{j_\mu} \colon \beta_i \in \mathrm{GF}(q), 1 \le i \le \mu\} \tag{2.57}$$

forms a μ-flat in $\mathrm{EG}(m,q)$ passing through the origin (a μ-dimensional subspace of the m-dimensional vector space V_m over $\mathrm{GF}(q)$). For simplicity, we use $\{\beta_1\alpha^{j_1} + \beta_2\alpha^{j_2} + \cdots + \beta_\mu\alpha^{j_\mu}\}$ to denote the μ-flat given by (2.57). Let α^{j_0} be a point not on $\mathcal{F}^{(\mu)}$. Then

$$\alpha^{j_0} + \mathcal{F}^{(\mu)} = \{\alpha^{j_0} + \beta_1\alpha^{j_1} + \cdots + \beta_\mu\alpha^{j_\mu}\} \tag{2.58}$$

is a μ-flat passing through the point α^{j_0} and parallel to the μ-flat $\mathcal{F}^{(\mu)}$. It follows from (2.57) and (2.58) that, for each μ-flat of $\mathrm{EG}(m,q)$ passing through the origin, there are $q^{m-\mu} - 1$ μ-flats parallel to it. $\mathcal{F}^{(\mu)}$ and the $q^{m-\mu} - 1$ μ-flats parallel to it form a parallel bundle of μ-flats. The μ-flats of $\mathrm{EG}(m,q)$ can be partitioned into $K_{\mathrm{EG}}(m,\mu)$ parallel bundles of μ-flats, where $K_{\mathrm{EG}}(m,\mu)$ is given by (2.51).

If $\alpha^{j_{\mu+1}}$ is not a point on the μ-flat $\{\alpha^{j_0} + \beta_1\alpha^{j_0} + \cdots + \beta_\mu\alpha^{j_\mu}\}$, then the $(\mu+1)$-flat

$$\{\alpha^{j_0} + \beta_1\alpha^{j_1} + \cdots + \beta_\mu\alpha^{j_\mu} + \beta_{\mu+1}\alpha^{j_{\mu+1}}\}$$

contains the μ-flat $\{\alpha^{j_0} + \beta_1\alpha^{j_1} + \cdots + \beta_\mu\alpha^{j_\mu}\}$. Let $\alpha^{j'_{\mu+1}}$ be a point not on the $(\mu+1)$-flat

$$\{\alpha^{j_0} + \beta_1\alpha^{j_1} + \cdots + \beta_{\mu+1}\alpha^{j_{\mu+1}}\}.$$

Then the two $(\mu+1)$-flats, $\{\alpha^{j_0} + \beta_1\alpha^{j_0} + \cdots + \beta_\mu\alpha^{j_\mu} + \beta_{\mu+1}\alpha^{j_{\mu+1}}\}$ and $\{\alpha^{j_0} + \beta_1\alpha^{j_1} + \cdots + \beta_\mu\alpha^{j_\mu} + \beta_{\mu+1}\alpha^{j'_{\mu+1}}\}$, intersect on the μ-flat $\{\alpha^{j_0} + \beta_1\alpha^{j_1} + \cdots +$

$\beta_\mu \alpha^{j_\mu}\}$. For a given μ-flat $\mathcal{F}^{(\mu)}$, the number of $(\mu+1)$-flats in EG(m,q) that intersect on $\mathcal{F}^{(\mu)}$ is given by

$$g(m, \mu+1, \mu) = \frac{q^{m-\mu}-1}{q-1}, \tag{2.59}$$

which is obtained from (2.52) with μ replaced by $\mu+1$.

Let $\mathcal{F}^{(\mu)} = \{\alpha^{j_0} + \beta_1 \alpha^{j_1} + \cdots + \beta_\mu \alpha^{j_\mu}\}$ be a μ-flat not passing through the origin of EG(m,q) (i.e., $\alpha^{j_0} \neq 0$). For $0 \leq i < q^m - 1$, the set of q^μ points

$$\alpha^i \mathcal{F}^{(\mu)} = \{\alpha^{j_0+i} + \beta_1 \alpha^{j_1+i} + \cdots + \beta_\mu \alpha^{j_\mu+i}\}$$

also forms a μ-flat of EG(m,q) (or the subgeometry EG*(m,q)) not passing through the origin. It can also be proved that $\mathcal{F}^{(\mu)}, \alpha\mathcal{F}^{(\mu)}, \ldots, \alpha^{q^m-2}\mathcal{F}^{(\mu)}$ are $q^m - 1$ different μ-flats in EG(m,q) not passing through the origin [5]. This is to say that every μ-flat not passing through the origin of EG(m,q) is primitive. A μ-flat not passing through the origin of EG(m,q) and its $q^m - 2$ cyclic-shifts are said to form a cyclic class of μ-flats. The μ-flats in EG*(m,q) not passing through the origin can be partitioned into cyclic classes of μ-flats of size $q^m - 1$.

In Chapters 10 and 11, low-density parity-check codes will be constructed from the parallel, intersecting, and cyclic structures of the lines and flats of finite Euclidean geometries.

2.7.2 Projective Geometries

Projective geometries form another family of geometries over finite fields, consisting of finite numbers of points, lines, and flats [1–4]. For any positive integer m and any finite field GF(q), there exists an m-dimensional projective geometry over GF(q), denoted by PG(m,q). A point of PG(m,q) is a *nonzero* $(m+1)$-tuple over GF(q). Let β be a primitive element of GF(q). For any point $(a_0, a_1, \ldots, a_m)$ in PG(m,q) and $0 \leq j < q-1$, the $(m+1)$-tuple $(\beta^j a_0, \beta^j a_1, \ldots, \beta^j a_m)$ represents the same point $(a_0, a_1, \ldots, a_m)$. That is to say that the $q-1$ nonzero $(m+1)$-tuples in the set

$$\{(\beta^j a_0, \beta^j a_1, \ldots, \beta^j a_m) : \beta^j \in \mathrm{GF}(q), 0 \leq j < q-2\}$$

represent the same point $(a_0, a_1, \ldots, a_m)$. Therefore, no point in PG(m,q) is a *multiple* (or scalar product) of another point in PG(m,q). Since all the points of PG(m,q) are nonzero $(m+1)$-tuples over GF(q), PG(m,q) does not have an origin.

The m-dimensional projective geometry PG(m,q) over GF(q) can be constructed from the extension field GF(q^{m+1}) of GF(q). We view GF(q^{m+1}) as the $(m+1)$-dimensional vector space V_{m+1} over GF(q). Each element in GF(q^{m+1}) can be represented by an $(m+1)$-tuple over GF(q). Let α be a primitive element of GF(q^{m+1}). Then $\alpha^0, \alpha, \ldots, \alpha^{q^{m+1}-2}$ give all the nonzero elements of GF(q^{m+1}). Let

$$n = \frac{q^{m+1}-1}{q-1}. \tag{2.60}$$

Let $\beta = \alpha^n$. The order of β is $q-1$. The q elements $0, \beta^0 = 1, \beta, \ldots, \beta^{q-2}$ form the ground field GF(q) of GF(q^{m+1}).

Partition the nonzero elements of GF(q^m) into n disjoint subsets as follows:

$$\begin{aligned}
&\{\alpha^0, \beta\alpha^0, \ldots, \beta^{q-2}\alpha^0\},\\
&\{\alpha^1, \beta\alpha^1, \ldots, \beta^{q-2}\alpha^1\},\\
&\{\alpha^2, \beta\alpha^2, \ldots, \beta^{q-2}\alpha^2\},\\
&\vdots\\
&\{\alpha^{n-1}, \beta\alpha^{n-1}, \ldots, \beta^{q-2}\alpha^{n-1}\}.
\end{aligned} \tag{2.61}$$

Each set consists of $q-1$ nonzero elements of GF(q^{m+1}) and each element in the set is a multiple of the first element. We can represent each set by its first element as follows:

$$(\alpha^i) = \{\alpha^i, \beta\alpha^i, \ldots, \beta^{q-2}\alpha^i\} \tag{2.62}$$

with $0 \le i < n$. For any element α^k in GF(q^{m+1}), if $\alpha^k = \beta^j\alpha^i$ with $0 \le i < n$, then α^k is represented by α^i. If each element in GF(q^{m+1}) is represented by an $(m+1)$-tuple over GF(q), then (α^i) consists of $q-1$ nonzero $(m+1)$-tuples over GF(q), each is a multiple of the $(m+1)$-tuple representation of α^i. It follows from the definition of a point of PG(m,q) given above that (α^i) is a point in PG(m,q). Therefore,

$$(\alpha^0), (\alpha^1), \ldots, (\alpha^{n-1})$$

form all the points of PG(m,q) and the number of points in PG(m,q) is n given by (2.60).

Note that, if the 0-element of GF(q^m) is added to the set (α^i), we obtain a set $\{0, \alpha^i, \beta\alpha^i, \ldots, \beta^{q-2}\alpha^i\}$ of q elements. This set of q elements, viewed as $(m+1)$-tuples over GF(q), is simply a one-dimensional subspace of the vector space V_{m+1} of all the $(m+1)$-tuples over GF(q) and hence it is a line in the $(m+1)$-dimensional Euclidean geometry EG($m+1$,q) over GF(q), which passes through the origin of EG($m+1$,q). Therefore, we may regard a point (α^i) of PG(m,q) as a *projection* of a line of EG($m+1$,q) passing through the origin of EG($m+1$,q).

Let (α^i) and (α^j) be two distinct points in PG(q^{m+1}). The line that connects (α^i) and (α^j) consists of points of the following form:

$$(\delta_1\alpha^i + \delta_2\alpha^j), \tag{2.63}$$

where δ_1 and δ_2 are scalars from GF(q) and are *not both equal to zero*. We denote this line by

$$\{(\delta_1\alpha^i + \delta_2\alpha^j)\}. \tag{2.64}$$

There are q^2-1 choices of δ_1 and δ_2 from GF(q) (excluding $\delta_1 = \delta_2 = 0$). However, for each choice of (δ_1, δ_2), there are $q-2$ multiples of (δ_1, δ_2). Consequently, (δ_1, δ_2) and its $q-2$ multiples result in the same points of the form given by (2.63). Therefore, the line connecting points (α^i) and (α^j) consists of

$$\frac{q^2-1}{q-1} = q+1 \tag{2.65}$$

points. For any two points in PG(m,q), there is one and only one line connecting them. From the number of points and definition of a line in PG(m,q), we can enumerate the number of lines in PG(m,q), which is given by

$$J_{\mathrm{PG}}(m,1) = \frac{(q^{m+1}-1)(q^m-1)}{(q^2-1)(q-1)}. \tag{2.66}$$

Let (α^k) be a point not on the line that connects the points (α^i) and (α^j). Then the line $\{(\delta_1\alpha^i + \delta_2\alpha^j)\}$ and the line $\{(\delta_1\alpha^i + \delta_2\alpha^k)\}$ intersect at the point (α^i) (with $\delta_1 = 1$ and $\delta_2 = 0$). The number of lines in PG(m,q) that intersect at a point in PG(m,q) is given by

$$g_{\mathrm{PG}}(m,1,0) = \frac{q^m-1}{q-1}. \tag{2.67}$$

The lines that intersect at a point of PG(m,q) are said to form an intersecting bundle of lines at the point.

If we take all the q^2 linear combinations of α^i and α^j in $\{(\delta_1\alpha^i + \delta_2\alpha^j)\}$ (including $\delta_1 = \delta_2 = 0$), these q^2 linear combinations give q^2 vectors over GF(q) that form a two-dimensional subspace of the vector space V_{m+1} over GF(q). This two-dimensional space of V_{m+1} is a 2-flat of EG($m+1$,q) that passes through the origin of EG($m+1$,q). Thus a line of PG(m,q) may be regarded as a projection of a 2-flat (a two-dimensional plane) of EG($m+1$,q).

For $0 \le \mu < m$, let $(\alpha^{j_0}), (\alpha^{j_1}), \ldots, (\alpha^{j_\mu})$ be $\mu+1$ linearly independent points. Then a μ-flat in PG(m,μ) consists of points of the following form:

$$(\delta_0\alpha^{j_0} + \delta_1\alpha^{j_1} + \cdots + \delta_\mu\alpha^{j_\mu}), \tag{2.68}$$

where $\delta_k \in \mathrm{GF}(q)$ with $0 \le k \le \mu$, and not all of $\delta_0, \delta_1, \ldots, \delta_\mu$ are zero. We denote this μ-flat by

$$\{(\delta_0\alpha^{j_0} + \delta_1\alpha^{j_1} + \cdots + \delta_\mu\alpha^{j_\mu})\}. \tag{2.69}$$

It can be shown that the number of points on a μ-flat is

$$\frac{q^{\mu+1}-1}{q-1}. \tag{2.70}$$

There are

$$J_{\mathrm{PG}}(m,\mu) = \prod_{i=0}^{\mu} \frac{q^{m-i+1}-1}{q^{\mu-i+1}-1} \tag{2.71}$$

μ-flats in PG(m,q).

If we allow δ_i in (2.69) to be any element in GF(q) without restriction, this results in $q^{\mu+1}$ points in the $(m+1)$-dimensional Euclidean geometry EG($m+1$,q) over GF(q), which form a $(\mu+1)$-flat in EG($m+1$,q). Therefore, we may regard a μ-flat in the m-dimensional projective geometry PG(m,q) over GF(q) as a projection of a $(\mu+1)$-flat of the $(m+1)$-dimensional Euclidean geometry EG($m+1$,q) over GF(q).

For $1 \leq \mu < m$, let $\alpha^{j'_\mu}$ be a point not on the μ-flat $\{(\delta_0\alpha^{j_0} + \delta_1\alpha^{j_1} + \cdots + \delta_\mu\alpha^{j_\mu})\}$. Then the μ-flat

$$\{(\delta_0\alpha^{j_0} + \delta_1\alpha^{j_1} + \cdots + \delta_{\mu-1}\alpha^{j_{\mu-1}} + \delta_\mu\alpha^{j_\mu})\}$$

and the μ-flat

$$\{(\delta_0\alpha^{j_0} + \delta_1\alpha^{j_1} + \cdots + \delta_{\mu-1}\alpha^{j_{\mu-1}} + \delta_\mu\alpha^{j'_\mu})\}$$

intersect on the $(\mu-1)$-flat $\{(\delta_0\alpha^{j_0} + \delta_1\alpha^{j_1} + \cdots + \delta_{\mu-1}\alpha^{j_{\mu-1}}\}$. For $1 \leq \mu < m$, the number of μ-flats that intersect on a given $(\mu-1)$-flat is given by

$$g_{\mathrm{PG}}(m, \mu, \mu-1) = \frac{q^{m-\mu+1}-1}{q-1}. \tag{2.72}$$

The μ-flats in PG(q,m) that intersect on a $(\mu-1)$-flat $\mathcal{F}^{(\mu-1)}$ are called an intersecting bundle of μ-flats on $\mathcal{F}^{(\mu-1)}$.

Next we consider the cyclic structure of lines in an m-dimensional projective geometry PG(m,q) over GF(q). Let $\mathcal{L}_0 = \{(\delta_0\alpha^{j_0} + \delta_1\alpha^{j_1})\}$ be a line in PG(m,q). For $0 \leq i < n = (q^{m+1}-1)/(q-1)$,

$$\alpha^i\mathcal{L}_0 = (\delta_0\alpha^{j_0+i} + \delta_1\alpha^{j_1+i})$$

is also a line in PG(m,q), which is called the ith cyclic-shift of $\mathcal{L}_0$. If m is even, every line is primitive [5]. In this case, $\mathcal{L}_0, \alpha\mathcal{L}_0, \ldots, \alpha^{n-1}\mathcal{L}_0$ are all different lines and they form a cyclic class. Then the lines of PG(m,q) can be partitioned into

$$K^{(\mathrm{e})}_{\mathrm{c,PG}}(m, 1) = \frac{q^m-1}{q^2-1} \tag{2.73}$$

cyclic classes of size $(q^{m+1}-1)/(q-1)$. If m is odd, then there are $l_0 = (q^{m+1}-1)/(q^2-1)$ nonprimitive lines, which can be represented by $\alpha^i\mathcal{L}_0$ for a certain line $\mathcal{L}_0$ in PG(m,q), where $0 \leq i < l_0$ [5]. These l_0 lines form a cyclic class of size l_0. All the other lines in PG(m,q) are primitive and they can partitioned into

$$K^{(\mathrm{o})}_{\mathrm{c,PG}}(m, 1) = \frac{q(q^{m-1}-1)}{q^2-1} \tag{2.74}$$

cyclic classes of size $(q^{m+1}-1)/(q-1)$.

For $1 < \mu < m$, a μ-flat in PG(m,q) is in general not primitive, but it is primitive in the following cases [13]: (1) when $\mu = m-1$; and (2) when the number of points in a μ-flat and the number of points in PG(m,q) are relatively prime. For the first case, there are $(q^{m+1}-1)/(q-1)$ $(m-1)$-flats and they together form a single cyclic class of size $(q^{m+1}-1)/(q-1)$. For the second case, the μ-flats in PG(m,q) can be partitioned into

$$K_{\mathrm{c,PG}}(m, \mu) = \prod_{i=0}^{\mu-1} = \frac{q^{m-i}-1}{q^{\mu-i+1}-1} \tag{2.75}$$

cyclic classes.

Since the μ-flats of PG(m,q) are obtained from the $(\mu+1)$-flats of EG($m+1$,q) not passing through the origin, the μ-flats in PG(m,q) do not have parallel structure.

2.8 Graphs

The theory of graphs is an important part of combinatorial mathematics. A graph is a simple geometrical figure consisting of *vertices* (or *nodes*) and *edges* (*lines* or *branches*), which connect some of the vertices. Because of their diagrammatic representation, graphs have been found to be extremely useful in modeling systems arising in physical sciences, engineering, social sciences, economics, computer science, etc. In error-control coding, graphs are used to construct codes and to represent codes for specific ways of decoding. This will be seen in later chapters. In this section, we first introduce some basic concepts and fundamental properties of graphs and then present some special graphs that are useful for designing codes and decoding algorithms.

2.8.1 Basic concepts

Like for every mathematical theory, we have to begin with a list of definitions.

Definition 2.23. A *graph*, denoted $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, consists of an non-empty set $\mathcal{V} = \{v_1, v_2, \ldots\}$ of elements called *vertices* and a prescribed set $\mathcal{E} = \{(v_i, v_j)\}$ of unordered pairs of vertices called *edges*. The set $\mathcal{V}$ is called the *vertex-set* and the set $\mathcal{E}$ is called the *edge-set*. The vertices v_i and v_j associated with an edge (v_i, v_j) are called the *end vertices* of the edge (v_i, v_j).

If (v_i, v_j) is an edge in a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, the edge (v_i, v_j) is said to be *incident with* the end vertices v_i and v_j, and v_i and v_j are said to be *adjacent*. Two adjacent vertices v_i and v_j are said to be connected by edge (v_i, v_j). If (v_i, v_j) and (v_j, v_k) are two edges in $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, these two edges are incident with the common vertex v_j and they are said to be adjacent edges. A vertex is said to be an *isolated vertex* if there is no edge incident with it. A vertex is called a *pendant* if it is incident with only one edge. The edge (v_i, v_i) (if it exists) is called a *self-loop*. Multiple (two or more) edges are said to be parallel if they are incident with the same two vertices.

The most common and useful representation of a graph is by means of a diagram in which the vertices are represented as points on a plane and an edge (v_i, v_j) is represented by a line joining vertices v_i and v_j. For example, consider the graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with

$$\mathcal{V} = \{v_1, v_2, v_3, v_4, v_5, v_6\},$$
$$\mathcal{E} = \{(v_1, v_1), (v_1, v_2), (v_1, v_3), (v_1, v_4), (v_2, v_3), (v_3, v_4), (v_4, v_3), (v_4, v_5)\}.$$

The geometrical representation of this graph is shown in Figure 2.1. It is clear that (v_1, v_1) forms a self-loop, vertex v_6 is an isolated vertex, and v_5 is a pendant.

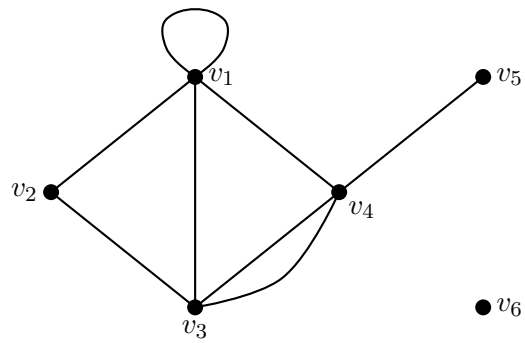


Figure 2.1 A graph with six vertices and eight edges.

Definition 2.24. A graph with finite numbers of vertices as well as a finite number of edges is called a *finite graph*; otherwise, it is called an *infinite graph*.

Definition 2.25. In a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, the *degree* of a vertex, denoted $d(v)$, is defined as the number of edges that are incident with v.

It is clear that the degree of an isolated vertex is zero and the degree of a pendant is 1. Consider the graph shown in Figure 2.1. The degrees of the vertices are $d(v_1) = 5$, $d(v_2) = 2$, $d(v_3) = 4$, $d(v_4) = 4$, $d(v_5) = 1$, and $d(v_6) = 0$.

Theorem 2.20. For a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with m edges and n vertices, $v_1, v_2, \ldots, v_n$, the sum of degrees of vertices is equal to $2m$, i.e.,

$$\sum_{i=1}^{n} d(v_i) = 2m.$$

Proof. The proof follows from the fact that each edge contributes two to the sum of degrees. □

A direct consequence of Theorem 2.20 is the following corollary.

Corollary 2.3. In a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, the number of vertices with odd degree is even.

Definition 2.26. A graph that has neither self-loops nor parallel edges is called a *simple graph*. A graph is said to be *regular* of degree k if the degree of each vertex is k.

The graph shown in Figure 2.2 is a simple graph, while the graph shown in Figure 2.3 is a simple regular graph of degree 3.

Definition 2.27. A simple graph is said to be *complete* if there exists an edge for every pair of vertices.

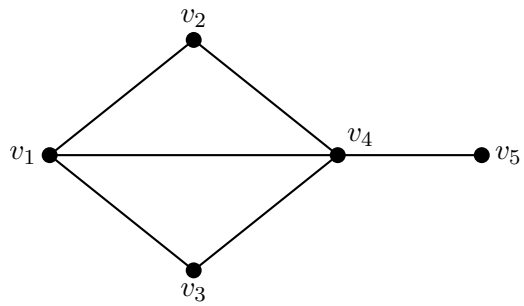


Figure 2.2 A simple graph with five vertices and six edges.

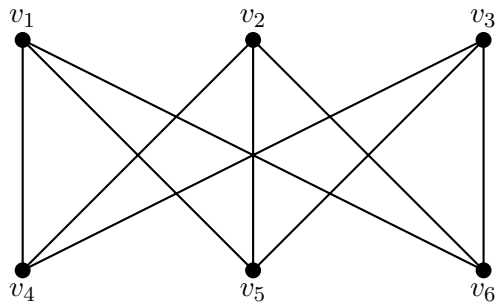


Figure 2.3 A regular graph of degree 3.

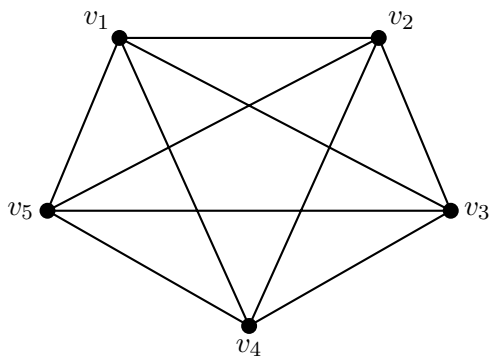


Figure 2.4 A complete graph with five vertices.

It is clear that, for a complete graph with n vertices, each vertex has degree $n-1$ and hence it is a regular graph of degree $n-1$. Figure 2.4 shows a simple complete graph.

Definition 2.28. A graph $\mathcal{G}_s = (\mathcal{V}_s, \mathcal{E}_s)$ is called a *subgraph* of graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ if $\mathcal{V}_s$ is a subset of $\mathcal{V}$ and $\mathcal{E}_s$ is a subset of $\mathcal{E}$.

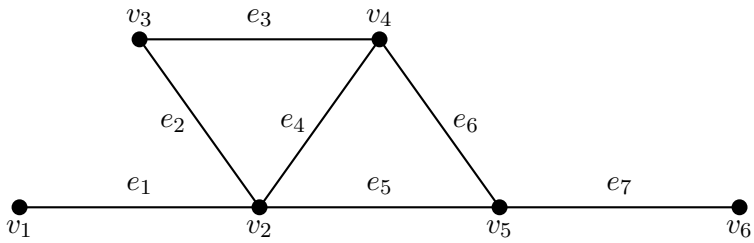


Figure 2.5 A simple graph with six vertices and seven edges.

In many applications, self-loops and parallel multiple edges do not arise. For our purpose in this book, we mostly restrict our attention to simple graphs.

A simple finite graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with n vertices and m edges can be represented by two matrices. Let $\mathcal{V} = \{v_1, v_2, \ldots, v_n\}$ and $\mathcal{E} = \{e_1, e_2, \ldots, e_m\}$. The first matrix is an $n \times n$ matrix, denoted $\mathbf{A}(\mathcal{G}) = [a_{i,j}]$, in which the rows and columns correspond to the n vertices of $\mathcal{G}$ and the entry $a_{i,j} = 1$ if (v_i, v_j) is an edge in $\mathcal{E}$, otherwise $a_{i,j} = 0$. $\mathbf{A}(\mathcal{G})$ is called the *adjacency matrix* of $\mathcal{G}$. The second matrix is an $m \times n$ matrix, denoted by $\mathbf{M}(\mathcal{G}) = [m_{i,j}]$, in which the rows correspond to the m edges of $\mathcal{G}$, the columns correspond to the n vertices of $\mathcal{G}$ and the entry $m_{i,j} = 1$ if the jth vertex v_j is an end-vertex of the ith edge e_i, otherwise $m_{i,j} = 0$. $\mathbf{M}(\mathcal{G})$ is called the *incidence matrix* of $\mathcal{G}$.

Consider the graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ shown in Figure 2.5. The adjacency and incidence matrices of this graph are given below:

$$\mathbf{A}(\mathcal{G}) = \begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \end{array} \begin{array}{c} \begin{array}{cccccc} v_1 & v_2 & v_3 & v_4 & v_5 & v_6 \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \end{array},$$

$$\mathbf{M}(\mathcal{G}) = \begin{array}{c} \\ e_1 \\ e_2 \\ e_3 \\ e_4 \\ e_5 \\ e_6 \\ e_7 \end{array} \begin{array}{c} \begin{array}{cccccc} v_1 & v_2 & v_3 & v_4 & v_5 & v_6 \end{array} \\ \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix} \end{array}.$$

2.8.2 Paths and Cycles

A sequence of connected edges that form a continuous route plays an important role in graph theory and has many practical applications.

Definition 2.29. A *path* in a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is defined as an alternating sequence of vertices and edges, such that each edge is incident with the vertices preceding and following it, and no vertex appears more than once. The number of edges on a path is called the *length* of the path.

A simple way to represent a path is by arranging the vertices on the path in consecutive order as follows:

$$v_{i_0}, v_{i_1}, \ldots, v_{i_{k-1}}, v_{i_k},$$

in which any two consecutive vertices are adjacent (or connected by an edge), where v_{i_0} and v_{i_k} are called the *starting* and *ending* vertices of the path. Basically, a path is a continuous sequence of adjacent edges joining two vertices. It is clear that each edge on a path appears once and only once. Consider the graph shown in Figure 2.1. The path v_1, v_2, v_3, v_4, v_5 is a path of length 4 with v_1 and v_5 as the starting and ending vertices, respectively. Consider the simple graph shown in Figure 2.5. The path $v_1, v_2, v_3, v_4, v_5, v_6$ is a path of length 5 with v_1 and v_6 as the starting and ending vertices, respectively.

Two vertices in a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ are said to be *connected* if there exists a path connecting them. A graph is said to be connected if any two vertices in $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ are connected by at least one path; otherwise it is said to be disconnected. The graphs shown in Figures 2.3, 2.4, and 2.5 are simple and connected graphs. In a connected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, the *distance* between two of its vertices v_i and v_j, denoted by $d(v_i, v_j)$, is defined as the length of the shortest path (i.e., the number of edges on the shortest path) between v_i and v_j. Consider the connected graph shown in Figure 2.5. There are three paths connecting v_1 and v_6: (1) v_1, v_2, v_5, v_6; (2) v_1, v_2, v_4, v_5, v_6; and (3) $v_1, v_2, v_3, v_4, v_5, v_6$. They have lengths 3, 4, and 5, respectively. The shortest path between v_1 and v_6 is v_1, v_2, v_5, v_6, which has length 3, and hence the distance between v_1 and v_6 is 3.

Definition 2.30. The *eccentricity* $E(v_i)$ of a vertex v_i in a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is defined as the distance from v_i to the vertex farthest from v_i in $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, i.e.,

$$E(v_i) = \max\{d(v_i, v_j) \colon v_j \in \mathcal{V}\}.$$

A vertex with minimum eccentricity in graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is called a *center*, denoted by $c(\mathcal{G})$, of $\mathcal{G} = (\mathcal{V}, \mathcal{E})$.

It is clear that a graph may have more than one center. Consider the graph shown in Figure 2.6. The graph is a simple connected graph with 14 vertices. The eccentricity of each vertex is shown next to the vertex. This graph has seven

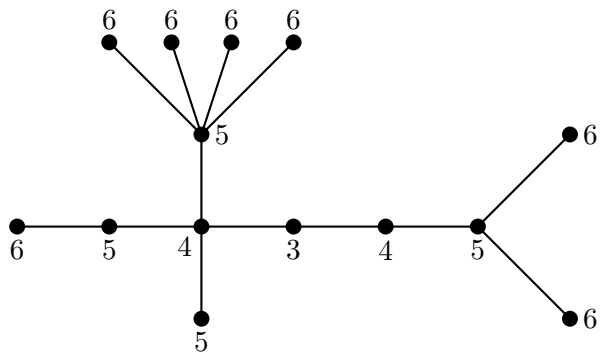


Figure 2.6 Eccentricities of the vertices and a center of a graph.

vertices with eccentricities of 6, three vertices with eccentricities of 5, two vertices with eccentricities of 4, and one vertex (the center) with eccentricity 3.

Definition 2.31. The *diameter* of a connected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, denoted by $\operatorname{dia}(\mathcal{G})$, is defined as the largest distance between two vertices in the graph, i.e.,

$$\operatorname{dia}(\mathcal{G}) = \max_{v_i, v_j \in \mathcal{V}} d(v_i, v_j).$$

The *radius* of a connected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, denoted by $r(\mathcal{G})$, is defined as the eccentricity of a center of the graph, i.e.,

$$r(\mathcal{G}) = \min\{E(v_i) \colon v_i \in \mathcal{V}\}.$$

It follows from the definitions of eccentricity of a vertex and diameter of a graph that the diameter of a connected graph equals the maximum of the vertex eccentricities, i.e.,

$$\operatorname{dia}(\mathcal{G}) = \max\{E(v_i) \colon v_i \in \mathcal{V}\}.$$

Consider the connected graph shown in Figure 2.6. The diameter and radius are 6 and 3, respectively.

If two vertices v_i and v_j in a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ are connected, there may be more than one path between them. Among the paths connecting v_i and v_j, the one (or ones) with the shortest length is (or are) called the *shortest path* (or paths) between v_i and v_j. Finding shortest paths between connected vertices has many applications in communication networks, operations research, and other areas. There are several known algorithms for finding shortest paths between two vertices, which can be found in many texts on graphs.

Definition 2.32. A closed path in a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ that begins and ends at the same vertex is called a *cycle*. The number of edges on the cycle is called the *length* of the cycle. The *girth* of a graph with cycles is defined as the length of its shortest cycle. A graph with no cycles has infinite girth.

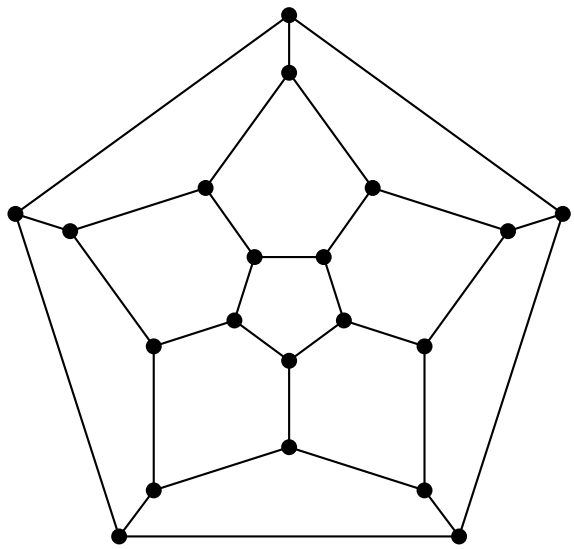


Figure 2.7 A graph with girth 5.

Consider the graph shown in Figure 2.7. It has many cycles. The girth of this graph is 5.

Definition 2.33. A graph with no cycle is said to be *acyclic*. A connected acyclic graph is called a *tree*. An edge in a tree is called a *branch*. A *pendant* in a tree is called an *end-vertex*.

The graph shown in Figure 2.6 contains no cycle and hence is a tree. There are six end-vertices.

2.8.3 Bipartite Graphs

In this section, we present a special type of graph, known as a *bipartite graph*, that is particularly useful in the design (or construction) of a class of Shannon-limit-approaching codes and provides a pictorial explanation of a suboptimal decoding algorithm for this class of codes.

Definition 2.34. A graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is called a *bipartite graph* if its vertex-set $\mathcal{V}$ can be decomposed into two disjoint subsets $\mathcal{V}_1$ and $\mathcal{V}_2$ such that every edge in the edge-set $\mathcal{E}$ joins a vertex in $\mathcal{V}_1$ and a vertex in $\mathcal{V}_2$, and no two vertices in either $\mathcal{V}_1$ or $\mathcal{V}_2$ are connected.

It is obvious that a bipartite graph contains no self-loop. Figure 2.8 shows a bipartite graph with $\mathcal{V}_1 = \{v_1, v_2, v_3, v_4, v_5, v_6, v_7\}$ and $\mathcal{V}_2 = \{v_8, v_9, v_{10}, v_{11}, v_{12}, v_{13}, v_{14}\}$. It is a simple connected graph. Since all the vertices of the graph have the same degree, 3, it is also a regular graph.

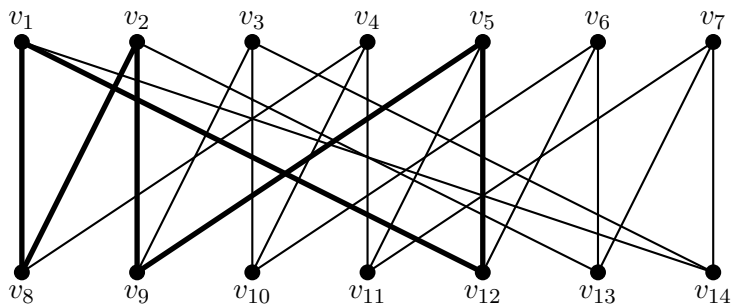


Figure 2.8 A bipartite graph.

Theorem 2.21. If a bipartite graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ contains cycles, then all cycles have even lengths.

Proof. Suppose we trace a cycle in $\mathcal{G}$ beginning from a vertex v_i in $\mathcal{V}_1$. The first edge of this cycle must connect vertex v_i to a vertex v_j in $\mathcal{V}_2$. The second edge of the cycle must connect v_j to a vertex v_k in $\mathcal{V}_1$. If v_k is identical to the starting vertex v_i, then we have completed the tracing of a cycle of length 2. If v_k is not identical to the starting vertex v_i, we must trace the third edge of the cycle which connects v_k to a vertex $v_l \in \mathcal{V}_2$. Then, we must take the fourth edge of the cycle to return to a vertex in $\mathcal{V}_1$. If this vertex is not identical to the starting vertex v_i, we need to continue tracing the next edge of the cycle. This back and forth tracing process continues until the last edge of the cycle terminates at the starting vertex v_i. Leaving a vertex in $\mathcal{V}_1$ and returning to a vertex in $\mathcal{V}_1$ requires two edges each time. Therefore, the length of the cycle must be a multiple of 2, an even number. □

If a bipartite graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is a simple graph (i.e., there exist no multiple edges between a vertex in $\mathcal{V}_1$ and a vertex in $\mathcal{V}_2$), then it has no cycle of length 2. In this case, the shortest cycle in $\mathcal{G}$ has a length of at least 4. For a simple bipartite graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, if no two vertices in $\mathcal{V}_1$ are jointly connected (or jointly adjacent) to two vertices in $\mathcal{V}_2$ or vice versa, then $\mathcal{G}$ contains no cycle of length 4. In this case, the shortest cycle in $\mathcal{G}$ has a length of at least 6. That is to say, the girth of $\mathcal{G}$ is at least 6. Consider the bipartite graph shown in Figure 2.8. Since it is a simple graph, it has no cycle of length 2. Since no two vertices in $\mathcal{V}_1$ are jointly connected to two vertices in $\mathcal{V}_2$, the graph has no cycle of length 4. It does have cycles of length 6; one such cycle is marked by heavy lines. Therefore, the girth of this bipartite graph is 6. How to construct bipartite graphs with large girth is a very challenging problem.

Consider a simple bipartite graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with disjoint subsets of vertices, $\mathcal{V}_1$ and $\mathcal{V}_2$. Let m and n be the numbers of vertices in $\mathcal{V}_1$ and $\mathcal{V}_2$, respectively. Then, this bipartite graph can be represented by an $m \times n$ adjacency matrix $\mathbf{A}(\mathcal{G}) = [a_{i,j}]$ whose rows correspond to the vertices of $\mathcal{V}_1$ and whose columns correspond

to the vertices of $\mathcal{V}_2$, where $a_{i,j} = 1$ if and only if the ith vertex of $\mathcal{V}_1$ is adjacent to the jth vertex of $\mathcal{V}_2$, otherwise $a_{i,j} = 0$. The adjacency matrix of the bipartite graph $\mathcal{G}$ shown in Figure 2.8 is given below:

$$\mathbf{A}(\mathcal{G}) = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

For a simple bipartite graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, if no two vertices in $\mathcal{V}_1$ are jointly connected to two vertices in $\mathcal{V}_2$, then there are not four ones at the four corners of a rectangle in its adjacency matrix $\mathbf{A}(\mathcal{G})$. Conversely, if there are not four ones at the four corners of a rectangle in the adjacency matrix $\mathbf{A}(\mathcal{G})$ of a simple bipartite graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, then no two vertices in $\mathcal{V}_1$ are jointly connected to two vertices in $\mathcal{V}_2$. This implies that, if there is no rectangle in the adjacency matrix of a bipartite graph with four ones at its four corners, then the graph has no cycle of length 4.

Some useful books are [6–15].

Problems

2.1 Construct the group under modulo-11 addition.

2.2 Construct the group under modulo-11 multiplication.

2.3 Let m be a positive integer. If m is not a prime, prove that the set $\{1, 2, \ldots, m-1\}$ does not form a group under modulo-m multiplication.

2.4 Find all the generators of the multiplicative group constructed in Problem 2.2.

2.5 Find a subgroup of the additive group constructed in Example 2.5 and give its cosets.

2.6 Find a cyclic subgroup of the multiplicative group constructed using the prime integer 13 and give its cosets.

2.7 Prove the following properties of a field F.

1. For every element a in F, $a \cdot 0 = 0 \cdot a = 0$.
2. For any two nonzero elements a and b in F, $a \cdot b \neq 0$.
3. For two elements a and b in F, $a \cdot b = 0$ implies that either $a = 0$ or $b = 0$.
4. For $a \neq 0$, $a \cdot b = a \cdot c$ implies that $b = c$.
5. For any two elements a and b in F, $-(a \cdot b) = (-a) \cdot b = a \cdot (-b)$.

2.8 Let m be a positive integer. If m is not a prime, prove that the set $\{0, 1, \ldots, m-1\}$ does not form a field under modulo-m addition and multiplication.

2.9 Consider the integer group $G = \{0, 1, 2, \ldots, 31\}$ under modulo-32 addition. Show that $H = \{0, 4, 8, 12, 16, 20, 24, 28\}$ forms a subgroup of G. Decompose G into cosets with respect to H (or modulo H).

2.10 Prove that every finite field has at least one primitive element.

2.11 Prove the following properties of a vector space V over a field F.

1. Let 0 be the zero element of F. For any vector $\mathbf{v}$ in V, $0 \cdot \mathbf{v} = 0$.
2. For any element a in F, $a \cdot \mathbf{0} = \mathbf{0}$.
3. For any element a in F and any vector $\mathbf{v}$ in V, $(-a) \cdot \mathbf{v} = a \cdot (-\mathbf{v}) = -(a \cdot \mathbf{v})$.

2.12 Prove that two different bases of a vector space over a field have the same number of linearly independent vectors.

2.13 Prove that the inner product of two n-tuples over GF(q) has the following properties.

1. $\mathbf{u} \cdot \mathbf{v} = \mathbf{v} \cdot \mathbf{u}$ (commutative law).
2. $\mathbf{u} \cdot (\mathbf{v} + \mathbf{w}) = \mathbf{u} \cdot \mathbf{v} + \mathbf{u} \cdot \mathbf{w}$ (distributive law).
3. For any element c in GF(q), $(c\mathbf{u}) \cdot \mathbf{v} = c(\mathbf{u} \cdot \mathbf{v})$ (associative law).
4. $\mathbf{0} \cdot \mathbf{u} = \mathbf{0}$.

2.14 Prove that the addition and multiplication of two polynomials over GF(q) defined by (2.17), (2.18), and (2.19) satisfy the associative, commutative, and distributive laws.

2.15 Let m be a positive integer. Prove that the set $\{0, 1, \ldots, m-1\}$ forms a ring under modulo-m addition and multiplication.

2.16 Let $a(X) = 3X^2 + 1$ and $b(X) = X^6 + 3X + 2$ be two polynomials over the prime field GF(5). Divide $b(X)$ by $a(X)$ and find the quotient and remainders of the division.

2.17 Let $a(X) = 3X^3 + X + 2$ be a polynomial over the prime field GF(5). Determine whether or not $a(X)$ is irreducible over GF(5).

2.18 Let $f(X)$ be a polynomial of degree n over GF(q). The reciprocal of $f(X)$ is defined as $f^*(X) = X^n f(1/X)$.

1. Prove that $f^*(X)$ is irreducible over GF(q) if and only if $f(X)$ is irreducible over GF(q).
2. Prove that $f^*(X)$ is primitive if and only if $f(X)$ is primitive.

2.19 Prove that the extension field $\mathrm{GF}(p^m)$ of the prime field $\mathrm{GF}(p)$ is an m-dimensional vector space over $\mathrm{GF}(p)$.

2.20 Prove Theorem 2.12.

2.21 Let β be a nonzero element of $\mathrm{GF}(q^m)$. Let e be the smallest non-negative integer such that $\beta^{q^e} = \beta$. Prove that e divides m.

2.22 Prove Theorem 2.19.

2.23 Consider the Galois field $\mathrm{GF}(2^5)$ given by Table 2.10. Find the minimum polynomials of α^5 and α^7.

2.24 If $q-1$ is a prime, prove that every nonzero element of $\mathrm{GF}(q)$ not equal to the unit element 1 is primitive.

2.25 Consider the Galois field $\mathrm{GF}(2^4)$ given by Table 2.14. Let $\beta = \alpha^5$. Then $\{0, 1, \beta^1, \beta^2\}$ form a subfield $\mathrm{GF}(2^2)$ of $\mathrm{GF}(2^4)$. Regard $\mathrm{GF}(2^4)$ as the two-dimensional Euclidean geometry $\mathrm{EG}(2,2^2)$ over $\mathrm{GF}(2,2^2)$. Find the lines that intersect at point α^3.

2.26 Construct the Galois field $\mathrm{GF}(2^6)$ based on the primitive polynomial $p(X) = 1 + X + X^6$ over $\mathrm{GF}(2)$. Let α be a primitive element of $\mathrm{GF}(2^6)$ and $\beta = \alpha^{21}$. Then $\{0, 1, \beta, \beta^2\}$ form a subfield $\mathrm{GF}(2^2)$ of $\mathrm{GF}(2^6)$. Regard $\mathrm{GF}(2^6)$ as the three-dimensional Euclidean geometry $\mathrm{EG}(3,2^2)$ over $\mathrm{GF}(2^2)$. Find all the 2-flats that intersect on the 1-flat, $\{\alpha^{63} + \delta\alpha\}$, where $\delta \in \mathrm{GF}(2^2)$.

2.27 Let $\mathcal{L}$ be a line in the m-dimensional Euclidean geometry $\mathrm{EG}(m,q)$ over $\mathrm{GF}(q)$, not passing through the origin. Let α be a primitive element of $\mathrm{GF}(q^m)$. Prove that $\mathcal{L}, \alpha\mathcal{L}, \ldots, \alpha^{q^m-2}\mathcal{L}$ form $q^m - 1$ different lines in $\mathrm{EG}(m,q)$, not passing through the origin.

2.28 Find the cyclic classes of lines in $\mathrm{EG}(3,2^2)$ constructed in Problem 2.26.

2.29 Find the points of the two-dimensional projective geometry $\mathrm{PG}(2,2^2)$ over $\mathrm{GF}(2^2)$. Choose a point in $\mathrm{PG}(2,2^2)$ and give the lines that intersect at this point.

2.30 For odd m, prove that there are $(q^{m+1} - 1)/(q^2 - 1)$ nonprimitive lines.

References

[1] R. Lidl and H. Niederreiter, *Introduction to Finite Fields and Their Applications*, revised edn., Cambridge, Cambridge University Press, 1994.

[2] I. F. Blake and R. C. Mullin, *The Mathematical Theory of Coding*, New York, Academic Press, 1975.

[3] R. D. Carmichael, *Introduction to the Theory of Groups of Finite Orders*, Boston, MA, Ginn & Co., 1937.

[4] H. B. Mann, *Analysis and Design of Experiments*, New York, Dover, 1949.

[5] H. Tang, J. Xu, S. Lin, and K. Abdel-Ghaffar, "Codes on finite geometries," *IEEE Trans. Information Theory*, vol. 51, no. 2, pp. 572–596, February 2005.

[6] G. Birkhoff and S. Maclane, *A Survey of Modern Algebra*, New York, Macmillan, 1953,
[7] A. Clark, *Elements of Abstract Algebra*, New York, Dover, 1984.
[8] R. A. Dean, *Classical Abstract Algebra*, New York, Harper & Row, 1990.
[9] N. Deo, *Graph Theory with Applications to Engineering and Computer Science*, Englewood Cliffs, NJ, Prentice-Hall, 1974.
[10] D. S. Dummit and R. M. Foote, *Abstract Algebra*, 3rd edn., New York, John Wiley & Sons, 2004.
[11] T. W. Hungerford, *Abstract Algebra: An Introduction*, 2nd edn., New York, Saunders College Publishing, 1997.
[12] S. Lang, *Algebra*, 2nd edn., Reading, MA, Addison-Wesley, 1984.
[13] J. E. Maxfield and M. W. Maxfield, *Abstract Algebra and Solution by Radicals*, New York, Dover, 1992.
[14] B. L. Van der Waerden, *Modern Algebra*, vols. 1 and 2, New York, Ungar, 1949.
[15] D. B. West, *Introduction to Graph Theory*, 2nd edn., Upper Saddle River, NJ, Prentice-Hall, 2001.

Appendix A Table of Primitive Polynomials

Characteristic 2

m		m	
2	X^2+X+1	14	$X^{14}+X^{13}+X^{12}+X^2+1$
3	X^3+X^2+1	15	$X^{15}+X^{14}+1$
4	X^4+X^3+1	16	$X^{16}+X^{15}+X^{13}+X^4+1$
5	X^5+X^3+1	17	$X^{17}+X^{14}+1$
6	X^6+X^5+1	18	$X^{18}+X^{11}+1$
7	X^7+X^6+1	19	$X^{19}+X^{18}+X^{17}+X^{14}+1$
8	$X^8+X^7+X^6+X+1$	20	$X^{20}+X^{17}+1$
9	X^9+X^5+1	21	$X^{21}+X^{19}+1$
10	$X^{10}+X^7+1$	22	$X^{22}+X^{21}+1$
11	$X^{11}+X^9+1$	23	$X^{23}+X^{18}+1$
12	$X^{12}+X^{11}+X^{10}+X^4+1$	24	$X^{24}+X^{23}+X^{22}+X^{17}+1$
13	$X^{13}+X^{12}+X^{11}+X^8+1$		

Characteristic 3

m		m	
2	X^2+X+2	7	$X^7+X^6+X^4+1$
3	X^3+2X^2+1	8	X^8+X^5+2
4	X^4+X^3+2	9	$X^9+X^7+X^5+1$
5	$X^5+X^4+X^2+1$	10	$X^{10}+X^9+X^7+2$
6	X^6+X^5+2	11	$X^{11}+X^{10}+X^4+1$

Characteristic 5

m		m	
2	$X^2 + X + 2$	6	$X^6 + X^5 + 2$
3	$X^3 + X^2 + 2$	7	$X^7 + X^6 + 2$
4	$X^4 + X^3 + X + 3$	8	$X^8 + X^5 + X^3 + 3$
5	$X^5 + X^2 + 2$	9	$X^9 + X^7 + X^6 + 3$

Characteristic 7

m		m	
2	$X^2 + X + 3$	5	$X^5 + X^4 + 4$
3	$X^3 + X^2 + X + 2$	6	$X^6 + X^5 + X^4 + 3$
4	$X^4 + X^3 + X^2 + 3$	7	$X^7 + X^5 + 4$

Characteristic 11

m		m	
2	$X^2 + X + 7$	4	$X^4 + X^3 + 8$
3	$X^3 + X^2 + 3$	5	$X^5 + X^4 + X^3 + 3$

3 Linear Block Codes

There are two structurally different types of codes, block and convolutional codes. Both types of codes have been widely used for error control in communication and/or storage systems. Block codes can be divided into two categories, linear and nonlinear block codes. Nonlinear block codes are never used in practical applications and not much investigated. This chapter gives an introduction to linear block codes. The coverage of this chapter includes (1) fundamental concepts and structures of linear block codes; (2) specifications of these codes in terms of their generator and parity-check matrices; (3) their error-correction capabilities; (4) several important classes of linear block codes; and (5) encoding of two special classes of linear block codes, namely cyclic and quasi-cyclic codes. In our presentation, no proofs or derivations are provided; only descriptions and constructions of codes are given. We will begin our introduction with linear block codes with symbols from the binary field GF(2). Linear block codes over nonbinary fields will be given at the end of this chapter.

There are many excellent texts on the subject of error-control coding theory [1–15], which have extensive coverage of linear block codes. For in-depth study of linear block codes, readers are referred to these texts.

3.1 Introduction to Linear Block Codes

We assume that the output of an information source is a continuous sequence of binary symbols over GF(2), called an *information sequence*. The binary symbols in an information sequence are commonly called information bits, where a "bit" stands for a binary digit. In block coding, an information sequence is segmented into message blocks of fixed length; each message block consists of k information bits. There are 2^k distinct messages. At the channel encoder, each input message $\mathbf{u} = (u_0, u_1, \ldots, u_{k-1})$ of k information bits is encoded into a longer binary sequence $\mathbf{v} = (v_0, v_1, \ldots, v_{n-1})$ of n binary digits with $n > k$, according to certain encoding rules. This longer binary sequence $\mathbf{v}$ is called the *codeword* of the message $\mathbf{u}$. The binary digits in a codeword are called code bits. Since there are 2^k distinct messages, there are 2^k codewords, one for each distinct message. This set of 2^k codewords is said to form an (n,k) block code. For a block code to be useful, the 2^k codewords for the 2^k distinct messages must be distinct. The $n - k$ bits added to each input message by the channel encoder are called *redundant*

bits. These redundant bits carry no new information and their main function is to provide the code with the capability of *detecting* and *correcting* transmission errors caused by the channel noise or interferences. How to form these redundant bits such that an (n,k) block has good error-correcting capability is a major concern in designing the channel encoder. The ratio $R = k/n$ is called the *code rate*, which is interpreted as the average number of information bits carried by each code bit.

For a block code of length n with 2^k codewords, unless it has certain special structural properties, the encoding and decoding apparatus would be prohibitively complex for large k since the encoder has to store 2^k codewords of length n in a dictionary and the decoder has to perform a table (with 2^n entries) look-up to determine the transmitted codeword. Therefore, we must restrict our attention to block codes that can be implemented in a practical manner. A desirable structure for a block code is *linearity*.

Definition 3.1. A binary block code of length n with 2^k codewords is called an (n,k) *linear block code* if and only if its 2^k codewords form a k-dimensional subspace of the vector space V of all the n-tuples over GF(2).

3.1.1 Generator and Parity-Check Matrices

Since a binary (n,k) linear block code $\mathcal{C}$ is a k-dimensional subspace of the vector space of all the n-tuples over GF(2), there exist k *linearly independent codewords*, $\mathbf{g}_0, \mathbf{g}_1, \ldots, \mathbf{g}_{k-1}$, in $\mathcal{C}$ such that every codeword $\mathbf{v}$ in $\mathcal{C}$ is a *linear combination* of these k linearly independent codewords. These k linearly independent codewords in $\mathcal{C}$ form a *basis* $\mathcal{B}_c$ of $\mathcal{C}$. Using this basis, encoding can be done as follows. Let $\mathbf{u} = (u_0, u_1, \ldots, u_{k-1})$ be the message to be encoded. The codeword $\mathbf{v} = (v_0, v_1, \ldots, v_{n-1})$ for this message is given by the following linear combination of $\mathbf{g}_0, \mathbf{g}_1, \ldots, \mathbf{g}_{k-1}$, with the k message bits of $\mathbf{u}$ as the coefficients:

$$\mathbf{v} = u_0\mathbf{g}_0 + u_1\mathbf{g}_1 + \cdots + u_{k-1}\mathbf{g}_{k-1}. \tag{3.1}$$

We may arrange the k linearly independent codewords, $\mathbf{g}_0, \mathbf{g}_1, \ldots, \mathbf{g}_{k-1}$, of $\mathcal{C}$ as rows of a $k \times n$ matrix over GF(2) as follows:

$$\mathbf{G} = \begin{bmatrix} \mathbf{g}_0 \\ \mathbf{g}_1 \\ \vdots \\ \mathbf{g}_{k-1} \end{bmatrix} = \begin{bmatrix} g_{0,0} & g_{0,1} & \cdots & g_{0,n-1} \\ g_{1,0} & g_{1,1} & \cdots & g_{1,n-1} \\ \vdots & \vdots & \ddots & \vdots \\ g_{k-1,0} & g_{k-1,1} & \cdots & g_{k-1,n-1} \end{bmatrix}. \tag{3.2}$$

Then the codeword $\mathbf{v} = (v_0, v_1, \ldots, v_{n-1})$ for message $\mathbf{u} = (u_0, u_1, \ldots, u_{k-1})$ given by (3.1) can be expressed as the matrix product of $\mathbf{u}$ and $\mathbf{G}$ as follows:

$$\mathbf{v} = \mathbf{u} \cdot \mathbf{G}. \tag{3.3}$$

Therefore, the codeword $\mathbf{v}$ for a message $\mathbf{u}$ is simply a linear combination of the rows of matrix $\mathbf{G}$ with the information bits in the message $\mathbf{u}$ as the coefficients.

$\mathbf{G}$ is called a *generator matrix* of the (n,k) linear block code $\mathcal{C}$. Since $\mathcal{C}$ is spanned by the rows of $\mathbf{G}$, it is called the *row space* of $\mathbf{G}$. In general, an (n,k) linear block code has more than one basis. Consequently, a generator matrix of a given (n,k) linear block code is not unique. Any choice of a basis of $\mathcal{C}$ gives a generator matrix of $\mathcal{C}$. Obviously, the rank of a generator matrix of a linear block code $\mathcal{C}$ is equal to the dimension of $\mathcal{C}$.

Since a binary (n,k) linear block code $\mathcal{C}$ is a k-dimensional subspace of the vector space V of all the n-tuples over GF(2), its *null* (or *dual*) space, denoted $\mathcal{C}_d$, is an $(n-k)$-dimensional subspace of V that is given by the following set of n-tuples in V:

$$\mathcal{C}_d = \{\mathbf{w} \in V : \langle \mathbf{w}, \mathbf{v} \rangle = 0 \text{ for all } \mathbf{v} \in \mathcal{C}\}, \tag{3.4}$$

where $\langle \mathbf{w}, \mathbf{v} \rangle$ denotes the inner product of $\mathbf{w}$ and $\mathbf{v}$ (see Section 2.4). $\mathcal{C}_d$ may be regarded as a binary $(n,n-k)$ linear block code and is called the *dual code* of $\mathcal{C}$. Let $\mathcal{B}_d$ be a basis of $\mathcal{C}_d$. Then $\mathcal{B}_d$ consists of $n-k$ linearly independent codewords in $\mathcal{C}_d$. Let $\mathbf{h}_0, \mathbf{h}_1, \ldots, \mathbf{h}_{n-k-1}$ be the $n-k$ linearly independent codewords in $\mathcal{B}_d$. Then every codeword in $\mathcal{C}_d$ is a linear combination of these $n-k$ linearly independent codewords in $\mathcal{B}_d$. Form the following $(n-k) \times n$ matrix over GF(2):

$$\mathbf{H} = \begin{bmatrix} \mathbf{h}_0 \\ \mathbf{h}_1 \\ \vdots \\ \mathbf{h}_{n-k-1} \end{bmatrix} = \begin{bmatrix} h_{0,0} & h_{0,1} & \cdots & h_{0,n-1} \\ h_{1,0} & h_{1,1} & \cdots & h_{1,n-1} \\ \vdots & \vdots & \ddots & \vdots \\ h_{n-k-1,0} & h_{n-k-1,1} & \cdots & h_{n-k-1,n-1} \end{bmatrix}. \tag{3.5}$$

Then $\mathbf{H}$ is a generator matrix of the dual code $\mathcal{C}_d$ of the binary (n,k) linear block code $\mathcal{C}$. It follows from (3.2), (3.4), and (3.5) that $\mathbf{G} \cdot \mathbf{H}^{\mathrm{T}} = \mathbf{O}$, where $\mathbf{O}$ is a $k \times (n-k)$ zero matrix. Furthermore, $\mathcal{C}$ is also uniquely specified by the $\mathbf{H}$ matrix as follows: a binary n-tuple $\mathbf{v} \in V$ is a codeword in $\mathcal{C}$ if and only if $\mathbf{v} \cdot \mathbf{H}^{\mathrm{T}} = \mathbf{0}$ (the all-zero $(n-k)$-tuple), i.e.,

$$\mathcal{C} = \{\mathbf{v} \in V : \mathbf{v} \cdot \mathbf{H}^{\mathrm{T}} = \mathbf{0}\}. \tag{3.6}$$

$\mathbf{H}$ is called a *parity-check* matrix of $\mathcal{C}$ and $\mathcal{C}$ is said to be the *null space* of $\mathbf{H}$. Therefore, a linear block code is uniquely specified by two matrices, a generator matrix and a parity-check matrix. In general, encoding of a linear block code is based on a generator matrix of the code using (3.3) and decoding is based on a parity-check matrix of the code. Many classes of well-known linear block codes are constructed in terms of their parity-check matrices, as will be seen later in this chapter and other chapters. A parity-check matrix $\mathbf{H}$ of a linear block code is said to be a *full-rank* matrix if its rank is equal to the number of rows of $\mathbf{H}$. However, in many cases, a parity-check matrix of an (n,k) linear block code is not given as a full-rank matrix, i.e., the number of its rows is greater than its row rank, $n-k$. In this case, some rows of the given parity-check matrix $\mathbf{H}$ are linear combinations of a set of $n-k$ linearly independent rows. These extra rows are called *redundant rows*. The low-density parity-check codes constructed in Chapters 10 to 13 are specified by parity-check matrices that are in general not full-rank matrices.

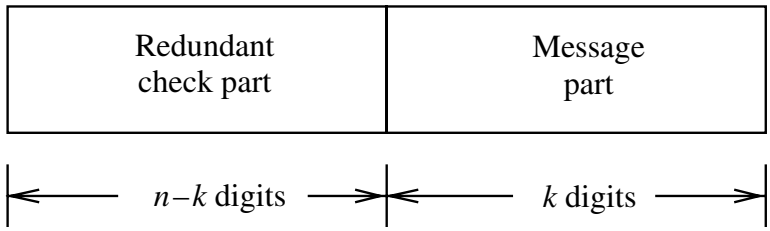


Figure 3.1 Systematic format of a codeword.

It is also desirable to require a codeword to have the format shown in Figure 3.1 where a codeword is divided into parts, namely the message part and the redundant check part. The message part consists of the k unaltered information digits and the redundant check part consists of $n-k$ parity-check digits. A linear block code with this structure is referred to as a *linear systematic code*. A linear systematic (n,k) block code is completely specified by a $k \times n$ generator matrix of the following form:

$$\mathbf{G} = \begin{bmatrix} \mathbf{g}_0 \\ \mathbf{g}_1 \\ \vdots \\ \mathbf{g}_{k-1} \end{bmatrix} = \begin{bmatrix} \underbrace{\begin{matrix} p_{0,0} & p_{0,1} & \cdots & p_{0,n-k-1} \\ p_{1,0} & p_{1,1} & \cdots & p_{1,n-k-1} \\ \vdots & \vdots & \ddots & \vdots \\ p_{k-1,0} & p_{k-1,1} & \cdots & p_{k-1,n-k-1} \end{matrix}}_{\mathbf{P}\text{ matrix}} & \underbrace{\begin{matrix} 1\,0 \cdots 0 \\ 0\,1 \cdots 0 \\ \vdots\,\vdots\,\ddots\,\vdots \\ 0\,0 \cdots 1 \end{matrix}}_{k\times k\text{ identity matrix }\mathbf{I}_k} \end{bmatrix}. \tag{3.7}$$

The generator matrix $\mathbf{G}$ consists of two submatrices, a $k \times (n-k)$ submatrix $\mathbf{P}$ on the left with entries over GF(2) and a $k \times k$ identity matrix $\mathbf{I}_k$ on the right. For simplicity, we write $\mathbf{G} = [\mathbf{P} \quad \mathbf{I}_k]$. Let $\mathbf{u} = (u_0, u_1, \ldots, u_{k-1})$ be the message to be encoded. Taking the linear combination of the rows of $\mathbf{G}$ given by (3.7) with the information bits in $\mathbf{u}$ as coefficients, we obtain the following corresponding codeword:

$$\begin{aligned} \mathbf{v} &= (v_0, v_1, \ldots, v_{n-k-1}, v_{n-k}, \ldots, v_{n-1}) \\ &= (u_0, u_1, \ldots, u_{k-1}) \cdot \mathbf{G}, \end{aligned} \tag{3.8}$$

where the n code bits are given by

$$v_{n-k+l} = u_l, \text{ for } l = 0, 1, \ldots, k-1 \tag{3.9}$$

and

$$v_j = u_0 p_{0,j} + u_1 p_{1,j} + \cdots + u_{k-1} p_{k-1,j}, \text{ for } j = 0, 1, \ldots, n-k-1. \tag{3.10}$$

Expression (3.9) shows that the rightmost k code bits of codeword $\mathbf{v}$ are identical to k information bits $u_0, u_1, \ldots, u_{k-1}$ to be encoded, and (3.10) shows that the leftmost $n-k$ code bits of $\mathbf{v}$ are linear sums of information bits. These $n-k$ code bits given by linear sums of information bits are called *parity-check bits* (or simply parity bits) and they are completely specified by the $n-k$ columns of the $\mathbf{P}$ submatrix of $\mathbf{G}$ given by (3.7). The $n-k$ equations given by (3.10) are called

parity-check equations of the code and the $\mathbf{P}$ submatrix of $\mathbf{G}$ is called the parity submatrix of $\mathbf{G}$. A generator matrix in the form given by (3.7) is said to be in systematic form.

Given a $k \times n$ generator matrix $\mathbf{G}'$ of an (n,k) linear block code $\mathcal{C}'$ not in systematic form, a generator matrix $\mathbf{G}$ in the systematic form of (3.7) can always be obtained by performing elementary operations on the rows of $\mathbf{G}'$ and then possibly taking column permutations. The $k \times n$ matrix $\mathbf{G}$ is called a *combinatorially equivalent matrix* of $\mathbf{G}'$. The systematic (n,k) linear block code $\mathcal{C}$ generated by $\mathbf{G}$ is called a *combinatorially equivalent code* of $\mathcal{C}'$. $\mathcal{C}$ and $\mathcal{C}'$ are possibly different in the arrangement (or order) of code bits in their codewords, i.e., a codeword in $\mathcal{C}$ is obtained by a fixed permutation of the positions of the code bits in a codeword of $\mathcal{C}'$, and vice versa. Two combinatorially equivalent (n,k) linear block codes give the same error performance.

If a generator matrix of an (n,k) linear block code $\mathcal{C}$ is given in the systematic form of (3.7), then its corresponding parity-check matrix in systematic form is given below:

$$\begin{aligned}\mathbf{H} &= \begin{bmatrix}\mathbf{I}_{n-k} & \mathbf{P}^{\mathrm{T}}\end{bmatrix} \\ &= \begin{bmatrix} 1 & 0 & \cdots & 0 & p_{0,0} & p_{1,0} & \cdots & h_{k-1,0} \\ 0 & 1 & \cdots & 0 & p_{0,1} & p_{1,1} & \cdots & h_{k-1,1} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 & p_{0,n-k-1} & p_{1,n-k-1} & \cdots & h_{k-1,n-k-1} \end{bmatrix}. \end{aligned} \tag{3.11}$$

It can be easily proved that $\mathbf{G} \cdot \mathbf{H}^{\mathrm{T}} = \mathbf{O}$.

3.1.2 Error Detection with Linear Block Codes

Consider an (n,k) linear block code $\mathcal{C}$ with an $(n-k) \times n$ parity-check matrix $\mathbf{H}$. Suppose a codeword $\mathbf{v} = (v_0, v_1, \ldots, v_{n-1})$ in $\mathcal{C}$ is transmitted over a BSC (or any binary-input, binary-output channel). Let $\mathbf{r} = (r_0, r_1, \ldots, r_{n-1})$ be the corresponding hard-decision received vector (n-tuple or sequence) at the input of the channel decoder. Because of the channel noise, the received vector $\mathbf{r}$ and the transmitted codeword $\mathbf{v}$ may differ in some places. Define the following vector sum of $\mathbf{r}$ and $\mathbf{v}$:

$$\begin{aligned}\mathbf{e} &= \mathbf{r} + \mathbf{v} \\ &= (e_0, e_1, \ldots, e_{n-1}) \\ &= (r_0 + v_0, r_1 + v_1, \ldots, r_{n-1} + v_{n-1}),\end{aligned} \tag{3.12}$$

where $e_j = r_j + v_j$ for $0 \le j < n$ and the addition $+$ is modulo-2 addition. Then, $e_j = 1$ for $r_j \neq v_j$ and $e_j = 0$ for $r_j = v_j$. Therefore, the positions of the 1-components in the n-tuple $\mathbf{e}$ are the places where $\mathbf{r}$ and $\mathbf{v}$ differ. At these places, transmission errors have occurred. Since $\mathbf{e}$ displays the pattern of transmission errors in $\mathbf{r}$, we called $\mathbf{e}$ the *error pattern* (or *vector*) during the transmission of the codeword $\mathbf{v}$. The 1-components in $\mathbf{e}$ are called *transmission errors* caused by the channel noise. For a BSC, an error can occur at any place with the same

probability p over the span of n places (the length of the code). There are $2^n - 1$ possible nonzero error patterns.

It follows from (3.12) that we can express the received vector $\mathbf{r}$ as

$$\mathbf{r} = \mathbf{v} + \mathbf{e}. \tag{3.13}$$

At the receiver, neither the transmitted codeword $\mathbf{v}$ nor the error pattern $\mathbf{e}$ is known. Upon receiving $\mathbf{r}$, the decoder must first determine whether there are transmission errors in $\mathbf{r}$. If the presence of errors is detected, then the decoder must estimate the error pattern $\mathbf{e}$ on the basis of the code $\mathcal{C}$ and the channel information provided to the decoder. Let $\mathbf{e}^*$ denote the estimated error pattern. Then the estimated transmitted codeword is given by $\mathbf{v}^* = \mathbf{r} + \mathbf{e}^*$.

To check whether a received vector $\mathbf{r}$ contains transmission errors, we compute the following $(n-k)$-tuples over GF(2),

$$\begin{aligned} \mathbf{s} &= (s_0, s_1, \ldots, s_{n-k-1}) \\ &= \mathbf{r} \cdot \mathbf{H}^{\mathrm{T}}. \end{aligned} \tag{3.14}$$

Note that $\mathbf{r}$ is an n-tuple in the vector space V of all the n-tuples over GF(2). Recall that an n-tuple $\mathbf{r}$ in V is a codeword in $\mathcal{C}$ if and only if $\mathbf{r} \cdot \mathbf{H}^{\mathrm{T}} = 0$. Therefore, if $\mathbf{s} \neq \mathbf{0}$, $\mathbf{r}$ is not a codeword in $\mathcal{C}$. In this case, the transmitter transmitted a codeword but the receiver received a vector that is not a codeword. Hence, the presence of transmission errors is detected. If $\mathbf{s} = \mathbf{0}$, then $\mathbf{r}$ is a codeword in $\mathcal{C}$. In this case, the channel decoder assumes that $\mathbf{r}$ is error-free and accepts $\mathbf{r}$ as the transmitted codeword. However, in the event that $\mathbf{r}$ is a codeword in $\mathcal{C}$ but differs from the transmitted codeword $\mathbf{v}$, on accepting $\mathbf{r}$ as the transmitted codeword, the decoder commits a decoding error. This occurs when the error pattern $\mathbf{e}$ caused by the noise changes the transmitted codeword $\mathbf{v}$ into another codeword in $\mathcal{C}$. This happens when and only when the error pattern $\mathbf{e}$ is identical to a nonzero codeword in $\mathcal{C}$. An error pattern of this type is called an *undetectable error pattern.* There are $2^k - 1$ such undetectable error patterns. Since the $(n-k)$-tuple $\mathbf{s}$ over GF(2) is used for detecting whether the received vector $\mathbf{r}$ contains transmission errors, it is called the *syndrome* of $\mathbf{r}$.

3.1.3 Weight Distribution and Minimum Hamming Distance of A Linear Block Code

Let $\mathbf{v} = (v_0, v_1, \ldots, v_{n-1})$ be an n-tuple over GF(2). The *Hamming weight* (or simply weight) of $\mathbf{v}$, denoted $w(\mathbf{v})$, is defined as the number of nonzero components in $\mathbf{v}$. Consider an (n,k) linear block code $\mathcal{C}$ with code symbols from GF(2). For $0 \leq i \leq n$, let A_i be the number of codewords in $\mathcal{C}$ with Hamming weight i. Then the numbers $A_0, A_1, \ldots, A_n$ are called the *weight distribution* of $\mathcal{C}$. It is clear that $A_0 + A_1 + \cdots + A_n = 2^k$. Since there is one and only one all-zero codeword in a linear block code, $A_0 = 1$. The *smallest weight* of the nonzero codewords in $\mathcal{C}$, denoted $w_{\min}(\mathcal{C})$, is called the *minimum weight* of $\mathcal{C}$. Mathematically, the minimum weight of $\mathcal{C}$ is given as follows:

$$w_{\min}(\mathcal{C}) = \min\{w(\mathbf{v}) \colon \mathbf{v} \in \mathcal{C}, \mathbf{v} \neq 0\}. \tag{3.15}$$

Suppose $\mathcal{C}$ is used for error control over a BSC with transition probability p. Recall that an undetectable error pattern is an error pattern that is identical to a nonzero codeword in $\mathcal{C}$. When such an error pattern occurs, the decoder will not be able to detect the presence of transmission errors and hence will commit a decoding error. There are A_i undetectable error patterns of weight i; each occurs with probability $p^i(1-p)^{n-i}$. Hence, the total probability that an undetectable error pattern with i errors occurs is $A_i p^i(1-p)^{n-i}$. Then the probability that the decoder fails to detect the presence of transmission errors, called the *probability of an undetected error*, is equal to

$$P_u(E) = \sum_{i=1}^{n} A_i p^i (1-p)^{n-i}. \tag{3.16}$$

Therefore, the weight distribution of a linear block code completely determines its probability of an undetected error. It has been proved that in the *ensemble* of (n,k) linear block codes over GF(2), there exist codes with the probability of an undetected error, $P_u(E)$, upper bounded by $2^{-(n-k)}$, i.e.,

$$P_u(E) \leq 2^{-(n-k)}. \tag{3.17}$$

A code that satisfies the above upper bound is said to be a good error-detection code.

Let $\mathbf{v}$ and $\mathbf{w}$ be two n-tuples over GF(2). The Hamming distance (or simply distance) between $\mathbf{v}$ and $\mathbf{w}$, denoted $d(\mathbf{v},\mathbf{w})$, is defined as the number of places where $\mathbf{v}$ and $\mathbf{w}$ differ. The Hamming distance is a metric function that satisfies the *triangle inequality*. Let $\mathbf{v}$, $\mathbf{w}$, and $\mathbf{x}$ be three n-tuples over GF(2). Then

$$d(\mathbf{v},\mathbf{w}) + d(\mathbf{w},\mathbf{x}) \geq d(\mathbf{v},\mathbf{x}). \tag{3.18}$$

It follows from the definition of the Hamming distance between two n-tuples and the Hamming weight of an n-tuple that the Hamming distance between $\mathbf{v}$ and $\mathbf{w}$ is equal to the Hamming weight of the vector sum of $\mathbf{v}$ and $\mathbf{w}$, i.e., $d(\mathbf{v},\mathbf{w}) = w(\mathbf{v}+\mathbf{w})$.

The minimum distance of an (n,k) linear block code $\mathcal{C}$, denoted by $d_{\min}(\mathcal{C})$, is defined as the *smallest* Hamming distance between two different codewords in $\mathcal{C}$, i.e.,

$$d_{\min}(\mathcal{C}) = \min\{d(\mathbf{v},\mathbf{w})\colon \mathbf{v},\mathbf{w} \in \mathcal{C}, \mathbf{v} \neq \mathbf{w}\}. \tag{3.19}$$

Using the fact that $d(\mathbf{v},\mathbf{w}) = w(\mathbf{v}+\mathbf{w})$, we can easily prove that the minimum distance $d_{\min}(\mathcal{C})$ of $\mathcal{C}$ is equal to the minimum weight $w_{\min}(\mathcal{C})$ of $\mathcal{C}$. This follows from (3.19),

$$\begin{aligned} d_{\min}(\mathcal{C}) &= \min\{d(\mathbf{v},\mathbf{w})\colon \mathbf{v},\mathbf{w} \in \mathcal{C}, \mathbf{v} \neq \mathbf{w}\} \\ &= \min\{w(\mathbf{v}+\mathbf{w})\colon \mathbf{v},\mathbf{w} \in \mathcal{C}, \mathbf{v} \neq \mathbf{w}\} \\ &= \min\{w(\mathbf{x})\colon \mathbf{x} \in \mathcal{C}, \mathbf{x} \neq \mathbf{0}\} \\ &= w_{\min}(\mathcal{C}). \end{aligned} \tag{3.20}$$

Therefore, for a linear block code, determining its minimum distance is equivalent to determining its minimum weight. The weight structure (or weight distribution) of a linear block code $\mathcal{C}$ is related to a parity-check matrix $\mathbf{H}$ of the code, as given by the following three theorems. We state these theorems without proofs.

Theorem 3.1. Let $\mathcal{C}$ be an (n,k) linear block code with a parity-check matrix $\mathbf{H}$. For each codeword in $\mathcal{C}$ with weight i, there exist i columns in $\mathbf{H}$ whose vector sum gives a zero (column) vector. Conversely, if there are i columns in $\mathbf{H}$ whose vector sum results in a zero (column) vector, there is a codeword in $\mathcal{C}$ with weight i.

Theorem 3.2. The minimum weight (or minimum distance) of an (n,k) linear block code $\mathcal{C}$ with a parity-check matrix $\mathbf{H}$ is equal to the smallest number of columns in $\mathbf{H}$ whose vector sum is a zero vector.

Theorem 3.3. For an (n,k) linear block code $\mathcal{C}$ given by the null space of a parity-check matrix $\mathbf{H}$, if there are no $d-1$ or fewer columns in $\mathbf{H}$ that sum to a zero vector, the minimum distance (or weight) of $\mathcal{C}$ is at least d.

Theorem 3.3 gives a *lower bound* on the minimum distance (or weight) of a linear block code. In general, it is very hard to determine the exact minimum distance (or weight) of a linear block code; however, it is much easier to give a lower bound on its minimum distance (or weight). This will be seen later.

The weight distribution of a linear block code $\mathcal{C}$ actually gives the *distance distribution* of the nonzero codewords with respect to the all-zero codeword $\mathbf{0}$. For $1 \leq i \leq n$, the number A_i of codewords in $\mathcal{C}$ with weight i is simply equal to the number of codewords that are at distance i from the all-zero codeword $\mathbf{0}$. Owing to the linear structure of $\mathcal{C}$, A_i also gives the number of codewords in $\mathcal{C}$ that are at a distance i from *any* fixed codeword $\mathbf{v}$ in $\mathcal{C}$. Therefore, the weight distribution, $\{A_0, A_1, \ldots, A_n\}$, of $\mathcal{C}$ is also the distance distribution of $\mathcal{C}$ with respect to any codeword in $\mathcal{C}$.

The capabilities of a linear block code $\mathcal{C}$ for detecting and correcting random errors over a BSC with hard-decision decoding are determined by the minimum distance of $\mathcal{C}$ and its error performance with soft-decision MLD is determined by its distance (or weight) distribution.

For an (n,k) linear block code $\mathcal{C}$ with minimum distance $d_{\min}(\mathcal{C})$, no error pattern with $d_{\min}(\mathcal{C})-1$ or fewer errors can change a transmitted codeword into another codeword in $\mathcal{C}$. Therefore, any error pattern with $d_{\min}(\mathcal{C})-1$ or fewer errors will result in a received vector that is not a codeword in $\mathcal{C}$ and hence its syndrome will not be equal to zero. Therefore, all the error patterns with $d_{\min}(\mathcal{C})-1$ or fewer errors are detectable by the channel decoder. However, if a codeword $\mathbf{v}$ is transmitted and an error pattern with $d_{\min}(\mathcal{C})$ errors that happens to be a codeword in $\mathcal{C}$ at a distance $d_{\min}(\mathcal{C})$ from $\mathbf{v}$ occurs, then the received vector $\mathbf{r}$ is a codeword and its syndrome is zero. Such an error pattern is an undetectable error. This is to say that all the error patterns with $d_{\min}(\mathcal{C})-1$ or fewer errors are guaranteed to be detectable; however, detection is not guaranteed for error patterns with $d_{\min}(\mathcal{C})$ or more errors. For this reason, $d_{\min}(\mathcal{C})-1$ is called the *error-detecting capability*

of the code $\mathcal{C}$. The number of guaranteed detectable error patterns is

$$\binom{n}{1} + \binom{n}{2} + \cdots + \binom{n}{d_{\min}(\mathcal{C}) - 1},$$

where $\binom{n}{i}$ is a binomial coefficient. For large n, this number of guaranteed detectable error patterns is only a small fraction of the $2^n - 2^k + 1$ detectable error patterns.

So far, we have focused on error detection with a linear block code. Decoding and the error-correcting capability of a linear block code will be discussed in the next section. We will show that, with hard-decision decoding, a linear block code $\mathcal{C}$ with minimum distance $d_{\min}(\mathcal{C})$ is capable of correcting $\lfloor (d_{\min}(\mathcal{C}) - 1)/2 \rfloor$ or fewer random errors over a span of n transmitted code digits.

3.1.4 Decoding of Linear Block Codes

Consider an (n,k) linear block code $\mathcal{C}$ with a parity-check matrix $\mathbf{H}$ and minimum distance $d_{\min}(\mathcal{C})$. Suppose a codeword in $\mathcal{C}$ is transmitted and $\mathbf{r}$ is the received vector. For maximum-likelihood decoding (MLD) as described in Chapter 1, $\mathbf{r}$ is decoded into a codeword $\mathbf{v}$ such that the conditional probability $P(\mathbf{r}|\mathbf{v})$ is maximized. For a BSC, this is equivalent to decoding $\mathbf{r}$ into a codeword $\mathbf{v}$ such that the Hamming distance $d(\mathbf{r}, \mathbf{v})$ between $\mathbf{r}$ and $\mathbf{v}$ is minimized. This is called *minimum-distance* (or *nearest-neighbor*) decoding. With minimum-distance decoding, the decoder has to compute the distance between $\mathbf{r}$ and every codeword in $\mathcal{C}$ and then choose a codeword $\mathbf{v}$ (not necessarily unique) that is closest to $\mathbf{r}$ (i.e., $d(\mathbf{r}, \mathbf{v})$ is the smallest) as the decoded codeword. This decoding is called a *complete error-correction decoding* and requires a total of 2^k computations of the distances between $\mathbf{r}$ and the 2^k codewords in $\mathcal{C}$. For large k, implementation of this complete decoder is practically impossible. However, for many linear block codes, efficient algorithms have been developed for incomplete error-correction decoding to achieve good error performance with vastly reduced decoding complexity.

No matter which codeword in $\mathcal{C}$ is transmitted over a noisy channel, the received vector $\mathbf{r}$ is one of the 2^n n-tuples over GF(2). Let $\mathbf{v}_0 = \mathbf{0}, \mathbf{v}_1, \ldots, \mathbf{v}_{2^k-1}$ be the codewords in $\mathcal{C}$. Any decoding scheme used at the decoder is a rule to partition the vector space V of all the n-tuples over GF(2) into 2^k regions; each region contains one and only one codeword in $\mathcal{C}$, as shown in Figure 3.2. Decoding is done to find the region that contains the received vector $\mathbf{r}$. Then decode $\mathbf{r}$ into the codeword $\mathbf{v}$ that is contained in the region. These regions are called *decoding regions*. For MLD, the decoding regions for the 2^k codewords are, for $0 \le i < 2^k$,

$$D(\mathbf{v}_i) = \{\mathbf{r} \in V : P(\mathbf{r}|\mathbf{v}_i) \ge P(\mathbf{r}|\mathbf{v}_j), j \ne i\}.$$

For minimum-distance decoding (MLD for the BSC), the decoding regions for the 2^k codewords are, for $0 \le i < 2^k$,

$$D(\mathbf{v}_i) = \{\mathbf{r} \in V : d(\mathbf{r}, \mathbf{v}_i) \le d(\mathbf{r}, \mathbf{v}_j), j \ne i\}.$$

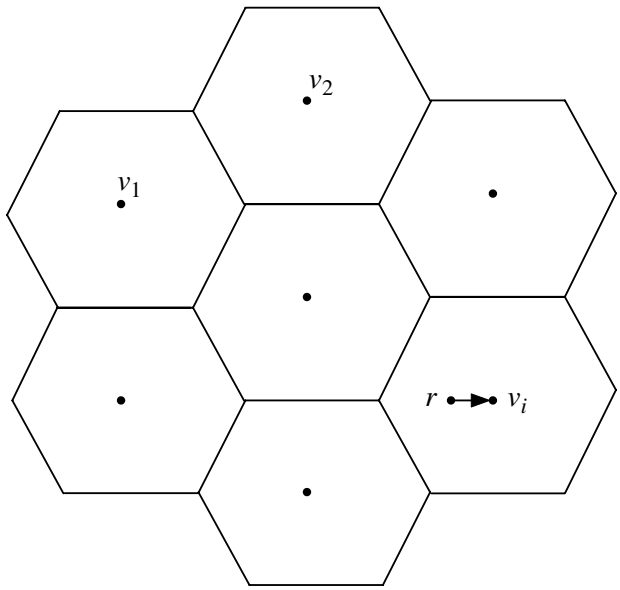


Figure 3.2 Decoding regions.

An algebraic method to partition the 2^n possible received vectors into 2^k decoding regions can be implemented as follows. First, we arrange the 2^k codewords in $\mathcal{C}$ as the top row (or 0th row) of a $2^{n-k} \times 2^k$ array with the all-zero codeword $\mathbf{v}_0 = \mathbf{0}$ as the first entry as shown in Table 3.1. Then we form the rest of the rows of the array one at a time. Suppose we have formed the $(j-1)$th row of the array $1 \leq j \leq 2^{n-k}$. To form the jth row of the array, we choose a vector $\mathbf{e}_j$ in V that is not contained in the previous $j-1$ rows of the array. Form the jth row of the array by adding $\mathbf{e}_j$ to each codeword $\mathbf{v}_i$ in the top row of the array and placing the vector sum $\mathbf{e}_j + \mathbf{v}_i$ under $\mathbf{v}_i$. The array is completed when no vector can be chosen from V. This array is called a *standard array* for the code $\mathcal{C}$. Each row of the array is called a *coset* of $\mathcal{C}$. The first element of each coset is called the *coset leader*. For $0 \leq j < 2^{n-k}$, the jth coset is given by

$$\mathbf{e}_j + \mathcal{C} = \{\mathbf{e}_j + \mathbf{v}_i \colon \mathbf{v}_i \in \mathcal{C}, 0 \leq i < 2^k\}, \tag{3.21}$$

where $\mathbf{e}_j$ is the coset leader.

A standard array for an (n,k) linear block code $\mathcal{C}$ with an $(n-k) \times n$ parity-check matrix $\mathbf{H}$ has the following structural properties that can be easily proved (see Problem 3.8): (1) the sum of two vectors in the same coset is a codeword in $\mathcal{C}$; (2) no two vectors in the same coset or in two different cosets are the same; (3) every n-tuple in the vector space V of all the n-tuples over GF(2) appears once and only once in the array; (4) all the vectors in a coset have the same syndrome which is the syndrome of the coset leader, i.e., $(\mathbf{e}_j + \mathbf{v}_i) \cdot \mathbf{H}^{\mathrm{T}} = \mathbf{e}_j \cdot \mathbf{H}^{\mathrm{T}}$ (since $\mathbf{v}_i \cdot \mathbf{H}^{\mathrm{T}} = \mathbf{0}$); and (5) different cosets have different syndromes. Since there are 2^{n-k} different $(n-k)$-tuple syndromes with respect to the parity-check matrix $\mathbf{H}$ of the code $\mathcal{C}$ and there are 2^{n-k} cosets in a standard array for $\mathcal{C}$, it follows from

Table 3.1. A standard array for an (n,k) linear block code

Cosets	Coset leaders					
$\mathcal{C}$	$\mathbf{e}_0 = \mathbf{v}_0 = \mathbf{0}$	$\mathbf{v}_1$	$\cdots$	$\mathbf{v}_i$	$\cdots$	$\mathbf{v}_{2^k-1}$
$\mathbf{e}_1 + \mathcal{C}$	$\mathbf{e}_1$	$\mathbf{e}_1 + \mathbf{v}_1$	$\cdots$	$\mathbf{e}_1 + \mathbf{v}_i$	$\cdots$	$\mathbf{e}_1 + \mathbf{v}_{2^k-1}$
$\mathbf{e}_2 + \mathcal{C}$	$\mathbf{e}_2$	$\mathbf{e}_2 + \mathbf{v}_1$	$\cdots$	$\mathbf{e}_2 + \mathbf{v}_i$	$\cdots$	$\mathbf{e}_2 + \mathbf{v}_{2^k-1}$
$\vdots$	$\vdots$	$\vdots$	$\cdots$	$\vdots$	$\cdots$	$\vdots$
$\mathbf{e}_{2^{n-k}-1} + \mathcal{C}$	$\mathbf{e}_{2^{n-k}-1}$	$\mathbf{e}_{2^{n-k}-1} + \mathbf{v}_1$	$\cdots$	$\mathbf{e}_{2^{n-k}-1} + \mathbf{v}_i$	$\cdots$	$\mathbf{e}_{2^{n-k}-1} + \mathbf{v}_{2^k-1}$

properties (4) and (5) that there is a *one-to-one correspondence* between a coset leader and an $(n-k)$-tuple syndrome.

A standard array for an (n,k) linear block code $\mathcal{C}$ consists of 2^k columns; each column contains one and only one codeword at the top of the column. For $0 \le i < 2^k$, the ith column consists of the following set of 2^{n-k} n-tuples:

$$D(\mathbf{v}_i) = \{\mathbf{v}_i, \mathbf{e}_1 + \mathbf{v}_i, \mathbf{e}_2 + \mathbf{v}_i, \ldots, \mathbf{e}_{2^{n-k}-1} + \mathbf{v}_i\}, \tag{3.22}$$

where each element is the vector sum of the ith codeword $\mathbf{v}_i$ and a coset leader $\mathbf{e}_j$ (note that $\mathbf{e}_0 = \mathbf{0}$) with $0 \le j < 2^{n-k}$. We see that the ith codeword $\mathbf{v}_i$ is the only codeword in $D(\mathbf{v}_i)$. The 2^k columns of a standard array for $\mathcal{C}$ form a partition of the vector space V of all the n-tuples over GF(2). These 2^k columns of the array can be used as the regions for decoding $\mathcal{C}$. If the received vector $\mathbf{r}$ is found in the ith column $D(\mathbf{v}_i)$, we decode $\mathbf{r}$ into $\mathbf{v}_i$. From the structure of a standard array for $\mathcal{C}$, we can easily check that (1) if the ith codeword $\mathbf{v}_i$ is transmitted and the error pattern caused by the channel noise is a coset leader $\mathbf{e}_j$, then the received vector $\mathbf{r} = \mathbf{v}_i + \mathbf{e}_j$ is in the column $D(\mathbf{v}_i)$ which contains $\mathbf{v}_i$; and (2) if $\mathbf{v}_i$ is transmitted but the error pattern is not a coset leader, then the received vector $\mathbf{r}$ is not in column $D(\mathbf{v}_i)$ (see Problem 3.9). Therefore, using the columns of a standard array of an (n,k) linear block code $\mathcal{C}$ as decoding regions, decoding is correct (i.e., $\mathbf{r}$ is decoded into the transmitted codeword) if and only if a codeword $\mathbf{v}_i$ is transmitted and the error pattern caused by the channel noise is identical to a coset leader. This is to say that the $2^{n-k} - 1$ nonzero coset leaders of a standard array are all the correctable error patterns (i.e., they result in correct decoding). To minimize the probability of a decoding error, the error patterns that are *most likely* to occur for a given channel should be chosen as the coset leaders.

For a BSC, an error pattern of smaller weight (or smaller number of errors) is more probable than an error pattern with larger weight (or larger number of errors). So, when a standard array for a linear block code $\mathcal{C}$ is formed, each coset leader should be chosen to be an n-tuple of the least weight from the remaining available n-tuples in V. Choosing coset leaders in this manner, each coset leader has the minimum weight in each coset. In this case, decoding based on a standard array for $\mathcal{C}$ is minimum-distance decoding (or MLD for a BSC). A standard array formed in this way is called an *optimal standard array* for $\mathcal{C}$.

The minimum distance $d_{\min}(\mathcal{C})$ of a linear block code $\mathcal{C}$ is either odd or even. Let $t = \lfloor (d_{\min}(\mathcal{C}) - 1)/2 \rfloor$ where $\lfloor x \rfloor$ denotes the integer part of x (or the largest

Table 3.2. A look-up decoding table

Syndromes	Correctable error patterns
$\mathbf{0}$	$\mathbf{e}_0 = \mathbf{0}$
$\mathbf{s}_1$	$\mathbf{e}_1$
$\mathbf{s}_2$	$\mathbf{e}_2$
$\cdots$	$\cdots$
$\mathbf{s}_{2^{n-k}-1}$	$\mathbf{e}_{2^{n-k}-1}$

integer equal to or less than x). Then $2t+1 \leq d_{\min}(\mathcal{C}) \leq 2t+2$. It can be shown that all the n-tuples over GF(2) of weight t or less can be used as coset leaders in an optimal standard array for $\mathcal{C}$ (see Problem 3.10). Also, it can be shown that there is at least one n-tuple of weight $t+1$ that cannot be used as a coset leader (see Problem 3.10). When this error pattern occurs, received vector $\mathbf{r}$ will be decoded incorrectly. Therefore, for a linear code $\mathcal{C}$ with minimum distance $d_{\min}(\mathcal{C})$, any error pattern with $t = \lfloor (d_{\min}(\mathcal{C})-1)/2 \rfloor$ or fewer errors is guaranteed correctable (i.e., resulting in correct decoding), but not all the error patterns with $t+1$ or more errors. The parameter $t = \lfloor (d_{\min}(\mathcal{C})-1)/2 \rfloor$ is called the *error-correction capability* of $\mathcal{C}$. We say that $\mathcal{C}$ is capable of correcting t or fewer random errors and $\mathcal{C}$ is called a t-error-correcting code.

Decoding of an (n,k) linear block code $\mathcal{C}$ based on an optimal standard array for the code requires a memory to store 2^n n-tuples. For large n, the size of the memory will be prohibitively large and implementation of a standard-array-based decoding becomes practically impossible. However, the decoding can be significantly simplified by using the following facts: (1) the coset leaders form all the correctable error patterns; and (2) there is a one-to-one correspondence between an $(n-k)$-tuple syndrome and a coset leader. From these two facts, we form a table with only two columns that consists of 2^{n-k} coset leaders (correctable error patterns) in one column and their corresponding syndromes in another column, as shown in Table 3.2. Decoding of a received vector $\mathbf{r}$ is carried out in three steps.

1. Compute the syndrome of $\mathbf{r}$, $\mathbf{s} = \mathbf{r} \cdot \mathbf{H}^{\mathrm{T}}$.
2. Find the coset leader $\mathbf{e}$ in the table whose syndrome is equal to $\mathbf{s}$. Then $\mathbf{e}$ is assumed to be the error pattern caused by the channel noise.
3. Decode $\mathbf{r}$ into the codeword $\mathbf{v} = \mathbf{r} + \mathbf{e}$.

The above decoding is called *syndrome decoding* or *table-look-up decoding*. With this decoding, the decoder complexity is drastically reduced compared with standard-array-based decoding.

For a long code with large $n-k$, a complete table-look-up decoder is still very complex, requiring a very large memory to store the look-up table. If we limit ourself to correcting only the error patterns guaranteed by the error-correcting capability $t = \lfloor (d_{\min}(\mathcal{C})-1)/2 \rfloor$ of the code, then the size of the look-up table can

be further reduced. The new table consists of only

$$N_t = \binom{n}{0} + \binom{n}{1} + \cdots + \binom{n}{t}$$

correctable error patterns guaranteed by the minimum distance $d_{\min}(\mathcal{C})$ of the code. With this new table, decoding of a received vector $\mathbf{r}$ consists of the following four steps.

1. Compute the syndrome $\mathbf{s}$ of $\mathbf{r}$.
2. If the syndrome $\mathbf{s}$ corresponds to a correctable error pattern $\mathbf{e}$ listed in the table, go to Step 3, otherwise go to Step 4.
3. Decode $\mathbf{r}$ into the codeword $\mathbf{v} = \mathbf{r} + \mathbf{e}$.
4. Declare a *decoding failure*. In this case, the presence of errors is detected but the decoder fails to correct the errors.

With the above partial table-look-up decoding, the number of errors to be corrected is bounded by the error-correction capability $t = \lfloor (d_{\min}(\mathcal{C}) - 1)/2 \rfloor$ of the code. This is called *bound-distance decoding*.

Many classes of linear block codes with good error-correction capabilities have been constructed over the years. Efficient algorithms to carry out the bound-distance decoding of these classes of codes have been devised. Two classes of these codes will be presented in later sections of this chapter.

3.2 Cyclic Codes

Let $\mathbf{v} = (v_0, v_1, v_2, \ldots, v_{n-1})$ be an n-tuple over GF(2). If we shift every component of $\mathbf{v}$ cyclically one place to the right, we obtain the following n-tuple:

$$\mathbf{v}^{(1)} = (v_{n-1}, v_0, v_1, \ldots, v_{n-2}), \tag{3.23}$$

which is called the *right cyclic-shift* (or simply cyclic-shift) of $\mathbf{v}$.

Definition 3.2. An (n,k) linear block code $\mathcal{C}$ is said to be *cyclic* if the cyclic-shift of each codeword in $\mathcal{C}$ is also a codeword in $\mathcal{C}$.

Cyclic codes form a very special type of linear block codes. They have encoding advantage over many other types of linear block codes. Encoding of this type of code can be implemented with simple shift-registers with feedback connections. Many classes of cyclic codes with large minimum distances have been constructed; and, furthermore, efficient algebraic hard-decision decoding algorithms for some of these classes of cyclic codes have been developed. This section gives an introduction to cyclic codes.

To analyze the structural properties of a cyclic code, a codeword $\mathbf{v} = (v_0, v_1, \ldots, v_{n-1})$ is represented by a polynomial over GF(2) of degree $n-1$ or less with the components of $\mathbf{v}$ as coefficients as follows:

$$\mathbf{v}(X) = v_0 + v_1 X + \cdots + v_{n-1} X^{n-1}. \tag{3.24}$$

This polynomial is called a *code polynomial*. In polynomial form, an (n,k) cyclic code $\mathcal{C}$ consists of 2^k code polynomials. The code polynomial corresponding to the all-zero codeword is the zero polynomial. All the other $2^k - 1$ code polynomials corresponding to the $2^k - 1$ nonzero codewords in $\mathcal{C}$ are nonzero polynomials.

Some important structural properties of cyclic codes are presented in the following without proofs. References [1–6,11] contain good and extensive coverage of the structure and construction of cyclic codes.

In an (n,k) cyclic code $\mathcal{C}$, every nonzero code polynomial has degree *at least* $n-k$ but not greater than $n-1$. There exists one and only one code polynomial $\mathbf{g}(X)$ of degree $n-k$ of the following form:

$$\mathbf{g}(X) = 1 + g_1 X + g_2 X^2 + \cdots + g_{n-k-1} X^{n-k-1} + X^{n-k}. \tag{3.25}$$

Therefore, $\mathbf{g}(X)$ is a nonzero code polynomial of *minimum degree* and is *unique*. Every code polynomial $\mathbf{v}(X)$ in $\mathcal{C}$ is divisible by $\mathbf{g}(X)$, i.e., is a *multiple* of $\mathbf{g}(X)$. Moreover, every polynomial over GF(2) of degree $n-1$ or less that is divisible by $\mathbf{g}(X)$ is a code polynomial in $\mathcal{C}$. Therefore, an (n,k) cyclic code $\mathcal{C}$ is completely specified by the unique polynomial $\mathbf{g}(X)$ of degree $n-k$ given by (3.25). This unique nonzero code polynomial $\mathbf{g}(X)$ of minimum degree in $\mathcal{C}$ is called the *generator polynomial* of the (n,k) cyclic code $\mathcal{C}$. The degree of $\mathbf{g}(X)$ is simply the number of parity-check bits of the code.

Since each code polynomial $\mathbf{v}(X)$ in $\mathcal{C}$ is a multiple of $\mathbf{g}(X)$ (including the zero code polynomial), it can be expressed as the following product:

$$\mathbf{v}(X) = \mathbf{m}(X)\mathbf{g}(X), \tag{3.26}$$

where $\mathbf{m}(X) = m_0 + m_1 X + \cdots + m_{k-1} X^{k-1}$ is a polynomial over GF(2) of degree $k-1$ or less. If $\mathbf{m} = (m_0, m_1, \ldots, m_{k-1})$ is the message to be encoded, then $\mathbf{m}(X)$ is the polynomial representation of $\mathbf{m}$ (called a *message polynomial*) and $\mathbf{v}(X)$ is the corresponding code polynomial of the message polynomial $\mathbf{m}(X)$. With this encoding, the corresponding $k \times n$ generator matrix of the (n,k) cyclic code $\mathcal{C}$ is given as follows:

$$\mathbf{G} = \begin{bmatrix} 1 & g_1 & g_2 & \cdots & g_{n-k-1} & 1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 1 & g_1 & \cdots & g_{n-k-2} & g_{n-k-1} & 1 & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & g_1 & g_2 & g_3 & \cdots & g_{n-k-1} & 1 \end{bmatrix}. \tag{3.27}$$

Note that $\mathbf{G}$ is simply obtained by using the n-tuple representation of the generator polynomial $\mathbf{g}(X)$ as the first row and its $k-1$ right cyclic-shifts as the other $k-1$ rows. $\mathbf{G}$ is not in systematic form but can be put into systematic form by elementary row operations without column permutations.

A very important property of the generator polynomial $\mathbf{g}(X)$ of an (n,k) cyclic code $\mathcal{C}$ over GF(2) is that $\mathbf{g}(X)$ divides $X^n + 1$. Consequently, $X^n + 1$ can be expressed as the following product:

$$X^n + 1 = \mathbf{g}(X)\mathbf{f}(X), \tag{3.28}$$

where $\mathbf{f}(X) = 1 + f_1 X + \cdots + f_{k-1}X^{k-1} + X^k$ is a polynomial of degree k over GF(2). Let

$$\begin{aligned}\mathbf{h}(X) &= X^k \mathbf{f}(X^{-1}) \\ &= 1 + h_1 X + \cdots + h_{k-1}X^{k-1} + X^k \end{aligned} \tag{3.29}$$

be the *reciprocal polynomial* of $\mathbf{f}(X)$. It is easy to prove that $\mathbf{h}(X)$ also divides $X^n + 1$. Form the following $(n-k) \times n$ matrix over GF(2) with the n-tuple representation $\mathbf{h} = (1, h_1, \ldots, h_{k-1}, 1, 0, \ldots, 0)$ of $\mathbf{h}(X)$ as the first row and its $n-k-1$ cyclic-shifts as the other $n-k-1$ rows:

$$\mathbf{H} = \begin{bmatrix} 1 & h_1 & h_2 & \cdots & h_{k-1} & 1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 1 & h_1 & \cdots & h_{k-2} & h_{k-1} & 1 & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & h_1 & h_2 & h_3 & \cdots & h_{k-1} & 1 \end{bmatrix}. \tag{3.30}$$

Then $\mathbf{H}$ is a parity-check matrix of the (n,k) cyclic code $\mathcal{C}$ corresponding to $\mathbf{G}$ given by (3.27). The polynomial $\mathbf{h}(X)$ given by (3.29) is called the *parity-check polynomial* of $\mathcal{C}$. In fact, $\mathbf{h}(X)$ is the generator polynomial of the dual code, an $(n,n-k)$ cyclic code, of the (n,k) cyclic code $\mathcal{C}$.

Systematic encoding of an (n,k) cyclic code with generator polynomial $\mathbf{g}(X)$ can be accomplished easily. Suppose $\mathbf{m} = (m_0, m_1, \ldots, m_{k-1})$ is the message to be encoded. On multiplying the message polynomial $\mathbf{m}(X) = m_0 + m_1 X + \cdots + m_{k-1}X^{k-1}$ by X^{n-k}, we obtain

$$X^{n-k}\mathbf{m}(X) = m_0 X^{n-k} + m_1 X^{n-k+1} + \cdots + m_{k-1}X^{n-1}, \tag{3.31}$$

which is a polynomial of degree $n-1$ or less. On dividing $X^{n-k}\mathbf{m}(X)$ by the generator polynomial $\mathbf{g}(X)$, we have

$$X^{n-k}\mathbf{m}(X) = \mathbf{a}(X)\mathbf{g}(X) + \mathbf{b}(X), \tag{3.32}$$

where $\mathbf{a}(X)$ and $\mathbf{b}(X)$ are the quotient and remainder, respectively. Since the degree of $\mathbf{g}(X)$ is $n-k$, the degree of the remainder $\mathbf{b}(X)$ must be $n-k-1$ or less. Then $\mathbf{b}(X)$ must be of the following form:

$$\mathbf{b}(X) = b_0 + b_1 X + \cdots + b_{n-k-1}X^{n-k-1}. \tag{3.33}$$

We rearrange the expression of (3.32) as follows:

$$\mathbf{b}(X) + X^{n-k}\mathbf{m}(X) = \mathbf{a}(X)\mathbf{g}(X). \tag{3.34}$$

Expression (3.34) shows that $\mathbf{b}(X) + X^{n-k}\mathbf{m}(X)$ is divisible by $\mathbf{g}(X)$. Since the degree of $\mathbf{b}(X) + X^{n-k}\mathbf{m}(X)$ is $n-1$ or less, $\mathbf{b}(X) + X^{n-k}\mathbf{m}(X)$ is hence a code polynomial of the (n,k) cyclic code $\mathcal{C}$ with $\mathbf{g}(X)$ as its generator polynomial. The n-tuple representation of the code polynomial $\mathbf{b}(X) + X^{n-k}\mathbf{m}(X)$ is

$$(b_0, b_1, \ldots, b_{n-k-1}, m_0, m_1, \ldots, m_{k-1}),$$

which is in systematic form, where the $n-k$ parity-check bits, $b_0, b_1, \ldots, b_{n-k-1}$ are simply the coefficients of the remainder $\mathbf{b}(X)$ given by (3.33).

For $0 \le i < k$, let $\mathbf{m}_i(X) = X^i$ be the message polynomial with a single nonzero information bit at the ith position of the message $\mathbf{m}_i$ to be encoded. Dividing $X^{n-k}\mathbf{m}_i(X) = X^{n-k+i}$ by $\mathbf{g}(X)$, we obtain

$$X^{n-k+i} = \mathbf{a}_i(X)\mathbf{g}(X) + \mathbf{b}_i(X), \tag{3.35}$$

where the remainder $\mathbf{b}_i(X)$ is of the following form:

$$\mathbf{b}_i(X) = b_{i,0} + b_{i,1}X + \cdots + b_{i,n-k-1}X^{n-k-1}. \tag{3.36}$$

Since $\mathbf{b}_i + X^{n-k+i}$ is divisible by $\mathbf{g}(X)$, it is a code polynomial in $\mathcal{C}$. On arranging the n-tuple representations of the $n-k$ code polynomials, $\mathbf{b}_i + X^{n-k+i}$ for $0 \le i < k$, as the rows of a $k \times n$ matrix over GF(2), we obtain

$$\mathbf{G}_{\text{c,sys}} = \begin{bmatrix} b_{0,0} & b_{0,1} & \cdots & b_{0,n-k-1} & 1 & 0 & 0 & \cdots & 0 \\ b_{1,0} & b_{1,1} & \cdots & b_{1,n-k-1} & 0 & 1 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ b_{k-1,0} & b_{k-1,1} & \cdots & b_{k-1,n-k-1} & 0 & 0 & 0 & \cdots & 1 \end{bmatrix}, \tag{3.37}$$

which is the generator matrix of the (n,k) cyclic code $\mathcal{C}$ in systematic form. The corresponding parity-check matrix of $\mathcal{C}$ in systematic form is

$$\mathbf{H}_{\text{c,sys}} = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 & b_{0,0} & b_{1,0} & \cdots & b_{k-1,0} \\ 0 & 1 & 0 & \cdots & 0 & b_{0,1} & b_{1,1} & \cdots & b_{k-1,1} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & b_{0,n-k-1} & b_{1,n-k-1} & \cdots & b_{k-1,n-k-1} \end{bmatrix}. \tag{3.38}$$

From (3.32) and (3.34), we see that encoding of an (n,k) cyclic code $\mathcal{C}$ can be achieved with a division circuit that divides the message polynomial $X^{n-k}\mathbf{m}(X)$ by its generator polynomial $\mathbf{g}(X)$ and takes the remainder as the parity part of the codeword. This division circuit can be implemented with an $(n-k)$-stage shift-register with feedback connections based on the coefficients of the generator polynomial as shown in Figure 3.3. The pre-multiplication of the message polynomial $\mathbf{m}(X)$ by X^{n-k} is accomplished by shifting the message polynomial from the right-hand end of the encoding feedback shift-register as shown in Figure 3.3.

Suppose an (n,k) cyclic code $\mathcal{C}$ with a generator polynomial $\mathbf{g}(X)$ is used for error control over a noisy channel. Let $\mathbf{r}(X) = r_0 + r_1X + \cdots + r_{n-1}X^{n-1}$ be the

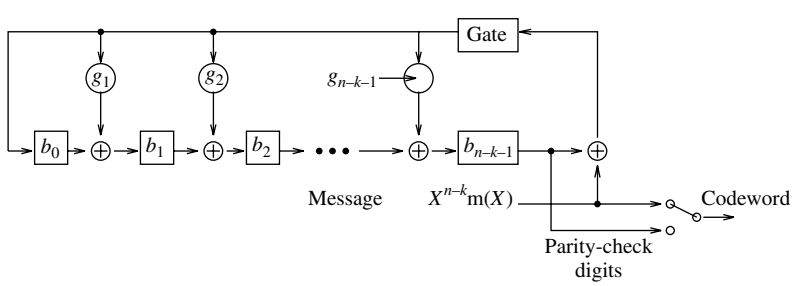


Figure 3.3 An encoding circuit for an (n,k) cyclic code with generator polynomial $\mathbf{g}(X) = 1 + g_1X + \cdots + g_{n-k-1}X^{n-k-1} + X^{n-k}$.

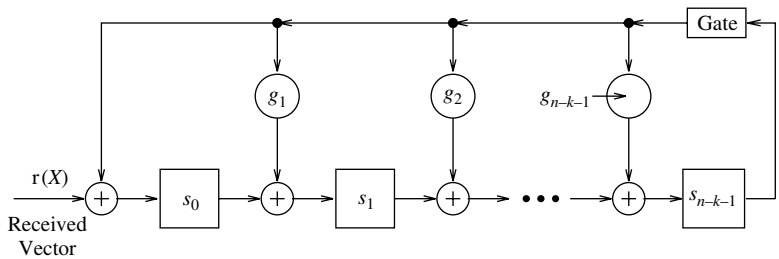


Figure 3.4 A syndrome-computation circuit for an (n,k) cyclic code with generator polynomial $\mathbf{g}(X) = 1 + g_1 X + \cdots + g_{n-k-1} X^{n-k-1} + X^{n-k}$.

received polynomial. Then, $\mathbf{r}(X)$ is the sum of a transmitted polynomial $\mathbf{v}(X)$ and an error polynomial $\mathbf{e}(X)$, i.e., $\mathbf{r}(X) = \mathbf{v}(X) + \mathbf{e}(X)$. The first step in decoding $\mathbf{r}(X)$ is to compute its syndrome. The syndrome of $\mathbf{r}(X)$, denoted $\mathbf{s}(X)$, is given by the remainder obtained from dividing $\mathbf{r}(X)$ by the generator polynomial $\mathbf{g}(X)$ of code $\mathcal{C}$. If $\mathbf{s}(X) = 0$, then $\mathbf{r}(X)$ is a code polynomial and is accepted by the receiver as the transmitted code polynomial. If $\mathbf{s}(X) \neq 0$, then $\mathbf{r}(X)$ is not a code polynomial and the presence of transmission errors is detected. Computation of the syndrome $\mathbf{s}(X)$ of $\mathbf{r}(X)$ again can be accomplished with a division circuit as shown in Figure 3.4.

Similarly to the decoding of a linear block code described in Section 3.1.4, decoding of a cyclic code involves associating the computed syndrome $\mathbf{s}(X)$ of a received polynomial $\mathbf{r}(X)$ with a specific correctable error pattern $\mathbf{e}(X)$. Then the estimated transmitted code polynomial $\hat{\mathbf{v}}(X)$ is obtained by removing the estimated error pattern $\mathbf{e}(X)$ from the received polynomial $\mathbf{r}(X)$, i.e., $\hat{\mathbf{v}}(X) = \mathbf{r}(X) - \mathbf{e}(X)$. For bounded distance decoding, the architecture and complexity of the decoding circuit very much depend on the decoding algorithm devised for a specific class of codes. For more on decoding of cyclic codes, readers are referred to references [1,5,11].

Consider a systematic (n,k) cyclic code $\mathcal{C}$. Let l be a non-negative integer less than k. Consider the set of code polynomials whose l leading high-order information digits, $v_{n-l}, \ldots, v_{n-2}, v_{n-1}$, are zeros. There are 2^{k-l} such code polynomials. If the l zero information digits are deleted from each of these code polynomials, we obtain a set of 2^{k-l} polynomials over GF(2) with degree $n-l-1$ or less. These 2^{k-l} shortened polynomials form an $(n-l,k-l)$ linear block code. This code is called a *shortened cyclic code* (or *polynomial code*) and it is not cyclic. A shortened cyclic code has at least the same error-correction capability as the code from which it is shortened. The encoding and syndrome computation for a shortened cyclic code can be accomplished by the same circuits as employed by the original cyclic code. This is so because the deleted l leading zero information digits do not affect the parity-check and syndrome computation. The decoding circuit for the original cyclic code can be used for decoding the shortened code simply by prefixing each received vector with l zeros. This prefixing can be eliminated by modifying the decoding circuit.

So far, we have discussed only the structure, encoding, and decoding of cyclic codes, saying nothing about the existence of cyclic codes. Recall that the generator

polynomial $\mathbf{g}(X)$ of an (n,k) cyclic code divides $X^n + 1$. With this fact, it can be proved that any factor $\mathbf{g}(X)$ over GF(2) of $X^n + 1$ with degree $n - k$ can be used as a generator polynomial of an (n,k) cyclic code. In general, any factor over GF(2) of $X^n + 1$ generates a cyclic code.

3.3 BCH Codes

BCH (Bose–Chaudhuri–Hocquenghem) codes form a large class of cyclic codes for correcting multiple random errors. This class of codes was first discovered by Hocquenghem in 1959 [16] and then independently by Bose and Chaudhuri in 1960 [17]. The first algorithm for decoding of binary BCH code was devised by Peterson [10]. Since then, the Peterson decoding algorithm has been improved and generalized by others [1, 18]. In this section, we introduce a subclass of binary BCH codes that is the most important subclass from the standpoint of both theory and implementation.

3.3.1 Code Construction

BCH codes are specified in terms of the roots of their generator polynomials in finite fields. For any positive integers m and t with $m \geq 3$ and $t < 2^{m-1}$, there exists a binary BCH code of length $2^m - 1$ with minimum distance at least $2t + 1$ and at most mt parity-check digits. This BCH code is capable of correcting t or fewer random errors over a span of $2^m - 1$ transmitted code bits. It is called a *t-error-correcting* BCH code.

Construction of a t-error-correcting BCH code begins with a Galois field GF(2^m). Let α be a primitive element in GF(2^m). The generator polynomial $\mathbf{g}(X)$ of the t-error-correcting binary BCH code of length $2^m - 1$ is the *smallest-degree* (or *minimum-degree*) polynomial over GF(2) that has the following $2t$ consecutive powers of α:

$$\alpha, \alpha^2, \alpha^3, \ldots, \alpha^{2t}, \tag{3.39}$$

as *roots*. It follows from Theorem 2.11 that $\mathbf{g}(X)$ has $\alpha, \alpha^2, \ldots, \alpha^{2t}$ and their conjugates as all of its roots. Since all the nonzero elements of GF(2^m) are roots of $X^{2^m-1} + 1$ and all the roots of $\mathbf{g}(X)$ are elements in GF(2^m), $\mathbf{g}(X)$ divides $X^{2^m-1} + 1$. Since $\mathbf{g}(X)$ is a factor of $X^{2^m-1} + 1$, it can be used to generate a cyclic code of length $2^m - 1$.

For $1 \leq i \leq 2t$, let $\phi_i(X)$ be the minimal polynomial of α^i. Then the generator polynomial of the t-error-correcting binary BCH code of length $2^m - 1$ is given by the *least common multiple* (*LCM*) of $\phi_1(X), \phi_2(X), \ldots, \phi_{2t}(X)$, i.e.,

$$\mathbf{g}(X) = \text{LCM}\{\phi_1(X), \phi_2(X), \ldots, \phi_{2t}(X)\}. \tag{3.40}$$

If i is an even integer, it can be expressed as a product of an odd integer i' and a power of 2 as follows:

$$i = i'2^l.$$

Then α^i is a conjugate of $\alpha^{i'}$, and hence α^i and $\alpha^{i'}$ have the same minimal polynomial, i.e.,

$$\phi_i(X) = \phi_{i'}(X).$$

Therefore, every even power of α in the sequence of (3.39) has the same minimal polynomial as some previous odd power of α in the sequence. As a result, the generator polynomial of the t-error-correcting binary BCH code of length $2^m - 1$ given by (3.40) can be simplified as follows:

$$\mathbf{g}(X) = \text{LCM}\{\phi_1(X), \phi_3(X), \ldots, \phi_{2t-1}(X)\}. \tag{3.41}$$

There are only t minimal polynomials in the LCM expression of (3.41). Since the degree of a minimal polynomial of an element in $\text{GF}(2^m)$ is *at most* m, the degree of $\mathbf{g}(X)$ is then at most mt. Therefore, the t-error-correcting binary BCH code generated by $\mathbf{g}(X)$ has at most mt parity-check bits and hence its dimension is at least $2^m - mt - 1$. For a given field $\text{GF}(2^m)$ with $m \geq 3$, the above construction gives a family of binary BCH codes of length $2^m - 1$ with various rates and error-correction capabilities. The BCH codes defined above are called *primitive* (or *narrow-sense*) BCH codes since they are defined by consecutive powers of a primitive element α in $\text{GF}(2^m)$.

Consider a primitive t-error-correcting BCH code $\mathcal{C}$ constructed using $\text{GF}(2^m)$, whose generator polynomial $\mathbf{g}(X)$ has $\alpha, \alpha^2, \ldots, \alpha^{2t}$ and their conjugates as roots. Then, for $1 \leq i \leq 2t$, $\mathbf{g}(\alpha^i) = 0$. Let $\mathbf{v}(X) = v_0 + v_1 X + \cdots + v_{2^m-2} X^{2^m-2}$ be a code polynomial in $\mathcal{C}$. Since $\mathbf{v}(X)$ is divisible by $\mathbf{g}(X)$, a root of $\mathbf{g}(X)$ is also a root of $\mathbf{v}(X)$. Hence, for $1 \leq i \leq 2t$,

$$\mathbf{v}(\alpha^i) = v_0 + v_1\alpha^i + v_2\alpha^{2i} + \cdots + v_{2^m-2}\alpha^{(2^m-2)i} = 0. \tag{3.42}$$

The above equality can be written as the following matrix product:

$$(v_0, v_1, \ldots, v_{2^m-2}) \cdot \begin{pmatrix} 1 \\ \alpha^i \\ \alpha^{2i} \\ \vdots \\ \alpha^{(2^m-2)i} \end{pmatrix} = 0. \tag{3.43}$$

Equation (3.43) simply says that the inner product of the codeword $\mathbf{v} = (v_0, v_1, \ldots, v_{2^m-1})$ and $(1, \alpha^i, \ldots, a^{(2^m-2)i})$ over $\text{GF}(2^m)$ is equal to zero. Form the following $2t \times (2^m - 1)$ matrix over $\text{GF}(2^m)$:

$$\mathbf{H} = \begin{bmatrix} 1 & \alpha & \alpha^2 & \cdots & \alpha^{2^m-2} \\ 1 & \alpha^2 & (\alpha^2)^2 & \cdots & (\alpha^2)^{2^m-2} \\ 1 & \alpha^3 & (\alpha^3)^2 & \cdots & (\alpha^3)^{2^m-2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha^{2t} & (\alpha^{2t})^2 & \cdots & (\alpha^{2t})^{2^m-2} \end{bmatrix}. \tag{3.44}$$

It follows from (3.43) that, for every codeword $\mathbf{v} = (v_0, v_1, \ldots, v_{2^m-2})$ in the t-error-correcting BCH code $\mathcal{C}$ of length $2^m - 1$ generated by $\mathbf{g}(X)$, the following

condition holds:

$$\mathbf{v} \cdot \mathbf{H}^{\mathrm{T}} = \mathbf{0}. \tag{3.45}$$

On the other hand, if a $(2^m - 1)$-tuple $\mathbf{v} = (v_0, v_1, \ldots, v_{2^m-1})$ over GF(2) satisfies the condition given by (3.45), then it follows from (3.43) and (3.42) that its corresponding polynomial $\mathbf{v}(X) = v_0 + v_1 X + \cdots + v_{2^m-2} X^{2^m-2}$ has $\alpha, \alpha^2, \ldots, \alpha^{2t}$ and their conjugates as roots. As a result, $\mathbf{v}(X)$ is divisible by the generator polynomial $\mathbf{g}(X)$ of the t-error-correcting BCH code of length $2^m - 1$, and hence is a code polynomial. Therefore, the t-error-correcting primitive BCH code of length $2^m - 1$ generated by the generator polynomial $\mathbf{g}(X)$ given by (3.41) is the null space over GF(2) of $\mathbf{H}$; and $\mathbf{H}$ is a parity-check matrix of the code.

For $d \leq 2t$, any $d \times d$ submatrix of the parity-check matrix $\mathbf{H}$ is a *Vandermonde matrix* and its determinant is nonzero. This implies that any d columns of $\mathbf{H}$ are linearly independent and hence cannot be summed to a zero column vector. It follows from Theorem 3.3 that the BCH code given by the null space of $\mathbf{H}$ has minimum distance at least $2t + 1$. Hence, the BCH code can correct any error pattern with t or fewer random errors during the transmission of a codeword over the BSC. The parameter $2t + 1$ is called the *designed minimum distance* of the t-error-correcting BCH code. The true minimum distance of the code may be greater than its designed minimum distance $2t + 1$. So, $2t + 1$ is a lower bound on the minimum distance of the BCH code. The proof of this bound is based on the fact that the generator polynomial of the code has $2t$ consecutive powers of a primitive element α in GF(2^m) as roots. This bound is referred to as the *BCH bound*.

Example 3.1. Let GF(2^6) be the code construction field which is generated by the primitive polynomial $p(X) = 1 + X + X^6$. The elements of this field in three forms are given in Table 3.3. Let α be a primitive element of GF(2^6). Set $t = 3$. Then the generator polynomial $\mathbf{g}(X)$ of the triple-error-correcting primitive BCH code of length 63 has

$$\alpha, \alpha^2, \alpha^3, \alpha^4, \alpha^5, \alpha^6,$$

and their conjugates as roots. Note that α, α^2, and α^4 are conjugate elements and have the same minimal polynomial, which is $1 + X + X^6$. The elements α^3 and α^6 are conjugates and their minimal polynomial is $1 + X + X^2 + X^4 + X^6$. The minimal polynomial of α^5 is $1 + X + X^2 + X^5 + X^6$. It follows from (3.41) that the generator polynomial of the triple-error-correction primitive BCH code of length 63 is given by

$$\begin{aligned}\mathbf{g}(X) &= (1 + X + X^6)(1 + X + X^2 + X^4 + X^6)(1 + X + X^2 + X^5 + X^6) \\ &= 1 + X + X^2 + X^3 + X^6 + X^7 + X^9 + X^{15} + X^{16} + X^{17} + X^{18}.\end{aligned}$$

Since the degree of $\mathbf{g}(X)$ is 18, the BCH code generated by $\mathbf{g}(X)$ is a (63,45) code with designed minimum distance 7. However, it can be proved that the true minimum distance is also 7.

Table 3.3. GF(2^6) generated by the primitive polynomial $p(X) = 1 + X + X^6$ over GF(2)

Power representation	Vector representation	Power representation	Vector representation
0	(0 0 0 0 0 0)	α^{32}	(1 0 0 1 0 0)
1	(1 0 0 0 0 0)	α^{33}	(0 1 0 0 1 0)
α	(0 1 0 0 0 0)	α^{34}	(0 0 1 0 0 1)
α^2	(0 0 1 0 0 0)	α^{35}	(1 1 0 1 0 0)
α^3	(0 0 0 1 0 0)	α^{36}	(0 1 1 0 1 0)
α^4	(0 0 0 0 1 0)	α^{37}	(0 0 1 1 0 1)
α^5	(0 0 0 0 0 1)	α^{38}	(1 1 0 1 1 0)
α^6	(1 1 0 0 0 0)	α^{39}	(0 1 1 0 1 1)
α^7	(0 1 1 0 0 0)	α^{40}	(1 1 1 1 0 1)
α^8	(0 0 1 1 0 0)	α^{41}	(1 0 1 1 1 0)
α^9	(0 0 0 1 1 0)	α^{42}	(0 1 0 1 1 1)
α^{10}	(0 0 0 0 1 1)	α^{43}	(1 1 1 0 1 1)
α^{11}	(1 1 0 0 0 1)	α^{44}	(1 0 1 1 0 1)
α^{12}	(1 0 1 0 0 0)	α^{45}	(1 0 0 1 1 0)
α^{13}	(0 1 0 1 0 0)	α^{46}	(0 1 0 0 1 1)
α^{14}	(0 0 1 0 1 0)	α^{47}	(1 1 1 0 0 1)
α^{15}	(0 0 0 1 0 1)	α^{48}	(1 0 1 1 0 0)
α^{16}	(1 1 0 0 1 0)	α^{49}	(0 1 0 1 1 0)
α^{17}	(0 1 1 0 0 1)	α^{50}	(0 0 1 0 1 1)
α^{18}	(1 1 1 1 0 0)	α^{51}	(1 1 0 1 0 1)
α^{19}	(0 1 1 1 1 0)	α^{52}	(1 0 1 0 1 0)
α^{20}	(0 0 1 1 1 1)	α^{53}	(0 1 0 1 0 1)
α^{21}	(1 1 0 1 1 1)	α^{54}	(1 1 1 0 1 0)
α^{22}	(1 0 1 0 1 1)	α^{55}	(0 1 1 1 0 1)
α^{23}	(1 0 0 1 0 1)	α^{56}	(1 1 1 1 1 0)
α^{24}	(1 0 0 0 1 0)	α^{57}	(0 1 1 1 1 1)
α^{25}	(0 1 0 0 0 1)	α^{58}	(1 1 1 1 1 1)
α^{26}	(1 1 1 0 0 0)	α^{59}	(1 0 1 1 1 1)
α^{27}	(0 1 1 1 0 0)	α^{60}	(1 0 0 1 1 1)
α^{28}	(0 0 1 1 1 0)	α^{61}	(1 0 0 0 1 1)
α^{29}	(0 0 0 1 1 1)	α^{62}	(1 0 0 0 0 1)
α^{30}	(1 1 0 0 1 1)	$\alpha^{63} = 1$	
α^{31}	(1 0 1 0 0 1)		

3.3.2 Decoding

Suppose a code polynomial $\mathbf{v}(X)$ of a t-error-correcting primitive BCH code is transmitted over the BSC channel and $\mathbf{r}(X)$ is the corresponding received

polynomial. The syndrome of $\mathbf{r}(X)$ is given by

$$\mathbf{S} = (S_1, S_2, \ldots, S_{2t}) = \mathbf{r} \cdot \mathbf{H}^{\mathrm{T}}, \tag{3.46}$$

which consists of $2t$ syndrome components. It follows from (3.44), (3.45), and (3.46) that, for $1 \le i \le 2t$, the ith syndrome component is given by

$$S_i = \mathbf{r}(\alpha^i) = r_0 + r_1\alpha^i + \cdots + r_{2^m-2}\alpha^{(2^m-2)i}, \tag{3.47}$$

which is an element in $\mathrm{GF}(2^m)$. An effective algebraic hard-decision decoding algorithm, called the *Berlekamp–Massey* algorithm [1, 18], has been devised assuming as input the syndrome $\mathbf{S}$ (also see [2, 5, 11] for details).

Let $\mathbf{e}(X) = e_0 + e_1X + \cdots + e_{2^m-2}X^{2^m-2}$ be the error pattern induced by the channel noise during the transmission of the code polynomial $\mathbf{v}(X)$. Then the received polynomial $\mathbf{r}(X) = \mathbf{v}(X) + \mathbf{e}(X)$. It follows from (3.47) that, for $1 \le i \le 2t$,

$$S_i = \mathbf{r}(\alpha^i) = \mathbf{v}(\alpha^i) + \mathbf{e}(\alpha^i). \tag{3.48}$$

Since $\mathbf{v}(\alpha^i) = 0$, we have

$$S_i = \mathbf{e}(\alpha^i), \tag{3.49}$$

for $1 \le i \le 2t$. Suppose $\mathbf{e}(X)$ contains ν errors at the locations $j_1, j_2, \ldots, j_\nu$, where $0 \le j_1 < j_2 < \cdots < j_\nu < n$. Then

$$\mathbf{e}(X) = X^{j_1} + X^{j_2} + \cdots + X^{j_\nu}. \tag{3.50}$$

From (3.49) and (3.50), we obtain the following $2t$ equalities that relate the error locations, $j_1, j_2, \ldots, j_\nu$, of $\mathbf{e}(X)$ and the $2t$ syndrome components, $S_1, S_2, \ldots, S_{2t}$:

$$\begin{aligned} S_1 &= \mathbf{e}(\alpha) = \alpha^{j_1} + \alpha^{j_2} + \cdots + \alpha^{j_\nu}, \\ S_2 &= \mathbf{e}(\alpha^2) = (\alpha^{j_1})^2 + (\alpha^{j_2})^2 + \cdots + (\alpha^{j_\nu})^2, \\ &\vdots \\ S_{2t} &= \mathbf{e}(\alpha^{2t}) = (\alpha^{j_1})^{2t} + (\alpha^{j_2})^{2t} + \cdots + (\alpha^{j_\nu})^{2t}. \end{aligned} \tag{3.51}$$

Note that the $2t$ syndrome components $S_1, S_2, \ldots, S_{2t}$ are known but the error locations $j_1, j_2, \ldots, j_\nu$ are unknown. If we can solve the above $2t$ equations, we can determine $\alpha^{j_1}, \alpha^{j_2}, \ldots, \alpha^{j_\nu}$, whose exponents then give the locations of errors in the error pattern $\mathbf{e}(X)$. Since the elements $\alpha^{j_1}, \alpha^{j_2}, \ldots, \alpha^{j_\nu}$ in $\mathrm{GF}(2^m)$ give the locations of the errors in $\mathbf{e}(X)$, they are referred to as the *error-location numbers*.

For $1 \le l \le \nu$, define

$$\beta_l = \alpha^{j_l}. \tag{3.52}$$

Then the $2t$ equations given by (3.51) can be expressed in a simpler form as follows:

$$\begin{aligned} S_1 &= \beta_1 + \beta_2 + \cdots + \beta_\nu, \\ S_2 &= \beta_1^2 + \beta_2^2 + \cdots + \beta_\nu^2, \\ &\vdots \\ S_{2t} &= \beta_1^{2t} + \beta_2^{2t} + \cdots + \beta_\nu^{2t}. \end{aligned} \tag{3.53}$$

The $2t$ equations given by (3.53) are called the *power-sum symmetric functions*. Since they are nonlinear equations, solving them directly would be very difficult, if not impossible. In the following, an indirect method for solving these equations is provided.

Define the following polynomial of degree ν over GF(2^m):

$$\begin{aligned}\sigma(X) &= (1+\beta_1 X)(1+\beta_2 X)\cdots(1+\beta_\nu X)\\ &= \sigma_0 + \sigma_1 X + \cdots + \sigma_\nu X^\nu,\end{aligned} \tag{3.54}$$

where $\sigma_0 = 1$ and

$$\begin{aligned}&\sigma_1 = \beta_1 + \beta_2 + \cdots + \beta_\nu,\\ &\sigma_2 = \beta_1\beta_2 + \beta_1\beta_3 + \cdots + \beta_{\nu-1}\beta_\nu,\\ &\sigma_3 = \beta_1\beta_2\beta_3 + \beta_1\beta_2\beta_4 + \cdots + \beta_{\nu-1}\beta_{\nu-2}\beta_{\nu-3},\\ &\vdots\\ &\sigma_\nu = \beta_1\beta_2\cdots\beta_\nu.\end{aligned} \tag{3.55}$$

The polynomial $\sigma(X)$ has $\beta_1^{-1}, \beta_2^{-1}, \ldots, \beta_\nu^{-1}$ (the *inverses* of the location numbers $\beta_1, \beta_2, \ldots, \beta_\nu$) as roots. This polynomial is called the *error-location polynomial*. If this polynomial can be found, then the inverses of its roots give the error-location numbers. Using (3.52), we can determine the error locations of the error pattern $\mathbf{e}(X)$. The ν equalities given by (3.55) are called the *elementary-symmetric functions* which relate the coefficients of the error-location polynomial $\sigma(X)$ to the error-location numbers, $\beta_1, \beta_2, \ldots, \beta_\nu$.

From (3.53) and (3.55), we see that the syndrome components $S_1, S_2, \ldots, S_{2t}$ and the coefficients of the error-location polynomial $\sigma(X)$ are related through the error-location numbers $\beta_1, \beta_2, \ldots, \beta_v$. From these two sets of equations, it is possible to derive the following equations that relate the coefficients of the error-location polynomial $\sigma(X)$ to the syndrome components computed from the received polynomial $\mathbf{r}(X)$ [1, 10, 18]:

$$\begin{aligned}&S_1 + \sigma_1 = 0,\\ &S_2 + \sigma_1 S_1 + 2\sigma_2 = 0,\\ &S_3 + \sigma_1 S_2 + \sigma_2 S_1 + 3\sigma_3 = 0,\\ &\vdots\\ &S_\nu + \sigma_1 S_{\nu-1} + \sigma_2 S_{\nu-2} + \cdots + \sigma_{\nu-1} S_1 + \nu\sigma_\nu = 0,\\ &S_{\nu+1} + \sigma_1 S_\nu + \sigma_2 S_{\nu-1} + \cdots + \sigma_{\nu-1} S_2 + \sigma_\nu S_1 = 0,\\ &\vdots\end{aligned} \tag{3.56}$$

Since $1 + 1 = 0$ under modulo-2 addition, we have $i\sigma_i = \sigma_i$ for odd i and $i\sigma_i = 0$ for even i. The identities given by (3.56) are called the *Newton identities*.

If we can determine the coefficients, $\sigma_1, \sigma_2, \ldots, \sigma_\nu$, of the error-location polynomial $\sigma(X)$ from the Newton identities, then we can determine $\sigma(X)$. Once $\sigma(X)$ has been determined, we find its roots. By taking the inverses of the roots of $\sigma(X)$, we obtain the error-location numbers, $\beta_1, \beta_2, \ldots, \beta_\nu$, of the error pattern $\mathbf{e}(X)$. From (3.52) and (3.50), we can determine the error pattern $\mathbf{e}(X)$. On removing $\mathbf{e}(X)$ from the received polynomial $\mathbf{r}(X)$ (i.e., adding $\mathbf{e}(X)$ to $\mathbf{r}(X)$ using

modulo-2 addition of their corresponding coefficients), we obtain the decoded codeword $\mathbf{v}(X)$. From the description given above, a procedure for decoding a BCH code is as follows.

1. Compute the syndrome $\mathbf{S} = (S_1, S_2, \ldots, S_{2t})$ of the received polynomial $\mathbf{r}(X)$.
2. Determine the error-location polynomial $\sigma(X)$ from the Newton identities.
3. Determine the roots, $\beta_1^{-1}, \beta_2^{-1}, \ldots, \beta_\nu^{-1}$, of $\sigma(X)$ in GF(2^m). Take the inverses of these roots to obtain the error-location numbers, $\beta_1 = \alpha^{j_1}$, $\beta_2 = \alpha^{j_2}, \ldots,$ $\beta_\nu = \alpha^{j_\nu}$. Form the error pattern
$$\mathbf{e}(X) = X^{j_1} + X^{j_2} + \cdots + X^{j_\nu}.$$
4. Perform the error correction by adding $\mathbf{e}(X)$ to $\mathbf{r}(X)$. This gives the decoded code polynomial $\mathbf{v}(X) = \mathbf{r}(X) + \mathbf{e}(X)$.

Steps 1, 3, and 4 can be carried out easily. However, Step 2 involves solving the Newton identities. There will be in general more than one error pattern for which the coefficients of its error-location polynomial satisfy the Newton identities. To minimize the probability of a decoding error, we need to find the *most probable* error pattern for error correction. For the BSC, finding the most probable error pattern means determining the error-location polynomial of minimum degree whose coefficients satisfy the Newton identities given by (3.56). This can be achieved iteratively by using an algorithm first devised by Berlekamp [1] and then improved by Massey [18]. Such an algorithm is commonly referred to as the Berlekamp–Massey (BM) algorithm.

The BM algorithm is used to find the error-location polynomial $\sigma(X)$ iteratively in $2t$ steps. For $1 \le k \le 2t$, the algorithm at the kth step gives an error-location polynomial of minimum degree,
$$\sigma^{(k)}(X) = \sigma_0^{(k)} + \sigma_1^{(k)} X + \cdots + \sigma_{l_k}^{(k)} X^{l_k}, \tag{3.57}$$
whose coefficients satisfy the *first* k Newton identities. At the $(k+1)$th step, the algorithm is used to find the next minimum-degree error-location polynomial $\sigma^{(k+1)}(X)$ whose coefficients satisfy the first $k+1$ Newton identities based on $\sigma^{(k)}(X)$. First, we check whether the coefficients of $\sigma^{(k)}(X)$ also satisfy the $(k+1)$th Newton identity. If yes, then
$$\sigma^{(k+1)}(X) = \sigma^{(k)}(X) \tag{3.58}$$
is the minimum-degree error-location polynomial whose coefficients satisfy the first $k+1$ Newton identities. If no, a correction term is added to $\sigma^{(k)}(X)$ to form a minimum-degree error-location polynomial,
$$\sigma^{(k+1)}(X) = \sigma_0^{(k+1)} + \sigma_1^{(k+1)} X + \cdots + \sigma_{l_{k+1}}^{(k+1)} X^{l_{k+1}}, \tag{3.59}$$
whose coefficients satisfy the first $k+1$ Newton identities. To test whether the coefficients of $\sigma^{(k)}(X)$ satisfy the $(k+1)$th Newton identity, we compute
$$d_k = S_{k+1} + \sigma_1^{(k)} S_k + \sigma_2^{(k)} S_{k-1} + \cdots + \sigma_{l_k}^{(k)} S_{k+1-l_k}. \tag{3.60}$$

Note that the sum on the right-hand side of the equality (3.60) is actually equal to the left-hand side of the $(k+1)$th Newton identity given by (3.56). If $d_k = 0$, then the coefficients of $\sigma^{(k)}(X)$ satisfy the $(k+1)$th Newton identity. If $d_k \neq 0$, then the coefficients of $\sigma^{(k)}(X)$ do not satisfy the $(k+1)$th Newton identity. In this case, $\sigma^{(k)}(X)$ needs to be adjusted to obtain a new minimum-degree error-location polynomial $\sigma^{(k+1)}(X)$ whose coefficients satisfy the first $k+1$ Newton identities. The quantity d_k is called the kth *discrepancy*.

If $d_k \neq 0$, a *correction term* is added to $\sigma^{(k)}(X)$ to obtain the next minimum-degree error-location polynomial $\sigma^{(k+1)}(X)$. First, we go back to the steps prior to the kth step and determine a step i at which the minimum-degree error-location polynomial is $\sigma^{(i)}(X)$ such that the ith discrepancy $d_i \neq 0$ and $i - l_i$ has the largest value, where l_i is the degree of $\sigma^{(i)}(X)$. Then

$$\sigma^{(k+1)}(X) = \sigma^{(k)}(X) + d_k d_i^{-1} X^{k-i} \sigma^{(i)}(X), \tag{3.61}$$

where

$$d_k d_i^{-1} X^{k-i} \sigma^{(i)}(X) \tag{3.62}$$

is the correction term. Since i is chosen to maximize $i - l_i$, this choice of i is equivalent to minimizing the degree $k - (i - l_i)$ of the correction term. We repeat the above testing and correction process until we reach the $2t$th step. Then

$$\sigma(X) = \sigma^{(2t)}(X). \tag{3.63}$$

To begin using the above BM iterative algorithm to find $\sigma(X)$, two sets of initial conditions are needed: (1) for $k = -1$, set $\sigma^{(-1)}(X) = 1$, $d_{-1} = 1$, $l_{-1} = 0$, and $-1 - l_{-1} = -1$; and (2) for $k = 0$, set $\sigma^{(0)}(X) = 1$, $d_0 = S_1$, $l_0 = 0$, and $0 - l_0 = 0$. Then the BM algorithm for finding the error-location polynomial $\sigma(X)$ can be formulated as follows.

Algorithm 3.1 The Berlekamp–Massey Algorithm for Finding the Error-Location Polynomial of a BCH Code

---

Initialization. For $k = -1$, set $\sigma^{(-1)}(X) = 1$, $d_{-1} = 1$, $l_{-1} = 0$, and $-1 - l_{-1} = -1$. For $k = 0$, set $\sigma^{(0)}(X) = 1$, $d_0 = S_1$, $l_0 = 0$ and $0 - l_0 = 0$.

1. If $k = 2t$, output $\sigma^{(k)}(X)$ as the error-location polynomial $\sigma(X)$; otherwise go to Step 2.
2. Compute d_k and go to Step 3.
3. If $d_k = 0$, set $\sigma^{(k+1)}(X) = \sigma^{(k)}(X)$; otherwise, set $\sigma^{(k+1)}(X) = \sigma^{(k)}(X) + d_k d_i^{-1} X^{k-i} \sigma^{(i)}(X)$. Go to Step 4.
4. $k \leftarrow k+1$. Go to Step 1.

If the number ν of errors induced by the channel noise during the transmission of a codeword in a t-error-correcting primitive BCH is t or fewer (i.e., $\nu \leq t$), the BM algorithm guarantees to produce a *unique* error-location polynomial $\sigma(X)$ of degree ν that has ν roots in GF(2^m). The inverses of the roots of $\sigma(X)$ give the

correct locations of the errors in the received sequence $\mathbf{r}(X)$. Flipping the received bits at these locations will result in a correct decoding.

The BM algorithm can be executed by setting up and filling in the following table:

Step k	Partial solution $\sigma^{(k)}(X)$	Discrepancy d_k	Degree l_k	Step/degree difference $k - l_k$
-1	1	1	0	-1
0	1	S_1	0	0
1	$\sigma^{(1)}(X)$	d_1	l_1	$1 - l_1$
2	$\sigma^{(2)}(X)$	d_2	l_2	$2 - l_2$
$\vdots$				
$2t$	$\sigma^{(2t)}(X)$	—	—	—

Example 3.2. Let $\mathrm{GF}(2^4)$ be a field constructed using the primitive polynomial $p(X) = 1 + X + X^4$ over $\mathrm{GF}(2)$ from Table 2.14. Let α be a primitive element of $\mathrm{GF}(2^4)$. Suppose we want to construct the triple-error-correction BCH code of length $2^4 - 1 = 15$. The generator polynomial $\mathbf{g}(X)$ of this code is the minimum-degree polynomial over $\mathrm{GF}(2)$ that has $\alpha, \alpha^2, \alpha^3, \alpha^4, \alpha^5, \alpha^6$, and their conjugates as roots. The minimal polynomials of α, α^3, and α^5 are $1 + X + X^4$, $1 + X + X^2 + X^3 + X^4$, and $1 + X + X^2$, respectively. Then it follows from (3.41) that the generator polynomial $\mathbf{g}(X)$ of the triple-error-correction BCH code is

$$\begin{aligned}\mathbf{g}(X) &= (1 + X + X^4)(1 + X + X^2 + X^3 + X^4)(1 + X + X^2)\\ &= 1 + X + X^2 + X^4 + X^5 + X^8 + X^{10}.\end{aligned}$$

Since the degree of $\mathbf{g}(X)$ is 10, the triple-error-correction BCH code generated by $\mathbf{g}(X)$ is a (15,5) code.

Suppose the zero code polynomial is transmitted and $\mathbf{r}(X) = X^3 + X^5 + X^{12}$ is received. In this case, the error polynomial $\mathbf{e}(X)$ is identical to the received polynomial. To decode $\mathbf{r}(X)$, we compute its syndrome $\mathbf{S} = (S_1, S_2, S_3, S_4, S_5, S_6)$, where

$$\begin{aligned}S_1 &= \mathbf{r}(\alpha) = \alpha^3 + \alpha^5 + \alpha^{12} = 1, & S_2 &= \mathbf{r}(\alpha^2) = \alpha^6 + \alpha^{10} + \alpha^{24} = 1,\\ S_3 &= \mathbf{r}(\alpha^3) = \alpha^9 + \alpha^{15} + \alpha^{36} = \alpha^{10}, & S_4 &= \mathbf{r}(\alpha^4) = \alpha^{12} + \alpha^{20} + \alpha^{48} = 1,\\ S_5 &= \mathbf{r}(\alpha^5) = \alpha^{15} + \alpha^{25} + \alpha^{60} = \alpha^{10}, & S_6 &= \mathbf{r}(\alpha^6) = \alpha^{18} + \alpha^{30} + \alpha^{72} = \alpha^5.\end{aligned}$$

With the above computed syndrome components, executing the BM algorithm results in Table 3.4.

From Table 3.4, we find that the error-location polynomial is $\sigma(X) = \sigma^{(6)}(X) = 1 + X + \alpha^5 X^3$. On substituting the indeterminate X of $\sigma(X)$ by the nonzero elements of $\mathrm{GF}(2^4)$, we find that the roots of $\sigma(X)$ are α^3, α^{10}, and α^{12}. The inverses of these roots are $\alpha^{-3} = \alpha^{12}$, $\alpha^{-10} = \alpha^5$, and $\alpha^{-12} = \alpha^3$, respectively. Therefore, the locations of errors of the error pattern $\mathbf{e}(X)$ are X^3, X^5, and X^{12}, and the error pattern is

Table 3.4. The decoding table for the (15,5) BCH code given in Example 3.2

Step k	Partial solution $\sigma^{(k)}(X)$	Discrepancy d_k	Degree l_k	Step/degree difference $k - l_k$
-1	1	1	0	-1
0	1	1	0	0
1	$1 + X$	0	1	0
2	$1 + X$	α^5	1	1
3	$1 + X + \alpha^5 X^2$	0	2	1
4	$1 + X + \alpha^5 X^2$	α^{10}	2	2
5	$1 + X + \alpha^5 X^3$	0	3	2
6	$1 + X + \alpha^5 X^3$	—	—	—

$\mathbf{e}(X) = X^3 + X^5 + X^{12}$. On adding $\mathbf{e}(X)$ to $\mathbf{r}(x)$, we obtain the zero code polynomial which is the transmitted code polynomial. This completes the decoding process.

Finding the roots of the error-location polynomial $\sigma(X)$, determining the error-location numbers, and the error correction can be carried out systematically all together. The received polynomial $\mathbf{r}(X) = r_0 + r_1 X + \cdots + r_{2^m-1-i} X^{2^m-1-i} + \cdots + r_{2^m-2} X^{2^m-2}$ is read out from a buffer register one bit at a time from the high-order end (i.e., r_{2^m-2} first). At the same time, the indeterminate X of $\sigma(X)$ is replaced by the nonzero elements of GF(2^m) in the order $\alpha, \alpha^2, \ldots, \alpha^{2^m-2}, \alpha^{2^m-1}$, one at a time with α first. For $1 \leq i \leq 2^m - 1$, if α^i is a root of $\sigma(X)$ (i.e., $\sigma(\alpha^i) = 0$), then $\alpha^{-i} = \alpha^{2^m-1-i}$ is an error-location number. In this case, the received bit r_{2^m-1-i} is changed either from 0 to 1 or from 1 to 0 as it is shifted out of the buffer register.

The BM algorithm can be used to find the error-location polynomial for either a binary BCH code or a q-ary BCH code over GF(q) with q as a power of a prime (see the next section). For application to a binary BCH code, it is possible to prove that, if the first, third, $\ldots$, $(2t-1)$th Newton identities are satisfied, then the second, fourth, $\ldots$, $2t$th Newton identities are also satisfied. This implies that, with the BM algorithm for finding the error-location polynomial $\sigma(X)$, the solution $\sigma^{(2k-1)}(X)$ at the $(2k-1)$th step of the BM algorithm is also the solution $\sigma^{(2k)}(X)$ at the $2t$th step of the BM algorithm, i.e.,

$$\sigma^{(2k)}(X) = \sigma^{(2k-1)}(X), \tag{3.64}$$

$1 \leq k \leq t$. The is demonstrated in Table 3.4.

Given the fact above, in executing the BM algorithm the $(2k-1)$th and $2k$th steps can be combined into one step. Consequently, for decoding a binary BCH code, the BM algorithm can be simplified as follows.

Algorithm 3.2 The Simplified BM Algorithm for Finding the Error-Location Polynomial of a Binary BCH Code

Initialization. For $k = -1/2$, set $\sigma^{(-1/2)}(X) = 1$, $d_{-1/2} = 1$, $l_{-1/2} = 0$, and $2(-1/2) - l_{-1/2} = -1$. For $k = 0$, set $\sigma^{(0)}(X) = 1$, $d_0 = S_1$, $l_0 = 0$, and $0 - l_0 = 0$.

1. If $k = t$, output $\sigma^{(k)}(X)$ as the error-location polynomial $\sigma(X)$; otherwise go to Step 2.
2. Compute $d_k = S_{2k+1} + \sigma_1^{(k)} S_{2k} + \sigma_2^{(k)} S_{2k-1} + \cdots + \sigma_{l_k}^{(k)} S_{2k+1-l_k}$ and go to Step 3.
3. If $d_k = 0$, set $\sigma^{(k+1)}(X) = \sigma^{(k)}(X)$; otherwise, set
$$\sigma^{(k+1)}(X) = \sigma^{(k)}(X) + d_k d_i^{-1} X^{2(k-i)} \sigma^{(i)}(X), \tag{3.65}$$
where i is a step prior to the kth step at which the discrepancy $d_i \neq 0$ and $2i - l_i$ is the largest. Go to Step 4.
4. $k \leftarrow k + 1$. Go to Step 1.

The simplified BM algorithm can be executed by setting up and filling in the following table.

Step k	Partial solution $\sigma^{(k)}(X)$	Discrepancy d_k	Degree l_k	Step/degree difference $2k - l_k$
$-1/2$	1	1	0	-1
0	1	S_1	0	0
1	$\sigma^{(1)}(X)$	d_1	l_1	$2 - l_1$
2	$\sigma^{(2)}(X)$	d_2	l_2	$4 - l_2$
$\vdots$				
t	$\sigma^{(t)}(X)$	—	—	—

Example 3.3. Consider the binary (15,5) triple-error-correction BCH code given in Example 3.2. Using the simplified BM algorithm for binary BCH code, Table 3.4 is reduced to Table 3.5.

3.4 Nonbinary Linear Block Codes and Reed–Solomon Codes

So far, we have considered only block codes with symbols from the binary field GF(2). Block codes with symbols from nonbinary fields can be constructed in exactly the same manner as for binary block codes. Block codes with code symbols from GF(q), where q is a power of a prime, are called q-ary block codes (or block codes over GF(q)). A q-ary (n,k) block code has length n and q^k codewords. A message for a q-ary (n,k) block code consists of k information symbols from GF(q).

Table 3.5. A simplified decoding table for the (15,5) BCH code given in Example 3.2

Step k	Partial solution $\sigma^{(k)}(X)$	Discrepancy d_k	Degree l_k	Step/degree difference $2k - l_k$
-1	1	1	0	-1
0	1	1	0	0
1	$1 + X$	α^5	1	1
2	$1 + X + \alpha^5 X^2$	α^{10}	2	2
3	$1 + X + \alpha^5 X^3$	0	3	3

Definition 3.3. A q-ary (n,k) block code of length n with q^k codewords is called a *q-ary (n,k) linear block code* if and only if its q^k codewords form a k-dimensional subspace of the vector space of all the q^n n-tuples over GF(q).

All the fundamental concepts and structural properties developed for binary linear block codes (including cyclic codes) in the previous three sections apply to q-ary linear block codes with few modifications. We simply replace GF(2) by GF(q). A q-ary (n,k) linear block code is specified by either a $k \times n$ generator matrix $\mathbf{G}$ or an $(n-k) \times n$ parity-check matrix $\mathbf{H}$ over GF(q). Generator and parity-check matrices of a q-ary (n,k) linear block code in systematic form have exactly the same forms as given by (3.7) and (3.11), except that their entries are from GF(q). Encoding and decoding of q-ary linear block codes are the same as for binary codes, except that operations and computations are performed over GF(q).

A q-ary (n,k) cyclic code $\mathcal{C}$ is generated by a monic polynomial of degree $n-k$ over GF(q),

$$\mathbf{g}(X) = g_0 + g_1 X + \cdots + g_{n-k-1} X^{n-k-1} + X^{n-k}, \tag{3.66}$$

where $g_0 \neq 0$ and $g_i \in \mathrm{GF}(q)$. This generator polynomial $\mathbf{g}(X)$ is a factor of $X^n - 1$. A polynomial $\mathbf{v}(X)$ of degree $n-1$ or less over GF(q) is a code polynomial if and only if $\mathbf{v}(X)$ is divisible by the generator polynomial of the code.

Let GF(q^m) be an extension field of GF(q) and α be a primitive element of GF(q^m). A t-symbol-error-correcting primitive BCH code of length $q^m - 1$ over GF(q) is a cyclic code generated by the lowest-degree polynomial $\mathbf{g}(X)$ over GF(q) that has $\alpha, \alpha^2, \ldots, \alpha^{2t}$ and their conjugates as roots. For $1 \leq i \leq 2t$, let $\phi_i(X)$ be the (monic) minimal polynomial of α^i over GF(q). Then

$$\mathbf{g}(X) = \mathrm{LCM}\{\phi_1(X), \phi_2(X), \ldots, \phi_{2t}(X)\}. \tag{3.67}$$

Since the degree of a minimal polynomial of an element in GF(q^m) is at most m, the degree of $\mathbf{g}(X)$ is at most $2mt$ and $\mathbf{g}(X)$ divides $X^{q^m-1} - 1$. The q-ary t-symbol-correcting primitive BCH code has length $q^m - 1$ with dimension at least $q^m - 2mt - 1$. Its parity-check matrix $\mathbf{H}$ given in terms of its roots has exactly the same form as that given by (3.44). In the same manner, we can prove that no $2t$ or fewer columns of $\mathbf{H}$ can be added to a zero column vector. Hence, the code has minimum

distance at least $2t+1$ (BCH bound) and is capable of correcting t or fewer random symbol errors over a span of q^m-1 symbol positions. For a given field GF(q^m), a family of q-ary BCH codes can be constructed. The BM algorithm given in the previous section can be used for decoding both binary and q-ary BCH codes.

The most important and most widely used class of q-ary codes is the class of Reed–Solomon (RS) codes which was discovered in 1960 [19], in the same year as the discovery of binary BCH codes by Bose and Chaudhuri [17]. For an RS code, the symbol field and construction field are the same. Such codes can be put into either cyclic or non-cyclic form. We define RS codes in cyclic form. Let α be a primitive element of GF(q). For a positive integer t such that $2t<q$, the generator polynomial of a t-symbol-error-correcting cyclic RS code over GF(q) of length $q-1$ is given by

$$\begin{aligned}\mathbf{g}(X) &= (X-\alpha)(X-\alpha^2)\cdots(X-\alpha^{2t})\\ &= g_0+g_1X+\cdots+g_{2t-1}X^{2t-1}+X^{2t},\end{aligned} \tag{3.68}$$

with $g_i\in$ GF(q). Since $\alpha,\alpha^2,\ldots,\alpha^{2t}$ are roots of $X^{q-1}-1$, $\mathbf{g}(X)$ divides $X^{q-1}-1$. Therefore, $\mathbf{g}(X)$ generates a cyclic RS code of length $q-1$ with exactly $2t$ parity-check symbols. Actually, RS codes form a special subclass of q-ary BCH codes with $m=1$. However, since they were discovered independently and before q-ary BCH codes and are much more important than other q-ary BCH codes in practical applications, we consider them as an independent class of q-ary codes. Since the BCH bound gives a minimum distance (or minimum weight) of at least $2t+1$ and $\mathbf{g}(X)$ is a code polynomial with $2t+1$ nonzero terms (weight $2t+1$), its minimum distance is exactly $2t+1$. Note that the minimum distance $2t+1$ of an RS code is 1 greater than the number $2t$ of its parity-check symbols. A code with minimum distance that is 1 greater than the number of its parity-check symbols is called a *maximum-distance-separable* (*MDS*) code. In summary, a t-symbol-error-correction RS code over GF(q) has the following parameters:

Length	$q-1$,
Number of parity-check symbols	$2t$,
Dimension	$q-2t-1$,
Minimum distance	$2t+1$.

Such an RS code is commonly called a ($q-1$,$q-2t-1$,$2t+1$) RS code over GF(q) with the length, dimension, and minimum distance displayed.

In all practical applications of RS codes in digital communication or data-storage systems, q is commonly chosen as a power of 2, say $q=2^s$, and the code symbols are from GF(2^s). If each code symbol is represented by an s-tuple over GF(2), then an RS code can be transmitted using binary signaling, such as BPSK. In decoding, every s received bits are grouped into a received symbol over GF(2^s). This results in a received sequence of 2^s-1 symbols over GF(2^s). Then decoding is performed on this received symbol sequence. Since RS codes are special q-ary BCH codes, they can be decoded with the BM algorithm. The Euclidean algorithm is also commonly used for decoding RS codes [1, 2, 5, 20].

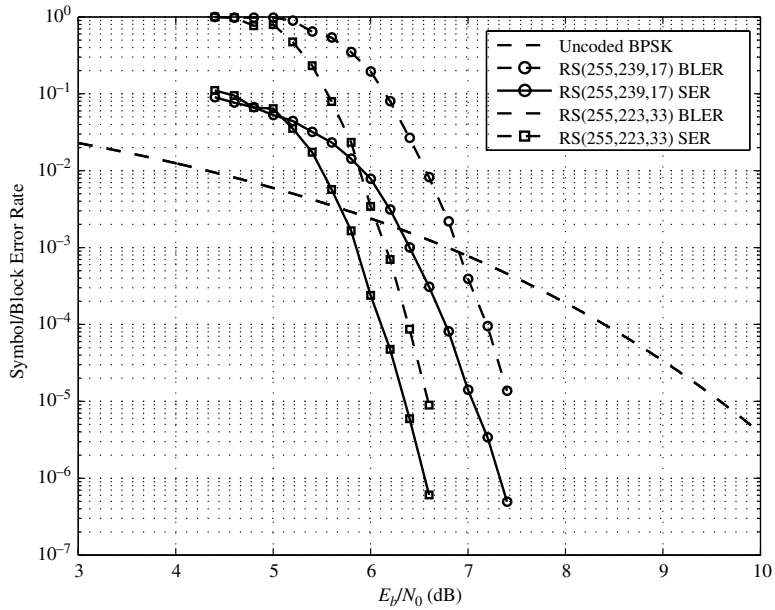


Figure 3.5 Error performances of the (255,239,17) and (255,223,33) RS codes over GF(2^8).

Example 3.4. A widely used RS code in optical communications, data-storage systems, and hard-disk drives is the (255,239,17) RS code over GF(2^8) which is capable of correcting eight or fewer symbol errors. Another important RS code is the (255,223,33) RS code over GF(2^8), which is the NASA standard code for deep-space and satellite communications. This code is capable of correcting 16 or fewer random symbol errors. The symbol and block error performances of these two codes for the binary-input AWGN channel are shown in Figure 3.5.

The $(q-1, q-2t-1, 2t+1)$ cyclic RS code $\mathcal{C}$ over GF(q) generated by $\mathbf{g}(X) = (X-a)(X-\alpha^2)\cdots(X-\alpha^{2t})$ can be extended by adding an extra code symbol to each codeword in $\mathcal{C}$. Let $\mathbf{v}(X) = v_0 + v_1X + \cdots + v_{q-2}X^{q-2}$ be a nonzero code polynomial in $\mathcal{C}$. Its corresponding codeword is $\mathbf{v} = (v_0, v_1, \ldots, v_{q-2})$. Extend this codeword by adding the following code symbol (replacing X of $\mathbf{v}(X)$ by 1):

$$\begin{aligned} v_\infty &= \mathbf{v}(1) \\ &= v_0 + v_1 + \cdots + v_{q-2}. \end{aligned} \tag{3.69}$$

This added code symbol is called the *overall parity-check symbol* of $\mathbf{v}$. The extended codeword is then $\mathbf{v}_{\text{ext}} = (v_\infty, v_0, v_1, \ldots, v_{q-2})$. The extension results in a $(q, q-2t-1, 2t+2)$ code $\mathcal{C}_{\text{ext}}$, called an *extended* RS code. To determine the minimum distance of the extended RS code, there are two cases to be considered. The first case is that $v_\infty = \mathbf{v}(1) = 0$. This implies that 1 is a root of $\mathbf{v}(X)$ and hence $\mathbf{v}(X)$ is divisible by $\mathbf{g}^*(X) = (X-1)\mathbf{g}(X)$. Consequently, $\mathbf{v}(X)$ is a

nonzero codeword in the $(q-1,q-2t-2,2t+2)$ RS code generated by $\mathbf{g}^*(X) = (X-1)\mathbf{g}(X) = (X-1)(X-\alpha)\cdots(X-\alpha^{2t})$ and the weight of $\mathbf{v}(X)$ must be at least $2t+2$. Therefore, the extended codeword $(v_\infty = 0, v_0, \ldots, v_{q-2})$ has weight at least $2t+2$. The second case is that $v_\infty = \mathbf{v}(1) \neq 0$. In this case, adding the overall parity-check symbol v_∞ to $\mathbf{v} = (v_0, v_1, \ldots, v_{q-2})$ increases the weight of $\mathbf{v}$ by 1. This implies that the weight of the extended codeword $\mathbf{v}_{\text{ext}} = (v_\infty, v_1, \ldots, v_{q-2})$ is at least $2t+2$. Note that $\mathbf{g}(X)$ does not have 1 as a root. Hence the overall parity-check symbol for the codeword corresponding to $\mathbf{g}(X)$ is nonzero. Adding the overall parity-check symbol to the codeword corresponding to $\mathbf{g}(X)$ results in an extended codeword of weight exactly $2t+2$. It follows from the above analysis that the minimum weight of the extended $(q,q-2t-1,2t+2)$ RS code $\mathcal{C}_{\text{ext}}$ is exactly $2t+2$. This implies that the minimum distance of the extended RS code $\mathcal{C}_{\text{ext}}$ is exactly $2t+2$.

Suppose $q-1$ is not a prime and can be factored as $q-1 = cn$. Let α be a primitive element in GF(q). Let $\beta = \alpha^c$. Then the order β is n. The n elements, $\beta^0 = 1, \beta, \beta^2, \ldots, \beta^{n-1}$, form a cyclic subgroup of the multiplicative group of GF(q). They also form all the roots of $X^n - 1$. For a positive integer t such that $2t < n$, let $\mathbf{g}(X) = (X-\beta)(X-\beta^2)\cdots(X-\beta^{2t})$. It is clear that $\mathbf{g}(X)$ divides $X^n - 1$. Then all the polynomials of degree $n-1$ or less over GF(q) that are divisible by $\mathbf{g}(X)$ give a *nonprimitive* cyclic $(n,n-2t,2t+1)$ RS code. The parity-check matrix of this nonprimitive RS code in terms of its roots is given by

$$\mathbf{H} = \begin{bmatrix} 1 & \beta & \beta^2 & \cdots & \beta^{n-1} \\ 1 & \beta^2 & (\beta^2)^2 & \cdots & (\beta^2)^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \beta^{2t} & (\beta^{2t})^2 & \cdots & (\beta^{2t})^{n-1} \end{bmatrix}. \tag{3.70}$$

Suppose a $(q-1,q-2t-1,2t+1)$ RS code over GF(q) is used for error control over a noisy channel. Suppose a code polynomial $\mathbf{v}(X) = v_0 + v_1 X + \cdots + v_{q-2}X^{q-2}$ is transmitted. Let $\mathbf{r}(X) = r_0 + r_1 X + \cdots + r_{q-2}X^{q-2}$ be the received polynomial and let

$$\mathbf{e}(X) = e_{j_1}X^{j_1} + e_{j_2}X^{j_2} + \cdots + e_{j_\nu}X^{j_\nu} \tag{3.71}$$

be the error pattern, where $0 \le j_1 < j_2 < \cdots < j_\nu < q-1$ are the locations of the errors in $\mathbf{e}(X)$ and $e_{j_1}, e_{j_2}, \ldots, e_{j_\nu}$ are the values of errors at the locations $j_1, j_2, \ldots, j_\nu$ which are elements in GF(q). Any algebraic hard-decision decoding method must find the locations and values of errors in $\mathbf{e}(X)$ given the $2t$ syndrome components, $S_1 = \mathbf{r}(\alpha), S_2 = \mathbf{r}(\alpha^2), \ldots, S_{2t} = \mathbf{r}(\alpha^{2t})$. Two well-known and efficient algorithms for decoding RS codes (or q-ary BCH codes) are the BM algorithm and Euclid's algorithm [20]. The computational complexities of these two algorithms are about the same; however, Euclid's algorithm is much easier to understand. In the following, we present the decoding procedure for Euclid's algorithm.

For $1 \le i \le \nu$, define

$$\beta_i \triangleq \alpha^{j_i}. \tag{3.72}$$

The exponents of $\alpha^{j_1}, \alpha^{j_2}, \ldots, \alpha^{j_\nu}$ are simply the locations of the errors in $\mathbf{e}(X)$. We call $\beta_1, \beta_2, \ldots, \beta_\nu$ the *error-location numbers*. Define the following polynomial:

$$\begin{aligned}\sigma(X) &= (1-\beta_1 X)(1-\beta_2 X)\cdots(1-\beta_\nu X)\\ &= \sigma_0 + \sigma_1 X + \sigma_2 X^2 + \cdots + \sigma_\nu X^\nu,\end{aligned} \tag{3.73}$$

where $\sigma_0 = 1$. Note that $\sigma(X)$ has $\beta_1^{-1}, \beta_2^{-1}, \ldots, \beta_\nu^{-1}$ (the reciprocals (or inverses) of the error-location numbers) as roots. If we can determine $\sigma(X)$ from the $2t$ syndrome components $S_1, S_2, \ldots, S_{2t}$, then the roots of $\sigma(X)$ give us the reciprocals of the error-location numbers. From the error-location numbers, we can determine the error locations, $j_1, j_2, \ldots, j_\nu$. The polynomial $\sigma(X)$ is called the *error-location polynomial*.

Define

$$\mathbf{S}(X) = S_1 + S_2 X + \cdots + S_{2t} X^{2t-1}, \tag{3.74}$$

and

$$\begin{aligned}\mathbf{Z}_0(X) = S_1 &+ (S_2 + \sigma_1 S_1)X + (S_3 + \sigma_1 S_2 + \sigma_2 S_1)X^2 + \cdots\\ &+ (S_0 + \sigma_1 S_{\nu-1} + \cdots + \sigma_{\nu-1} S_1)X^{\nu-1}.\end{aligned} \tag{3.75}$$

$\mathbf{S}(X)$ and $\mathbf{Z}_0(X)$ are called the *syndrome polynomial* and *error-value evaluator*, respectively. Once $\sigma(X)$ and $\mathbf{Z}_0(X)$ have been found, the locations and values of errors in the error pattern $\mathbf{e}(X)$ can be determined. Note that the degree of $\mathbf{Z}_0(X)$ is at least one less than the degree of the error-location polynomial $\sigma(X)$. The three polynomials $\sigma(X)$, $\mathbf{S}(X)$, and $\mathbf{Z}_0(X)$ are related by the following equation [1]:

$$\sigma(X)\mathbf{S}(X) \equiv \mathbf{Z}_0(X) \ \texttt{modulo}\ X^{2t} \tag{3.76}$$

(i.e. $\sigma(X)S(X) - \mathbf{Z}_0$ is divisible by X^{2t}), which is called the *key-equation*. Any method of solving this key-equation to find $\sigma(X)$ and $\mathbf{Z}_0(X)$ is a decoding method for RS codes (or q-ary BCH codes). If the number of errors in $\mathbf{e}(X)$ is less than or equal to t (the error-correction capability of the code), then the key-equation has a unique pair of solutions ($\sigma(X)$ and $\mathbf{Z}_0(X)$), with

$$\deg[\mathbf{Z}_0(X)] < \deg[\sigma(X)] \le t. \tag{3.77}$$

The key-equation of (3.76) can be solved by using *Euclid's iterative division algorithm* to find the *greatest common divisor* (GCD) of X^{2t} and $\mathbf{S}(X)$. The first step is to divide X^{2t} by $\mathbf{S}(X)$. This results in the following expression:

$$X^{2t} = \mathbf{q}_1(X)\mathbf{S}(X) + \mathbf{Z}_0^{(1)}(X), \tag{3.78}$$

where $\mathbf{q}_1(X)$ and $\mathbf{Z}_0^{(1)}(X)$ are the quotient and remainder, respectively. Then, we divide $\mathbf{S}(X)$ by $\mathbf{Z}_0^{(1)}(X)$. Let $\mathbf{q}_2(X)$ and $\mathbf{Z}^{(2)}(X)$ be the resultant quotient and remainder, respectively. Then, we divide $\mathbf{Z}_0^{(1)}(X)$ by $\mathbf{Z}_0^{(2)}(X)$. We repeat the above division process. Let $\mathbf{Z}_0^{(i-2)}(X)$ and $\mathbf{Z}_0^{(i-1)}(X)$ be the remainders at the $(i-2)$th and $(i-1)$th division steps, respectively. Euclid's algorithm for finding $\sigma(X)$ and $\mathbf{Z}_0(X)$ carries out the following two computations iteratively.

1. At the ith division step with $i = 1, 2, \ldots,$ divide $\mathbf{Z}_0^{(i-2)}(X)$ by $\mathbf{Z}_0^{(i-1)}(X)$ to obtain the quotient $\mathbf{q}_i(X)$ and remainder $\mathbf{Z}_0^{(i)}(X)$.
2. Find $\sigma^{(i)}(X)$ from

$$\sigma^{(i)}(X) = \sigma^{(i-2)}(X) - \mathbf{q}_i(X)\sigma^{(i-1)}(X). \tag{3.79}$$

Iteration begins with the following initial conditions:

$$\begin{aligned} &\mathbf{Z}_0^{(-1)}(X) = X^{2t}, \quad && \mathbf{Z}_0^{(0)}(X) = \mathbf{S}(X), \\ &\sigma^{(-1)}(X) = 0, && \sigma^{(0)}(X) = 1. \end{aligned} \tag{3.80}$$

Iteration stops when a step ρ is reached for which

$$\deg[\mathbf{Z}_0^{(\rho)}(X)] < \deg[\sigma^{(\rho)}(X)] \leq t. \tag{3.81}$$

Then the error-location polynomial $\sigma(X)$ and error-value evaluator are given by

$$\sigma(X) = \sigma^{(\rho)}(X), \tag{3.82}$$

$$\mathbf{Z}_0(X) = \mathbf{Z}_0^{(\rho)}(X). \tag{3.83}$$

If the number of errors in $\mathbf{e}(X)$ is t or less, there always exists a step $\rho \leq 2t$ for which the condition given by (3.81) holds.

Once $\sigma(X)$ and the error-value evaluator $\mathbf{Z}_0(X)$ have been found, the locations and values of errors in the error pattern $\mathbf{e}(X)$ can be determined as follows.

1. Find the roots of $\sigma(X)$ and take the reciprocal of each of the roots, which gives the error-location numbers $\alpha^{j_1}, \alpha^{j_2}, \ldots, \alpha^{j_\nu}$. Then the exponents of α, $j_1, j_2, \ldots, j_\nu$, give the locations of errors in the error pattern $\mathbf{e}(X)$.
2. Let $\sigma'(X)$ be the derivative of $\sigma(X)$. Then the error value at location j_i is given by

$$e_{j_i} = \frac{-\mathbf{Z}_0(\alpha^{-j_i})}{\sigma'(\alpha^{-j_i})}. \tag{3.84}$$

The above two steps completely determine the error pattern $\mathbf{e}(X)$. Then the estimated transmitted code polynomial is given by $\mathbf{v}^*(X) = \mathbf{r}(X) - \mathbf{e}(X)$. The iteration process for finding $\sigma(X)$ and $\mathbf{Z}_0(X)$ can be carried out by setting up and filling a table as shown below:

Iteration step i	$\mathbf{Z}_0^{(i)}(X)$	$\mathbf{q}_i(X)$	$\sigma^{(i)}(X)$
-1	X^{2t}	—	0
0	$\mathbf{S}(X)$	—	1
1			
2			
$\vdots$			
i			
$\vdots$			

Table 3.6. The steps of Euclid's algorithm for finding the error-location polynomial and the error-value evaluator of the (15,9,7) RS code given in Example 3.5

i	$\mathbf{Z}_0^{(i)}(X)$	$\mathbf{q}_i(X)$	$\sigma^{(i)}(X)$
-1	X^6	—	0
0	$\mathbf{S}(X)$	—	1
1	$\alpha^8+\alpha^3X+\alpha^5X^2+\alpha^5X^3+\alpha^6X^4$	$\alpha+\alpha X$	$\alpha+\alpha X$
2	$\alpha^3+\alpha^2X$	$\alpha^{11}+\alpha^8X$	$\alpha^{11}+\alpha^8X+\alpha^9X^2$

Example 3.5. Let GF(2^4) be the field constructed from the primitive polynomial $p(X) = 1+X+X^4$ (see Table 2.14). Let α be a primitive element of GF(2^4). Consider the triple-error-correction (15,9,7) RS code over GF(2^4) generated by

$$\mathbf{g}(X) = (X+\alpha)(X+\alpha^2)(X+\alpha^3)(X+\alpha^4)(X+\alpha^5)(X+\alpha^6).$$

Suppose a code polynomial $\mathbf{v}(X)$ is transmitted and the received polynomial is $\mathbf{r}(X) = \alpha^7X^3+\alpha^{11}X^{10}$. The syndrome components of $\mathbf{r}(X)$ are

$$\begin{aligned}
S_1 &= \mathbf{r}(\alpha) = \alpha^{10}+\alpha^{20} = \alpha^7,\\
S_2 &= \mathbf{r}(\alpha^2) = \alpha^{13}+\alpha^{31} = \alpha^{12},\\
S_3 &= \mathbf{r}(\alpha^3) = \alpha^{16}+\alpha^{41} = \alpha^{6},\\
S_4 &= \mathbf{r}(\alpha^4) = \alpha^{19}+\alpha^{51} = \alpha^{12},\\
S_5 &= \mathbf{r}(\alpha^5) = \alpha^{7}+\alpha = \alpha^{14},\\
S_6 &= \mathbf{r}(\alpha^6) = \alpha^{10}+\alpha^{11} = \alpha^{14}.
\end{aligned}$$

The syndrome polynomial is

$$\mathbf{S}(X) = \alpha^7+\alpha^{12}X+\alpha^6X^2+\alpha^{12}X^3+\alpha^{14}X^4+\alpha^{14}X^5.$$

On executing Euclid's algorithm, we obtain Table 3.6.

We find that at iteration step 2 the condition given by (3.81) holds. Hence

$$\begin{aligned}
\sigma(X) &= \sigma^{(2)}(X)\\
&= \alpha^{11}+\alpha^8X+\alpha^9X^2 = \alpha^{11}(1+\alpha^{10}X)(1+\alpha^3X),\\
\mathbf{Z}_0(X) &= \alpha^3+\alpha^2X.
\end{aligned}$$

The roots of $\sigma(X)$ are α^5 and α^{12}. Their reciprocals are α^{10} and α^3 and hence there are two errors in the estimated error pattern $\mathbf{e}(X)$ at the locations 3 and 10. The error values at the locations 3 and 10 are

$$\begin{aligned}
e_3 &= \frac{-\mathbf{Z}_0(\alpha^{-3})}{\sigma'(\alpha^{-3})} = \frac{\alpha^3+\alpha^2\alpha^{-3}}{\alpha^{11}\alpha^3(1+\alpha^{10}\alpha^{-3})} = \frac{1}{\alpha^8} = \alpha^7,\\
e_{10} &= \frac{-\mathbf{Z}_0(\alpha^{-10})}{\sigma'(\alpha^{-10})} = \frac{\alpha^3+\alpha^2\alpha^{-10}}{\alpha^{11}\alpha^{10}(1+\alpha^3\alpha^{-10})} = \frac{\alpha^4}{\alpha^8} = \alpha^{11}.
\end{aligned}$$

Therefore, the estimated error pattern is $\mathbf{e}(X) = \alpha^7X^3+\alpha^{11}X^{10}$ and the decoded codeword $\mathbf{v}^*(X) = \mathbf{r}(X)-\mathbf{e}(X)$ is the all-zero codeword.

3.5 Product, Interleaved, and Concatenated Codes

In this section, we present three well-known and widely used coding techniques to construct long, powerful codes from short component codes. These coding techniques are called *product* [21], *interleaving*, and *concatenation* [22], which are effective in correcting mixed types of errors, such as combinations of random errors and bursts of errors (or random errors and erasures) [5].

3.5.1 Product Codes

Let $\mathcal{C}_1$ be a binary (n_1,k_1) linear block code, and $\mathcal{C}_2$ be a binary (n_2,k_2) linear block code. A code with n_1n_2 symbols can be constructed by making a *rectangular array* of n_1 columns and n_2 rows in which every row is a codeword in $\mathcal{C}_1$ and every column is a codeword in $\mathcal{C}_2$, as shown in Figure 3.5. The k_1k_2 symbols in the *upper-right quadrant* of the array are information symbols. The $k_2(n_1 - k_1)$ symbols in the *upper-left quadrant* of the array are parity-check symbols formed from the parity-check rules for code $\mathcal{C}_1$, and the $k_1(n_2 - k_2)$ symbols in the *lower-right quadrant* are parity-check symbols formed from the parity-check rules for $\mathcal{C}_2$. The $(n_1 - k_1)(n_2 - k_2)$ parity-check symbols in the *lower-left quadrant* can be formed by using either the parity-check rules for $\mathcal{C}_2$ on columns or the parity-check rules for $\mathcal{C}_1$ on rows. The rectangular array shown in Figure 3.6 is called a *code array* that consists of k_1k_2 information symbols and $n_1n_2 - k_1k_2$ parity-check symbols. There are $2^{k_1k_2}$ such code arrays. The sum of two code arrays is an array obtained by adding either their corresponding rows or their corresponding columns. Since the rows (or columns) are codewords in $\mathcal{C}_1$ (or $\mathcal{C}_2$), the sum of two corresponding rows (or columns) in two code arrays is a codeword in $\mathcal{C}_1$ (or in $\mathcal{C}_2$). Consequently, the sum of two code arrays is another code array. Hence, the $2^{k_1k_2}$ code arrays form a two-dimensional (n_1n_2,k_1k_2) linear block code, denoted $\mathcal{C}_1 \times \mathcal{C}_2$, which is called the *direct product* (or simply product) of $\mathcal{C}_1$ and $\mathcal{C}_2$.

Encoding can be accomplished in two stages. A sequence of k_1k_2 information symbols is first arranged as a $k_2 \times k_1$ array $\mathbf{A}$, the upper-right quadrant of the code array given in Figure 3.6. At the first stage of encoding, each row of $\mathbf{A}$ is encoded into a codeword in $\mathcal{C}_1$. This results in a $k_2 \times n_1$ array $\mathbf{B}$. The first stage of encoding

$$\left[\begin{array}{cccc:ccc}
v_{0,0} & v_{0,1} & \cdots & v_{0,n_1-k_1-1} & v_{0,n_1-k_1} & \cdots & v_{0,n_1-1} \\
v_{1,0} & v_{1,1} & \cdots & v_{1,n_1-k_1-1} & v_{1,n_1-k_1} & \cdots & v_{1,n_1-1} \\
\vdots & \vdots & \ddots & \vdots & \vdots & & \vdots \\
v_{k_2-1,0} & v_{k_2-1,1} & \cdots & v_{k_2-1,n_1-k_1-1} & v_{k_2-1,n_1-k_1} & \cdots & v_{k_2-1,n_1-1} \\
\hdashline
v_{k_2,0} & v_{k_2,1} & \cdots & v_{k_2,n_1-k_1-1} & v_{k_2,n_1-k_1} & \cdots & v_{k_2,n_1-1} \\
\vdots & \vdots & \ddots & \vdots & \vdots & & \vdots \\
v_{n_2-1,0} & v_{n_2-1,1} & \cdots & v_{n_2-1,n_1-k_1-1} & v_{n_2-1,n_1-k_1-1} & \cdots & v_{n_2-1,n_1-1}
\end{array}\right]$$

Figure 3.6 A code array in a two-dimensional product code.

is referred to as the *row encoding*. At the second stage of encoding, each column of B is encoded into a codeword in $\mathcal{C}_2$. After the completion of the second stage of encoding, we obtain an $n_2 \times n_1$ code array as shown in Figure 3.6. The second stage of encoding is referred to as *column encoding*. The above two-stage encoding of the information array $\mathbf{A}$ is referred to as *row/column* encoding. Clearly, two-stage encoding can be accomplished by first performing column encoding of the information array $\mathbf{A}$ and then the row encoding. This two-stage encoding is referred to as *column/row* encoding. With row/column encoding, a code array is transmitted column by column. A column codeword in $\mathcal{C}_2$ is transmitted as soon as it is formed. With column/row encoding, a code array is transmitted row by row. A row codeword in $\mathcal{C}_1$ is transmitted as soon as it is formed. With two-stage encoding, a transmitter buffer is needed to store the code array.

If the minimum weights (or minimum distances) of $\mathcal{C}_1$ and $\mathcal{C}_2$ are d_1 and d_2, respectively, then the minimum weight (or distance) of the product code $\mathcal{C}_1 \times \mathcal{C}_2$ is $d_1 d_2$. A minimum-weight code array in the product code can be formed as follows: (1) choose a minimum-weight codeword $\mathbf{v}_1$ in $\mathcal{C}_1$ and a minimum-weight codeword $\mathbf{v}_2$ in $\mathcal{C}_2$; and (2) form a code array in which all columns corresponding to the zero-components of $\mathbf{v}_1$ are zero columns and all columns corresponding to the 1-components of $\mathbf{v}_1$ are $\mathbf{v}_2$.

Decoding of a product code can be accomplished in two stages. Suppose a code array is transmitted, either column by column or row by row. At the receiving end, the received $n_1 n_2$ symbols are arranged back into an $n_2 \times n_1$ array, called a *received array*. Then decoding is carried out first on columns and then on rows (or first on rows and then on columns), which is referred to as column/row decoding (or row/column decoding). Residue errors that are not corrected at the first stage will be, with high probability, corrected at the second stage. Decoding performance can be improved by carrying out the column/row (row/column) decoding iteratively [5, 21]. Residue errors that are not corrected in one iteration will be, with high probability, corrected in the next iteration. This type of decoding is referred to as *iterative decoding*, which will be discussed in detail in Chapter 7. The complexity of two-stage decoding is roughly the sum of the complexities of the two component code decodings.

A product code over GF(q) can be constructed by using two component codes over GF(q).

3.5.2 Interleaved Codes

Let $\mathcal{C}$ be an (n,k) linear block code and λ a positive integer. We can construct a $(\lambda n, \lambda k)$ linear block code $\mathcal{C}(\lambda)$ by arranging λ codewords from $\mathcal{C}$ as rows of a $\lambda \times n$ array as shown in Figure 3.7 and then transmitting the array, column by column. This process basically interleaves λ codewords in $\mathcal{C}$ such that two neighboring bits of a codeword are separated by $\lambda - 1$ bits from other $\lambda - 1$ codewords. This code-construction technique is called *block interleaving* and the parameter λ is referred

$$\begin{bmatrix} v_{0,0} & v_{0,1} & \cdots & v_{0,n-1} \\ v_{1,0} & v_{1,1} & \cdots & v_{1,n-1} \\ \vdots & \vdots & \ddots & \vdots \\ v_{\lambda-1,0} & v_{\lambda-1,1} & \cdots & v_{\lambda-1,n-1} \end{bmatrix}$$

Figure 3.7 An block-interleaved array.

to as the *interleaving depth* (or *degree*). $\mathcal{C}(\lambda)$ is called an interleaved code with interleaving depth λ.

At the receiving end, before decoding, every λn received symbols must be rearranged column by column back to an $\lambda \times n$ array, called a received array. This process is referred to as *de-interleaving*. Then each row of the received array is decoded on the basis of $\mathcal{C}$ and a decoding algorithm. Interleaving is a very effective technique for correcting *multiple bursts of errors*. Let $\mathcal{C}$ be a t-error-correcting code. Suppose a codeword in the interleaved code $\mathcal{C}(\lambda)$ is transmitted and multiple bursts of errors occur during its transmission. After de-interleaving, if the multiple bursts of errors in the received sequence do not cause more than t errors in each row of the received array, then row decoding based on $\mathcal{C}$ will correct the errors in each row and hence corrects the multiple bursts of errors in the received sequence. The de-interleaving simply makes the multiple bursts of errors in the received sequence appear as random errors in each row of the received array.

3.5.3 Concatenated Codes

Let GF(2^m) be an extension field of GF(2). Then each element in GF(2^m) can be represented by an m-tuple over GF(2). A simple concatenated code is formed by two component codes, an (n_1,k_1) code $\mathcal{C}_1$ over GF(2^m) and a binary (n_2,m) code $\mathcal{C}_2$. Let $\mathbf{u} = (u_0, u_1, \ldots, u_{n_1-1})$ be a codeword in $\mathcal{C}_1$. On expanding each code symbol of $\mathbf{u}$ into an m-tuple of GF(2), we obtain an mn_1-tuple $\mathbf{w} = (w_0, w_1, \ldots, w_{mn_1-1})$ over GF(2). This mn_1-tuple $\mathbf{w}$ is called the *binary image sequence* of $\mathbf{u}$. Encode every m consecutive binary symbols of $\mathbf{w}$ into a codeword in $\mathcal{C}_2$. This results in a sequence of n_1n_2 binary symbols, which consists of a sequence of n_1 binary codewords in $\mathcal{C}_2$. This concatenated sequence of n_1n_2 binary symbols contains k_1m information bits. Since there are 2^{k_1m} codewords in $\mathcal{C}_1$, there are 2^{k_1m} such concatenated n_1n_2-bit sequences, which form a binary (n_1n_2,k_1m) linear block code, called a *concatenated code* [7].

Encoding of a concatenated code consists of two stages as shown in Figure 3.8. First a binary information sequence of k_1m bits is divided into k_1 bytes of m information bits each. Each m-bit byte is regarded as an element in GF(2^m). At the first stage of encoding, the k_1 bytes, regarded as k_1 information symbols over GF(2^m), are encoded into an n_1-byte codeword $\mathbf{u}$ in $\mathcal{C}_1$. The first stage of encoding results in a coded sequence $\mathbf{w}$ of mn_1 bits (or n_1 m-bit bytes). At the

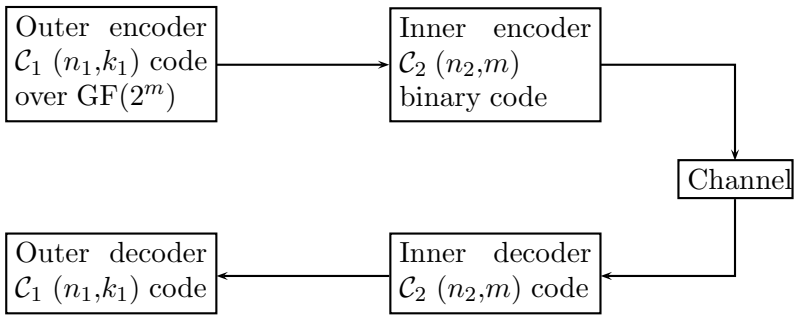


Figure 3.8 A concatenated coding system.

second stage of encoding, every group of m consecutive bits of $\mathbf{w}$ is encoded into an n_2-bit codeword in $\mathcal{C}_2$, resulting in a string of n_1 codewords in $\mathcal{C}_2$. This string of n_1 codewords in $\mathcal{C}_2$ is then transmitted, one $\mathcal{C}_2$ codeword at a time, in succession. Since $\mathcal{C}_1$ and $\mathcal{C}_2$ are used at *outer* and *inner* encoding stages, respectively, as shown in Figure 3.8, they are called outer and inner codes, respectively. If the minimum distances of $\mathcal{C}_1$ and $\mathcal{C}_2$ are d_1 and d_2, then the minimum distance of the concatenation of $\mathcal{C}_1$ and $\mathcal{C}_2$ is $d_1 d_2$.

Decoding of a concatenated code is also done in two stages. First, decoding is carried out for each inner n_2-bit received word as it arrives, based on a decoding method for the inner code $\mathcal{C}_2$, and the parity-check bits are then removed, leaving a sequence of n_1 m-bit bytes. This stage of decoding is called the *inner* decoding. The n_1 decoded bytes at the end of the inner decoding are then decoded based on the outer code $\mathcal{C}_1$ using a certain decoding method. This second decoding stage, called the *outer decoding*, results in k_1 decoded information bytes (or $k_1 m$ decoded information bits).

In concatenated coding systems, RS codes over extension fields of GF(2) are commonly used as outer codes. In the above presentation, a binary linear block code is used as the inner code. However, the convolutional codes to be presented in the next chapter can also be used as inner codes. Usually simple codes are used as inner codes and they are decoded with soft-decision decoding to achieve good error performance.

Binary codes can also be used as outer codes. A concatenated code with both binary outer and binary inner codes can be put into parallel form, in which a single information sequence is encoded by two encoders independently using a pseudo-random interleaver as shown in Figure 3.9. The encoding generates two independent (or uncorrelated) sets of parity-check bits for the same information sequence. At the receiving end, iterative decoding is performed using two decoders based on the two independent sets of parity-check bits, with messages passing between the two decoders. To achieve good error performance, both decoders use soft-decision decoding. Parallel concatenation is also known as *turbo coding*, which will be discussed in Chapter 7.

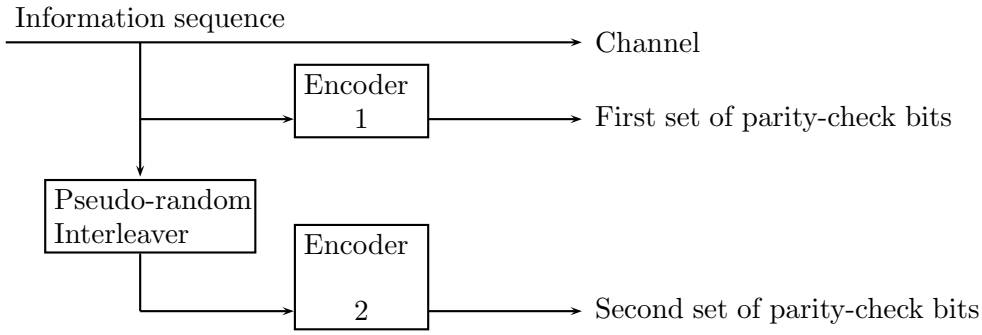


Figure 3.9 A parallel-concatenated coding scheme.

3.6 Quasi-Cyclic Codes

Let t and b be two positive integers. Consider a tb-tuple over GF(2),

$$\mathbf{v} = (\mathbf{v}_0, \mathbf{v}_1, \ldots, \mathbf{v}_{t-1}), \tag{3.85}$$

that consists of t sections of b bits each. For $0 \leq j < t$, the jth section of $\mathbf{v}$ is a b-tuple over GF(2),

$$\mathbf{v}_j = (v_{j,0}, v_{j,1}, \ldots, v_{j,b-1}). \tag{3.86}$$

Let $\mathbf{v}_j^{(1)}$ be the b-tuple over GF(2) obtained by cyclically shifting each component of $\mathbf{v}_j$ one place to the right. We call $\mathbf{v}_j^{(1)}$ the (right) cyclic-shift of $\mathbf{v}_j$. Let

$$\mathbf{v}^{(1)} = (\mathbf{v}_0^{(1)}, \mathbf{v}_1^{(1)}, \ldots, \mathbf{v}_{t-1}^{(1)}). \tag{3.87}$$

The tb-tuple $\mathbf{v}^{(1)}$ is called the *t-sectionized* cyclic-shift of $\mathbf{v}$.

Definition 3.4. Let b, k, and t be positive integers such that $k < tb$. A (tb,k) linear block code $\mathcal{C}_{qc}$ over GF(2) is called a *quasi-cyclic* (QC) code if the following conditions hold: (1) each codeword in $\mathcal{C}_{qc}$ consists of t sections of b bits each; and (2) every t-sectionized cyclic-shift of a codeword in $\mathcal{C}_{qc}$ is also a codeword in $\mathcal{C}_{qc}$. Such a QC code is called a *t-section QC code*.

If $t = 1$, $\mathcal{C}_{qc}$ is a cyclic code of length b. Therefore, cyclic codes form a subclass of QC codes. For $k = cb$ with $1 \leq c < t$, the generator matrix $\mathbf{G}_{qc}$ of a binary (tb,cb) QC code consists of a $c \times t$ *array* of $b \times b$ circulants over GF(2). A $b \times b$ circulant is a $b \times b$ matrix for which each row is a right cyclic-shift of the row above it and the first row is the right cyclic-shift of the last row. For a circulant, each column is a downward cyclic-shift of the column on its left and the first column is the downward cyclic-shift of the last column. The top row (or the leftmost column) of a circulant is called the *generator* of the circulant. The set of columns of a circulant, read from bottom to top (or from top to bottom), is the same as the set of rows of the circulant, read from left to right (or from right to left). Therefore,

the rows and columns of a circulant have the same weight. A $b \times b$ zero matrix may be regarded as a (trivial) circulant.

The generator matrix of a (tb,cb) QC code in systematic form is iven as follows:

$$\mathbf{G}_{qc,\text{sys}} = \begin{bmatrix} \mathbf{G}_0 \\ \mathbf{G}_1 \\ \vdots \\ \mathbf{G}_{c-1} \end{bmatrix} = \underbrace{\begin{bmatrix} \mathbf{G}_{0,0} & \mathbf{G}_{0,1} & \cdots & \mathbf{G}_{0,t-c-1} \\ \mathbf{G}_{1,0} & \mathbf{G}_{1,1} & \cdots & \mathbf{G}_{1,t-c-1} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{G}_{c-1,0} & \mathbf{G}_{c-1,1} & \cdots & \mathbf{G}_{c-1,t-c-1} \end{bmatrix}}_{\mathbf{P}} \underbrace{\begin{matrix} \mathbf{I} & \mathbf{O} & \cdots & \mathbf{O} \\ \mathbf{O} & \mathbf{I} & \cdots & \mathbf{O} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{O} & \mathbf{O} & \cdots & \mathbf{I} \end{matrix}}_{\mathbf{I}_{cb}}, \tag{3.88}$$

where $\mathbf{I}$ is a $b \times b$ identity matrix, $\mathbf{O}$ a $b \times b$ zero matrix, and $\mathbf{G}_{i,j}$ a $b \times b$ circulant with $0 \leq i < c$ and $0 \leq j < t - c$. The generator matrix of the form given by (3.88) is said to be in *systematic circular form*. We see that the parity submatrix $\mathbf{P}$ on the left-hand side of $\mathbf{G}_{qc,\text{sys}}$ is a $c \times (t - c)$ array of $b \times b$ circulants. $\mathbf{G}_{qc,\text{sys}}$ is a $cb \times tb$ matrix over GF(2). Let

$$\begin{aligned} \mathbf{u} &= (\mathbf{u}_0, \mathbf{u}_1, \ldots, \mathbf{u}_{c-1}) \\ &= (u_0, u_1, \ldots, u_{cb-1}) \end{aligned}$$

be an information sequence of cb bits that consists of c sections, of b bits each. The codeword for $\mathbf{u}$ in systematic form is given by

$$\mathbf{v} = \mathbf{u}\mathbf{G} = \mathbf{u}_0\mathbf{G}_0 + \mathbf{u}_1\mathbf{G}_1 + \cdots + \mathbf{u}_{c-1}\mathbf{G}_{c-1}. \tag{3.89}$$

Example 3.6. Consider the four-section (20,10) systematic QC code generated by the following generator matrix in systematic circular form:

$$\mathbf{G}_{qc,\text{sys}} = \left[\begin{array}{ccccc|ccccc|ccccc|ccccc}
1&0&1&0&0&1&0&0&0&1&1&0&0&0&0&0&0&0&0&0\\
0&1&0&1&0&1&1&0&0&0&0&1&0&0&0&0&0&0&0&0\\
0&0&1&0&1&0&1&1&0&0&0&0&1&0&0&0&0&0&0&0\\
1&0&0&1&0&0&0&1&1&0&0&0&0&1&0&0&0&0&0&0\\
0&1&0&0&1&0&0&0&1&1&0&0&0&0&1&0&0&0&0&0\\
\hline
1&1&0&0&0&0&1&0&1&0&0&0&0&0&0&1&0&0&0&0\\
0&1&1&0&0&0&0&1&0&1&0&0&0&0&0&0&1&0&0&0\\
0&0&1&1&0&1&0&0&1&0&0&0&0&0&0&0&0&1&0&0\\
0&0&0&1&1&0&1&0&0&1&0&0&0&0&0&0&0&0&1&0\\
1&0&0&0&1&1&0&1&0&0&0&0&0&0&0&0&0&0&0&1
\end{array}\right].$$

Suppose the information sequence to be encoded is $\mathbf{u} = (\mathbf{u}_0, \mathbf{u}_1) = (10000, 00011)$ which consists of two sections, 5 bits each. The corresponding codeword for $\mathbf{u}$ is

$$\begin{aligned} \mathbf{v} &= (\mathbf{v}_0, \mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3) \\ &= (00110, 01100, 10000, 000011), \end{aligned}$$

which consists of four sections, 5 bits each. On cyclically shifting each section of $\mathbf{v}$ one place to the right, we obtain the following vector:

$$\begin{aligned}\mathbf{v}^{(1)} &= (\mathbf{v}_0^{(1)}, \mathbf{v}_1^{(1)}, \mathbf{v}_2^{(1)}, \mathbf{v}_3^{(1)}) \\ &= (00011, 00110, 01000, 10001),\end{aligned}$$

which is the codeword for the information sequence $\mathbf{u}' = (01000, 10001)$.

Encoding of a QC code in systematic circular form can be implemented using simple shift-registers with linear complexity [23]. For $0 \le i < c$ and $0 \le j < t - c$, let $\mathbf{g}_{i,j}$ be the generator of the circulant $\mathbf{G}_{i,j}$ in the $\mathbf{P}$-matrix of the generator matrix $\mathbf{G}_{qc,\text{sys}}$ given by (3.88). For $0 \le l < b$, let $\mathbf{g}_{i,j}^{(l)}$ be the b-tuple obtained by cyclically shifting every component of $\mathbf{g}_{i,j}$ l places to the right. This b-tuple $\mathbf{g}_{i,j}^{(l)}$ is called the lth right cyclic shift of $\mathbf{g}_{i,j}$. It is clear that $\mathbf{g}_{i,j}^{(0)} = \mathbf{g}_{i,j}^{(b)} = \mathbf{g}_{i,j}$. Let $\mathbf{u} = (\mathbf{u}_0, \mathbf{u}_1, \ldots, \mathbf{u}_{c-1}) = (u_0, u_1, \ldots, u_{cb-1})$ be the information sequence of cb bits to be encoded. Divide this sequence into c sections of b bits each, where the ith section of $\mathbf{u}$ is $\mathbf{u}_i = (u_{ib}, u_{ib+1}, \ldots, u_{(i+1)b-1})$ for $0 \le i < c$. Then the codeword for $\mathbf{u}$ is $\mathbf{v} = \mathbf{u}\mathbf{G}_{qc,\text{sys}}$, which has the following systematic form:

$$\mathbf{v} = (\mathbf{p}_0, \mathbf{p}_1, \ldots, \mathbf{p}_{t-c}, \mathbf{u}_0, \mathbf{u}_1, \ldots, \mathbf{u}_{c-1}), \tag{3.90}$$

where, for $0 \le j < t - c$, $\mathbf{p}_j = (p_{j,0}, p_{j,1}, \ldots, p_{j,b-1})$ is a section of b parity-check bits. It follows from $\mathbf{v} = \mathbf{u}\mathbf{G}_{qc,\text{sys}}$ that, for $0 \le j < t - c$,

$$\mathbf{p}_j = \mathbf{u}_0\mathbf{G}_{0,j} + \mathbf{u}_1\mathbf{G}_{1,j} + \cdots + \mathbf{u}_{c-1}\mathbf{G}_{c-1,j}, \tag{3.91}$$

where, for $0 \le i < c$,

$$\mathbf{u}_i\mathbf{G}_{i,j} = u_{ib}\mathbf{g}_{i,j}^{(0)} + u_{ib+1}\mathbf{g}_{i,j}^{(1)} + \cdots + u_{(i+1)b-1}\mathbf{g}_{i,j}^{(b-1)}. \tag{3.92}$$

It follows from (3.90) and (3.91) that the jth parity-check section $\mathbf{p}_j$ can be computed, step by step, as the information sequence $\mathbf{u}$ is shifted into the encoder. The information sequence $\mathbf{u} = (\mathbf{u}_0, \mathbf{u}_1, \ldots, \mathbf{u}_{c-1})$ is shifted into the encoder in the order from $\mathbf{u}_{c-1}$ to $\mathbf{u}_0$, i.e., the section $\mathbf{u}_{c-1}$ is shifted into the encoder first and $\mathbf{u}_0$ last. For $1 \le l < c$, at the lth step, the accumulated sum

$$\mathbf{s}_{l,j} = \mathbf{u}_{c-1}\mathbf{G}_{c-1,j} + \mathbf{u}_{c-2}\mathbf{G}_{c-2,j} + \cdots + \mathbf{u}_{c-l}\mathbf{G}_{c-l,j} \tag{3.93}$$

is formed and stored in an accumulator. At the $(l+1)$th step, the partial sum $\mathbf{u}_{c-l-1}\mathbf{G}_{c-l-1,j}$ is computed from (3.92) and added to $\mathbf{s}_{l,j}$ to form the accumulated sum $\mathbf{s}_{l+1,j}$. At the cth step, the accumulated sum $\mathbf{s}_{c,j}$ gives the jth parity-check section $\mathbf{p}_j$.

By application of the above encoding process and (3.92), the jth parity-check section $\mathbf{p}_j$ can be formed with a *cyclic shift-register–adder–accumulator* (CSRAA) circuit as shown in Figure 3.10. At the beginning of the first step, $\mathbf{g}_{c-1,j}^{(b-1)}$, the $(b-1)$th right cyclic-shift of the generator $\mathbf{g}_{c-1,j}$ of the circulant $\mathbf{G}_{c-1,j}$ is stored

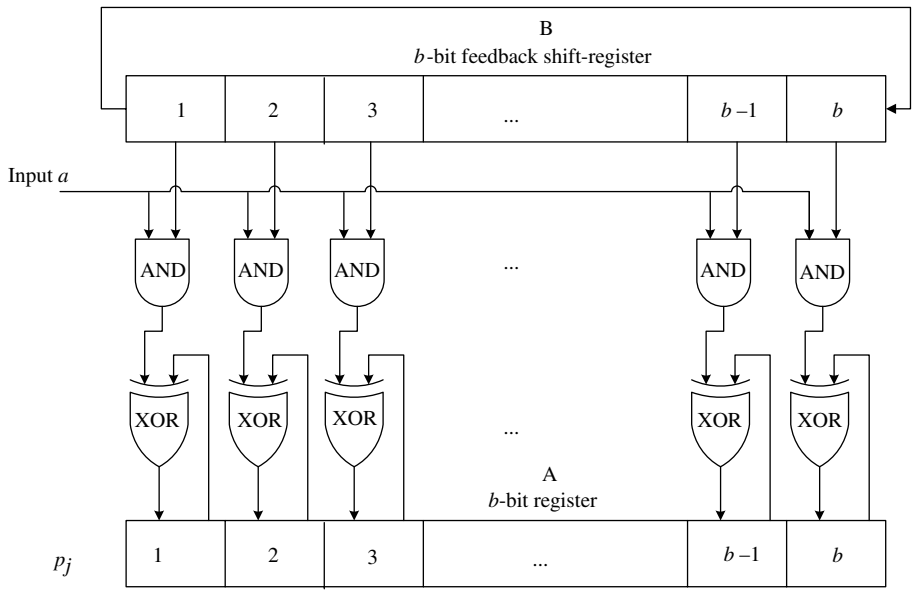


Figure 3.10 A CSRAA encoder circuit.

in the feedback shift-register B, and the content of register A is set to zero. When the information bit u_{cb-1} is shifted into the encoder and the channel, the product $u_{cb-1}\mathbf{g}_{c-1,j}^{(b-1)}$ is formed at the output of AND gates, and is added to the content stored in register A (zero at this time). The sum is then stored back into register A. The feedback shift-register B is shifted once to the left. The new content in A is $\mathbf{g}_{c-1,j}^{(b-2)}$. When the next information bit u_{cb-2} is shifted into the encoder and the channel, the product $u_{cb-2}\mathbf{g}_{c-1,j}^{(b-2)}$ is formed at the output of the AND gates. This product is then added to the sum $u_{cb-1}\mathbf{g}_{c-1,j}^{(b-1)}$ in the accumulator register A. The sum $u_{cb-2}\mathbf{g}_{c-1,j}^{(b-2)} + u_{cb-1}\mathbf{g}_{c-1,j}^{(b-1)}$ is then stored back into A. The above *shift–add–store* process continues. When the last information bit $u_{(c-1)b}$ of information section $\mathbf{u}_{c-1}$ has been shifted into the encoder, register A stores the partial sum $\mathbf{u}_{c-1}\mathbf{G}_{c-1,j}$, which is the contribution to the parity-check section $\mathbf{p}_j$ from the information section $\mathbf{u}_{c-1}$. At this time, $\mathbf{g}_{c-2,j}^{(b-1)}$ the $(b-1)$th right cyclic-shift of the generator $\mathbf{g}_{c-2,j}$ of circulant $\mathbf{G}_{c-2,j}$ is loaded into register B. The shift–add–store process repeats. When the information section $\mathbf{u}_{c-2}$ has been completely shifted into the encoder, register A stores the accumulated partial sum $\mathbf{u}_{c-2}\mathbf{G}_{c-2,j} + \mathbf{u}_{c-1}\mathbf{G}_{c-1,j}$, that is the contribution to the parity-check section $\mathbf{p}_j$ from the information sections $\mathbf{u}_{c-1}$ and $\mathbf{u}_{c-2}$. The above process repeats until the last information section $\mathbf{u}_0$ has been shifted into the encoder. At this time, the accumulator register A contains the jth parity-check section $\mathbf{p}_j$. To form $t-c$ parity-check sections, $t-c$ CSRAA circuits are needed, one for computing each parity-check section. A block diagram for the entire encoder is shown in

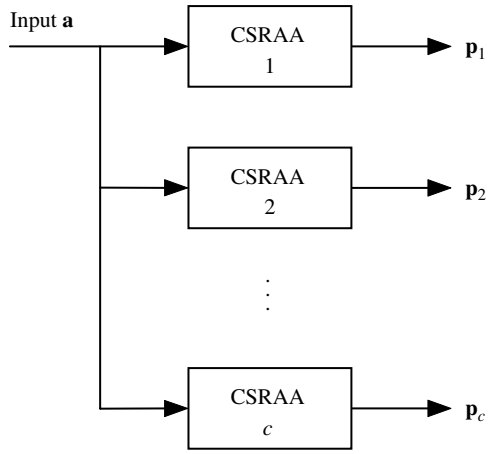


Figure 3.11 A CSRAA-based QC-LDPC code encoder.

Figure 3.11. All the parity-check sections are formed at the same time in parallel, and they are then shifted into the channel serially. The encoding circuit consists of $t-c$ CSRAA circuits with a total of $2(t-c)b$ flip-flops, $(t-c)b$ AND gates, and $(t-c)b$ two-input XOR gates (modulo-2 adders). The encoding is accomplished in linear time with complexity linearly proportional to the number of parity-check bits $(t-c)b$.

In construction, a binary QC code $\mathcal{C}_{qc}$ is commonly specified by a parity-check matrix, which is an array of $(t-c) \times t$ array of $b \times b$ circulants over GF(2) of the following form:

$$\mathbf{H} = \begin{bmatrix} \mathbf{A}_{0,0} & \mathbf{A}_{0,1} & \cdots & \mathbf{A}_{0,t-1} \\ \mathbf{A}_{1,0} & \mathbf{A}_{1,1} & \cdots & \mathbf{A}_{1,t-1} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{A}_{t-c-1,0} & \mathbf{A}_{t-c-1,1} & \cdots & \mathbf{A}_{t-c-1,t-1} \end{bmatrix}. \tag{3.94}$$

It is a $(t-c)b \times tb$ matrix over GF(2). $\mathcal{C}_{qc}$ is given by the null space of $\mathbf{H}$. If the rank r of $\mathbf{H}$ is equal to its number $(t-c)b$ of rows, we say that $\mathbf{H}$ is a *full-rank* matrix. In this case, $\mathcal{C}_{qc}$ is a (tb,cb) QC code. If $\mathbf{H}$ is not a full-rank matrix (i.e., the number of rows of $\mathbf{H}$ is greater than its rank), then the dimension of $\mathcal{C}_{qc}$ is greater than cb. Given the parity-check matrix $\mathbf{H}$ of a QC code $\mathcal{C}_{qc}$ in the form of (3.94), we need to find its generator matrix in the systematic circulant form of (3.88) for efficient encoding. There are two cases to be considered: (1) $\mathbf{H}$ is a full-rank matrix; and (2) $\mathbf{H}$ is not a full-rank matrix.

We first consider the case that the parity-check matrix $\mathbf{H}$ of a QC code given by (3.94) is a full-rank matrix, i.e., the rank r of $\mathbf{H}$ is $(t-c)b$. In this case, we assume that the columns of circulants of $\mathbf{H}$ given by (3.94) are arranged in such a way that the rank of the following $(t-c) \times (t-c)$ subarray of $\mathbf{H}$ (the leftmost

$t-c$ columns of $\mathbf{H}$) is $(t-c)b$, the rank of $\mathbf{H}$,

$$\mathbf{D} = \begin{bmatrix} \mathbf{A}_{0,0} & \mathbf{A}_{0,1} & \cdots & \mathbf{A}_{0,t-c-1} \\ \mathbf{A}_{1,0} & \mathbf{A}_{1,1} & \cdots & \mathbf{A}_{1,t-c-1} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{A}_{t-c-1,0} & \mathbf{A}_{t-c-1,1} & \cdots & \mathbf{A}_{t-c-1,t-c-1} \end{bmatrix}. \tag{3.95}$$

The necessary and sufficient condition for the matrix $\mathbf{G}_{qc,\text{sys}}$ given by (3.88) to be the generator matrix of $\mathcal{C}_{qc}$ with $\mathbf{H}$ given by (3.94) as a parity-check matrix is

$$\mathbf{H}\mathbf{G}_{qc,\text{sys}}^{\text{T}} = [\mathbf{O}], \tag{3.96}$$

where $[\mathbf{O}]$ is a $(t-c)b \times cb$ zero matrix.

For $0 \le i < c$ and $0 \le j < t-c$, let $\mathbf{g}_{i,j}$ be the generator of the circulant $\mathbf{G}_{i,j}$ in the $\mathbf{P}$-matrix of $\mathbf{G}_{qc,\text{sys}}$. Once we have found the $\mathbf{g}_{i,j}$s from (3.96), we can form all the circulants $\mathbf{G}_{i,j}$ of $\mathbf{G}_{qc,\text{sys}}$. Let $\mathbf{u} = (1, 0, \ldots, 0)$ be the unit b-tuple with a "1" in the first position, and $\mathbf{0} = (0, 0, \ldots, 0)$ be the all-zero b-tuple. For $0 \le i < c$, the first row of the submatrix $\mathbf{G}_i$ of $\mathbf{G}_{qc,\text{sys}}$ given by (3.88) is

$$\mathbf{g}_i = (\mathbf{g}_{i,0}\ \mathbf{g}_{i,1} \cdots \mathbf{g}_{i,t-c-1}\ \mathbf{0} \cdots \mathbf{0}\ \mathbf{u}\ \mathbf{0} \cdots \mathbf{0}), \tag{3.97}$$

where the unit b-tuple $\mathbf{u}$ is at the $(t-c+i)$th position of $\mathbf{g}_i$ (section-wise). The row $\mathbf{g}_i$ consists of t sections, each consisting of b bits. The first $t-c$ sections are simply the generators of the $t-c$ circulants $\mathbf{G}_{i,0}, \mathbf{G}_{i,1}, \ldots, \mathbf{G}_{i,t-c-1}$ of the ith submatrix $\mathbf{G}_i$ of $\mathbf{G}_{qc,\text{sys}}$. If follows from (3.96) that $\mathbf{H}\mathbf{g}_i^{\text{T}} = \mathbf{0}$ (the all-zero $(t-c)b$-tuple) for $0 \le i < c$. Define

$$\mathbf{z}_i = (\mathbf{g}_{i,0}\mathbf{g}_{i,1} \cdots \mathbf{g}_{i,t-c-1}) \tag{3.98}$$

and

$$\mathbf{M}_{t-c+i} = \begin{bmatrix} \mathbf{A}_{0,t-c+i} \\ \mathbf{A}_{1,t-c+i} \\ \vdots \\ \mathbf{A}_{t-c-1,t-c+i} \end{bmatrix} \tag{3.99}$$

with $0 \le i < c$. $\mathbf{M}_{t-c+i}$ is simply the $(t-c+i)$th column of circulants of $\mathbf{H}$. Then $\mathbf{H}\mathbf{g}_i^{\text{T}} = \mathbf{0}$ gives the following equality:

$$\mathbf{D}\mathbf{z}_i^{\text{T}} + \mathbf{M}_{t-c+i}\mathbf{u}^{\text{T}} = \mathbf{0}. \tag{3.100}$$

Since $\mathbf{D}$ is a $(t-c)b \times (t-c)b$ square matrix and has full rank $(t-c)b$, it is non-singular and has an inverse D^{-1}. Since all the matrices and vectors are over GF(2), it follows from (3.100) that

$$\mathbf{z}_i^{\text{T}} = \mathbf{D}^{-1}\mathbf{M}_{t-c+i}\mathbf{u}^{\text{T}}. \tag{3.101}$$

On solving (3.101) for $0 \le i < c$, we obtain $\mathbf{z}_0, \mathbf{z}_1, \ldots, \mathbf{z}_{c-1}$. From $\mathbf{z}_0, \mathbf{z}_1, \ldots, \mathbf{z}_{c-1}$ and (3.98), we obtain all the generators $\mathbf{g}_{i,j}$ of the circulants $\mathbf{G}_{i,j}$ of $\mathbf{G}_{qc,\text{sys}}$. Then $\mathbf{G}_{qc,\text{sys}}$ can be constructed readily.

Now we consider the case for which the rank r of the parity-check matrix $\mathbf{H}$ given by (3.94) is either less than the number $(t-c)b$ of rows of $\mathbf{H}$, or given by $r=(t-c)b$, but there does not exist a $(t-c)\times(t-c)$ subarray $\mathbf{D}$ in $\mathbf{H}$ with rank r. For this case, the generator matrix of the QC code $\mathcal{C}_{qc}$ given by the null space of $\mathbf{H}$ cannot be put exactly into the systematic circulant form given by (3.88), however, it can be put into a *semi-systematic circulant form* that allows encoding with simple shift-registers as shown in Figure 3.10. To construct the generator matrix in semi-systematic circulant form, we first find the *least number* of columns of circulants in $\mathbf{H}$, say l with $t-c\leq l\leq t$, such that these l columns of circulants form a $(t-c)\times l$ subarray $\mathbf{D}^*$, whose rank is equal to the rank r of $\mathbf{H}$. We permute the columns of circulants of $\mathbf{H}$ to form a new $(t-c)\times t$ array $\mathbf{H}^*$ of circulants, such that the first (or the leftmost) l columns of $\mathbf{H}^*$ form the array $\mathbf{D}^*$. Note that the null space of $\mathbf{H}^*$ gives a code that is combinatorially equivalent to the code given by the null space of $\mathbf{H}$, i.e., they give the same error performance with the same decoding.

Let

$$\mathbf{D}^*=\begin{bmatrix}\mathbf{A}_{0,0} & \mathbf{A}_{0,1} & \cdots & \mathbf{A}_{0,l-1}\\ \mathbf{A}_{1,0} & \mathbf{A}_{1,1} & \cdots & \mathbf{A}_{1,l-1}\\ \vdots & \vdots & \ddots & \vdots\\ \mathbf{A}_{t-c,0} & \mathbf{A}_{t-c,1} & \cdots & \mathbf{A}_{t-c,l-1}\end{bmatrix}. \tag{3.102}$$

Then, given $\mathbf{D}^*$, the generator matrix of the QC code $\mathcal{C}_{qc}$ given by the null space of $\mathbf{H}^*$ can be put in the following *semi-systematic circulant form*:

$$\mathbf{G}_{qc,\text{semi-sys}}=\begin{bmatrix}\mathbf{Q}\\ \hdashline \mathbf{G}^*_{qc,\text{sys}}\end{bmatrix}, \tag{3.103}$$

which consists of two submatrices $\mathbf{G}^*_{qc,\text{sys}}$ and $\mathbf{Q}$. The submatrix $\mathbf{G}^*_{qc,\text{sys}}$ is a $(t-l)b\times tb$ matrix in systematic cirulant form,

$$\mathbf{G}^*_{qc,\text{sys}}=\begin{bmatrix}\mathbf{G}_{0,0} & \mathbf{G}_{0,1} & \cdots & \mathbf{G}_{0,l-1} & \mathbf{I} & \mathbf{O} & \cdots & \mathbf{O}\\ \mathbf{G}_{1,0} & \mathbf{G}_{1,1} & \cdots & \mathbf{G}_{1,l-1} & \mathbf{O} & \mathbf{I} & \cdots & \mathbf{O}\\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots\\ \mathbf{G}_{t-l,0} & \mathbf{G}_{t-l,1} & \cdots & \mathbf{G}_{t-l,l-1} & \mathbf{O} & \mathbf{O} & \cdots & \mathbf{I}\end{bmatrix}. \tag{3.104}$$

where each $\mathbf{G}_{i,j}$ is a $b\times b$ circulant, $\mathbf{I}$ is a $b\times b$ identity matrix, and $\mathbf{O}$ is a $b\times b$ zero matrix. Then the generators $\mathbf{g}_{i,j}$ of the $b\times b$ circulants $\mathbf{G}_{i,j}$ in $\mathbf{G}^*_{qc,\text{sys}}$ with $0\leq i<t-l$ and $0\leq j<l$ can be obtained by solving Equation (3.100) with $\mathbf{D}$ replaced by $\mathbf{D}^*$ and setting the $bl-r$ linearly dependent elements in $\mathbf{z}_i=(\mathbf{g}_{i,0}\mathbf{g}_{i,1}\cdots\mathbf{g}_{i,l-1})$ to zeros. The linearly dependent elements in $\mathbf{z}_i$ correspond to the linearly dependent columns of $\mathbf{D}^*$.

The submatrix $\mathbf{Q}$ of $\mathbf{G}_{qc,\text{semi-sys}}$ is an $(lb-r)\times tb$ matrix whose rows are linearly independent, and also linearly independent of the rows of the submatrix $\mathbf{G}^*_{qc,\text{sys}}$. For $\mathbf{G}_{qc,\text{semi-sys}}$ to be a generator matrix of the null space of $\mathbf{H}^*$, $\mathbf{Q}$ must satisfy

the condition $\mathbf{H}^*\mathbf{Q}^{\mathrm{T}} = \mathbf{O}$, where $\mathbf{O}$ is a $(t-c)b \times (lb-r)$ zero matrix. To obtain $\mathbf{Q}$, let $d_0, d_1, \ldots, d_{l-1}$ be the number of linearly dependent columns in the 0th, 1st, …, $(l-1)$th columns of circulants in $\mathbf{D}^*$, respectively, such that

$$d_0 + d_1 + \cdots + d_{l-1} = lb - r. \tag{3.105}$$

For a $b \times b$ circulant, if its rank is b, then all the columns (or rows) are linearly independent. If its rank λ is less than b, then any λ consecutive columns (or rows) of the circulant are linearly independent and the other $b - \lambda$ columns (or rows) are linearly dependent. Starting from this structure, we take the last $b - d_j$ columns of the jth column of circulants of $\mathbf{D}^*$ as the linearly independent columns. Then the first $d_0, d_1, \ldots, d_{l-1}$ columns of the 0th, 1st, …, $(l-1)$th columns of circulants of $\mathbf{D}^*$ are linearly dependent columns. $\mathbf{Q}$ can be put into the following circulant form:

$$\mathbf{Q} = \begin{bmatrix} \mathbf{Q}_0 \\ \mathbf{Q}_1 \\ \vdots \\ \mathbf{Q}_{l-1} \end{bmatrix} = \begin{bmatrix} \mathbf{Q}_{0,0} & \mathbf{Q}_{0,1} & \cdots & \mathbf{Q}_{0,l-1} & \mathbf{O}_{0,0} & \mathbf{O}_{0,1} & \cdots & \mathbf{O}_{0,t-l-1} \\ \mathbf{Q}_{1,0} & \mathbf{Q}_{1,1} & \cdots & \mathbf{Q}_{1,l-1} & \mathbf{O}_{1,0} & \mathbf{O}_{1,1} & \cdots & \mathbf{O}_{1,t-l-1} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ \mathbf{Q}_{l-1,0} & \mathbf{Q}_{l-1,1} & \cdots & \mathbf{Q}_{l-1,l-1} & \mathbf{O}_{l-1,0} & \mathbf{O}_{l-1,1} & \cdots & \mathbf{Q}_{l-1,t-l-1} \end{bmatrix}, \tag{3.106}$$

where (1) $\mathbf{O}_{i,j}$ is a $d_i \times b$ zero matrix with $0 \le i < l$ and $0 \le j < t - l$; and (2) $\mathbf{Q}_{i,j}$ is a $d_i \times b$ *partial circulant* obtained by cyclically shifting its first row $d_i - 1$ times to form the other $d_i - 1$ rows with $0 \le i, j < l$.

For $0 \le i < l$, let

$$\begin{aligned} \mathbf{q}_i &= (\mathbf{w}_i, \mathbf{0}) \\ &= (q_{i,0}, q_{i,1}, \ldots, q_{i,lb-1}, 0, 0, \ldots, 0) \end{aligned} \tag{3.107}$$

be the first row of the ith submatrix $\mathbf{Q}_i$ of $\mathbf{Q}$, which consists of two parts, the right part $\mathbf{0}$ and left part $\mathbf{w}_i$. The right part $\mathbf{0} = (0, 0, \ldots, 0)$ consists of $(t-l)b$ zeros. The left part $\mathbf{w}_i = (q_{i,0}, q_{i,1}, \ldots, q_{i,lb-1})$ consists of lb bits, which correspond to the lb columns of $\mathbf{D}^*$. The $lb - r$ bits of $\mathbf{w}_i$ that correspond to the linearly dependent columns of $\mathbf{D}^*$, called dependent bits, have the following form:

$$(\mathbf{0}_0, \ldots, \mathbf{0}_{i-1}, \mathbf{u}_i, \mathbf{0}_{i+1}, \ldots, \mathbf{0}_{l-1}), \tag{3.108}$$

where, for $e \neq i$, $\mathbf{0}_e$ is a zero d_e-tuple and $\mathbf{u}_i = (1, 0, \ldots, 0)$ is a unit d_i-tuple. From the structure of $\mathbf{w}_i$, the number of unknown components in $\mathbf{w}_i$ is r, the rank of $\mathbf{D}^*$ (or $\mathbf{H}^*$).

The condition $\mathbf{H}^*\mathbf{Q}^{\mathrm{T}} = \mathbf{O}$ gives the following equation for $0 \le i < l$:

$$\mathbf{D}^*\mathbf{w}_i^{\mathrm{T}} = \begin{bmatrix} \mathbf{A}_{0,0} & \mathbf{A}_{0,1} & \cdots & \mathbf{A}_{0,l-1} \\ \mathbf{A}_{1,0} & \mathbf{A}_{1,1} & \cdots & \mathbf{A}_{1,l-1} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{A}_{t-c-1,0} & \mathbf{A}_{t-c-1,1} & \cdots & \mathbf{A}_{t-c-1,l-1} \end{bmatrix} \begin{bmatrix} q_{i,0} \\ q_{i,1} \\ \vdots \\ q_{i,lb-1} \end{bmatrix} = \mathbf{0}. \tag{3.109}$$

On solving (3.109), we find $\mathbf{w}_i = (q_{i,0}, q_{i,1}, \ldots, q_{i,lb-1})$ for $0 \le i < l$. Divide $\mathbf{w}_i$ into l sections, denoted $\mathbf{w}_{i,0}, \mathbf{w}_{i,1}, \ldots, \mathbf{w}_{i,l-1}$, each consisting of b consecutive components of $\mathbf{w}_i$. For $0 \le i, j < l$, the partial circulant $\mathbf{Q}_{i,j}$ of $\mathbf{Q}$ is obtained by using $\mathbf{w}_{i,j}$ as the first row, and then cyclically shifting it $d_i - 1$ times to form the other $d_i - 1$ rows. From the $\mathbf{Q}_{i,j}$s with $0 \le i, j < l$, we form the $\mathbf{Q}$ submatrix of $\mathbf{G}_{qc,\text{semi-sys}}$. By combining $\mathbf{Q}$ and $\mathbf{G}^*_{qc,\text{sys}}$ into the form of (3.108), we obtain the generator matrix $\mathbf{G}_{qc,\text{semi-sys}}$ of QC code $\mathcal{C}_{qc}$ given by the null space of a non-full-rank array $\mathbf{H}$ of circulants.

Given $\mathbf{G}_{qc,\text{sem-sys}}$ as described in (3.103), an encoder with l CSRAA circuits of the form given by Figures 3.10 and 3.11 can be implemented. Encoding consists of two phases. An information sequence $\mathbf{a} = (a_0, a_1, \ldots, a_{tb-r-1})$ of $tb - r$ bits is divided into two parts, $\mathbf{a}^{(1)} = (a_{lb-r}, a_{lb-r+1}, \ldots, a_{tb-r-1})$ and $\mathbf{a}^{(2)} = (a_0, a_1, \ldots, a_{lb-r-1})$. Then $\mathbf{a} = (\mathbf{a}^{(2)}, \mathbf{a}^{(1)})$. The information bits are shifted into the encoder serially in the order of bit a_{tb-r-1} first and bit a_0 last. The first part $\mathbf{a}^{(1)}$ of $\mathbf{a}$, consisting of $(t-l)b$ bits, is first shifted into the encoder and is encoded into a codeword $\mathbf{v}^{(1)}$ in the subcode $\mathcal{C}^{(1)}_{qc}$ generated by $\mathbf{G}^*_{qc,\text{sys}}$. The l parity sections are generated at the same time when $\mathbf{a}^{(1)}$ has been completely shifted into the encoder, as described in the first case. After encoding of $\mathbf{a}^{(1)}$, the second part $\mathbf{a}^{(2)}$ of $\mathbf{a}$ is then shifted into the encoder and is encoded into a codeword $\mathbf{v}^{(2)}$ in the subcode $\mathcal{C}^{(2)}_{qc}$ generated by $\mathbf{Q}$. By adding $\mathbf{v}^{(1)}$ and $\mathbf{v}^{(2)}$, we obtain the codeword $\mathbf{v} = \mathbf{v}^{(1)} + \mathbf{v}^{(2)}$ for the information sequence $\mathbf{a} = (\mathbf{a}^{(2)}, \mathbf{a}^{(1)})$.

To encode $\mathbf{a}^{(2)}$ using $\mathbf{Q}$, we divide $\mathbf{a}^{(2)}$ into l sections, $\mathbf{a}^{(2)}_0, \mathbf{a}^{(2)}_1, \ldots, \mathbf{a}^{(2)}_{l-1}$, with $d_0, d_1, \ldots, d_{l-1}$ bits, respectively. Then the codeword for $\mathbf{a}^{(2)}$ is of the form

$$\mathbf{v}^{(2)} = (\mathbf{v}^{(2)}_0, \mathbf{v}^{(2)}_1, \ldots, \mathbf{v}^{(2)}_{l-1}, \mathbf{0}, \mathbf{0}, \ldots, \mathbf{0}), \tag{3.110}$$

which consists of $t - l$ zero sections and l nonzero sections, $\mathbf{v}^{(2)}_0, \mathbf{v}^{(2)}_1, \ldots, \mathbf{v}^{(2)}_{l-1}$, each section, zero or nonzero, consisting of b bits. For $0 \le j < l$,

$$\mathbf{v}^{(2)}_j = \mathbf{a}^{(2)}_0 \mathbf{Q}_{0,j} + \mathbf{a}^{(2)}_1 \mathbf{Q}_{1,j} + \cdots + \mathbf{a}^{(2)}_{l-1} \mathbf{Q}_{l-1,j}. \tag{3.111}$$

Since each $\mathbf{Q}_{i,j}$ in $\mathbf{Q}$ with $0 \le i, j < l$ is a partial circulant with d_i rows, encoding of $\mathbf{a}^{(2)}$ can be accomplished with the same l CSRAA circuits as used for encoding $\mathbf{a}^{(1)}$. For $0 \le j < l$, at the end of encoding $\mathbf{a}^{(1)}$, the accumulator A of the jth CSRAA stores the jth parity-check section $\mathbf{v}^{(1)}_j$ of $\mathbf{v}^{(1)}$. In the second phase of encoding, $\mathbf{a}^{(2)}_0, \mathbf{a}^{(2)}_1, \ldots, \mathbf{a}^{(2)}_{l-1}$ are shifted into the encoder one at a time and the generators, $\mathbf{w}_{0,j}, \mathbf{w}_{1,j}, \ldots, \mathbf{w}_{l-1,j}$, of the partial circulants $\mathbf{Q}_{0,j}, \mathbf{Q}_{1,j}, \ldots, \mathbf{Q}_{l-1,j}$ are stored in register B of the jth CSRAA in turn. Then the cyclically shifts B, $d_0, d_1, \ldots, d_{l-1}$ times, respectively. At the end of $d_0 + d_1 + \cdots + d_{l-1} = lb - r$ shifts, the jth parity section, $\mathbf{v}^{(1)}_j + \mathbf{v}^{(2)}_j$, is stored in the accumulator register A of the jth CSRAA circuit.

Note that the codeword $\mathbf{v} = (v_0, v_1, \ldots, v_{tb-1})$ for the information sequence $\mathbf{a} = (\mathbf{a}^{(2)}, \mathbf{a}^{(1)})$ is not completely systematic. Only the rightmost $(t-l)b$ bits of $\mathbf{v}$ are

identical to the information bits in $\mathbf{a}^{(1)}$, i.e., $(v_{lb}, v_{lb+1}, \ldots, v_{tb-1}) = \mathbf{a}^{(1)}$. The next $lb - r$ bits, $v_r, v_{r+1}, \ldots, v_{lb-1}$, of $\mathbf{v}$ are not identical to the information bits in $\mathbf{a}^{(2)}$. The right-most bits $v_0, v_1, \ldots, v_{r-1}$ of $\mathbf{v}$ are parity-check bits.

3.7 Repetition and Single-Parity-Check Codes

Repetition and single-parity-check codes are two very simple types of linear block codes. A repetition code C_{rep} over GF(2) of length n is a binary $(n,1)$ linear code with a single information bit. The code is simply obtained by repeating a single information bit n times. Therefore, it consists of only two codewords, namely the all-zero codeword, $(00\ldots0)$, and the all-one codeword, $(11\ldots1)$. Obviously, its generator matrix is

$$\mathbf{G}_{\text{rep}} = [1\ 1 \ldots 1].$$

A single-parity-check (SPC) code C_{spc} over GF(2) of length n is a binary $(n,n-1)$ linear code for which each codeword consists of $n-1$ information bits and a single parity-check bit. Let $\mathbf{u} = (u_0, u_1, \ldots, u_{n-2})$ be the message to be encoded. Then a single parity-check bit c is added to it to form an n-bit codeword $(c, u_0, u_1, \ldots, u_{n-2})$. This single parity-check bit c is simply the modulo-2 sum of the $n-1$ information bits of the message $\mathbf{u}$, i.e.,

$$c = u_0 + u_1 + \cdots + u_{n-2}.$$

Hence, every codeword in C_{spc} has even weight. C_{spc} simply consists of all the n-tuples over GF(2) with even weight and hence its minimum distance is 2. Any error pattern with an odd number of errors will change a codeword in C_{spc} into a non-codeword and any error pattern with a nonzero even number of errors will change a codeword in C_{spc} into another codeword. This implies that C_{spc} is capable of detecting any error pattern containing an odd number of errors but not any error pattern containing a nonzero even number of errors. Such SPC codes are commonly used in communication and storage systems for simple error detection.

The generator matrix of an $(n, n-1)$ SPC code in systematic form is given as follows:

$$\mathbf{G}_{\text{spc}} = \left[\begin{array}{c|c} 1 & \\ 1 & \\ 1 & \mathbf{I}_{n-1} \\ \vdots & \\ 1 & \end{array}\right] = \left[\begin{array}{c|cccc} 1 & 1 & 0 & 0 & 0\cdots0 \\ 1 & 0 & 1 & 0 & 0\cdots0 \\ 1 & 0 & 0 & 1 & 0\cdots0 \\ \vdots & \vdots & & \ddots & \vdots \\ 1 & 0 & 0 & 0 & 0\cdots1 \end{array}\right],$$

where $\mathbf{I}_{n-1}$ is an $(n-1)\times(n-1)$ identity matrix. It is easy to check that the inner product of the single row of the generator matrix $\mathbf{G}_{\text{rep}}$ of the $(n, n-1)$ repetition code and any row of the generator matrix $\mathbf{G}_{spc}$ of the $(n, n-1)$ SPC code is zero,

i.e., $\mathbf{G}_{\text{rep}}\mathbf{G}_{spc}^T = 0$ (an $n-1$ zero-tuple). Therefore, the $(n,1)$ repetition code and the $(n,n-1)$ SPC code are dual codes.

Repetition and SPC codes are extremely simple codes; however, they are quite useful in many applications, as will be seen in later chapters.

Problems

3.1 Consider a binary linear block code with the following matrix as a generator matrix:

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

1. Put the given generator matrix into systematic form and find all the codewords.
2. Find the parity-check matrix of the code in systematic form. Determine the parity-check equations.
3. What is the minimum distance of the code?
4. Determine the weight distribution of the code.

3.2 Consider the code given in Problem 3.1 in systematic form. Suppose a codeword is transmitted over the BSC with transition probability $p = 0.01$ and $\mathbf{r} = (01110110)$ is the corresponding received vector.

1. Compute the syndrome of the received vector.
2. Find all the error patterns that have the syndrome you have computed.
3. Compute the probabilities of the error patterns you have found and determine the most probable error pattern.
4. Compute the probability of an undetected error of the code given in Problem 3.1.

3.3 Prove that the Hamming distance satisfies the triangle inequality given by (3.18).

3.4 Prove Theorem 3.1.

3.5 Prove Theorem 3.2.

3.6 Prove Theorem 3.3.

3.7 Construct a standard array of the code given in Problem 3.1 which realizes the maximum-likelihood decoding for a BSC with transition probability $p < 1/2$. Compute the probability of a decoding error based on the standard array that you have constructed.

3.8 In a standard array for an (n,k) linear block code $\mathcal{C}$, prove that (a) the sum of two vectors in the same coset is a codeword in $\mathcal{C}$; (b) no two vectors in the same

coset or two different cosets are the same; (c) all the vectors in a coset have the same syndrome; and (d) different cosets have different syndromes.

3.9 Using a standard array of a linear block code $\mathcal{C}$ for decoding, prove that, if an error pattern is not a coset leader, then the decoding of a received vector is incorrect.

3.10 Let $\mathcal{C}$ be a binary (n,k) linear block code with minimum distance $d_{\min}$. Let $t = \lfloor (d_{\min} - 1)/2 \rfloor$. Show that all the n-tuples over GF(2) with weights t or less can be used as coset leaders in a standard array of $\mathcal{C}$. Show that at least one n-tuple over GF(2) with weight $t+1$ can not be used as a coset leader.

3.11 Determine the code given by the null space of the following matrix over GF(2):

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

If we regard matrix $\mathbf{H}$ as the incidence matrix of a bipartite graph, draw this bipartite graph. What is the girth of this bipartite graph?

3.12 Let $\mathbf{H} = [1 \quad 1 \quad \cdots \quad 1]$ be a $1 \times n$ matrix over GF(2) where all the entries of the single row are the 1-element of GF(2). Determine the code given by the null space of $\mathbf{H}$ and its weight distribution. This code is called a *single-parity-check* (SPC) code. Draw the bipartite graph with $\mathbf{H}$ as the incidence matrix.

3.13 Let $\mathbf{g}(X) = 1 + X + X^2 + X^4 + X^5 + X^8 + X^{10}$ be the generator polynomial of a (15,5) cyclic code over GF(2).

1. Construct the generator matrix of the code in systematic form.
2. Find the parity-check polynomial of the code.

3.14 Let α be a primitive element of the Galois field GF(2^5) generated by the primitive polynomial $\mathbf{p}(X) = 1 + X^2 + X^5$ (see Table 2.10). Find the generator polynomial of the triple-error-correcting primitive BCH code of length 31.

3.15 Consider the primitive triple-error-correction BCH code over GF(2) constructed in Problem 3.14. Suppose a code polynomial $\mathbf{v}(X)$ is transmitted and $\mathbf{r}(X) = 1 + X^5 + X^{21}$ is the received polynomial. Decode $\mathbf{r}(X)$ using the BM algorithm.

3.16 Consider the primitive tripe-error-correction (63,45) BCH code over GF(2) constructed in Example 3.1. Suppose a code polynomial $\mathbf{v}(X)$ is transmitted and

$\mathbf{r}(X) = 1 + X^{61} + X^{62}$ is the received polynomial. Decode $\mathbf{r}(X)$ using the BM algorithm.

3.17 Using the Galois field GF(2^5) generated by the primitive polynomial $p(X) = 1 + X^2 + X^5$ (see Table 2.10) find the generator polynomial of the triple-error-correcting RS code $\mathcal{C}$ over GF(2^5) of length 31. Let α be a primitive element of GF(2^5). Suppose a code polynomial $\mathbf{v}(X)$ of $\mathcal{C}$ is transmitted and $\mathbf{r}(X) = \alpha^2 + \alpha^{21}X^{12} + \alpha^7 X^{20}$ is the corresponding received polynomial. Decode $\mathbf{r}(X)$ using Euclid's algorithm.

3.18 Decode the received polynomial $\mathbf{r}(X) = \alpha^2 + \alpha^{21}X^{12} + \alpha^7 X^{20}$ given in Problem 3.17 using the BM algorithm.

3.19 For $i = 1$ and 2, let $\mathcal{C}_i$ be an (n_i,k_i) linear block code over GF(2) with generator and parity-check matrices $\mathbf{G}_i$ and $\mathbf{H}_i$, respectively.

1. Find a generator matrix of the product code $\mathcal{C} = \mathcal{C}_1 \times \mathcal{C}_2$ in terms of $\mathbf{G}_1$ and $\mathbf{G}_2$.
2. Find a parity-check matrix of the product code $\mathcal{C} = \mathcal{C}_1 \times \mathcal{C}_2$ in terms of $\mathbf{H}_1$ and $\mathbf{H}_2$.

3.20 Let $\mathbf{g}(X)$ be the generator polynomial of an (n,k) cyclic code $\mathcal{C}$. Suppose we interleave $\mathcal{C}$ by a depth of λ. Prove that the interleaved code $\mathcal{C}_\lambda$ is also a cyclic code with the generator polynomial $\mathbf{g}(X^\lambda)$.

3.21 Let $\mathcal{C}_1$ and $\mathcal{C}_2$ be the outer and inner codes of a concatenated code $\mathcal{C}$ with minimum distances d_1 and d_2, respectively. Prove that the minimum distance of the concatenated code $\mathcal{C}$ is $d_1 \times d_2$.

References

[1] E. R. Berlekamp, *Algebraic Coding Theory*, revised edn., Laguna Hills, CA, Aegean Park Press, 1984.

[2] R. E. Blahut, *Algebraic Codes for Data Transmission*, Cambridge, Cambridge University Press, 2003.

[3] I. F. Blake and R.C. Mullin, *The Mathematical Theory of Coding*, New York, Academic Press, 1975.

[4] G. Clark and J. Cain, *Error-Correcting Codes for Digital Communications*, New York, Plenum Press, 1981.

[5] S. Lin and D. J. Costello, Jr., *Error Control Coding: Fundamentals and Applications*, 2nd edn., Upper Saddle River, NJ, Prentice-Hall, 2004.

[6] F. J. MacWilliams and N. J. A. Sloane, *The Theory of Error-Correcting Codes*, Amsterdam, North-Holland, 1977.

[7] R. J. McEliece, *The Theory of Information Theory and Coding*, Reading, MA, Addison-Wesley, 1977.

[8] A. M. Michaelson and A. J. Levesque, *Error Control Coding Techniques for Digital Communications*, New York, John Wiley & Sons, 1985.

[9] J. C. Moreira and P. G. Farrell, *Essentials of Error-Control Coding*, West Sussex, John Wiley & Sons, 2006.

[10] W. W. Peterson, "Encoding and error-correction procedures for the Bose–Chaudhuri codes," *IRE Trans. Information Theory*, vol. 6, no. 5, pp. 459–470, September 1960.

[11] W. W. Peterson and E. J. Weldon, Jr., *Error-Correcting Codes*, 2nd edn., Cambridge, MA, MIT Press, 1972.

[12] V. Pless, *Introduction to the Theory of Error Correcting Codes*, 3rd edn., New York, Wiley, 1998.

[13] A. Poli and L. Huguet, *Error Correcting Codes, Theory and Applications*, Hemel Hempstead, Prentice-Hall, 1992.

[14] R. M. Roth, *Introduction to Coding Theory*, Cambridge, Cambridge University Press, 2006.

[15] S. B. Wicker, *Error Control Systems for Digital Communication and Storage*, Englewood Cliffs, NJ, Prentice-Hall, 1995.

[16] A. Hocquenghem, "Codes correcteurs d'erreurs," Chiffres, vol. 2, pp. 147–156, 1959.

[17] R. C. Bose and D. K. Ray-Chaudhuri, "On a class of error correcting binary group codes," *Information Control*, vol. 3, pp. 68–79, March 1960.

[18] J. L. Massey, "Shift-register synthesis and BCH decoding," *IEEE Trans. Information Theory*, vol. 15, no. 1, pp. 122–127, January 1969.

[19] I. S. Reed and G. Solomon, "Polynomial codes over certain finite fields," *J. Soc. Indust. Appl. Math.* vol. 8, pp. 300–304, June 1960.

[20] Y. Sugiyama, M. Kasahara, and T. Namekawa, "A method for solving key equation for decoding Goppa codes," *Information Control*, vol. 27, pp. 87–99, January 1975.

[21] P. Elias, "Error-free coding," *IRE Trans. Information Theory*, vol. 4, no. 5, pp. 29–37, September 1954.

[22] G. D. Forney, Jr., *Concatenated Codes*, Cambridge, MA, MIT Press, 1966.

[23] Z. Li, L. Chen, L. Zeng, S. Lin, and W. Fong, "Efficient encoding of quasi-cyclic low-density parity-check codes," *IEEE Trans. Communications*, vol. 54, no. 1, pp. 71–81, January 2006.

4 Convolutional Codes

The class of convolutional codes was invented by Elias in 1955 [1] and has been widely in use for wireless, space, and broadcast communications since about 1970. Their popularity stems from the relative ease with which the maximum-likelihood sequence decoder may be implemented and from their effectiveness when concatenated with a Reed–Solomon code. Since the early 1990s, they have enjoyed new respect and popularity due to the efficacy of concatenations of multiple convolutional codes in turbo(-like) codes.

In this chapter, we introduce the class of convolutional codes. We first present algebraic descriptions of convolutional codes, which were pioneered by Forney [2] and extended by Johannesson and Zigangirov [3]. We also discuss various encoder realizations and matrix representations of convolutional codes. We then discuss a graphical (trellis) representation, which aids two optimal decoding algorithms, the Viterbi algorithm [4, 5] and the BCJR algorithm [6]. Finally, we present techniques for bounding the performance of convolutional codes based on the technique of Viterbi [7]. Our discussion in this chapter will include only binary convolutional codes. Extensions of the various details to the nonbinary case are straightforward. See [8, 9] for additional information on convolutional codes, including tables of optimal codes.

4.1 The Convolutional Code Archetype

Convolutional codes are linear codes with a very distinct algebraic structure. While they can be utilized in block-oriented (packet-based) situations, their encoders are frequently described as stream-oriented. That is, in contrast with a block code, whose encoder assigns an n-bit codeword to each block of k data bits, a convolutional encoder assigns code bits to an incoming information bit stream continuously, in a stream-oriented fashion.

The convolutional code archetype is a four-state, rate-1/2 code whose encoder is depicted in Figure 4.1. Observe that two code bits are produced for each data bit that enters the encoder; hence, the rate is 1/2. The state of the encoder is defined to be the contents of the two binary memory cells; hence, the encoder is a four-state device. As with binary block codes, all operations are over the binary field $\mathbb{F}_2$, so the two adders in Figure 4.1 are modulo-2 adders. Observe that

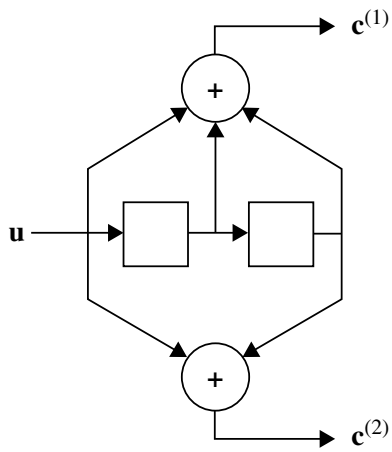


Figure 4.1 A four-state, rate-1/2 convolutional code encoder.

the top and bottom parts of the encoder act as two discrete-time finite-impulse-response filters with operations in $\mathbb{F}_2$. The top filter has impulse response $\mathbf{g}^{(1)} = [1\ 1\ 1]$ and the bottom filter has impulse response $\mathbf{g}^{(2)} = [1\ 0\ 1]$. It is from this perspective that the nomenclature "convolutional" originates: encoder output $\mathbf{c}^{(1)}$ is the convolution of the input $\mathbf{u}$ and the impulse response $\mathbf{g}^{(1)}$, and similarly for encoder output $\mathbf{c}^{(2)}$. Thus, we may write, for $j = 1, 2$,

$$\mathbf{c}^{(j)} = \mathbf{u} \circledast \mathbf{g}^{(j)}.$$

Moreover, since convolution in the time domain is multiplication in the transform domain, this equation may be rewritten as

$$c^{(j)}(D) = u(D)g^{(j)}(D),$$

where the coefficients of the polynomials in D are the elements of the corresponding vectors so that $g^{(1)}(D) = 1 + D + D^2$ and $g^{(2)}(D) = 1 + D^2$. Note that D is equivalently the discrete-time delay operator z^{-1}, although D is generally used in the coding literature.

We may also model the two encoding operations more compactly via the following matrix equation:

$$\begin{aligned}\begin{bmatrix} c^{(1)}(D) & c^{(2)}(D) \end{bmatrix} &= u(D)\begin{bmatrix} g^{(1)}(D) & g^{(2)}(D) \end{bmatrix} \\ &= u(D)\mathbf{G}(D).\end{aligned}$$

The 1×2 matrix $\mathbf{G}(D) = \begin{bmatrix} g^{(1)}(D) & g^{(2)}(D) \end{bmatrix}$ is the code's *generator matrix* and the polynomials $g^{(j)}(D)$ are called *generator polynomials*.

The codeword for such a convolutional encoder is generally taken to be the word formed by multiplexing the bits corresponding to the polynomials $c^{(1)}(D)$ and $c^{(2)}(D)$. For an input word length of ℓ (i.e., the degree of $u(D)$ is $\ell - 1$), the output word length is $2\ell + 4$. The added 4 bits are a consequence of two extra 0s

used to flush out the encoder memory, which results in four extra code bits. Thus, in practice the code rate is $\ell/(2\ell+4)$, which approaches $1/2$ for large ℓ. We remark that, in addition to low-complexity encoding (as in Figure 4.1), convolutional codes allow low-complexity decoding. This is a result of the highly structured but simple relationship between the data word and the codeword.

The following sections provide the framework necessary for a more thorough understanding of convolutional codes. Codes with rates other than $1/2$ will be considered, as will systematic codes. (Note that the code in Figure 4.1 is not systematic.) We first consider a more general algebraic description of convolutional codes than the one in the foregoing example. We then consider various encoder realizations for the various classes of convolutional codes. Next, we present various alternative representations of convolutional codes, including representation as a linear block code and graphical representations. Next, we present optimal decoding algorithms for convolutional codes operating over a binary-input AWGN channel, with an eye toward their use in iterative decoders, to be discussed in subsequent chapters. Finally, we present techniques for estimating the performance of these codes on the AWGN channel.

4.2 Algebraic Description of Convolutional Codes

In order to present an algebraic description of convolutional codes, the following notation for various algebraic structures is necessary. We denote by $\mathbb{F}_2[D]$ the *ring of polynomials over* $\mathbb{F}_2$ in the indeterminate D, where $\mathbb{F}_2$ is a shorthand for the finite field GF(2). Thus, an element of $\mathbb{F}_2[D]$ may be represented by $\sum_{i=0}^{s} a_i D^i$ for some integer $s \geq 0$, where $a_i \in \mathbb{F}_2$. We denote by $\mathbb{F}_2(D)$ the *field of rational functions over* $\mathbb{F}_2$ in the indeterminate D. Thus, an element of $\mathbb{F}_2(D)$ may be represented as a ratio of polynomials, $a(D)/b(D)$, for some $a(D)$ and $b(D) \neq 0$ in $\mathbb{F}_2[D]$. Finally, we denote by $\mathbb{F}_2\langle D\rangle$ the *field of Laurent series over* $\mathbb{F}_2$ in the indeterminate D. Thus, an element of $\mathbb{F}_2\langle D\rangle$ may be represented by $\sum_{i=r}^{\infty} s_i D^i$ for some integer r, where $s_i \in \mathbb{F}_2$. The following relationships are obvious: $\mathbb{F}_2[D] \subset \mathbb{F}_2(D) \subset \mathbb{F}_2\langle D\rangle$. Note that n-tuples of elements of $\mathbb{F}_2\langle D\rangle$, denoted by $\mathbb{F}_2^n\langle D\rangle$, form a vector space with $\mathbb{F}_2\langle D\rangle$ as the scalar field.

Example 4.1. The polynomials $1+D$, $1+D^2$, and $D+D^2+D^3$ are all elements of the ring $\mathbb{F}_2[D]$ (and, hence, are in $\mathbb{F}_2(D)$ and $\mathbb{F}_2\langle D\rangle$ as well). The rational functions

$$\frac{1+D}{D+D^2+D^3}, \quad \frac{1+D}{1+D+D^2}, \text{ and } \frac{1+D^2}{1+D}$$

are all elements of the field $\mathbb{F}_2(D)$. Note that the numerator of the third rational function factors as $1+D^2=(1+D)(1+D)$ and so the third rational function reduces to $1+D$, an element of $\mathbb{F}_2[D]$. The first two are not elements of $\mathbb{F}_2[D]$, however. Application of

long division to the second rational function yields

$$\frac{1+D}{1+D+D^2} = 1 + D^2 + D^3 + D^5 + D^6 + D^8 + D^9 + \cdots,$$

that is, a power series whose coefficients are 1011011011 .... Thus, $(1+D)/(1+D+D^2)$ is clearly in $\mathbb{F}_2\langle D\rangle$ as well as in $\mathbb{F}_2(D)$. The elements of $\mathbb{F}_2(D)$ must have power-series coefficients that are periodic after some initial transient, as seen in the foregoing example. To see this, first note that an element $a(D)/b(D) \in \mathbb{F}_2(D)$ satisfies $a(D) = b(D) \cdot \sum_{i=0}^{\infty} s_i D^i = b(D) \cdot s(D)$. Now note that the only way that the product between the power series $s(D)$ and the polynomial $b(D)$ can be a polynomial $a(D)$ (which has finite length, by definition) is if $s(D)$ is periodic. Thus, an example of an element of a series that is in $\mathbb{F}_2\langle D\rangle$ but not in $\mathbb{F}_2(D)$ is one whose coefficients are not periodic, e.g., are randomly chosen.

Given these definitions, we may now proceed to define a binary convolutional code. Whereas a linear binary block code of length n is a subspace of the vector space $\mathbb{F}_2^n$ of n-tuples over the field $\mathbb{F}_2$, a linear binary convolutional code of length n is a subspace of the vector space $\mathbb{F}_2^n\langle D\rangle$ of n-tuples over the field $\mathbb{F}_2\langle D\rangle$. Thus, given a convolutional code $\mathcal{C}$, a codeword $\mathbf{c}(D) \in \mathcal{C} \subset \mathbb{F}_2^n\langle D\rangle$ has the form

$$\mathbf{c}(D) = [c^{(1)}(D) \quad c^{(2)}(D) \ \ldots \ c^{(n)}(D)],$$

where $c^{(j)}(D) = \sum_{i=r}^{\infty} c_i^{(j)} D^i$ is an element of $\mathbb{F}_2\langle D\rangle$. Further, since $\mathcal{C}$ is a subspace of $\mathbb{F}_2^n\langle D\rangle$, each $\mathbf{c}(D) \in \mathcal{C}$ may be written as a linear combination of basis vectors $\{\mathbf{g}_i(D)\}_{i=1}^k$ from that subspace,

$$\mathbf{c}(D) = \sum_{i=1}^{k} u^{(i)}(D)\mathbf{g}_i(D), \tag{4.1}$$

where $u^{(i)}(D) \in \mathbb{F}_2\langle D\rangle$ and k is the dimension of $\mathcal{C}$. Analogously to linear block codes, we may rewrite (4.1) as

$$\mathbf{c}(D) = \mathbf{u}(D)\mathbf{G}(D), \tag{4.2}$$

where $\mathbf{u}(D) = [u^{(1)}(D) \quad u^{(2)}(D) \quad \ldots \quad u^{(k)}(D)]$ and the basis vectors $\mathbf{g}_i(D) = \left[g_i^{(1)}(D) \quad g_i^{(2)}(D) \quad \ldots \quad g_i^{(n)}(D)\right]$ form the rows of the $k \times n$ *generator matrix* $\mathbf{G}(D)$, so that the element in row i and column j of $\mathbf{G}(D)$ is $g_i^{(j)}(D)$.

In practice, the entries of $\mathbf{G}(D)$ are confined to the field of rational functions $\mathbb{F}_2(D)$ so that encoding and decoding are realizable. (Additional necessary constraints on $\mathbf{G}(D)$ are given in the next section.) Further, while $\left\{u^{(i)}(D)\right\}$ and $\left\{c^{(j)}(D)\right\}$ are still taken from $\mathbb{F}_2\langle D\rangle$, they are restricted to be delay-free, that is, $r = 0$. Observe that, if we multiply a matrix $\mathbf{G}(D)$ with entries in $\mathbb{F}_2(D)$ by

the least common multiple of the denominators of its entries $g_i^{(j)}(D)$, we obtain a generator matrix for the same code whose entries are confined to the ring of polynomials $\mathbb{F}_2[D]$. We denote this *polynomial form* of a generator matrix by $\mathbf{G}_{\text{poly}}(D)$ and emphasize that there exists a polynomial-form generator matrix for any realizable convolutional code. In that which follows, we will let $\mathbf{G}(D)$ represent a generic generator matrix, so that it may be in either systematic or non-systematic form, and its entries may be from $\mathbb{F}_2(D)$ or $\mathbb{F}_2[D]$.

There exists an $(n-k) \times n$ *parity-check matrix* $\mathbf{H}(D)$ whose null space is $\mathcal{C}$. This may be re-stated as

$$\mathbf{c}(D)\mathbf{H}^{\mathrm{T}}(D) = \mathbf{0}$$

for any $\mathbf{c}(D) \in \mathcal{C}$, or as

$$\mathbf{G}(D)\mathbf{H}^{\mathrm{T}}(D) = \mathbf{0}.$$

In the previous two equations, $\mathbf{0}$ represents a $1 \times (n-k)$ zero vector and a $k \times (n-k)$ zero matrix, respectively. As for linear block codes, $\mathbf{G}(D)$ and $\mathbf{H}(D)$ each have a *systematic form*, $\mathbf{G}_{\text{sys}}(D) = [\mathbf{I}|\mathbf{P}(D)]$ and $\mathbf{H}_{\text{sys}}(D) = [\mathbf{P}^{\mathrm{T}}(D)|\mathbf{I}]$. Also, as for linear block codes, $\mathbf{G}_{\text{sys}}(D)$ is obtained by row reduction of $\mathbf{G}(D)$ combined with possible column permutations to obtain the $[\mathbf{I}|\mathbf{P}(D)]$ format. As we will see in the following example, there also exists a polynomial form of the parity-check matrix, denoted by $\mathbf{H}_{\text{poly}}(D)$.

Example 4.2. Consider the generator matrix for a rate-2/3 convolutional code given by

$$\mathbf{G}(D) = \begin{bmatrix} \dfrac{1+D}{1+D+D^2} & 0 & 1+D \\ 1 & \dfrac{1}{1+D} & \dfrac{D}{1+D^2} \end{bmatrix}.$$

If we (a) multiply the first row by $(1+D+D^2)/(1+D)$, (b) add the new first row to the second row, and (c) multiply the new second row by $1+D$, with all operations over $\mathbb{F}_2(D)$, the result is the systematic form

$$\mathbf{G}_{\text{sys}}(D) = \begin{bmatrix} 1 & 0 & 1+D+D^2 \\ 0 & 1 & \dfrac{1+D^3+D^4}{1+D} \end{bmatrix}.$$

Noting that the submatrix $\mathbf{P}(D)$ is the rightmost column of $\mathbf{G}_{\text{sys}}(D)$, we may immediately write

$$\mathbf{H}_{\text{sys}}(D) = \begin{bmatrix} 1+D+D^2 & \dfrac{1+D^3+D^4}{1+D} & 1 \end{bmatrix}$$

because we know that $\mathbf{H}_{\text{sys}}(D) = [\mathbf{P}^{\text{T}}(D)|\mathbf{I}]$. The least common multiple of the denominators of the entries of $\mathbf{G}(D)$ is $(1+D+D^2)(1+D^2)$, from which

$$\mathbf{G}_{\text{poly}}(D) = \begin{bmatrix} (1+D)^3 & 0 & (1+D+D^2)(1+D)^3 \\ (1+D+D^2)(1+D^2) & (1+D+D^2)(1+D) & D(1+D+D^2) \end{bmatrix}.$$

Lastly, by multiplying each of the entries of $\mathbf{H}_{\text{sys}}(D)$ by $1+D$, we obtain

$$\mathbf{H}_{\text{poly}}(D) = \begin{bmatrix} 1+D^3 & 1+D^3+D^4 & 1+D \end{bmatrix}.$$

We emphasize that $\mathbf{G}(D)$, $\mathbf{G}_{\text{sys}}(D)$, and $\mathbf{G}_{\text{poly}}(D)$ all generate the same code, that is, their rows span the same subspace of $\mathbb{F}_2^n\langle D\rangle$. In (4.2), $\mathbf{u}(D)$ represents the data word to be encoded, and $\mathbf{u}(D)\mathbf{G}(D)$ represents the codeword corresponding to that data word. Thus, while the three generator matrices yield the same code, i.e., the same list of codewords, they correspond to different $\mathbf{u}(D) \mapsto \mathbf{c}(D)$ mappings, that is, to different encoders. In that which follows we use "encoder" and "generator matrix" interchangeably because they both describe the same $\mathbf{u}(D) \mapsto \mathbf{c}(D)$ mapping. In the next section we divide the different encoder possibilities for a given code into four classes and we describe two realizations for any given encoder, that is, for any given $\mathbf{G}(D)$.

4.3 Encoder Realizations and Classifications

When considering encoder realizations, it is useful to let D represent the unit-delay operator, which is equivalent to z^{-1} in the discrete-time signal-processing literature. In the encoding operation represented by $\mathbf{c}(D) = \mathbf{u}(D)\mathbf{G}(D)$, we note that there is a discrete-time transfer function $g_i^{(j)}(D)$ between input $u^{(i)}(D)$ and output $c^{(j)}(D)$. Further, when an element $g_i^{(j)}(D)$ of $\mathbf{G}(D)$ is a polynomial from $\mathbb{F}_2[D]$, this transfer function may be implemented as if it were a finite-impulse-response (FIR) filter, but with arithmetic over $\mathbb{F}_2$. When $g_i^{(j)}(D)$ is a rational function from $\mathbb{F}_2(D)$, it may be implemented as if it were an infinite-impulse-response (IIR) filter, but with arithmetic over $\mathbb{F}_2$.

When $g_j^{(i)}(D)$ is a rational function, that is, $g_i^{(j)}(D) = a(D)/b(D)$, we assume that it has the form

$$g_i^{(j)}(D) = \frac{a_0 + a_1 D + \cdots + a_m D^m}{1 + b_1 D + \cdots + b_m D^m} \tag{4.3}$$

so that $b(D)|_{D=0} = b_0 = 1$. This assumption implies that $g_i^{(j)}(D)$ is a causal transfer function and, hence, is realizable. To see this, note that, by definition, the transfer function $g_i^{(j)}(D)$ implies that

$$c^{(j)}(D) = u^{(i)}(D) g_i^{(j)}(D) = u^{(i)}(D) \frac{a(D)}{b(D)}$$

or, in the time domain with discrete-time parameter t,

$$\begin{aligned} b_0 c_t^{(j)} &= a_0 u_t^{(i)} + a_1 u_{t-1}^{(i)} + \cdots + a_m u_{t-m}^{(i)} \\ &\quad - b_1 c_{t-1}^{(j)} - b_2 c_{t-2}^{(j)} - \cdots - b_m c_{t-m}^{(j)}. \end{aligned}$$

Observe that $b_0 = 0$ makes the situation untenable, i.e., $c_t^{(j)}$ cannot be determined from the inputs $u_t^{(i)}, u_{t-1}^{(i)}, \ldots, u_{t-m}^{(i)}$ and the outputs $c_{t-1}^{(j)}, c_{t-2}^{(j)}, \ldots, c_{t-m}^{(j)}$.

Figure 4.2 depicts the *Type I IIR filter realization* of the transfer function $g_i^{(j)}(D) = a(D)/b(D)$ and Figure 4.3 presents the *Type II IIR filter realization.* The Type I form is also called the *controller canonical form* or the *direct canonical form.* The Type II form is also called the *observer canonical form* or the *transposed canonical form.* When $g_i^{(j)}(D)$ is a polynomial, that is, $g_i^{(j)}(D) = a(D)$, this is a special case of a rational function with $b(D) = 1$. Thus, the implementation of $g_i^{(j)}(D)$ in this case is also depicted in Figure 4.2 or Figure 4.3, but with $b_2 = b_3 = \ldots = b_m = 0$, that is, without the feedback. The derivations of both types of realizations are explored in Problems 4.4 and 4.5. See also Sections 6.3 and 6.4 of [10], which discuss implementations of IIR systems.

In that which follows, we shall require that $\mathbf{G}(D)$ be *delay-free*, that is, that $\mathbf{G}(D)$ correspond to a delay-free encoder. This implies that $\mathbf{G}(0)$ is not the zero matrix or, equivalently, that D^l, $l > 0$, cannot be factored out of $\mathbf{G}(D)$. We shall also require that $\mathbf{G}(D)$ be *realizable*, meaning that $\mathbf{G}(D)$ corresponds to a realizable encoder. When $\mathbf{G}(D)$ consists of rational functions, this means that the denominators of all entries of $\mathbf{G}(D)$ have nonzero constant terms. (Consider the impact of a zero constant term, $b_0 = 0$, in either Figure 4.2 or Figure 4.3.) $\mathbf{G}(D)$ in polynomial form is always realizable.

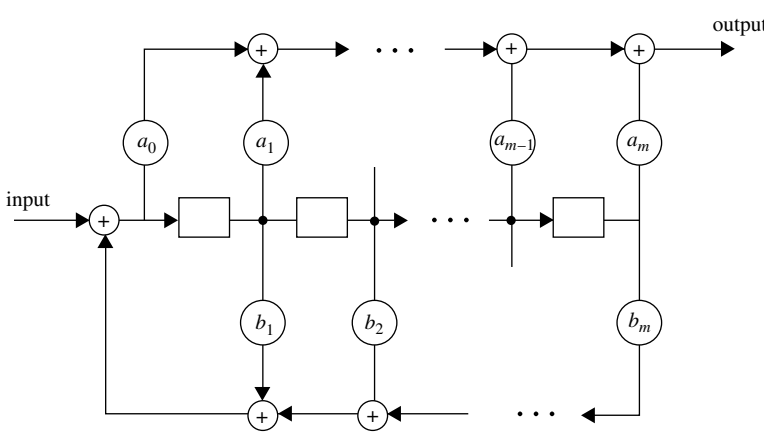


Figure 4.2 Type I realization of the transfer function $g_i^{(j)}(D) = a(D)/b(D)$, with $b_0 = 1$. The input is $u^{(i)}(D)$ and the output is $c^{(j)}(D)$.

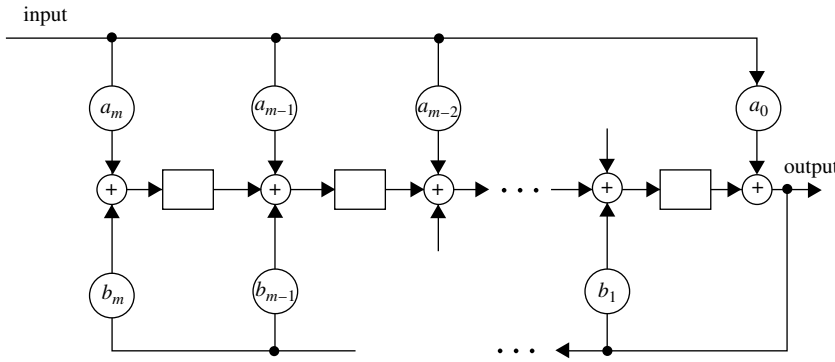


Figure 4.3 Type II realization of the transfer function $g_i^{(j)}(D) = a(D)/b(D)$, with $b_0 = 1$. The input is $u^{(i)}(D)$ and the output is $c^{(j)}(D)$.

Example 4.3. The Type I encoder realization of the rate-1/2 convolutional code with generator matrix $\mathbf{G}(D) = [1 + D + D^2 \quad 1 + D^2]$, originally depicted in Figure 4.1, is presented in Figure 4.4(a). The Type I encoder for the systematic form of this code, for which the generator matrix is

$$\mathbf{G}_{\text{sys}}(D) = \begin{bmatrix} 1 & \dfrac{1 + D^2}{1 + D + D^2} \end{bmatrix},$$

is presented in Figure 4.4(b). The Type II encoder for $\mathbf{G}_{\text{sys}}(D)$ is presented in Figure 4.4(c).

In the table below we give an example input/output pair for each of the encoders in Figures 4.4(a) and 4.4(b). The examples were specifically selected to demonstrate how different inputs lead to the same codeword in each case. The (semi-infinite) vector notation is used in the table. In this notation a polynomial (power series) is represented by a vector (semi-infinite vector) of its coefficients. Thus, the vector representation for $u(D) = u_0 + u_1 D + u_2 D^2 + \ldots$ is $\mathbf{u} = [u_0 \quad u_1 \quad u_2 \quad \ldots]$ and the vector representation for $c^{(j)}(D) = c_0^{(j)} + c_1^{(j)} D + c_2^{(j)} D^2 + \ldots$ is $\mathbf{c}^{(j)} = \begin{bmatrix} c_0^{(j)} & c_1^{(j)} & c_2^{(j)} & \ldots \end{bmatrix}$, $j = 1, 2$. As indicated in Figures 4.4(a) and 4.4(b), in the table the vector $\mathbf{c}$ is the codeword corresponding to multiplexing the bits of $\mathbf{c}^{(1)}$ and $\mathbf{c}^{(2)}$, as is done in practice.

	Figure 4.4(a) encoder	**Figure 4.4(b) encoder**
$u(D)$	1	$1 + D + D^2$
$\mathbf{u}$	$10\ldots$	$1110\ldots$
$c^{(1)}(D)$	$1 + D + D^2$	$1 + D + D^2$
$c^{(2)}(D)$	$1 + D^2$	$1 + D^2$
$\mathbf{c}^{(1)}$	$1110\ldots$	$1110\ldots$
$\mathbf{c}^{(2)}$	$1010\ldots$	$1010\ldots$
$\mathbf{c}$	$111011000\ldots$	$111011000\ldots$

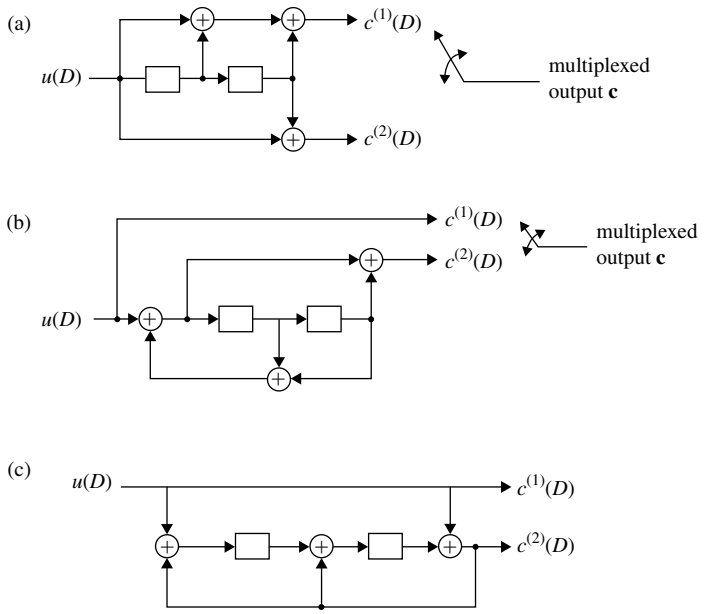


Figure 4.4 Encoder realizations of the non-systematic and systematic versions of the rate-1/2 convolutional code given in Example 4.3 with generator matrix $\mathbf{G}(D) = [1 + D + D^2 \quad 1 + D^2]$. (a) Type I realization of a non-systematic encoder. (b) Type I realization of a systematic encoder. (c) Type II realization of a systematic encoder.

Figure 4.5 presents the Type I encoder realization corresponding to the generator matrix $\mathbf{G}(D)$ of the rate-2/3 convolutional code given in Example 4.2,

$$\mathbf{G}(D) = \begin{bmatrix} \dfrac{1+D}{1+D+D^2} & 0 & 1+D \\ 1 & \dfrac{1}{1+D} & \dfrac{D}{1+D^2} \end{bmatrix}.$$

Note that, in an attempt to minimize the number of delay elements, $1 + D$ was implemented as $(1 + D^3)/(1 + D + D^2)$ and $1(1 + D)$ was implemented as $(1 + D)/(1 + D^2)$. However, we can reduce the number of encoder delay elements further for this code by implementing the encoder for

$$\mathbf{G}_{\text{sys}}(D) = \begin{bmatrix} 1 & 0 & 1+D+D^2 \\ 0 & 1 & \dfrac{1+D^3+D^4}{1+D} \end{bmatrix}$$

in the Type II form as depicted in Figure 4.6. Note that a single shift-register may be shared in the computation of the third column of $\mathbf{G}_{\text{sys}}(D)$ since all operations are linear. As we will see in Section 4.5, each elimination of a delay element results in a halving of the complexity of trellis-based decoders. In other words, decoder complexity is proportional to 2^μ, where μ is the number of encoder delay elements; μ is called the encoder *memory*.

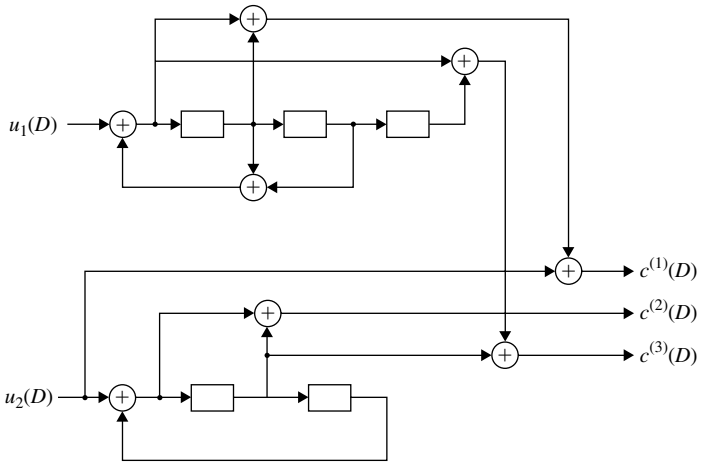


Figure 4.5 Type I encoder realization of $G(D)$ for the rate-2/3 encoder of the convolutional code given in Example 4.2 and discussed further in Example 4.3.

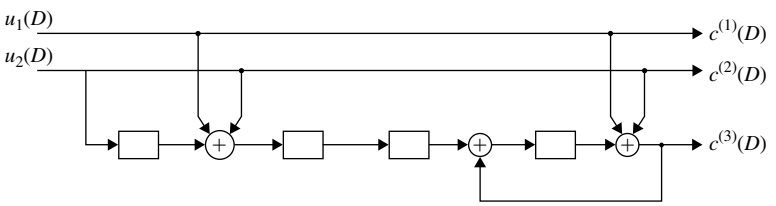


Figure 4.6 Type II encoder realization of $G_{\text{sys}}(D)$ for the rate-2/3 convolutional code given in Examples 4.2 and 4.3.

There exist four classes of encoders or, equivalently, four classes of generator matrices. Example 4.3 gives examples of encoders for three of these classes. The first example, corresponding to $\mathbf{G}(D)$ for the rate-1/2 code, is within the class of *non-recursive non-systematic convolutional* ($\bar{\text{R}}\bar{\text{S}}$C) encoders. The second and fourth examples, corresponding to $\mathbf{G}_{\text{sys}}(D)$ for the rate-1/2 and rate-2/3 codes, respectively, are instances of *recursive systematic convolutional* (RSC) encoders. The third example, corresponding to $\mathbf{G}(D)$ for the rate-2/3 code, is an instance of a *recursive non-systematic* convolutional (R$\bar{\text{S}}$C) encoder. Not included in Example 4.3 is an example of a *non-recursive systematic convolutional* ($\bar{\text{R}}$SC) encoder, a simple instance of which has $\mathbf{G}(D) = \begin{bmatrix} 1 & 1 + D + D^2 \end{bmatrix}$. The word "recursive" indicates the presence of at least one rational function in $\mathbf{G}(D)$ or, equivalently, the presence of feedback in the encoder realization. Conversely, a "non-recursive" encoder indicates the absence of any rational functions in $\mathbf{G}(D)$, that is, $\mathbf{G}(D)$ is in polynomial form, so that the encoder realization is devoid of any feedback. In fact, in the literature a recursive systematic encoder is often called a *systematic encoder with feedback*, and similar nomenclature exists for the other three classes.

Table 4.1. Convolutional encoder classes

Convolutional encoder class	Example $\mathbf{G}(D)$
Non-recursive non-systematic ($\bar{\text{R}}\bar{\text{S}}$C)	$\begin{bmatrix} 1+D+D^2 & 1+D^2 \end{bmatrix}$
Recursive systematic (RSC)	$\begin{bmatrix} 1 & \dfrac{1+D^2}{1+D+D^2} \end{bmatrix}$
Recursive non-systematic (R$\bar{\text{S}}$C)	$\begin{bmatrix} \dfrac{1+D}{1+D+D^2} & 0 & 1+D \\ 1 & \dfrac{1}{1+D} & \dfrac{D}{1+D^2} \end{bmatrix}$
Non-recursive systematic ($\bar{\text{R}}$SC)	$\begin{bmatrix} 1 & 1+D+D^2 \end{bmatrix}$

The four convolutional encoder classes are summarized in Table 4.1, together with their abbreviations and example generator matrices. A generator matrix $\mathbf{G}(D)$ may be realized as either a Type I or a Type II encoder for all four encoder classes listed in Table I. It is important to observe that, while we use $\mathbf{G}(D)$ and "encoder" interchangeably, an "encoder" (the $\mathbf{u}(D) \mapsto \mathbf{c}(D)$ mapping) is distinct from the "encoder realization" (Type I or Type II circuit).

Within a given encoder class there exist many encoders (generator matrices) for a given convolutional code. For example, scaling each entry of a generator matrix $\mathbf{G}(D)$ by the same arbitrary polynomial changes the encoder, but does not change the code. As another example, replacing a given row in $\mathbf{G}(D)$ by the sum of that row and another row does not change the code. The notion that many encoders exist for a single code and, on top of that, two encoder realizations exist, leads us to ask the following questions, each of which will be addressed below.

1. Is one encoder class superior to the others?
2. Are there any encoders to be avoided?
3. Since decoder complexity is proportional to 2^μ, can an encoder class and encoder realization be selected such that the encoder realization has the minimum number of delay elements?
4. What are typical values for n, k, and μ and, given such typical values, how does one design a convolutional code?

4.3.1 Choice of Encoder Class

The answer to the first question depends on the application. For example, many applications require that the encoder be systematic so that the data are readily observable in the code stream. As another example, parallel turbo codes (Chapter 7) require that the constituent convolutional codes be of the recursive type, and usually systematic. For many applications the encoder class matters very little, particularly applications in which the convolutional code is not a constituent

code of a turbo(-like) code. As seen in Example 4.3 and as we shall see in the discussion below of the third question, the choice of encoder class can affect the encoder memory μ.

4.3.2 Catastrophic Encoders

The answer to the second question is that catastrophic encoders are to be avoided, a topic we develop via the following example. Consider the rate-1/2 code with $\mathbf{G}(D) = \begin{bmatrix} 1+D^2 & 1+D \end{bmatrix}$. Note that the entries of $\mathbf{G}(D)$ have the common polynomial factor $1+D$. Now set the encoder input $u(D) = 1/(1+D)$ so that the resulting codeword is $\mathbf{c}(D) = [1+D \quad 1]$. Observe that the input $1/(1+D)$ expands into the power series $1 + D + D^2 + D^3 + \cdots$ corresponding to the binary input sequence of coefficients $\mathbf{u} = 1111\ldots$. Thus, the input has an infinite Hamming weight. On the other hand, the two output sequences are $\mathbf{c} = [\mathbf{c}^{(1)} \quad \mathbf{c}^{(2)}] = [11000\ldots \quad 1000\ldots]$, which has a Hamming weight of three. In practice, these two sequences would be multiplexed, yielding the code sequence $\mathbf{c}' = 111000\ldots$.

Now consider that any decoder implementation must have a finite memory "window." Even in the absence of noise, this decoder, which initially observes the three 1s from the encoder output, will eventually see only zeros in its memory window. Because the all-zeros input word maps to the all-zeros codeword, any decoder will eventually produce zeros at its output, whereas the correct encoder input sequence was all ones. Such a phenomenon is called *catastrophic error propagation* and this encoder is called a *catastrophic encoder*. Formally, an encoder is a catastrophic encoder if there exists an infinite-weight input that yields a finite-weight output. Clearly, a systematic encoder can never be catastrophic. (The literature often loosely uses the term "catastrophic code" when in fact it is only the encoder that can be catastrophic.)

For rate-$1/n$ convolutional codes, the generalization of the above example is as follows. Let $\mathbf{G}(D) = \begin{bmatrix} g^{(1)}(D) & g^{(2)}(D) & \ldots & g^{(n)}(D) \end{bmatrix}$, where only one subscript is necessary for the entries $g^{(j)}(D)$ since there is only one row. We let the $g^{(j)}(D)$ be polynomials without loss of generality since, if $\mathbf{G}(D)$ contains rational functions, it is a simple matter to put it into polynomial form. Suppose now that the $\{g^{(j)}(D)\}$ have a common polynomial factor, $a(D)$, so that $g^{(j)}(D) = a(D)f^{(j)}(D)$. Now let the encoder input be $u(D) = 1/a(D)$, which is clearly not a polynomial, but a rational function of infinite length when expanded into a power series. The codeword corresponding to $u(D)$ would then be $\mathbf{c}(D) = [f^{(1)}(D) \quad f^{(2)}(D) \quad \ldots \quad f^{(n)}(D)]$, which has finite weight since $\{f^{(j)}(D)\}$ are polynomials. Thus, a rate-$1/n$ encoder is catastrophic if (and only if) $\gcd\{g^{(j)}(D)\}$ is a non-monomial (a polynomial with more than one term).

To generalize the catastrophic encoder for rate-k/n convolutional codes, we first define $\{\Delta^{(i)}(D)\}$ to be the determinants of the set of $\binom{n}{k}$ possible $k \times k$ submatrices of the generator matrix $\mathbf{G}(D)$. Then the encoder is catastrophic if

$\gcd\{\Delta^{(i)}(D),\ i = 1, 2, \ldots, \binom{n}{k}\}$ is a non-monomial. See [8] or [3] for a proof of this fact.

4.3.3 Minimal Encoders

We now address the third question: encoders that achieve the minimum encoder memory are called *minimal encoders*. Consider first a rate-$1/n$ code represented by $\mathbf{G}(D) = \begin{bmatrix} g^{(1)}(D) & g^{(2)}(D) & \ldots & g^{(n)}(D) \end{bmatrix}$ in polynomial form. Clearly, if this code is catastrophic, the entries $g^{(j)}(D)$ of $\mathbf{G}(D)$ have a common (non-monomial) polynomial factor and this representation of the encoder is not minimal. However, if the greatest common factor of the $\{g^{(j)}(D)\}$ is factored out, then the resulting generator matrix (encoder) will be minimal. Alternatively, one may convert this generator matrix into the RSC form $\mathbf{G}_{\text{sys}}(D) = \begin{bmatrix} 1 & g^{(2)}(D)/g^{(1)}(D) & \ldots & g^{(n)}(D)/g^{(1)}(D) \end{bmatrix}$, which automatically removes any common factor after the rational functions $g^{(j)}(D)/g^{(1)}(D)$ are reduced to their lowest terms. Note that a rate-$1/n$ minimal encoder is non-catastrophic (and this fact extends to all code rates).

A close inspection reveals that the Type I realization is naturally suited to rate-$1/n$ codes in the RSC format. This is because each of the $n-1$ parity bits may be computed by "tapping off" of the same length-μ shift-register in the appropriate locations and summing (over $\mathbb{F}_2$). In this way, the Type I circuit of Figure 4.2 may be considered to be a single-input/multiple-output device, as is necessary when computing $n-1$ parity bits for each input bit. On the other hand, the Type II realization is particularly ill-suited as an encoder realization of $\mathbf{G}_{\text{sys}}(D)$ given above because it has rate $1/n$. In summary, a minimal encoder for a rate-$1/n$ convolutional code is obtained by applying the Type I encoder realization to $\mathbf{G}_{\text{sys}}(D)$, or to $\mathbf{G}(D)$ with the gcd factored out of its entries.

Example 4.4. For the rate-1/2 code of Example 4.3, the Type I encoder realization for $\mathbf{G}(D)$ and the Type I and Type II encoder realizations for $\mathbf{G}_{\text{sys}}(D)$ are all minimal with $\mu = 2$, whereas the Type II realization of $\mathbf{G}(D)$ has $\mu = 4$.

Consider the rate-1/3 code with $\mathbf{G}(D) = [1 + D^3 \quad 1 + D + D^2 + D^3 \quad 1 + D^2]$. If we divide out the gcd, $1 + D$, from each of the entries of $\mathbf{G}(D)$, we obtain $\mathbf{G}(D) = [1 + D + D^2 \quad 1 + D^2 \quad 1 + D]$. The Type I and Type II realizations for this generator matrix are presented in Figures 4.7(a) and (b), which we observe have memories $\mu = 2$ and $\mu = 5$, respectively. Noting that

$$\mathbf{G}_{\text{sys}}(D) = \begin{bmatrix} 1 & \dfrac{1 + D^2}{1 + D + D^2} & \dfrac{1 + D}{1 + D + D^2} \end{bmatrix},$$

the Type I and II realizations for this systematic generator matrix are presented in Figures 4.7(c) and (d), which we observe have memories $\mu = 2$ and $\mu = 4$, respectively.

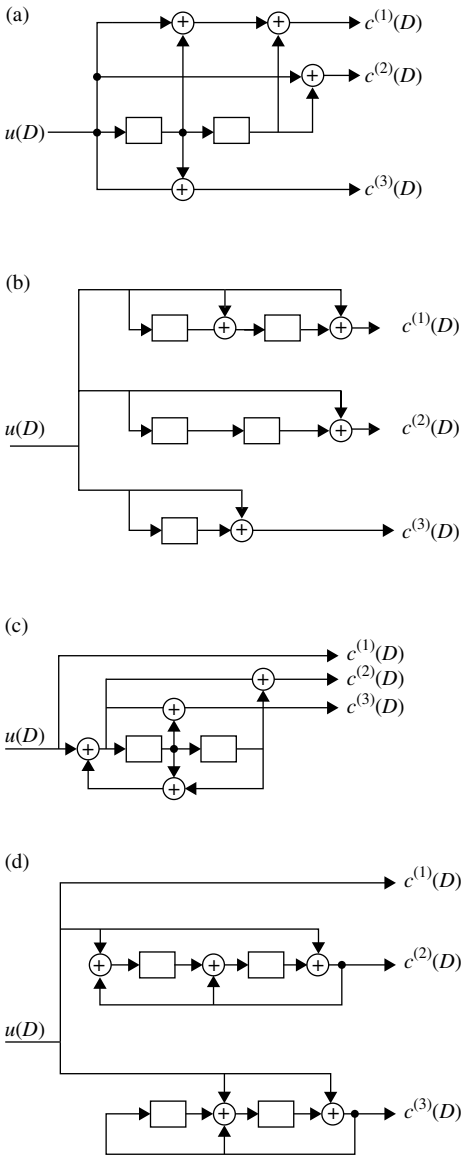


Figure 4.7 Encoder realizations for the rate-1/3 code of Example 4.4. (a) Type I $\bar{\text{R}}\bar{\text{S}}$C encoder. (b) Type II $\bar{\text{R}}\bar{\text{S}}$C encoder. (c) Type I RSC encoder. (d) Type II RSC encoder.

Note that the Type I realization applied to $\mathbf{G}_{\text{sys}}(D)$ yields a minimal encoder as claimed above. Note also that the Type I realization applied to $\mathbf{G}(D)$, after we have divided out the gcd from its entries, also yields a minimal encoder.

Next we consider a rate-$k/(k+1)$ code whose encoder is represented in polynomial form by

$$\mathbf{G}_{\text{poly}}(D) = \left[g_i^{(j)}(D)\right],$$

or in systematic form by

$$\mathbf{G}_{\text{sys}}(D) = \begin{bmatrix} & h^{(k)}(D)/h^{(0)}(D) \\ \mathbf{I}_k & h^{(k-1)}(D)/h^{(0)}(D) \\ & \vdots \\ & h^{(1)}(D)/h^{(0)}(D) \end{bmatrix}, \tag{4.4}$$

where $h^{(j)}(D)\big|_{j\neq 0}$ and $h^{(0)}(D)$ are relatively prime polynomials. We may immediately write, from (4.4),

$$\mathbf{H}_{\text{sys}}(D) = \begin{bmatrix} h^{(k)}(D)/h^{(0)}(D) & h^{(k-1)}(D)/h^{(0)}(D) & \dots & h^{(1)}(D)/h^{(0)}(D) & 1 \end{bmatrix} \tag{4.5}$$

and

$$\mathbf{H}_{\text{poly}}(D) = \begin{bmatrix} h^{(k)}(D) & h^{(k-1)}(D) & \dots & h^{(1)}(D) & h^{(0)}(D) \end{bmatrix}. \tag{4.6}$$

We now argue that the Type II realization of $\mathbf{G}_{\text{sys}}(D)$ as given by (4.4) is a minimal encoder since the rational functions in $\mathbf{G}_{\text{sys}}(D)$ are in lowest terms and the Type II realization is naturally suited to rate $k/(k+1)$. Note that the encoder realization is completely specified by the $k+1$ *parity-check polynomials* $\left\{h^{(j)}(D)\right\}_{j=0}^{k}$ and $\gcd\left\{h^{(0)}(D), h^{(1)}(D), \dots, h^{(k)}(D)\right\} = 1$ under the relatively prime assumption and the delay-free encoder assumption. The Type II realization is naturally suited to rate-$k/(k+1)$ codes in the systematic format in (4.4) because the single parity bit may be computed by "tapping into" the same length-μ shift-register at the appropriate adders. Thus, the Type II circuit of Figure 4.3 may be considered to be a multiple-input/single-output device as is necessary when computing a single parity bit from k input bits. On the other hand, the Type I realization is particularly ill-suited as an encoder realization of $\mathbf{G}_{\text{sys}}(D)$. In summary, a minimal encoder for a rate-$k/(k+1)$ convolutional code is obtained by applying the Type II encoder realization to $\mathbf{G}_{\text{sys}}(D)$ or, equivalently, to $\mathbf{H}_{\text{poly}}(D)$ with the gcd factored out of its entries.

Example 4.5. Returning to the rate-2/3 code of Example 4.2, we have

$$\mathbf{G}_{\text{poly}}(D) = \begin{bmatrix} (1+D)^3 & 0 & (1+D+D^2)(1+D)^3 \\ (1+D+D^2)(1+D^2) & (1+D+D^2)(1+D) & D(1+D+D^2) \end{bmatrix},$$

$$\mathbf{G}_{\text{sys}}(D) = \begin{bmatrix} 1 & 0 & \dfrac{1+D^3}{1+D} \\ 0 & 1 & \dfrac{1+D^3+D^4}{1+D} \end{bmatrix},$$

$$\mathbf{H}_{\text{sys}}(D) = \begin{bmatrix} \dfrac{1+D^3}{1+D} & \dfrac{1+D^3+D^4}{1+D} & 1 \end{bmatrix},$$

$$\mathbf{H}_{\text{poly}}(D) = \begin{bmatrix} 1+D^3 & 1+D^3+D^4 & 1+D \end{bmatrix}.$$

Note that the element $1+D+D^2$ in $\mathbf{G}_{\text{sys}}(D)$ and $\mathbf{H}_{\text{sys}}(D)$ in Example 4.2 has been adjusted to $(1+D^3)/(1+D)$ so that the expressions above conform to the forms given in (4.4), (4.5), and (4.6). Clearly, $h^{(0)}(D) = 1+D$. The Type II implementation of $\mathbf{G}_{\text{sys}}(D)$, which gives the minimal encoder, is depicted in Figure 4.6, where a memory size of $\mu = 4$ is evident. The Type I implementation of $\mathbf{G}_{\text{poly}}(D)$ requires $\mu = 9$ when a length-5 shift-register is used to implement the top row of $\mathbf{G}_{\text{poly}}(D)$ and a length-4 shift-register is used to implement the bottom row of $\mathbf{G}_{\text{poly}}(D)$. The shift-registers in each case are shared by the elements in a row, with appropriate taps connecting a binary adder to the shift-register. The Type II implementation of $\mathbf{G}_{\text{poly}}(D)$ requires $\mu = 9$ when circuit sharing is utilized: one circuit is used to implement the factor $(1+D)^3$ in the first row of $\mathbf{G}_{\text{poly}}(D)$ and one circuit is used to implement the factor $(1+D+D^2)$ in the second row of $\mathbf{G}_{\text{poly}}(D)$. The Type II implementation of $\mathbf{G}_{\text{sys}}(D)$ requires $\mu = 6$, which is achieved when the element $(1+D^3)/(1+D)$ in $\mathbf{G}_{\text{sys}}(D)$ is implemented as $1+D+D^2$.

In the foregoing discussion of minimal encoder realizations for rate-$1/n$ and rate-$k/(k+1)$ convolutional codes, we saw that the first step was to put the generator matrix in systematic form. The minimal realization for the rate-$1/n$ case is then achieved by the Type I form and that for the rate-$k/(k+1)$ case is achieved by the Type II form. For the general rate-k/n convolutional code, where k/n is neither $1/n$ nor $k/(k+1)$, the first step is again to put the generator matrix in systematic form. However, the minimal realization of this generator matrix may be neither Type I nor Type II. This topic is beyond our scope (see [3] for a comprehensive treatment), but we have the following remarks: (1) in practice, only rate-$1/n$ and rate-$k/(k+1)$ are used, and tables of these codes appear in minimal form; (2) in the rare event that a rate-k/n convolutional code is required, one may use standard sequential-circuit minimization techniques to achieve the minimal encoder realization.

4.3.4 Design of Convolutional Codes

Regarding the fourth question listed in Section 4.3, for reasons affecting decoder complexity, typical values of n and k are very small: $k = 1$, 2, 3 and $n = 2$, 3, 4. Typical code rates are 1/2, 1/3, 1/4, 2/3, and 3/4, with 1/2 by far the most common rate. Higher code rates are generally achieved by *puncturing* a lower-rate code, which means that code bits at the encoder output are periodically deleted and hence not transmitted. For example, to achieve a rate-2/3 convolutional code by puncturing a rate-1/2 encoder, one need only puncture every fourth code bit at the encoder output. Since each group of four code bits is produced by two information bits, but only three of the four will be sent, rate 2/3 has been achieved. Convolutional code rates close to unity (e.g., 0.95) are not viable in practice, with or without puncturing, because of the implications for encoder and decoder complexity as well as performance. Tables of puncturing patterns may be found in references [3, 8, 9].

As mentioned briefly in Example 4.3, the complexity of the (optimum) convolutional decoder (see Section 4.5) is proportional to 2^{μ}, where μ is the encoder memory. Thus, typical values of μ are less than 10, with $\mu = 6$ corresponding to the industry-standard rate-1/2 convolutional code, which has generator polynomials $g^{(1)}(D) = 1 + D + D^2 + D^3 + D^6$ and $g^{(2)}(D) = 1 + D^2 + D^3 + D^5 + D^6$. It is conventional to represent these polynomials as octal numbers, where the coefficients $g_0^{(1)}$ and $g_0^{(2)}$ of D^0 correspond to the least-significant bit of the octal numbers. Thus, in octal notation, these generator polynomials are $\mathbf{g}_{\text{octal}}^{(1)} = 117$ and $\mathbf{g}_{\text{octal}}^{(2)} = 155$.

While convolutional codes possess very nice algebraic structure, no satisfactory algebraic design algorithm exists for designing convolutional codes. Code design has been performed in the past via computer search. In the computer-aided-design approach, one usually specifies n, k, and μ, and chooses a (recursive) systematic encoder realization to ensure that the code is non-catastrophic. Then the computer algorithm runs through an exhaustive list of encoder connections (polynomials), each time examining the resulting code for its distance spectrum. The distance spectra ($d_{\min}$ and beyond, plus multiplicities) are determined via a Viterbi-like algorithm (see Section 4.5 for a discussion of the Viterbi algorithm).

4.4 Alternative Convolutional Code Representations

In the previous two sections we favored an algebraic description of convolutional codes, which we have seen is very useful in terms of encoder realization and classification. Under this section, we consider alternative representations of convolutional codes that are useful for many other aspects of convolutional codes, such as decoding, code design, and analysis.

4.4.1 Convolutional Codes as Semi-Infinite Linear Codes

In our algebraic description of a convolutional code, the information word $\mathbf{u}(D) = [u^{(1)}(D) \quad u^{(2)}(D) \quad \ldots \quad u^{(k)}(D)]$ was mapped to a codeword $\mathbf{c}(D) = [c^{(1)}(D) \quad c^{(2)}(D) \quad \ldots \quad c^{(n)}(D)]$ via a generator matrix (or encoder) $\mathbf{G}(D)$, where $u^{(i)}(D)$ and $c^{(j)}(D)$ are elements of the field $\mathbb{F}_2\langle D\rangle$ and the entries $g_i^{(j)}(D)$ of $\mathbf{G}(D)$ are elements of the field $\mathbb{F}_2(D)$ or the ring $\mathbb{F}_2[D]$. It is often useful to take a more system-theoretic view and represent $u^{(i)}(D)$, $c^{(j)}(D)$, and $g_i^{(j)}(D)$ by (generally semi-infinite) vectors of their coefficients, denoted by $\mathbf{u}^{(i)}$, $\mathbf{c}^{(j)}$, and $\mathbf{g}_i^{(j)}$, respectively. Thus, the vector representation of $u^{(i)}(D) = u_0^{(i)} + u_1^{(i)}D + u_2^{(i)}D^2 + \ldots$, is $\mathbf{u}^{(i)} = \begin{bmatrix} u_0^{(i)} & u_1^{(i)} & u_2^{(i)} & \ldots \end{bmatrix}$, and similarly for $\mathbf{c}^{(j)}$ and $\mathbf{g}_i^{(j)}$. Under this vector notation, we may write $\mathbf{u} = [\mathbf{u}^{(1)} \quad \mathbf{u}^{(2)} \quad \ldots \quad \mathbf{u}^{(k)}]$ and $\mathbf{c} = [\mathbf{c}^{(1)} \quad \mathbf{c}^{(2)} \quad \ldots \quad \mathbf{c}^{(n)}]$. We would like to determine a generator matrix that is a function of $\left\{\mathbf{g}_i^{(j)}\right\}$.

Since multiplication in the D domain is convolution in the coefficient domain, we may replace $c^{(j)}(D) = \sum_{i=1}^{k} u^{(i)}(D) g_i^{(j)}(D)$ by

$$\mathbf{c}^{(j)} = \sum_{i=1}^{k} \mathbf{u}^{(i)} \circledast \mathbf{g}_i^{(j)}. \tag{4.7}$$

Given this, one may loosely write $\mathbf{c} = \mathbf{u} \circledast \mathbf{G}$ for some generator matrix $\mathbf{G}$ depending on $\left\{\mathbf{g}_i^{(j)}\right\}$. However, this notation is awkward and not too useful. We can replace the convolution operation in (4.7) with a multiplication operation by employing a matrix that explicitly performs the convolution. Specifically, we have

$$\begin{aligned}
\mathbf{u}^{(i)} \circledast \mathbf{g}_i^{(j)} &= \begin{bmatrix} u_0^{(i)} & u_1^{(i)} & u_2^{(i)} & \ldots \end{bmatrix} \begin{bmatrix} g_{i,0}^{(j)} & g_{i,1}^{(j)} & g_{i,2}^{(j)} & \cdots & & \\ & g_{i,0}^{(j)} & g_{i,1}^{(j)} & g_{i,2}^{(j)} & \cdots & \\ & & g_{i,0}^{(j)} & g_{i,1}^{(j)} & g_{i,2}^{(j)} & \cdots \\ & & & \ddots & \ddots & \ddots \end{bmatrix} \\
&= \mathbf{u}^{(i)} \mathbf{G}_i^{(j)},
\end{aligned}$$

where $\mathbf{u}^{(i)} = \begin{bmatrix} u_0^{(i)} & u_1^{(i)} & u_2^{(i)} & \ldots \end{bmatrix}$, $\mathbf{g}_i^{(j)} = \begin{bmatrix} g_{i,0}^{(j)} & g_{i,1}^{(j)} & g_{i,2}^{(j)} & \ldots \end{bmatrix}$, and

$$\mathbf{G}_i^{(j)} = \begin{bmatrix} g_{i,0}^{(j)} & g_{i,1}^{(j)} & g_{i,2}^{(j)} & \cdots & & \\ & g_{i,0}^{(j)} & g_{i,1}^{(j)} & g_{i,2}^{(j)} & \cdots & \\ & & g_{i,0}^{(j)} & g_{i,1}^{(j)} & g_{i,2}^{(j)} & \cdots \\ & & & \ddots & \ddots & \ddots \end{bmatrix}.$$

We may then replace (4.7) by

$$\begin{aligned}\mathbf{c}^{(j)} &= \sum_{i=1}^{k} \mathbf{u}^{(i)} \mathbf{G}_i^{(j)} \\ &= \mathbf{u}\mathbf{G}^{(j)},\end{aligned} \tag{4.8}$$

where

$$\mathbf{G}^{(j)} = \begin{bmatrix} \mathbf{G}_1^{(j)} \\ \mathbf{G}_2^{(j)} \\ \vdots \\ \mathbf{G}_k^{(j)} \end{bmatrix}.$$

Finally, we have that encoding in the coefficient domain may be represented by the product

$$\mathbf{c} = \mathbf{u}\mathbf{G}, \tag{4.9}$$

where

$$\mathbf{G} = \begin{bmatrix} \mathbf{G}_1^{(1)} & \mathbf{G}_1^{(2)} & \cdots & \mathbf{G}_1^{(n)} \\ \mathbf{G}_2^{(1)} & \mathbf{G}_2^{(2)} & \cdots & \mathbf{G}_2^{(n)} \\ \vdots & \vdots & \cdots & \vdots \\ \mathbf{G}_k^{(1)} & \mathbf{G}_k^{(2)} & \cdots & \mathbf{G}_k^{(n)} \end{bmatrix}. \tag{4.10}$$

Encoding via (4.9) puts the codeword into the unmultiplexed format $\mathbf{c} = [\mathbf{c}^{(1)} \quad \mathbf{c}^{(2)} \quad \ldots \quad \mathbf{c}^{(n)}]$, which is analogous to $\mathbf{c}(D) = [c^{(1)}(D) \quad c^{(2)}(D) \quad \ldots \quad c^{(n)}(D)]$. However, in practice, the convolutional encoder outputs are multiplexed in a round-robin fashion, so that one bit is taken from $c^{(1)}(D)$, then $c^{(2)}(D)$, ..., then $c^{(n)}(D)$, after which a second bit is taken from $c^{(1)}(D)$, and so on. This leads to the multiplexed codeword format

$$\mathbf{c}' = \begin{bmatrix} c_0^{(1)} & c_0^{(2)} & \ldots & c_0^{(n)} & c_1^{(1)} & c_1^{(2)} & \ldots & c_1^{(n)} & c_2^{(1)} & c_2^{(2)} & \ldots & c_2^{(n)} & \cdots \end{bmatrix}. \tag{4.11}$$

It will be convenient to have also the multiplexed information word format

$$\mathbf{u}' = \begin{bmatrix} u_0^{(1)} & u_0^{(2)} & \ldots & u_0^{(k)} & u_1^{(1)} & u_1^{(2)} & \ldots & u_1^{(k)} & u_2^{(1)} & u_2^{(2)} & \ldots & u_2^{(k)} & \cdots \end{bmatrix}. \tag{4.12}$$

We may then write

$$\mathbf{c}' = \mathbf{u}'\mathbf{G}',$$

where $\mathbf{G}'$, also called a *generator matrix*, is given by

$$\mathbf{G}' = \begin{bmatrix} g_{1,0}^{(1)} & \cdots & g_{1,0}^{(n)} & g_{1,1}^{(1)} & \cdots & g_{1,1}^{(n)} & g_{1,2}^{(1)} & \cdots & g_{1,2}^{(n)} & \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \cdots \\ g_{k,0}^{(1)} & \cdots & g_{k,0}^{(n)} & g_{k,1}^{(1)} & \cdots & g_{k,1}^{(n)} & g_{k,2}^{(1)} & \cdots & g_{k,2}^{(n)} & \\ & & & g_{1,0}^{(1)} & \cdots & g_{1,0}^{(n)} & g_{1,1}^{(1)} & \cdots & g_{1,1}^{(n)} & \\ & & & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \cdots \\ & & & g_{k,0}^{(1)} & \cdots & g_{k,0}^{(n)} & g_{k,1}^{(1)} & \cdots & g_{k,1}^{(n)} & \\ & & & & & & g_{1,0}^{(1)} & \cdots & g_{1,0}^{(n)} & \\ & & & & & & \vdots & \ddots & \vdots & \cdots \\ & & & & & & g_{k,0}^{(1)} & \cdots & g_{k,0}^{(n)} & \\ & & & & & & & & & \ddots \end{bmatrix} \tag{4.13}$$

or

$$\mathbf{G}' = \begin{bmatrix} \mathbf{G}_0' & \mathbf{G}_1' & \mathbf{G}_2' & \mathbf{G}_3' & \cdots \\ & \mathbf{G}_0' & \mathbf{G}_1' & \mathbf{G}_2' & \cdots \\ & & \mathbf{G}_0' & \mathbf{G}_1' & \cdots \\ & & & \mathbf{G}_0' & \cdots \\ & & & & \ddots \end{bmatrix}, \tag{4.14}$$

where

$$\mathbf{G}_l' = \begin{bmatrix} g_{1,l}^{(1)} & \cdots & g_{1,l}^{(n)} \\ \vdots & \ddots & \vdots \\ g_{k,l}^{(1)} & \cdots & g_{k,l}^{(n)} \end{bmatrix}.$$

Example 4.6. We are given the rate-1/2 convolutional code with generator matrix $\mathbf{G}(D) = [g^{(1)}(D) \quad g^{(2)}(D)] = [1 + D + D^2 \quad 1 + D^2]$ so that $\mathbf{g}^{(1)} = \begin{bmatrix} g_0^{(1)} & g_1^{(1)} & g_2^{(1)} \end{bmatrix} = [1\ 1\ 1]$ and $\mathbf{g}_2 = \begin{bmatrix} g_2^{(0)} & g_2^{(1)} & g_2^{(2)} \end{bmatrix} = [1\ 0\ 1]$. The generator matrix corresponding to the

multiplexed codeword is given by

$$\mathbf{G}' = \begin{bmatrix} g_0^{(1)} g_0^{(2)} & g_1^{(1)} g_1^{(2)} & g_2^{(1)} g_2^{(2)} & & \\ & g_0^{(1)} g_0^{(2)} & g_1^{(1)} g_1^{(2)} & g_2^{(1)} g_2^{(2)} & \\ & & g_0^{(1)} g_0^{(2)} & g_1^{(1)} g_1^{(2)} & g_2^{(1)} g_2^{(2)} \\ & & \ddots & \ddots & \ddots \end{bmatrix}$$

$$= \begin{bmatrix} 11 & 10 & 11 & & & \\ & 11 & 10 & 11 & & \\ & & 11 & 10 & 11 & \\ & & & \ddots & \ddots & \ddots \end{bmatrix}.$$

Consider a rate-2/3 convolutional code with

$$\mathbf{G}(D) = \begin{bmatrix} 1+D & 0 & 1 \\ 1+D^2 & 1+D & 1+D+D^2 \end{bmatrix}.$$

From this, we have $\mathbf{g}_1^{(1)} = \begin{bmatrix} g_{1,0}^{(1)} & g_{1,1}^{(1)} & g_{1,2}^{(1)} \end{bmatrix} = [1\ 1\ 0]$, $\mathbf{g}_2^{(1)} = \begin{bmatrix} g_{2,0}^{(1)} & g_{2,1}^{(1)} & g_{2,2}^{(1)} \end{bmatrix} = [1\ 0\ 1]$, $\mathbf{g}_1^{(2)} = [0\ 0\ 0]$, and so on, so that from (4.13)

$$\mathbf{G}' = \begin{bmatrix} 1\,0\,1 & 1\,0\,0 & 0\,0\,0 & & & \\ 1\,1\,1 & 0\,1\,1 & 1\,0\,1 & & & \\ & 1\,0\,1 & 1\,0\,0 & 0\,0\,0 & & \\ & 1\,1\,1 & 0\,1\,1 & 1\,0\,1 & & \\ & & 1\,0\,1 & 1\,0\,0 & \ddots & \\ & & 1\,1\,1 & 0\,1\,1 & & \\ & & & 1\,0\,1 & \ddots & \\ & & & 1\,1\,1 & & \\ & & & & \ddots & \end{bmatrix}.$$

It is even possible for the multiplexed generator-matrix representation to accommodate the recursive systematic encoder

$$\mathbf{G}_{\text{sys}}(D) = \begin{bmatrix} 1 & 0 & \dfrac{1}{1+D} \\ 0 & 1 & \dfrac{D^2}{1+D} \end{bmatrix}$$

of this code. In this case, the following is obvious: $\mathbf{g}_1^{(1)} = [1\ 0\ 0\ \ldots]$, $\mathbf{g}_2^{(1)} = [0\ 0\ 0\ \ldots]$, $\mathbf{g}_1^{(2)} = [0\ 0\ 0\ \ldots]$, and $\mathbf{g}_2^{(2)} = [1\ 0\ 0\ \ldots]$. Since $g_{13}(D) = 1/(1+D)$ has the sequence representation $111\ldots$ and $g_{23}(D) = D^2/(1+D)$ has the sequence representation $00111\ldots$,

it follows that $\mathbf{g}_1^{(3)} = [1\ 1\ 1\ \ldots]$ and $\mathbf{g}_2^{(3)} = [0\ 0\ 1\ 1\ 1\ \ldots]$. We then have

$$\mathbf{G}' = \begin{bmatrix} 1\,0\,1 & 0\,0\,1 & 0\,0\,1 & 0\,0\,1 & \cdots \\ 0\,1\,0 & 0\,0\,0 & 0\,0\,1 & 0\,0\,1 & \\ & 1\,0\,1 & 0\,0\,1 & 0\,0\,1 & \ddots \\ & 0\,1\,0 & 0\,0\,0 & 0\,0\,1 & \\ & & 1\,0\,1 & 0\,0\,1 & \ddots \\ & & 0\,1\,0 & 0\,0\,0 & \\ & & & 1\,0\,1 & \ddots \\ & & & 0\,1\,0 & \\ & & & & \ddots \end{bmatrix}.$$

In Example 4.6, we gave the form of $\mathbf{G}'$ for a specific rate-2/3 RSC encoder. The form of $\mathbf{G}'$ for the general systematic encoder of the form

$$\mathbf{G}_{\text{sys}}(D) = \begin{bmatrix} & g_1^{(n-k+1)}(D) & \cdots & g_1^{(n)}(D) \\ \mathbf{I} & \vdots & \ddots & \vdots \\ & g_k^{(n-k+1)}(D) & \cdots & g_k^{(n)}(D) \end{bmatrix},$$

where each $g_i^{(j)}(D)$ is a rational function in general, is given by

$$\mathbf{G}'_{\text{sys}} = \begin{bmatrix} \mathbf{I}\ \mathbf{P}_0 & \mathbf{O}\ \mathbf{P}_1 & \mathbf{O}\ \mathbf{P}_2 & \cdots & & & \\ & \mathbf{I}\ \mathbf{P}_0 & \mathbf{O}\ \mathbf{P}_1 & \mathbf{O}\ \mathbf{P}_2 & \cdots & & \\ & & \mathbf{I}\ \mathbf{P}_0 & \mathbf{O}\ \mathbf{P}_1 & \mathbf{O}\ \mathbf{P}_2 & \cdots & \\ & & \ddots & \ddots & \ddots & & \end{bmatrix},$$

where $\mathbf{I}$ is the $k \times k$ identity matrix, $\mathbf{O}$ is the $k \times k$ all-zero matrix, and $\mathbf{P}_l$ is the $k \times (n-k)$ matrix given by

$$\mathbf{P}_l = \begin{bmatrix} g_{1,l}^{(n-k+1)} & \cdots & g_{1,l}^{(n)} \\ \vdots & \ddots & \vdots \\ g_{k,l}^{(n-k+1)} & \cdots & g_{k,l}^{(n)} \end{bmatrix}.$$

Here, $g_{i,l}^{(j)}$ is the coefficient of D^l in the series expansion of $g_i^{(j)}(D)$.

It is easy to check that the corresponding parity-check matrix $\mathbf{H}'_{\text{sys}}$ is given by

$$\mathbf{H}'_{\text{sys}} = \begin{bmatrix} \mathbf{P}_0^{\mathrm{T}}\ \mathbf{I} & & & & \\ \mathbf{P}_1^{\mathrm{T}}\ \mathbf{O} & \mathbf{P}_0^{\mathrm{T}}\ \mathbf{I} & & & \\ \mathbf{P}_2^{\mathrm{T}}\ \mathbf{O} & \mathbf{P}_1^{\mathrm{T}}\ \mathbf{O} & \mathbf{P}_0^{\mathrm{T}}\ \mathbf{I} & & \\ \mathbf{P}_3^{\mathrm{T}}\ \mathbf{O} & \mathbf{P}_2^{\mathrm{T}}\ \mathbf{O} & \mathbf{P}_1^{\mathrm{T}}\ \mathbf{O} & \mathbf{P}_0^{\mathrm{T}}\ \mathbf{I} & \\ \vdots & \ddots & \ddots & \ddots & \ddots \end{bmatrix}. \tag{4.15}$$

Unless the code rate is 1/2, finding $\mathbf{H}'$ for the non-systematic case is non-trivial. For the rate-1/2 non-systematic case for which $\mathbf{G}(D) = [g^{(1)}(D) \quad g^{(2)}(D)]$, clearly $\mathbf{H}(D) = [g^{(2)}(D) \quad g^{(1)}(D)]$ since $\mathbf{G}(D)\mathbf{H}^{\mathrm{T}}(D) = 0$ in this case. The non-multiplexed parity-check matrix is then

$$\mathbf{H} = \begin{bmatrix} \mathbf{G}_2 & \mathbf{G}_1 \end{bmatrix}$$

(so that $\mathbf{G}\mathbf{H}^{\mathrm{T}} = \mathbf{0}$), where, for $j = 1, 2$,

$$\mathbf{G}^{(j)} = \begin{bmatrix} g_0^{(j)} & g_1^{(j)} & g_2^{(j)} & \cdots & & & \\ & g_0^{(j)} & g_1^{(j)} & g_2^{(j)} & \cdots & & \\ & & g_0^{(j)} & g_1^{(j)} & g_2^{(j)} & \cdots & \\ & & \ddots & \ddots & \ddots & \ddots & \end{bmatrix}.$$

The multiplexed version of the parity-check matrix is then

$$\mathbf{H}' = \begin{bmatrix} \mathbf{H}'_0 & \mathbf{H}'_1 & \mathbf{H}'_2 & \mathbf{H}'_3 & \cdots \\ & \mathbf{H}'_0 & \mathbf{H}'_1 & \mathbf{H}'_2 & \cdots \\ & & \mathbf{H}'_0 & \mathbf{H}'_1 & \cdots \\ & & & \mathbf{H}'_0 & \cdots \\ & & & & \ddots \end{bmatrix}, \tag{4.16}$$

where

$$\mathbf{H}'_l = \begin{bmatrix} g_l^{(2)} & g_l^{(1)} \end{bmatrix}.$$

For the general non-systematic case, i.e., for code rates other than 1/2, one usually puts $\mathbf{G}(D)$, and hence $\mathbf{G}$, in systematic form, from which the parity-check matrix $\mathbf{H}'_{\mathrm{sys}}$ given by (4.15) follows.

Example 4.7. Consider again the rate-1/2 convolutional code with generator matrix $\mathbf{G}(D) = [1 + D + D^2 \quad 1 + D^2]$. We have $\mathbf{g}^{(1)} = \begin{bmatrix} g_0^{(1)} & g_1^{(1)} & g_2^{(1)} \end{bmatrix} = [1\ 1\ 1]$ and $\mathbf{g}^{(2)} = \begin{bmatrix} g_0^{(2)} & g_1^{(2)} & g_2^{(2)} \end{bmatrix} = [1\ 0\ 1]$, so that

$$\mathbf{H} = \begin{bmatrix} 101 & & & & 111 & & & \\ & 101 & & & & 111 & & \\ & & 101 & & & & 111 & \\ & & & \ddots & & & & \ddots \end{bmatrix}$$

and

$$\mathbf{H}' = \begin{bmatrix} 11 & 01 & 11 & & & \\ & 11 & 01 & 11 & & \\ & & 11 & 01 & 11 & \\ & & & \ddots & \ddots & \ddots \end{bmatrix}.$$

4.4.2 Graphical Representations for Convolutional Code Encoders

There exist several graphical representations for the encoders of convolutional codes, with each of these representations playing different roles. We find it useful, and sufficient, to describe the graphical models for our archetypal encoder, specifically, the rate-1/2 encoder described by $\mathbf{G}(D) = [1 + D + D^2 \quad 1 + D^2]$. The procedure for deriving graphical models for other convolutional codes is identical.

We start with the *finite-state transition-diagram* (FSTD), or *state-diagram*, graphical model for $\mathbf{G}(D)$. The encoder realization for $\mathbf{G}(D)$ was presented earlier in Figure 4.4(a) and the *encoder state* is defined to be the contents of the two memory elements in the encoder circuit (read from left to right). From that figure and the state definition, we may produce the following state-transition table from which the code's FSTD may be easily drawn, as depicted in Figure 4.8(a).

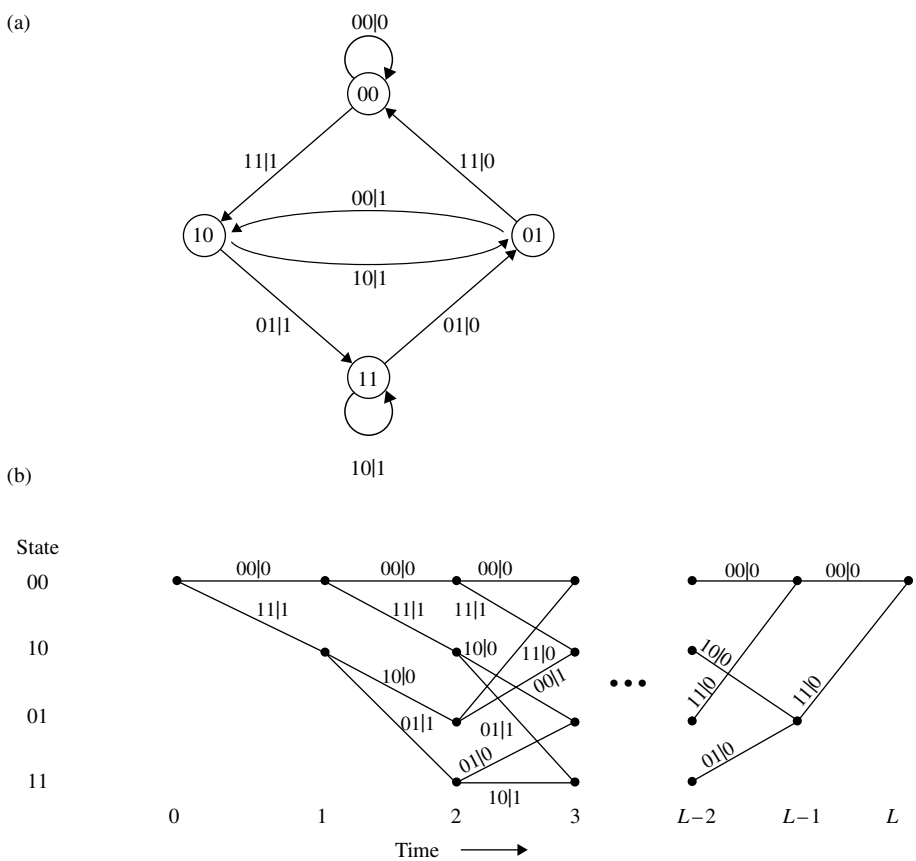


Figure 4.8 (a) FSTD for the encoder of Figure 4.4(a). The edge labels are $c_1(i)c_2(i)|u(i)$ or output|input. (b) A trellis diagram for the encoder of Figure 4.4(a) for input length L and input containing two terminating zeros.

The state diagram is useful for analytically determining the distance spectrum of a convolutional code [8, 9] and it also leads directly to the code's *trellis diagram,* which is simply the state diagram with the dimension of discrete time added in the horizontal direction. The trellis diagram for our example code is presented in Figure 4.8(b); it has been drawn under the assumption that the encoder starts in the 00 state, as is usually the case in practice. Note that the trellis stages between times i and $i+1$, $i \geq 2$, are essentially exact replicas of the state diagram. We remark that a list of code sequences of any given length may be obtained from the trellis by tracing through all possible trellis paths corresponding to that length, along the way picking off the code symbols which label the *branches* (or *edges*) within each path:

Input u_i	Current state $u_{i-1}u_{i-2}$	Next state u_iu_{i-1}	Output $c_i^{(1)}c_i^{(2)}$
0	00	00	00
1	00	10	11
0	01	00	11
1	01	10	00
0	10	01	10
1	10	11	01
0	11	01	01
1	11	11	10

When we consider trellis-based decoding in the next subsection, we will see that, unless the final trellis state is known, the last several decoded bits (about 5μ bits) will be somewhat unreliable. To counter this circumstance, the system designer will frequently *terminate* the trellis to a known state. Clearly this necessitates the appending of μ bits to the information sequence, resulting in an increase in coding overhead (or, equivalently, a decrease in code rate). A clever way to circumvent this inefficiency while still maintaining the advantage of trellis termination is via *tail-biting.* The encoder of a tail-biting convolutional code assumes the $\bar{\text{R}}$SC or $\bar{\text{R}}\bar{\text{S}}$C forms and initializes the encoder state with the final μ bits in the information sequence so that the initial and the final encoder state are identical. Note that this has the effect of rendering both the initial and the final encoder state unknown (albeit identical). However, with a little increase in decoder complexity, the decoder can be made to be near optimal and, in particular, the reliability of all of the decoded bits is made more uniform.

4.5 Trellis-Based Decoders

There exist several decoders for convolutional codes, including the majority-logic decoder, the sequential decoder, the Viterbi decoder, and the BCJR decoder [8]. The choice of algorithm depends, of course, on the application. Because of their

relative importance in the field of convolutional and turbo codes, we discuss only two convolutional decoders. The first is the maximum-likelihood sequence decoder (MLSD) which is implemented via the Viterbi algorithm [4]. The MLSD minimizes the probability of code-sequence error. The second is the bit-wise maximum *a posteriori* (MAP) decoder which is implemented via the BCJR algorithm [6]. We will often call the bit-wise MAP decoder simply the MAP decoder. The MAP decoder minimizes the probability of information-bit error. Generally, the performance characteristics of the two decoders, as measured either in bit error rate or in codeword error rate, are very similar. For this reason and the fact that the BCJR is approximately three times as complex as the Viterbi algorithm, the BCJR algorithm was ignored for many years. However, the BCJR algorithm was resurrected as a necessary ingredient to a turbo decoder.

4.5.1 MLSD and the Viterbi Algorithm

Our focus is the binary symmetric channel (BSC) and the binary-input AWGN channel (BI-AWGNC). For the sake of uniformity, for both channels, we denote the ith channel input by x_i and the ith channel output by y_i. Given channel input $x_i = c_i \in \{0, 1\}$ and channel output $y_i \in \{0, 1\}$, the BSC has channel transition probabilities $P(y_i|x_i)$ given by

$$\begin{aligned} P(y_i \neq x|x_i = x) &= \varepsilon, \\ P(y_i = x|x_i = x) &= 1 - \varepsilon, \end{aligned}$$

where ε is the crossover probability. (For simplicity of notation, we renumber the code bits in (4.11) so that the superscripts may be dropped.) For the BI-AWGNC, the code bits are mapped to the channel inputs as $x_i = (-1)^{c_i} \in \{\pm 1\}$. The BI-AWGNC has the channel transition probability density function (pdf) $p(y_i|x_i)$ given by

$$p(y_i|x_i) = \frac{1}{\sqrt{2\pi}\sigma} \exp\Big[-(y_i - x_i)^2 / (2\sigma^2)\Big],$$

where σ^2 is the variance of the zero-mean Gaussian-noise sample n_i that the channel adds to the transmitted value x_i (so that $y_i = x_i + n_i$). Considering first the BSC, the ML decision is

$$\hat{\mathbf{c}} = \arg\max_{\mathbf{c}} P(\mathbf{y}|\mathbf{x}),$$

which was shown in Chapter 1 to reduce to

$$\hat{\mathbf{c}} = \arg\min_{\mathbf{c}} d_{\mathrm{H}}(\mathbf{y}, \mathbf{x}),$$

where $d_{\mathrm{H}}(\mathbf{y}, \mathbf{x})$ represents the Hamming distance between $\mathbf{y}$ and $\mathbf{x}$ (note that $\mathbf{x} = \mathbf{c}$). For the BI-AWGNC, the ML decision is

$$\hat{\mathbf{c}} = \arg\max_{\mathbf{c}} p(\mathbf{y}|\mathbf{x}),$$

which was shown in Chapter 1 to reduce to

$$\hat{\mathbf{c}} = \arg\min_{\mathbf{c}} d_{\mathrm{E}}(\mathbf{y}, \mathbf{x}),$$

where $d_{\mathrm{E}}(\mathbf{y}, \mathbf{x})$ represents the Euclidean distance between $\mathbf{y}$ and $\mathbf{x}$ (note that $\mathbf{x} = (-1)^{\mathbf{c}}$).

Thus, for the BSC (BI-AWGNC), the MLSD chooses the code sequence $\mathbf{c}$ that is closest to the channel output $\mathbf{y}$ in a Hamming (Euclidean) distance sense. In principle, one could use the trellis to exhaustively search for the sequence that is closest to $\mathbf{y}$, since the trellis enumerates all of the code sequences. The computation for L trellis branches would be

$$\Gamma_L = \sum_{l=1}^{L} \lambda_l, \tag{4.17}$$

where λ_l is the lth *branch metric*, given by

$$\lambda_l = \sum_{j=1}^{n} d_{\mathrm{H}}\left(y_l^{(j)}, x_l^{(j)}\right) = \sum_{j=1}^{n} y_l^{(j)} \oplus x_l^{(j)} \qquad \text{for the BSC,} \tag{4.18}$$

$$\lambda_l = \sum_{j=1}^{n} d_{\mathrm{E}}^2\left(y_l^{(j)}, x_l^{(j)}\right) = \sum_{j=1}^{n} \left(y_l^{(j)} - x_l^{(j)}\right)^2 \qquad \text{for the BI-AWGNC,} \tag{4.19}$$

where we have returned to the notation of (4.11) so that $x_l^{(j)} = c_l^{(j)}$ for the BSC and $x_l^{(j)} = (-1)^{c_l^{(j)}}$ for the BI-AWGNC. The metric in (4.18) is called a *Hamming metric* and the metric in (4.19) is called a *Euclidean metric*.

Clearly, the number of computations implied by $\arg\min_{\mathbf{c}} d_{\mathrm{H}}(\mathbf{y}, \mathbf{x})$ and $\arg\min_{\mathbf{c}} d_{\mathrm{E}}(\mathbf{y}, \mathbf{x})$ is enormous, since L trellis stages imply 2^{kL} code sequences for a rate-$k/(k+1)$ code. However, Viterbi [4] noticed that the complexity can be reduced to $O(2^{\mu})$ with no loss in optimality under the following observation. Consider the two code sequences in Figure 4.8(b) which begin with [00 00 00 ...] and [11 10 11 ...], corresponding to input sequences [0 0 0 ...] and [1 0 0 ...]. Observe that the trellis paths corresponding to these two sequences diverge at state 0/time 0 and remerge at state 0 after three branches (equivalently, after three trellis stages). The significance of this remerging after three stages is that thereafter the two paths have identical extensions into the latter trellis stages. Thus, if one of these two paths possesses the superior *cumulative metric* Γ_l at time $l = 3$, then that path plus its extensions will also possess superior cumulative metrics. Consequently, we may remove the inferior path from further consideration at time $l = 3$. The path that is maintained is called the *survivor*. This argument applies to all merging paths in the trellis and to trellises with more than two merging paths per trellis node at times $l > 3$, and it leads us to the *Viterbi-algorithm* implementation of the MLSD.

Algorithm 4.1 The Viterbi Algorithm

Definitions

1. $\lambda_l(s', s)$ = branch metric from state s' at time $l-1$ to state s at time l, where $s', s \in \{0, 1, \ldots, 2^\mu - 1\}$.
2. $\Gamma_{l-1}(s')$ = cumulative metric for the survivor state s' at time $l-1$; the sum of branch metrics for the surviving path.
3. $\Gamma_l(s', s)$ = tentative cumulative metric for the paths extending from state s' at time $l-1$ to state s at time l; $\Gamma_l(s', s) = \Gamma_{l-1}(s') + \lambda_l(s', s)$.

Add–Compare–Select Iteration

Initialize. Set $\Gamma_0(0) = 0$ and $\Gamma_0(s') = -\infty$ for all $s' \in \{1, \ldots, 2^\mu - 1\}$. (The encoder, and hence the trellis, is initialized to state 0.)

for $l = 1$ to L

1. Compute the possible branch metrics $\lambda_l(s', s)$.
2. For each state s' at time $l-1$ and all possible states s at time l that may be reached from s', compute the tentative cumulative metrics $\Gamma_l(s', s) = \Gamma_{l-1}(s') + \lambda_l(s', s)$ for the paths extending from state s' to state s.
3. For each state s at time l, select and store the path possessing the minimum among the metrics $\Gamma_l(s', s)$. The cumulative metric for state s will be $\Gamma_l(s) = \min_{s'}\{\Gamma_l(s', s)\}$. (See Figure 4.9 for a depiction of Steps 2 and 3.)

end

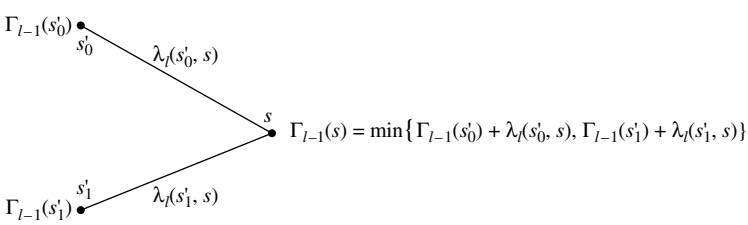


Figure 4.9 Depiction of Steps 2 and 3 of the ACS operation of the Viterbi algorithm for a rate-$1/n$ code and a distance metric. For a rate-$k/(k+1)$ code, 2^k branches merge into each state. For the correlation metric discussed below, "min" becomes "max."

Decision Stage

There are several ways to choose the ML code sequence (hence, ML information sequence) and we list a few here.

Block-oriented approach 1. Assuming an information sequence kL bits in length, after L add–compare–select (ACS) iterations (equivalently, L trellis stages), choose the trellis path with the best cumulative metric (ties are decided arbitrarily). L here must be relatively large to realize the full strength of the code, say, $L > 50\mu$.

Block-oriented approach 2. Assuming a non-recursive convolutional code, append μ zeros to the information sequences so that the trellis is forced to the zero state at the end of the information sequence. After the final ACS iteration, the ML trellis path will be the survivor at state zero. Again, L should be relatively large.

Stream-oriented approach 1. This approach exploits the fact that survivors tend to have a common "tail," as demonstrated in Figure 4.10. After the lth ACS iteration, the decoder traces back δ branches along an arbitrary surviving path and produces as its output decision the information bit(s) that label the branch at the $(l - \delta)$th trellis stage. Thus, this is effectively a *sliding-window decoder* that stores and processes information within δ trellis stages as it slides down the length of the trellis. The value of the *decoding delay* δ used is usually quoted to be four or five times the memory size. However, it is better to employ computer simulations to determine this value since it depends on the code. Alternatively, one can employ a computer search algorithm over the code's trellis to determine the maximum path length among the minimum-weight code sequences that diverge from the all-zeros path at state 0/time 0 and remerge later. δ would be set to a value greater than this maximum path length.

Stream-oriented approach 2. After the lth ACS iteration, choose the trellis path with the best cumulative metric and trace back δ branches. The corresponding output decision is selected as the bit(s) that labels the branch at the $(l - \delta)$th trellis stage.

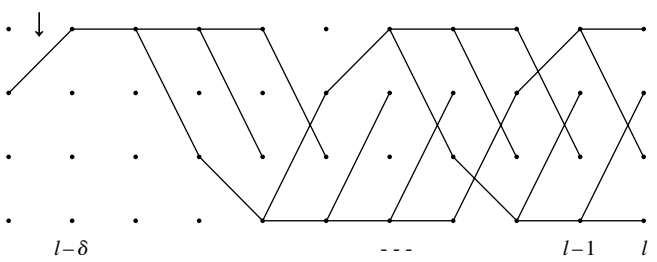


Figure 4.10 An example of survivors sharing a common tail in a four-state trellis. Because of this tail, a decision may be made for the $(l - \delta)$th trellis stage, in accordance with the label on the trellis branch indicated by the arrow.

The Euclidean distance metric given in (4.19) for the BI-AWGNC can be simplified as follows. Observe that

$$\begin{aligned}
\arg\min_{\mathbf{c}} d_{\mathrm{E}}^2(\mathbf{y},\mathbf{x}) &= \arg\min_{\mathbf{c}} \sum_{l=1}^{L}\sum_{j=1}^{n}\left(y_l^{(j)} - x_l^{(j)}\right)^2 \\
&= \arg\min_{\mathbf{c}} \sum_{l=1}^{L}\sum_{j=1}^{n}\left[\left(y_l^{(j)}\right)^2 + \left(x_l^{(j)}\right)^2 - 2y_l^{(j)}x_l^{(j)}\right] \\
&= \arg\max_{\mathbf{c}} \sum_{l=1}^{L}\sum_{j=1}^{n} y_l^{(j)}x_l^{(j)},
\end{aligned}$$

where the last line follows since the squared terms on the second line are independent of $\mathbf{c}$. Thus, the AWGN branch metric in (4.19) can be replaced by the *correlation metric*

$$\lambda_l = \sum_{j=1}^{n} y_l^{(j)}x_l^{(j)} \tag{4.20}$$

under the BI-AWGNC assumption.

Example 4.8. Figure 4.11 presents a Viterbi decoding example on the BSC using block-oriented decision approach 1 on the BSC. Figure 4.12 presents a Viterbi decoding example on the BI-AWGNC using block-oriented decision approach 1 and the correlation metric.

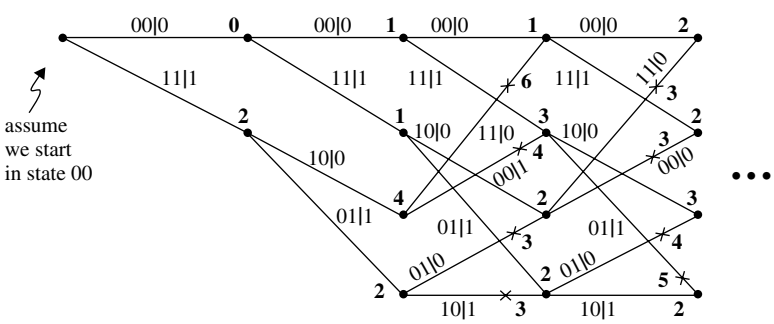


Figure 4.11 A Viterbi decoding example for the code of Figure 4.8 on the BSC. Here $\mathbf{y} = [00, 01, 00, 01]$. Non-surviving paths are indicated by an "X" and cumulative metrics are written in bold near merging branches. There is no outright minimum-distance path in this example. Any of the paths of distance 2 will do as the ML path.

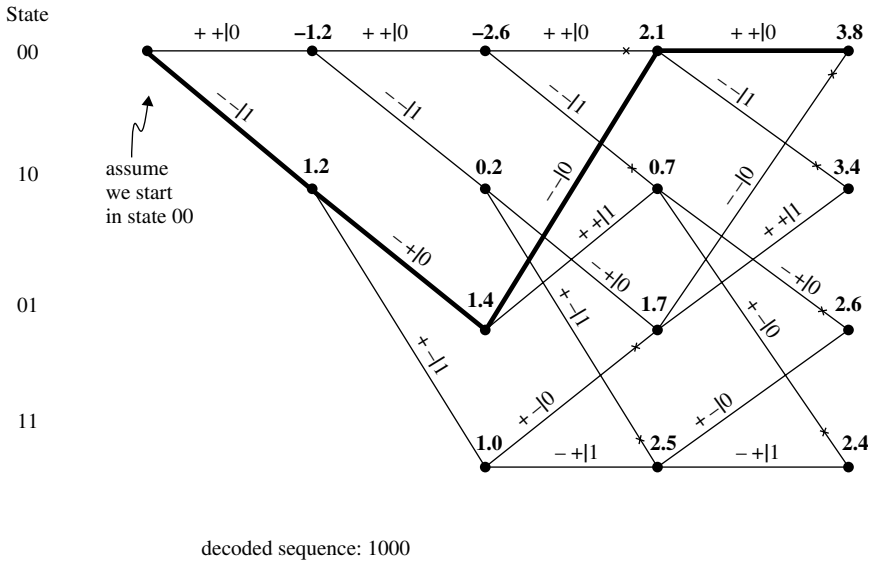


Figure 4.12 A Viterbi decoding example for the code of Figure 4.8 on the BI-AWGNC. Here $\mathbf{y} = [-0.7\ -0.5,\ -0.8\ -0.6,\ -1.1\ +0.4,\ +0.9\ +0.8]$. Non-surviving paths are indicated by an "X" and cumulative metrics are written in bold near merging branches. The ML path is the one with the cumulative metric of 3.8.

4.5.2 Differential Viterbi Decoding

It is possible to employ a (lower-complexity) *differential Viterbi algorithm* for the decoding of certain rate-$1/n$ non-recursive, non-systematic convolutional codes on the BI-AWGNC [11]. In particular, the algorithm applies to such convolutional codes whose generator polynomials have the same degree μ and have first and last coefficients equal to "1": $g_0^{(1)} = \cdots = g_0^{(n)} = 1$ and $g_\mu^{(1)} = \cdots = g_\mu^{(n)} = 1$. This characteristic holds for most of the best rate-$1/n$ codes. For example, it applies to the industry-standard convolutional code for which $\mathbf{g}_{\text{octal}}^{(1)} = 117$ and $\mathbf{g}_{\text{octal}}^{(2)} = 155$, and it applies to the rate-1/2, memory-2 code with $\mathbf{g}_{\text{octal}}^{(1)} = 7$ and $\mathbf{g}_{\text{octal}}^{(2)} = 5$ that was examined in the previous section.

The differential algorithm derives from the fact that (after the initial μ stages) trellises for rate-$1/n$ $\bar{\text{R}}\bar{\text{S}}$C codes are composed of multiple "butterflies" of the form shown in Figure 4.13. Note that the two states at time $l-1$ have the forms $(\mathbf{u}, 0)$ and $(\mathbf{u}, 1)$, where $\mathbf{u} = (u_{l-1}, u_{l-2}, \ldots, u_{l-\mu+1})$ is the contents of the first $\mu - 1$ memory elements in the memory-μ encoder. At time l, the two states have the forms $(0, \mathbf{u})$ and $(1, \mathbf{u})$. Note also that the labels on the two branches that diverge from a given state at time l are complements of each other. Lastly, the two labels on one pair of diverging branches are equal to the two labels on the other pair. These facts allow us to display the various metrics for the butterfly as in Figure 4.14. As seen in the figure, the four branch metrics $\{\lambda_l(s', s)\}$ in Figure 4.14 can

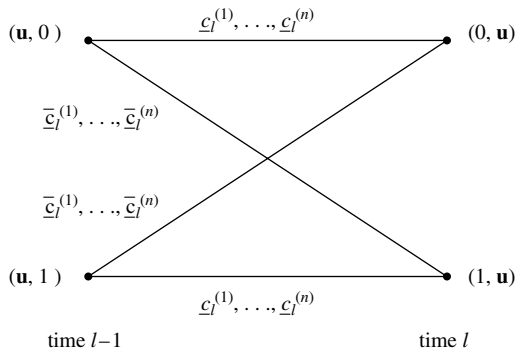


Figure 4.13 A generic butterfly from a trellis of a rate-$1/n$ convolutional codes whose generator polynomial coefficients are "1" in the first and last place.

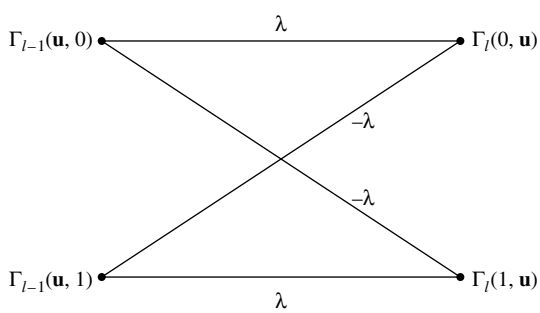


Figure 4.14 The butterfly corresponding to the butterfly of Figure 4.13 with cumulative and branch metrics displayed.

take on only one of two values, $+\lambda$ or $-\lambda$, a consequence of the fact that diverging branches are complements of each other (see also Equation (4.20)).

In view of Figure 4.14, observe that the tentative cumulative metric difference at state $(0, \mathbf{u})$ is

$$D_l(0, \mathbf{u}) = \Delta\Gamma_{l-1} + 2\lambda,$$

where $\Delta\Gamma_{l-1} \triangleq \Gamma_{l-1}(\mathbf{u}, 0) - \Gamma_{l-1}(\mathbf{u}, 1)$. Similarly, the tentative cumulative metric difference at state $(1, \mathbf{u})$ is

$$D_l(1, \mathbf{u}) = \Delta\Gamma_{l-1} - 2\lambda.$$

This can be compactly written as

$$D_l(u_l) = \Delta\Gamma_{l-1} + (-1)^{u_l} 2\lambda,$$

where $u_l \in \{0, 1\}$ is the encoder input at time l.

From Figure 4.14, the following survivor-selection rules may be determined:

the survivor into state $(0, \mathbf{u})$ comes from
state $(\mathbf{u}, 0)$, if $D_l(0, \mathbf{u}) > 0$
state $(\mathbf{u}, 1)$, if $D_l(0, \mathbf{u}) < 0$;
the survivor into state $(1, \mathbf{u})$ comes from
state $(\mathbf{u}, 0)$, if $D_l(1, \mathbf{u}) > 0$
state $(\mathbf{u}, 1)$, if $D_l(0, \mathbf{u}) < 0$;

where an arbitrary decision may be made if $D_l = 0$. It is easily shown that these survivor-selection rules can be alternatively represented as in Figure 4.15. Observe from this that in order to determine the two survivors in each butterfly within the full trellis one must first determine $\max\{|\Delta\Gamma_{l-1}|, |2\lambda|\}$ and then find the sign of $\Delta\Gamma_{l-1}$ if $|\Delta\Gamma_{l-1}|$ is maximum, or find the sign of λ if $|2\lambda|$ is maximum. Once the survivors have been found, their cumulative metrics can be determined via

$$\Gamma_l(u_l, \mathbf{u}) = \Gamma_{l-1}(\mathbf{u}, u_{l-\mu}) + (-1)^{u_l}(-1)^{u_{l-\mu}}\lambda.$$

Note that the core operation is effectively *compare–select–add* (CSA) as opposed to add–compare–select. It can be shown that a reduction of approximately 33% in additions is achieved by employing the differential Viterbi decoder instead of the conventional one.

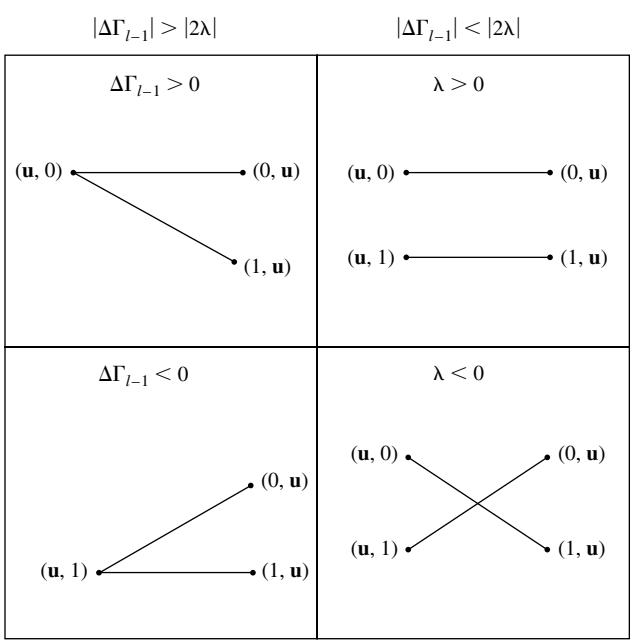


Figure 4.15 Survivor-selection rules for the butterfly of Figure 4.14.

4.5.3 Bit-wise MAP Decoding and the BCJR Algorithm

Whereas the MLSD minimizes the probability of sequence error, the bit-wise MAP decoder minimizes the bit-error probability. While they have similar performance characteristics, both bit-wise and sequence-wise, they have different applications. For example, the MAP decoder is used in turbo decoders and turbo equalizers where soft decoder outputs are necessary.

The bit-wise MAP decoding criterion is given by

$$\hat{u}_l = \arg\max_{u_l} P(u_l|\mathbf{y}),$$

where $P(u_l|\mathbf{y})$ is the *a posteriori* probability (APP) of the information bit u_l given the received word $\mathbf{y}$. For this discussion, we will favor $u_l \in \{+1, -1\}$ over $u_l \in \{0, 1\}$ under the mapping $\{0 \leftrightarrow +1, 1 \leftrightarrow -1\}$, and similarly for other binary variables. With $u_l \in \{+1, -1\}$, the bit-wise MAP rule simplifies to

$$\hat{u}_l = \text{sign}[L(u_l)], \tag{4.21}$$

where $L(u_l)$ is the logarithmic *a posteriori* probability (log-APP) ratio defined as

$$L(u_l) \triangleq \log\left[\frac{P(u_l = +1|\mathbf{y})}{P(u_l = -1|\mathbf{y})}\right].$$

The log-APP ratio is typically called the log-likelihood ratio (LLR) in the literature and we shall follow this convention here.

For convenience, we consider the BCJR-algorithm implementation of the MAP decoder applied to a rate-1/2 RSC code on a BI-AWGNC. Generalizing to other code rates is straightforward. The BCJR algorithm for the BSC will be considered in Problem 4.20. Further, the transmitted codeword $\mathbf{c}$ will have the form $\mathbf{c} = [c_1, c_2, \ldots, c_L] = [u_1, p_1, u_2, p_2, \ldots, u_L, p_L]$ with $c_l \triangleq [u_l, p_l]$, where u_l signifies the systematic bit, p_l signifies the parity bit, and L is the length of the encoder input sequence including termination bits. The received word $\mathbf{y} = \mathbf{c} + \mathbf{n}$ will have the form $\mathbf{y} = [y_1, y_2, \ldots, y_L] = [y_1^u, y_1^p, y_2^u, y_2^p, \ldots, y_L^u, y_L^p]$, where $y_l \triangleq [y_l^u, y_l^p]$, and similarly for $\mathbf{n}$.

Our goal is the development of the BCJR algorithm for computing the LLR $L(u_l)$ given the received word $\mathbf{y}$. In order to incorporate the RSC code trellis into this computation, we rewrite $L(u_l)$ as

$$L(u_l) = \log\left[\frac{\sum_{U^+} p(s_{l-1} = s', s_l = s, \mathbf{y})}{\sum_{U^-} p(s_{l-1} = s', s_l = s, \mathbf{y})}\right], \tag{4.22}$$

where s_l is the encoder state at time l, U^+ is the set of pairs (s', s) for the state transitions $(s_{l-1} = s') \rightarrow (s_l = s)$ which correspond to the event $u_l = +1$, and U^- is similarly defined for the event $u_l = -1$. To write (4.22) we used Bayes' rule, total probability, and then canceled out $1/p(\mathbf{y})$ in the numerator and denominator. We see from (4.22) that we need only compute $p(s', s, \mathbf{y}) = p(s_{l-1} = s', s_l = s, \mathbf{y})$

for all state transitions and then sum over the appropriate transitions in the numerator and denominator. We now present the crucial results which facilitate the computation of $p(s', s, \mathbf{y})$.

Lemma 4.1. *The pdf* $p(s', s, \mathbf{y})$ *may be factored as*

$$p(s', s, \mathbf{y}) = \alpha_{l-1}(s')\gamma_l(s', s)\beta_l(s), \tag{4.23}$$

where

$$\begin{aligned}
\alpha_l(s) &\triangleq p\Big(s_l = s, \mathbf{y}_1^l\Big), \\
\gamma_l(s', s) &\triangleq p(s_l = s, y_l | s_{l-1} = s'), \\
\beta_l(s) &\triangleq p\big(\mathbf{y}_{l+1}^L | s_l = s\big),
\end{aligned}$$

and $\mathbf{y}_a^b \triangleq [y_a, y_{a+1}, \ldots, y_b]$.

Proof. By several applications of Bayes' rule, we have

$$\begin{aligned}
p(s', s, \mathbf{y}) &= p\Big(s', s, \mathbf{y}_1^{l-1}, y_l, \mathbf{y}_{l+1}^L\Big) \\
&= p\Big(\mathbf{y}_{l+1}^L | s', s, \mathbf{y}_1^{l-1}, y_l\Big) p\Big(s', s, \mathbf{y}_1^{l-1}, y_l\Big) \\
&= p\Big(\mathbf{y}_{l+1}^L | s', s, \mathbf{y}_1^{l-1}, y_l\Big) p\Big(s, y_l | s', \mathbf{y}_1^{l-1}\Big) \cdot p\Big(s', \mathbf{y}_1^{l-1}\Big) \\
&= p\big(\mathbf{y}_{l+1}^L | s\big) p(s, y_l | s') p\Big(s', \mathbf{y}_1^{l-1}\Big) \\
&= \beta_l\,(s)\gamma_l\,(s', s)\alpha_{l-1}\,(s'),
\end{aligned}$$

where the fourth line follows from the third because the variables omitted on the fourth line are conditionally independent. □

Lemma 4.2. *The probability* $\alpha_l(s)$ *may be computed in a "forward recursion" via*

$$\alpha_l(s) = \sum_{s'} \gamma_l(s', s)\alpha_{l-1}\,(s'), \tag{4.24}$$

where the sum is over all possible encoder states.

Proof. By several applications of Bayes' rule and the theorem on total probability, we have

$$\begin{aligned}
\alpha_l(s) &\triangleq p\left(s, \mathbf{y}_1^l\right) \\
&= \sum_{s'} p\left(s', s, \mathbf{y}_1^l\right) \\
&= \sum_{s'} p\left(s, y_l | s', \mathbf{y}_1^{l-1}\right) p\left(s', \mathbf{y}_1^{l-1}\right) \\
&= \sum_{s'} p(s, y_l | s') p\left(s', \mathbf{y}_1^{l-1}\right) \\
&= \sum_{s'} \gamma_l\left(s', s\right) \alpha_{l-1}(s'),
\end{aligned}$$

where the fourth line follows from the third due to conditional independence of $\mathbf{y}_1^{l-1}$. □

Lemma 4.3. *The probability $\beta_l(s)$ may be computed in a "backward recursion" via*

$$\beta_{l-1}(s') = \sum_s \beta_l(s)\, \gamma_l(s', s). \tag{4.25}$$

Proof. Applying Bayes' rule and the theorem on total probability, we have

$$\begin{aligned}
\beta_{l-1}(s') &\triangleq p\big(\mathbf{y}_l^L | s'\big) \\
&= \sum_s p\big(\mathbf{y}_l^L, s | s'\big) \\
&= \sum_s p\big(\mathbf{y}_{l+1}^L | s', s, y_l\big) p(s, y_l | s') \\
&= \sum_s p\big(\mathbf{y}_{l+1}^L | s\big) p(s, y_l | s') \\
&= \sum_s \beta_l(s) \gamma_l(s', s),
\end{aligned}$$

where conditional independence led to the omission of variables on the fourth line. □

The recursion for the $\{\alpha_l(s)\}$ is initialized according to

$$\alpha_0(s) = \begin{cases} 1, & s = 0, \\ 0, & s \neq 0, \end{cases}$$

following from the reasonable assumption that the convolutional encoder is initialized to the zero state. The recursion for the $\{\beta_l(s)\}$ is initialized according to

$$\beta_L(s) = \begin{cases} 1, & s = 0, \\ 0, & s \neq 0, \end{cases}$$

which assumes that "termination bits" have been appended at the end of the data word so that the convolutional encoder is again in state zero at time L.

All that remains at this point is the computation of $\gamma_l(s', s) = p(s, y_l|s')$. Observe that $\gamma_l(s', s)$ may be written as

$$\begin{aligned} \gamma_l(s', s) &= \frac{P(s', s)}{P(s')} \cdot \frac{p(s', s, y_l)}{P(s', s)} \\ &= P(s|s')p(y_l|s', s) \\ &= P(u_l)p(y_l|u_l), \end{aligned} \tag{4.26}$$

where the event u_l corresponds to the event $s' \to s$. Note that $P(s|s') = \Pr(s' \to s) = 0$ if s is not a valid state from state s' and $\Pr(s' \to s) = 1/2$ otherwise (since we assume a binary-input encoder with equal *a priori* probabilities $P(u_l) = P(s|s')$). Hence, $\gamma_l(s', s) = 0$ if $s' \to s$ is not valid and, otherwise,

$$\gamma_l(s', s) = \frac{P(u_l)}{2\pi\sigma^2} \exp\left[-\frac{\|y_l - c_l\|^2}{2\sigma^2}\right] \tag{4.27}$$

$$= \frac{1}{2(2\pi)\sigma^2} \exp\left[-\frac{(y_l^u - u_l)^2 + (y_l^p - p_l)^2}{2\sigma^2}\right], \tag{4.28}$$

where $\sigma^2 = N_0/2$.

In summary, we may compute $L(u_l)$ via (4.22) using (4.23), (4.24), (4.25), and (4.27). This "probability-domain" version of the BCJR algorithm is numerically unstable for long and even moderate codeword lengths, so we now present the stable "log-domain" version of it. (Note that, in the presentation of the Viterbi algorithm, it was easy to immediately go to the log domain, although there exists a probability-domain Viterbi algorithm.)

In the log-BCJR algorithm, $\alpha_l(s)$ is replaced by the *forward metric*

$$\begin{aligned} \tilde{\alpha}_l(s) &\triangleq \log(\alpha_l(s)) \\ &= \log \sum_{s'} \alpha_{l-1}(s')\gamma_l(s', s) \\ &= \log \sum_{s'} \exp(\tilde{\alpha}_{l-1}(s') + \tilde{\gamma}_l(s', s)), \end{aligned} \tag{4.29}$$

where the *branch metric* $\tilde{\gamma}_l(s', s)$ is given by

$$\begin{aligned}\tilde{\gamma}_l(s', s) &= \log \gamma_l(s', s) \\ &= -\log\left(4\pi\sigma^2\right) - \frac{\|y_l - c_l\|^2}{2\sigma^2}.\end{aligned} \tag{4.30}$$

We will see that the first term in (4.30) may be dropped. Note that (4.29) not only defines $\tilde{\alpha}_l(s)$, but also gives its recursion. These log-domain forward metrics are initialized as

$$\tilde{\alpha}_0(s) = \begin{cases} 0, & s = 0, \\ -\infty, & s \neq 0. \end{cases} \tag{4.31}$$

The probability $\beta_{l-1}(s')$ is replaced by the *backward metric*

$$\begin{aligned}\tilde{\beta}_{l-1}(s') &\triangleq \log(\beta_{l-1}(s')) \\ &= \log\left(\sum_s \exp\left(\tilde{\beta}_l(s) + \tilde{\gamma}_l(s', s)\right)\right)\end{aligned} \tag{4.32}$$

with initial conditions

$$\tilde{\beta}_L(s) = \begin{cases} 0, & s = 0, \\ -\infty, & s \neq 0, \end{cases} \tag{4.33}$$

under the assumption that the encoder has been terminated to the zero state.

As before, $L(u_l)$ is computed as

$$\begin{aligned}L(u_l) &= \log\left[\frac{\sum_{U^+} \alpha_{l-1}(s')\,\gamma_l(s', s)\,\beta_l(s)}{\sum_{U^-} \alpha_{l-1}(s')\,\gamma_l(s', s)\beta_l(s)}\right] \\ &= \log\left[\sum_{U^+} \exp\left(\tilde{\alpha}_{l-1}(s') + \tilde{\gamma}_l(s', s) + \tilde{\beta}_l(s)\right)\right] \\ &\quad - \log\left[\sum_{U^-} \exp\left(\tilde{\alpha}_{l-1}(s') + \tilde{\gamma}_l(s', s) + \tilde{\beta}_l(s)\right)\right].\end{aligned} \tag{4.34}$$

It is evident from (4.34) that the constant term in (4.30) may be ignored since it may be factored all the way out of both summations. At first glance, Equations (4.29)–(4.34) do not look any simpler than the probability-domain algorithm, but we use the following results to attain the simplification.

It can be shown (Problem 4.19) that

$$\max(x, y) = \log\left(\frac{e^x + e^y}{1 + e^{-|x-y|}}\right). \tag{4.35}$$

Now define

$$\max^*(x, y) \triangleq \log\left(e^x + e^y\right) \tag{4.36}$$

so that, from (4.35),

$$\max^*(x, y) = \max(x, y) + \log\Big(1 + e^{-|x-y|}\Big). \tag{4.37}$$

This may be extended to more than two variables. For example,

$$\max^*(x, y, z) \triangleq \log(e^x + e^y + e^z),$$

which may be computed in pair-wise fashion according to

$$\max^*(x, y, z) = \max^*[\max^*(x, y), z].$$

Given the function $\max^*(\cdot)$, we may now rewrite (4.29), (4.32), and (4.34) as

$$\tilde{\alpha}_l(s) = \max_{s'}{}^*[\tilde{\alpha}_{l-1}(s') + \tilde{\gamma}_l(s', s)], \tag{4.38}$$

$$\tilde{\beta}_{l-1}(s') = \max_{s}{}^*\Big[\tilde{\beta}_l(s) + \tilde{\gamma}_l(s', s)\Big], \tag{4.39}$$

and

$$\begin{aligned} L(u_l) = &\max_{U^+}{}^*\Big[\tilde{\alpha}_{\,l-1}(s') + \tilde{\gamma}_l(s', s) + \tilde{\beta}_l(s)\Big] \\ &- \max_{U^-}{}^*\Big[\tilde{\alpha}_{\,l-1}(s') + \tilde{\gamma}_l(s', s) + \tilde{\beta}_l(s)\Big]. \end{aligned} \tag{4.40}$$

Figure 4.16 illustrates pictorially the trellis-based computations that these three equations represent. It is also illuminating to compare Figure 4.16 with Figure 4.9. Consider also Problem 4.22. The log-domain BCJR algorithm is presented below.

Algorithm 4.2 The Log-Domain BCJR Algorithm

Assumptions

We assume as above a rate-1/2 RSC encoder, a data block $\mathbf{u}$ of length L, and that the encoder starts and terminates in the zero state (the last μ bits of $\mathbf{u}$ are so selected). In practice, the value $-\infty$ used in initialization is simply some large-magnitude negative number.

The Algorithm

Initialize $\tilde{\alpha}_0(s)$ and $\tilde{\beta}_L(s)$ according to (4.31) and (4.33).
for $l = 1$ to L

- get $y_l = [y_l^u, y_l^p]$
- compute $\tilde{\gamma}_l(s', s) = -\|y_l - c_l\|^2/(2\sigma^2)$ for all allowable state transitions $s' \to s$ (note that $c_l = c_l(s', s)$ here)[1]
- compute $\tilde{\alpha}_l(s)$ for all s using the recursion (4.38)

end

[1] We may alternatively use $\tilde{\gamma}_l(s', s) = u_l y_l^u/\sigma^2 + p_l y_l^p/\sigma^2$. (See Problem 4.21.)

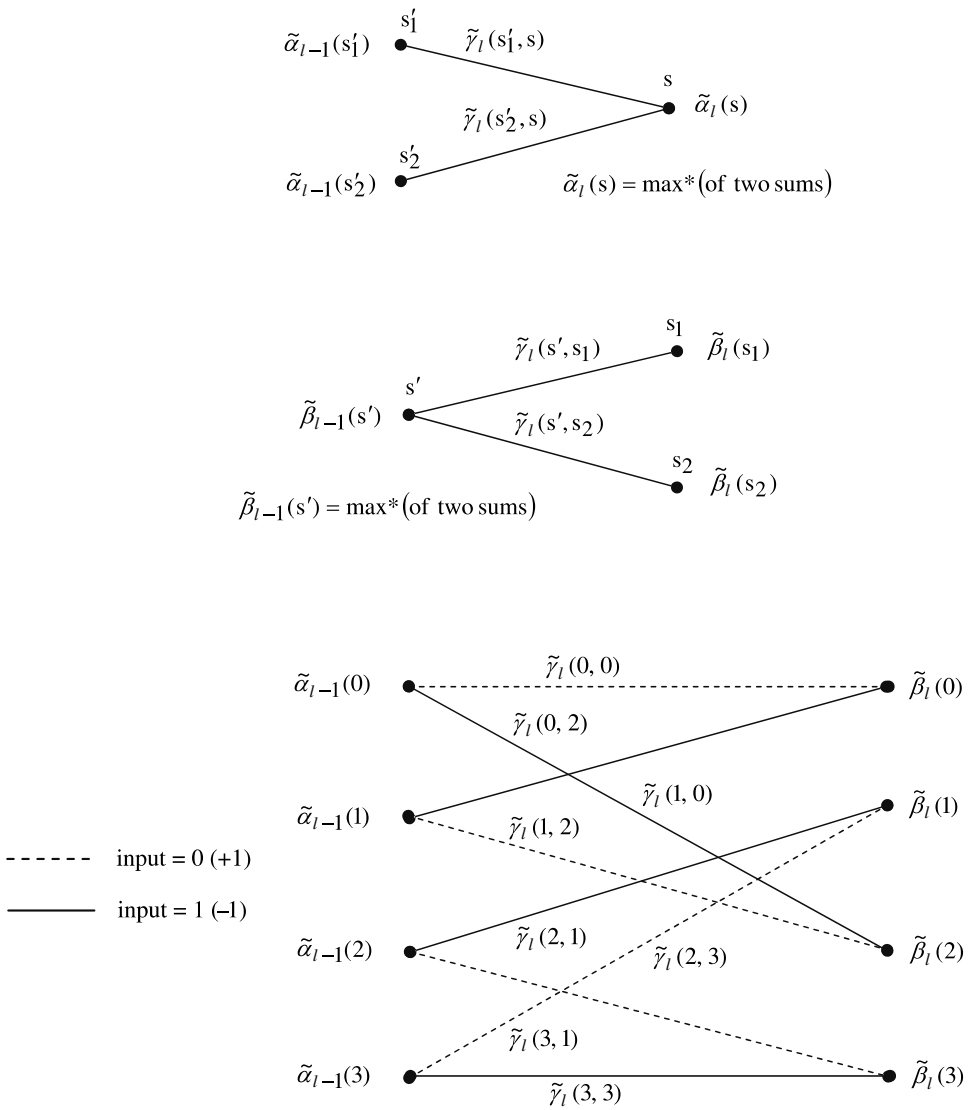


Figure 4.16 An illustration of the BCJR algorithm. The top diagram illustrates the forward recursion in (4.38), the middle diagram illustrates the backward recursion in (4.39), and the bottom diagram depicts the computation of $L(u_l)$ via (4.40).

for $l = L$ to 2 step -1

- compute $\tilde{\beta}_{l-1}(s')$ for all s' using (4.39)

end

for $l = 1$ to L

- compute $L(u_l)$ using (4.40)
- compute hard decisions via $\hat{u}_l = \text{sign}[L(u_l)]$

end

We see from (4.38), (4.39), and (4.40) how the log-domain computation of $L(u_l)$ is vastly simplified relative to the probability-domain computation. From (4.37), implementation of the $\max^*(\cdot)$ function involves only a two-input $\max(\cdot)$ function plus a look-up table for the "correction term" $\log\left(1 + e^{-|x-y|}\right)$. The size of the look-up table has been investigated in [12] for specific cases. When $\max^*(x, y)$ is replaced by $\max(\cdot)$ in (4.38) and (4.39), these recursions become forward and reverse Viterbi algorithms, respectively. The performance loss associated with this approximation in turbo decoding depends on the specific turbo code, but a loss of about 0.5 dB is typical [12].

4.6 Performance Estimates for Trellis-Based Decoders

4.6.1 ML Decoder Performance for Block Codes

Before we present formulas for estimating the performance of maximum-likelihood sequence decoding (Viterbi decoding) of convolutional codes on the BSC and BI-AWGN channels, we do this for linear binary block codes. This approach is taken because the block-code case is simpler conceptually, the block-code results are useful in their own right, and the block-code results provide a large step toward the convolutional-code results. We assume that the codeword length is N, the information word length is K, the code rate is R, and the minimum distance is $d_{\min}$.

To start, we observe that both channels are symmetric in the sense that for the BSC $\Pr(y = 1|x = 1) = \Pr(y = 0|x = 0)$ and for the BI-AWGN channel $p(y|x) = p(-y| - x)$, where x and y represent the channel input and output, respectively. Given these conditions together with code linearity, the probability of error given that some codeword $\mathbf{c}$ was transmitted is identical to the probability of error given that any other codeword $\mathbf{c}'$ was transmitted. Thus, it is convenient to assume that the all-zeros codeword $\mathbf{0}$ was transmitted in performance analyses. We can therefore bound the probability of codeword error P_{cw} using the union

bound as follows:

$$\begin{aligned} P_{cw} &= \sum_{\mathbf{c}} \Pr(\text{error}|\mathbf{c}) \Pr(\mathbf{c}) \\ &= \Pr(\text{error}|\mathbf{0}) \\ &= \Pr(\cup_{\hat{\mathbf{c}} \neq \mathbf{0}} \text{decide } \hat{\mathbf{c}}|\mathbf{0}) \\ &\leq \sum_{\hat{\mathbf{c}} \neq \mathbf{0}} \Pr(\text{decide } \hat{\mathbf{c}}|\mathbf{0}), \end{aligned} \tag{4.41}$$

where $\hat{\mathbf{c}}$ denotes the decoder output. The probability $\Pr(\text{decide } \hat{\mathbf{c}}|\mathbf{0})$, which is the probability that the decoder chooses $\hat{\mathbf{c}}$ given that $\mathbf{0}$ was transmitted, is called the *two-codeword error probability* (or *pair-wise error probability*). It is derived under the assumption that only two codewords are involved: $\hat{\mathbf{c}}$ and $\mathbf{0}$.

As we will see, both for the BSC and for the AWGN channel, $\Pr(\text{decide } \hat{\mathbf{c}}|\mathbf{0})$ is a function of the Hamming distance between $\hat{\mathbf{c}}$ and $\mathbf{0}$, that is, the Hamming weight w of $\hat{\mathbf{c}}$. Further, if $\hat{\mathbf{c}}$ and $\hat{\mathbf{c}}'$ both have weight w, then $\Pr(\text{decide } \hat{\mathbf{c}}|\mathbf{0}) = \Pr(\text{decide } \hat{\mathbf{c}}'|\mathbf{0})$ and we denote this common probability by P_w. In view of this, we may rewrite (4.41) as

$$P_{cw} \leq \sum_{w=d_{\min}}^{N} A_w P_w \ , \tag{4.42}$$

where A_w is the number of codewords of weight w.

For the BSC, when w is odd, $P_w \triangleq \Pr(\text{decide a codeword } \hat{\mathbf{c}} \text{ of weight } w \ |\mathbf{0})$ is the probability that the channel produces $\lceil w/2 \rceil$ or more errors that place the received word closer to $\hat{\mathbf{c}}$ than to $\mathbf{0}$ in a Hamming-distance sense. ($\lceil x \rceil$ is the integer greater than or equal to x.) Thus, we focus only on the w nonzero positions of $\hat{\mathbf{c}}$ to write

$$P_w = \begin{cases} \sum\limits_{j=\lceil w/2 \rceil}^{w} \binom{w}{j} \varepsilon^j (1-\varepsilon)^{w-j} & \text{for } w \text{ odd}, \\ \Pr(w/2) + \sum\limits_{j=w/2+1}^{w} \binom{w}{j} \varepsilon^j (1-\varepsilon)^{w-j} & \text{for } w \text{ even}, \end{cases} \tag{4.43}$$

where the term

$$\Pr(w/2) \triangleq \frac{1}{2} \binom{w}{w/2} \varepsilon^{w/2} (1-\varepsilon)^{w/2}$$

accounts for the fact that ties in the decoder are resolved arbitrarily when w is even.

For the BI-AWGN channel, let $m(\mathbf{c})$ denote the channel representation of the codeword $\mathbf{c}$. For example, $m(\mathbf{0}) = [+\sqrt{E_c}, +\sqrt{E_c}, +\sqrt{E_c}, +\sqrt{E_c}, \ldots]$ and $m(\hat{\mathbf{c}}) = m([1\,1\,1\,0\ \ldots]) = [-\sqrt{E_c}, -\sqrt{E_c}, -\sqrt{E_c}, +\sqrt{E_c}, \ldots]$, where E_c is the average code-bit energy on the channel and is related to the code rate R and the average data-bit energy E_b as $E_c = RE_b$. Now, P_w is the probability that the white-noise sequence results in a receiver output $\mathbf{r}$ that is closer to $m(\hat{\mathbf{c}})$ than it is to $m(\mathbf{0})$ in a Euclidean-distance sense. However, $m(\hat{\mathbf{c}})$ and $m(\mathbf{0})$ are separated by the Euclidean distance

$d_{\mathrm{E}} = 2\sqrt{wE_c} = 2\sqrt{wRE_b}$. Thus, since the white noise has variance $\sigma^2 = N_0/2$ in all directions, P_w is the probability that the noise in the direction of $\hat{\mathbf{c}}$ has magnitude greater than $d_{\mathrm{E}}/2 = \sqrt{wRE_b}$; that is,

$$P_w = Q\left(\frac{d_{\mathrm{E}}}{2\sigma}\right) = Q\left(\sqrt{\frac{2wRE_b}{N_0}}\right). \tag{4.44}$$

The bit error probability P_b, another commonly used performance measure, can be obtained from the above results. We first define $A_{i,w}$ to be the number of weight-w codewords produced by weight-i encoder inputs. Then the probability of bit error can be bounded as

$$\begin{aligned} P_b &\leq \frac{1}{K} \sum_{w=d_{\min}}^{N} \sum_{i=1}^{K} i A_{i,w} P_w \\ &= \frac{1}{K} \sum_{w=d_{\min}}^{N} B_w P_w \end{aligned} \tag{4.45}$$

where $B_w \triangleq \sum_{i=1}^{K} i A_{i,w}$ is the number of nonzero information bits corresponding to all of the weight-w codewords. P_w in the above expressions is given by (4.43) or (4.44), depending on whether the channel is BSC or BI-AWGN.

4.6.2 Weight Enumerators for Convolutional Codes

Because digital communication links are generally packet-based, convolutional codes are often used as block codes. That is, the inputs to a rate-k/n convolutional encoder in this case are blocks of K information bits, which produce blocks of $N = K(n/k)$ code bits. When a convolutional code is treated as a block code, all of the results of the preceding subsection apply. However, as we will see, it is more convenient to settle for approximations of the bounds given above. This is so because the computation of A_w and $A_{i,w}$ (or B_w) can be demanding (although it is possible; see Chapter 8). Recall that A_w is the number of weight-w codewords. Or, in the context of the code's trellis, it is the number of weight-w paths that diverge from the all-zeros trellis path *one or more times* in $L = K/k$ trellis stages. Finding $\{A_w\}$ is possible by computer for all practical codes, and so this is one possible avenue. Then, the formulas of the previous section may be utilized.

Traditionally, however, one uses an alternative weight spectrum $\{A'_w\}$, where A'_w is the number of weight-w paths that diverge from the all-zeros trellis path *one time* in L trellis stages. The attractiveness of A'_w stems from the fact that the entire spectrum $\{A'_w\}$ is easily found analytically for small-memory codes, or by computer for any practical code. These comments apply as well to the alternative weight spectrum $\{A'_{i,w}\}$, where $A'_{i,w}$ is the number of weight-w paths, corresponding to weight-i encoder inputs, that diverge from the all-zeros trellis path *one time* in L trellis stages.

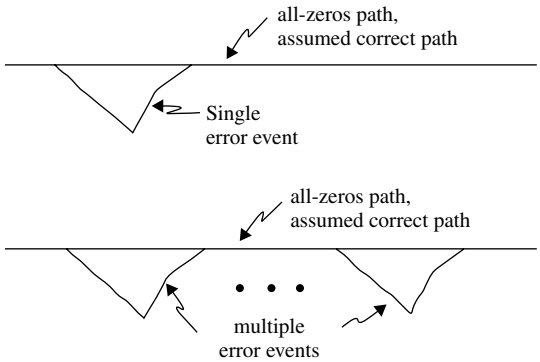


Figure 4.17 An illustration of nonzero code paths comprising single error events and multiple error events.

We remark, incidentally, that a portion of a path that diverges/remerges from/to the correct path is called an *error event*. Thus, as shown in Figure 4.17, we say that a path that diverges/remerges from/to the correct path multiple times has multiple error events; otherwise, it has a single error event.

Consider any convolutional code and its state diagram and trellis. It is possible to determine $\{A'_{i,w}\}$ via computer by having the algorithm traverse the L-stage trellis until all paths have been covered, updating the $A'_{i,w}$ tables along the way. Alternatively, the state diagram can be utilized to analytically determine $\{A'_{i,w}\}$. To do this, recognize that diverging from the all-zeros trellis path one time is equivalent to leaving state zero in the state diagram and returning to that state only once. Thus, we can count the cumulative information weight and codeword weight in either the trellis or the state diagram. However, signal flow-graph techniques allow us to perform this computation analytically using the state diagram. Note that $\{A'_w\}$ is obtainable from $\{A'_{i,w}\}$ since $A'_w = \sum_i A'_{i,w}$.

Example 4.9. Consider again the rate-1/2 encoder described by $\mathbf{G}(D) = [1 + D + D^2 \quad 1 + D^2]$ whose state diagram and trellis are presented in Figure 4.8. We are interested in all of the paths that leave state $S_0 = 00$ and then return to it some time later. Thus, we split the state diagram at state S_0. Then, in order to exploit signal flow-graph techniques, we re-label the state-transition branches with bivariate monomials whose exponents contain the information weight and code weight. For example, the label 11|1 would be replaced by the label $I^1W^2 = IW^2$ and the label 01|0 would be replaced by the label $I^0W^1 = W$. The resulting *split state diagram* is shown in Figure 4.18.

The split state diagram is now regarded as a transform-domain linear system with input S_0, and output S'_0, whose subsystem gains are given by the new branch labels. If one obtains the transfer function S'_0/S_0 of this linear system, one has actually obtained the *input–output weight enumerator* (IO-WE) $A'(I, W) = \sum_{i,w} A'_{i,w} I^i W^w$. This is because

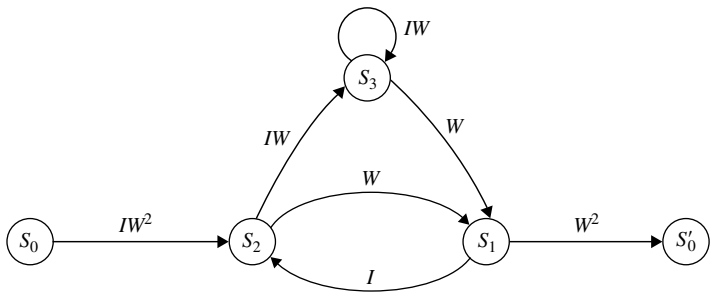


Figure 4.18 The branch-labeled split state diagram for the $\bar{\text{R}}\bar{\text{S}}$C rate-1/2 convolutional encoder with octal generators $(7, 5)$.

taking the product of the gains of the form $I^i W^w$ that label a series of branches has the effect of adding exponents, thereby counting the information weight and code weight along the series of branches. The transfer-function computation also keeps track of the number of paths from S_0 to S_0' for the weight profiles (i, w); these are the coefficients $A'_{i,w}$.

The most obvious way to compute $A'(I, W) = S_0'/S_0$ is to write down a system of linear equations based on the split state diagram and then solve for S_0'/S_0. For example, from Figure 4.18 we may write

$$\begin{aligned} S_2 &= IW^2 S_0, \\ S_1 &= WS_2 + WS_3, \\ S_3 &= IWS_3 + IWS_2, \end{aligned}$$

and so on. The solution is

$$\begin{aligned} A'(I, W) = S_0'/S_0 &= \frac{IW^5}{1 - 2IW} \\ &= IW^5 + 2I^2W^6 + 4I^3W^7 + 8I^4W^8 + \cdots . \end{aligned}$$

Thus, $A'_{1,5} = 1$ (one weight-5 path created by a weight-1 encoder input), $A'_{2,6} = 2$ (two weight-6 paths created by weight-2 encoder inputs), $A'_{3,7} = 4$ (four weight-7 paths created by weight-3 encoder inputs), and so on. Note that the codeword weight enumerator $A'(W)$ may be obtained from $A'(I, W)$ as

$$\begin{aligned} A'(W) &= A'(I, W)|_{I=1} \\ &= \frac{W^5}{1 - 2W} \\ &= W^5 + 2W^6 + 4W^7 + 8W^8 + \cdots . \end{aligned}$$

Thus, $A'_5 = 1$ (one weight-5 path), $A'_6 = 2$ (two weight-6 paths), $A'_7 = 4$ (four weight-7 paths), and so on. Of course, $A'(W)$ may be obtained more directly by using branch labels of the form W^w in the split state diagram (which effectively sets $I = 1$).

By analogy to the definition of B_w, we define $B'_w \triangleq \sum_{i=1}^{K} iA'_{i,w}$, the number of nonzero information bits corresponding to all of the weight-w paths the diverge from state zero (at an arbitrary time) and remerge later. The values B'_w may be obtained as coefficients of the enumerator

$$\begin{aligned} B'(W) &= \left[\frac{d}{dI}A'(I,W)\right]\Bigg|_{I=1} \\ &= \frac{W^5}{1-4W+4W^2} \\ &= W^5 + 4W^6 + 12W^7 + 32W^8 + \cdots . \end{aligned}$$

Example 4.10. An even simpler approach to arriving at $A'(I,W)$ avoids writing out a system of linear equations. The approach involves successively simplifying the split state diagram until one obtains a single branch with a single label that is equal to $A'(I,W)$. This is demonstrated in Figure 4.19 for the code of the previous example. Except for the last flow graph, whose derivation is obvious, explanations are given for the derivation of each flow graph from the one above it. For extremely large split state diagrams, one generally utilizes a computer program based on *Mason's gain rule*.

The techniques in the foregoing examples allow us to derive $A'(I,W)$ for information words and codewords of unlimited length. For finite-length codewords, the techniques may be simply modified to obtain an *augmented* weight enumerator $A'(I,W,B) = \sum_{i,w,b} A_{i,w,b} I^i W^w B^b$, where $A_{i,w,b}$ is the number of weight-w paths spanning b trellis branches, created by weight-i encoder inputs. The split-state-diagram labels in this case are of the form $I^i W^w B$, where the exponent of B is always unity since it labels and accounts for a single branch.

Following this procedure for our example code above results in

$$\begin{aligned} A'(I,W,B) &= \frac{IW^5B^3}{1-IW(B+B^2)} \\ &= IW^5B^3 + I^2W^6B^4 + I^2W^6B^5 + \cdots . \end{aligned}$$

Thus, the weight-5 path consists of three trellis branches. Also, one weight-6 path consists of four branches and the other consists of five (both correspond to weight-2 inputs). Note that $A'(I,W) = A'(I,W,B)|_{B=1}$.

Suppose now we are interested in codewords in the above code corresponding to length-K information words. Then we can find the weight enumerator for this

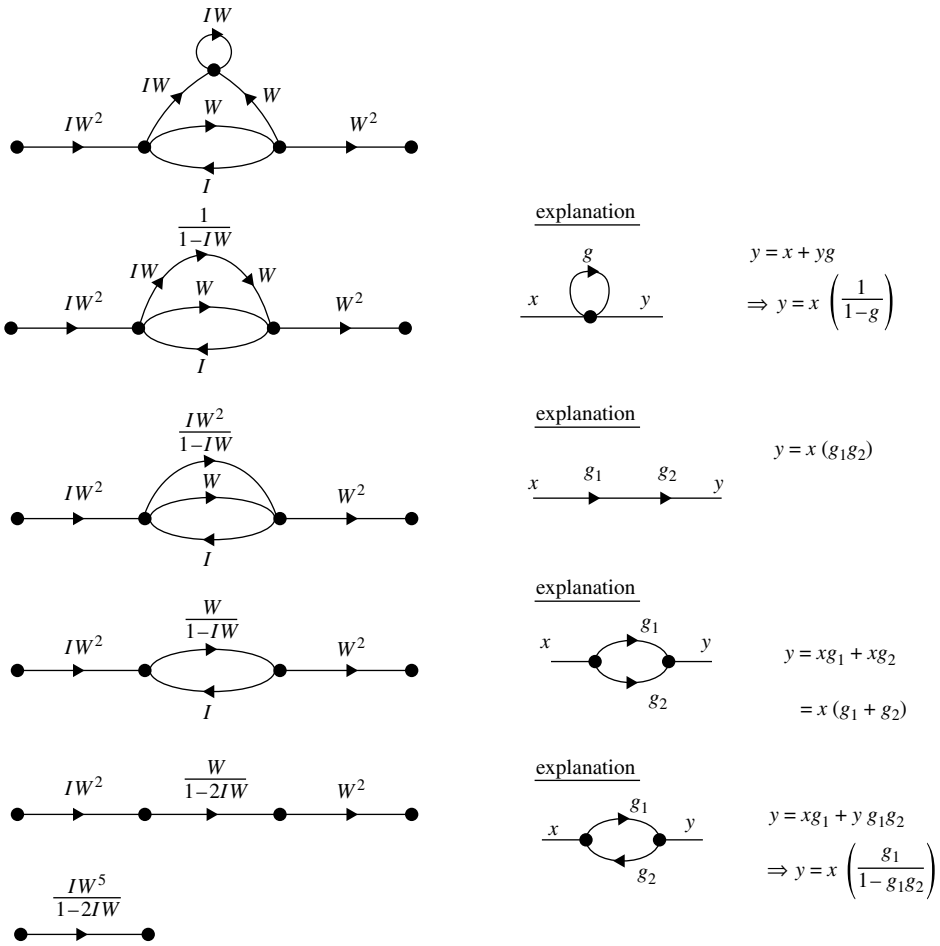


Figure 4.19 Successive reduction of the split state diagram of Figure 4.18 to obtain the transfer function.

situation by omitting the terms in $A'(I, W, B)$ whose B exponents are greater than K.

4.6.3 ML Decoder Performance for Convolutional Codes

For packet-based communications, we make the assumption that the rate-k/n convolutional encoder accepts K information bits and produces N code bits. As mentioned at the beginning of the previous section, traditional flow-graph techniques yield only weight-enumerator data for incorrect paths that have single error events (e.g., $A'_{i,w}$ or A'_w). On the other hand, our P_{cw} and P_b performance bounds for (N,K) linear codes, Equations (4.42) and (4.45), require enumerator data for all incorrect paths (A_w and $A_{i,w}$). Chapter 8 shows how one can carefully obtain

A_w and $A_{i,w}$ from $A'_{i,w}$ or A'_w. Here, we make a couple of approximations that simplify our work and result in simple formulas. We will see that the approximate formulas are sufficiently accurate for most purposes.

The first approximation is that at most one error event occurs in each code block. This is clearly inaccurate for extremely long code blocks and low SNR values, but it is quite accurate otherwise, as we will see in the example below. The second approximation is that the number of different locations at which an error event may occur in a code block is $L = K/k$. The more accurate value is $L - \ell$ locations, where ℓ is the length of the error event measured in trellis stages, but in practice $\ell \ll L$.

Given these two approximations, we may write

$$A_w \simeq LA'_w$$

and

$$A_{i,w} \simeq LA'_{i,w}.$$

Application of these relations to (4.42) and (4.45) yields

$$P_{cw} \lesssim \sum_{w=d_{\min}}^{N} LA'_w P_w = \frac{K}{k} \sum_{w=d_{\min}}^{N} A'_w P_w \tag{4.46}$$

and

$$\begin{aligned} P_b &\lesssim \frac{1}{K} \sum_{w=d_{\min}}^{N} \sum_{i=1}^{K} iLA'_{i,w} P_w \\ &= \frac{1}{K} \frac{K}{k} \sum_{w=d_{\min}}^{N} \left(\sum_{i=1}^{K} iA'_{i,w} \right) P_w \\ &= \frac{1}{k} \sum_{w=d_{\min}}^{N} B'_w P_w, \end{aligned} \tag{4.47}$$

where B'_w is as defined above. As before, P_w in the above expressions is given by (4.43) for the BSC and (4.44) for the BI-AWGN channel.

Example 4.11. We demonstrate the accuracy of Equations (4.46) and (4.47) in Figure 4.20, which displays these bounds together with simulation results for our running-example rate-1/2 code for which $\mu = 2$, $\mathbf{g}^{(1)}_{\text{octal}} = 7$, and $\mathbf{g}^{(2)}_{\text{octal}} = 5$. The figure presents P_{cw} curves for $K = 100$, 1000, and 5000. Observe that for all three P_{cw} curves and the P_b curve there is close agreement between the bounds (solid lines) and the simulation results (circles). Also included in the figure are the P_b bound (4.47) and P_b simulation results

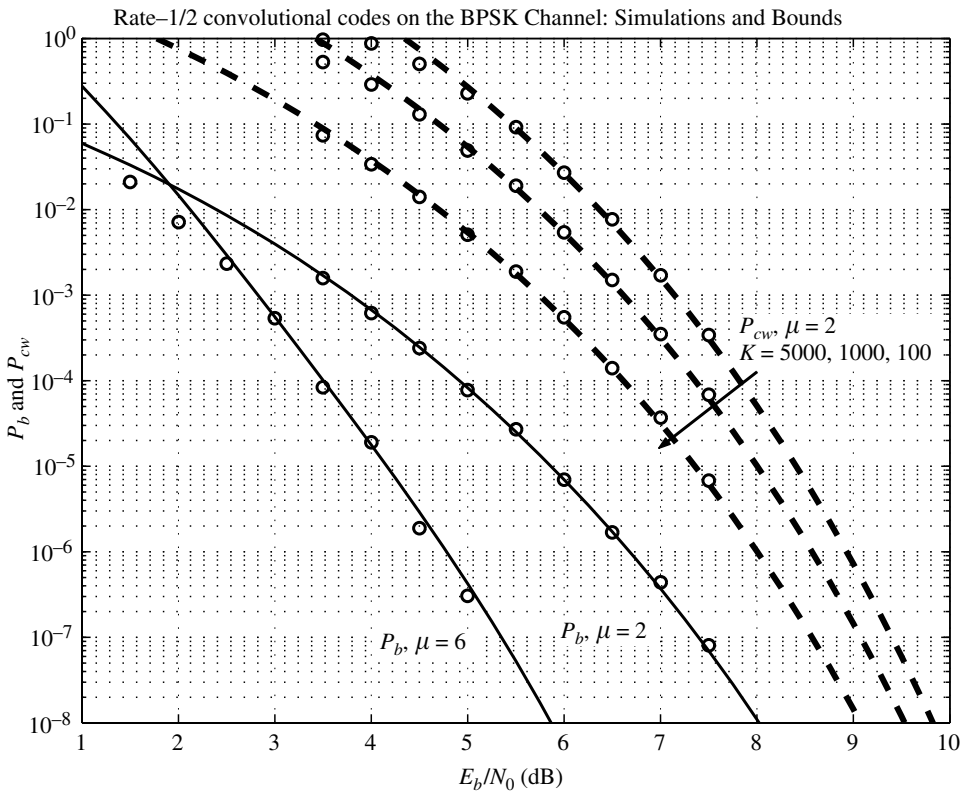


Figure 4.20 Maximum-likelihood decoding performance bounds and simulation results for the rate-1/2 convolutional codes of Example 4.11 on the BI-AWGN channel.

for the $\mu = 6$ industry-standard code for which $\mathbf{g}^{(1)}_{\text{octal}} = 117$ and $\mathbf{g}^{(2)}_{\text{octal}} = 155$ (see Section 4.3.4). It was found by computer search that, for this code, $[B_{10}, B_{11}, \ldots, B_{21}] = [36, 0, 211, 0, 1404, 0, 11633, 0, 77433, 0, 502690, 0]$. Clearly, from the figure, this limited-weight-spectrum information is sufficient to obtain a tight bound.

Problems

4.1 Find the power series $s(D)$ for the rational function

$$\frac{a(D)}{b(D)} = \frac{1 + D + D^2}{1 + D + D^3}$$

and note that, after a transient, its coefficients are periodic. Show that $a(D) = b(D)s(D)$.

4.2 Show that, if the entries of the generator matrix $\mathbf{G}(D)$ are confined to the field of rational functions, $\mathbb{F}_2(D)$, then the entries of the minimal representation of the parity-check matrix $\mathbf{H}(D)$ must also be confined to $\mathbb{F}_2(D)$.

4.3 Find $\mathbf{G}_{\text{sys}}(D)$, $\mathbf{H}_{\text{sys}}(D)$, $\mathbf{G}_{\text{poly}}(D)$, and $\mathbf{H}_{\text{poly}}(D)$ for the rate-2/3 convolutional-code generator matrix given by

$$\mathbf{G}(D) = \begin{bmatrix} \dfrac{1+D}{1+D+D^2} & 1+D & 0 \\ 1 & \dfrac{1}{1+D} & \dfrac{D}{1+D^2} \end{bmatrix}.$$

4.4 Show that the transfer function

$$g_{ij}(D) = \frac{a_0 + a_1 D + \cdots + a_m D^m}{1 + b_1 D + \cdots + b_m D^m}$$

can be implemented as in Figure 4.2. To do so write $c^{(j)}(D)$ as

$$c^{(j)}(D) = \left(u^{(i)}(D) \cdot \frac{1}{b(D)} \right) \cdot a(D). \tag{4.48}$$

Then sketch the direct implementation of the leftmost "filtering" operation $v(D) = u^{(i)}(D) \cdot 1/b(D)$, which can be determined from the difference equation

$$v_t = u_t^{(i)} - b_1 v_{t-1} - b_2 v_{t-2} - \cdots - b_m v_{t-m}.$$

Next, sketch the direct implementation of the second filtering operation in (4.48), $c^{(j)}(D) = v(D)a(D)$, which can be determined from the difference equation

$$c_t^{(j)} = a_0 v_t^{(i)} + a_1 v_{t-1}^{(i)} + \cdots + a_m v_{t-m}^{(i)}.$$

Finally, sketch the two "filters" in cascade, the $1/b(D)$ filter followed by the $a(D)$ filter (going from left to right), and notice that the m delay (memory) elements may be shared by the two filters.

4.5 Show that Figures 4.2 and 4.3 implement equivalent transfer functions and that Figure 4.3 is the transposed form of Figure 4.2. That is, note that Figure 4.3 may be obtained from Figure 4.2 by (a) changing all nodes to adders; (b) changing all adders to nodes; (c) reversing the directions of the arrows; and (d) naming the input the output, and vice versa.

4.6 Show that every non-systematic encoder is equivalent to a systematic encoder in the sense that they generate the same code. Show that for every non-recursive non-systematic encoder there exists an equivalent recursive systematic encoder in the sense that they generate the same code.

4.7 Consider the rate-2/3 convolutional code in Example 4.2. Let the input for the encoder $\mathbf{G}(D)$ be $\mathbf{u}(D) = [1 \quad 1+D]$ and find the corresponding codeword $\mathbf{c}(D) = [\mathbf{c}_1(D) \quad \mathbf{c}_2(D) \quad \mathbf{c}_3(D)]$. Find the input which yields the same codeword when the encoder is given by $\mathbf{G}_{\text{sys}}(D)$. Repeat for $\mathbf{G}_{\text{poly}}(D)$.

4.8 Show that every rate-$1/n$ convolutional code has a catastrophic encoder. Repeat for rate-k/n.

4.9 Determine whether either of the following encoders is catastrophic:

$$\mathbf{G}(D) = \begin{bmatrix} 1 + D + D^2 + D^3 + D^4 & D + D^6 \end{bmatrix},$$

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

4.10 Generalize the catastrophic-code condition for rate-$1/n$ convolutional codes to the case in which the entries of $\mathbf{G}(D)$ are rational functions. Using this result, determine whether either of the following encoders is catastrophic:

$$\mathbf{G}(D) = \begin{bmatrix} \dfrac{1+D}{1+D+D^2} & 1+D \end{bmatrix},$$

$$\mathbf{G}(D) = \begin{bmatrix} \dfrac{D}{1+D+D^2} & \dfrac{1+D^2}{1+D^3} \end{bmatrix}.$$

4.11 Consider the four generator matrices given in the previous two problems.

(a) Determine to which class of encoder each of these generator matrices belongs (RSC, $\bar{\text{R}}$SC, R$\bar{\text{S}}$C, or $\bar{\text{R}}\bar{\text{S}}$C). If an encoder is not systematic, find its systematic form.

(b) Sketch the Type I and Type II realizations of the systematic and non-systematic versions of each of these codes.

4.12 Consider a rate-2/3 convolutional code with

$$\mathbf{G}_{\text{poly}}(D) = \begin{bmatrix} 1+D & 0 & 1 \\ 1+D^2 & 1+D & 1+D+D^2 \end{bmatrix}.$$

Show that the memory required for the Type I and Type II realizations of $\mathbf{G}_{\text{poly}}(D)$ is $\mu = 3$ and $\mu = 6$, respectively. Show that

$$\mathbf{G}_{\text{sys}}(D) = \begin{bmatrix} 1 & 0 & \dfrac{1}{1+D} \\ 0 & 1 & \dfrac{D^2}{1+D} \end{bmatrix},$$

and that the memory required for its Type I realization is $\mu = 3$. Finally, show that the Type II realization of $\mathbf{G}_{\text{sys}}(D)$ requires only $\mu = 2$. Thus, the Type II realization of $\mathbf{G}_{\text{sys}}(D)$ is the minimal encoder as claimed in Section 4.3.3 for rate-$k/(k+1)$ convolutional codes.

4.13 Sketch the minimal encoder realizations for the code corresponding to the following generator matrices:

(a) $\mathbf{G}(D) = \begin{bmatrix} 1 + D^4 & 1 + D^3 & 1 + D + D^2 + D^3 \end{bmatrix}$,

(b) $\mathbf{G}(D) = \begin{bmatrix} 1+D^3 & 1+D^4 & (1+D+D^2)(1+D^3) \\ (1+D+D^2)(1+D^2) & 1+D^3 & 1+D+D^2+D^3 \end{bmatrix}$.

4.14 Suppose a generator matrix $\mathbf{G}(D)$ is in polynomial form and the maximum degree of any polynomial in $\mathbf{G}(D)$ is two. Give the form of (4.13) specialized to this case.

4.15 Find the multiplexed generator matrix $\mathbf{G}'$ and the multiplexed parity-check matrix $\mathbf{H}'$ for the rate-1/2 encoder matrix

$$\mathbf{G}(D) = \begin{bmatrix} 1+D^2+D^3+D^4 & 1+D+D^4 \end{bmatrix}.$$

Find $\mathbf{G}_{\text{sys}}(D)$ corresponding to $\mathbf{G}(D)$ and repeat for the encoder matrix $\mathbf{G}_{\text{sys}}(D)$.

4.16 Consider the rate-2/3 convolutional code with polynomial parity-check matrix

$$\begin{aligned} \mathbf{H}(D) &= \begin{bmatrix} h_2(D) & h_1(D) & h_0(D) \end{bmatrix} \\ &= \begin{bmatrix} 1+D & 1+D^2 & 1+D+D^2 \end{bmatrix}. \end{aligned}$$

Show that a generator matrix for this code is given by

$$\mathbf{G}(D) = \begin{bmatrix} h_1(D) & h_2(D) & 0 \\ 0 & h_0(D) & h_1(D) \end{bmatrix}.$$

Find the multiplexed generator matrix $\mathbf{G}'$ and the multiplexed parity-check matrix $\mathbf{H}'$. Find $\mathbf{G}_{\text{sys}}(D)$ and repeat for $\mathbf{G}_{\text{sys}}(D)$.

4.17 Assuming the BI-AWGNC, simulate Viterbi decoding of the rate-1/2 convolutional code whose encoder matrix is given by

$$\mathbf{G}(D) = \begin{bmatrix} 1+D^2+D^3+D^4 & 1+D+D^4 \end{bmatrix}.$$

Plot the bit error rate from $P_b = 10^{-1}$ to $P_b = 10^{-6}$. Repeat for the BCJR decoder and comment on the performance of the two decoders.

4.18 Assuming the BI-AWGNC, simulate Viterbi decoding of the rate-2/3 convolutional code whose parity-check matrix is given by (see Problem 4.16)

$$\begin{aligned} \mathbf{H}(D) &= \begin{bmatrix} h_2(D) & h_1(D) & h_0(D) \end{bmatrix} \\ &= \begin{bmatrix} 1+D & 1+D^2 & 1+D+D^2 \end{bmatrix}. \end{aligned}$$

Plot the bit error rate from $P_b = 10^{-1}$ to $P_b = 10^{-6}$. Repeat for the BCJR decoder and comment on the performance of the two decoders.

4.19 Show that

$$\max(x, y) = \log\left(\frac{e^x + e^y}{1 + e^{-|x-y|}}\right).$$

Hint: First suppose $x > y$.

4.20 Derive the BCJR algorithm for the BSC. Initialization from the channel output values proceeds as follows. Let $y_i \in \{0, 1\}$ be the BSC output and $c_i \in$

$\{0,1\}$ be the BSC output at time i. Define $\varepsilon = \Pr(y_i = b^c | c_i = b)$ to be the error probability. Then

$$\Pr(c_i = b|y_i) = \begin{cases} 1-\varepsilon & \text{when } y_i = b, \\ \varepsilon & \text{when } y_i = b^c, \end{cases}$$

and, from this, we have

$$L(c_i|y_i) = (-1)^{y_i} \log\left(\frac{1-\varepsilon}{\varepsilon}\right).$$

4.21 Show that in the log-domain BCJR algorithm we may alternatively use the branch metric $\tilde{\gamma}_l(s', s) = u_l y_l^u/\sigma^2 + p_l y_l^p/\sigma^2$.

4.22 Show that there is a BCJR algorithm equivalent to the one presented in the chapter, as follows. Define

$$\begin{aligned} \min{}^*(x, y) &\triangleq -\log\left(e^{-x} + e^{-y}\right) \\ &= \min(x, y) + \log\left(1 + e^{-|x-y|}\right). \end{aligned}$$

A similar definition exists when more than two variables are involved, for example, $\min^*(x, y, z) = -\log(e^{-x} + e^{-y} + e^{-z})$. Define also

$$\begin{aligned} \breve{\alpha}_l(s) &\triangleq -\log(\alpha_l(s)) \\ &= -\log \sum_{s'} \exp(-\breve{\alpha}_{l-1}(s') - \breve{\gamma}_l(s', s)) \\ &= \min{}^*_{s'}\{\tilde{\alpha}_{l-1}(s') + \tilde{\gamma}_l(s', s)\}, \end{aligned}$$

$$\begin{aligned} \breve{\gamma}_l(s', s) &\triangleq -\log \gamma_l(s', s) \\ &= \frac{\|y_l - c_l\|^2}{2\sigma^2} \text{ (ignores a useless constant)}, \end{aligned}$$

$$\begin{aligned} \breve{\beta}_{l-1}(s') &\triangleq -\log\left(\beta_{l-1}(s')\right) \\ &= -\log\left(\sum_s \exp\left(-\breve{\beta}_l(s) - \breve{\gamma}_l(s', s)\right)\right) \\ &= \min{}^*_s\left\{\breve{\beta}_l(s) + \breve{\gamma}_l(s', s)\right\}. \end{aligned}$$

Specify the initial conditions and show that $L(u_l)$ can be computed as

$$\begin{aligned} L(u_l) &= \log\left[\frac{\sum_{U^+} \alpha_{l-1}(s')\,\gamma_l(s', s)\,\beta_l(s)}{\sum_{U^-} \alpha_{l-1}(s')\,\gamma_l(s', s)\,\beta_l(s)}\right] \\ &= \min_{U^+}{}^*\left[\breve{\alpha}_{l-1}(s') + \breve{\gamma}_l(s', s) + \breve{\beta}_l(s)\right] \\ &\quad - \min_{U^-}{}^*\left[\breve{\alpha}_{l-1}(s') + \breve{\gamma}_l(s', s) + \breve{\beta}_l(s)\right]. \end{aligned}$$

4.23 Repeat Example 4.9 for the RSC code with

$$\mathbf{G}(D) = \begin{bmatrix} 1 & \dfrac{1+D}{1+D+D^2} \end{bmatrix}.$$

Find also the augmented weight enumerator $A'(I, W, B)$. You should find, after dividing the denominator of $A'(I, W, B)$ into its numerator, that the series begins as $I^3W^5B^3 + I^2W^6B^4 + \left(I^3W^7 + I^4W^6\right)B^5 + \cdots$. Show in the state diagram of the code the four paths corresponding to these four terms.

4.24 Repeat Example 4.10 and Figure 4.16 for the RSC code with

$$\mathbf{G}(D) = \begin{bmatrix} 1 & \dfrac{1+D}{1+D+D^2} \end{bmatrix}.$$

Compare your results with those of the $\bar{\text{R}}\bar{\text{S}}$C code of Figure 4.16 for which $\mathbf{G}(D) = \begin{bmatrix} 1+D+D^2 & 1+D \end{bmatrix}$.

References

[1] P. Elias, "Coding for noisy channels," *IRE Conv. Rep.*, Pt. 4, pp. 37–47, 1955.

[2] G. D. Forney, Jr., "Convolutional codes I: Algebraic structure," *IEEE Trans. Information Theory*, vol. 16, no. 11, pp. 720–738, November 1970.

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

[4] A. J. Viterbi, "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm," *IEEE Trans. Information Theory*, vol. 13, no. 4, pp. 260–269, April 1967.

[5] G. D. Forney, Jr., "The Viterbi algorithm," *Proc. IEEE*, pp. 268–278, March 1973.

[6] L. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate," *IEEE Trans. Information Theory*, vol. 20, no. 3, pp. 284–287, March 1974.

[7] A. J. Viterbi, "Convolutional codes and their performance in communication systems," *IEEE Trans. Commun. Technol.*, pp. 751–772, October 1971.

[8] S. Lin and D. J. Costello, Jr., *Error Control Coding*, 2nd edn., New Saddle River, NJ, Prentice-Hall, 2004.

[9] S. Wicker, *Error Control Systems for Digital Communication and Storage*, Englewood Cliffs, NJ, Prentice-Hall, 1995.

[10] A. V. Oppenheim and R. W. Schafer, *Discrete-Time Signal Processing*, Englewood Cliffs, NJ, Prentice-Hall, 1989.

[11] M. Fossorier and S. Lin, "Differential trellis decoding of convolutional codes," *IEEE Trans. Information Theory*, vol. 46, no. 5, pp. 1046–1053, May 2000.

[12] P. Robertson, E. Villebrun, and P. Hoeher, "A comparison of optimal and suboptimal MAP decoding algorithms operating in the log domain," *Proc. 1995 Int. Conf. on Communications*, pp. 1009–1013.

5 Low-Density Parity-Check Codes

Low-density parity-check (LDPC) codes are a class of linear block codes with implementable decoders, which provide near-capacity performance on a large set of data-transmission and data-storage channels. LDPC codes were invented by Gallager in his 1960 doctoral dissertation [1] and were mostly ignored during the 35 years that followed. One notable exception is the important work of Tanner in 1981 [2], in which Tanner generalized LDPC codes and introduced a graphical representation of LDPC codes, now called a Tanner graph. The study of LDPC codes was resurrected in the mid 1990s with the work of MacKay, Luby, and others [3–6], who noticed, apparently independently of Gallager's work, the advantages of linear block codes with sparse (low-density) parity-check matrices.

This chapter introduces LDPC codes and creates a foundation for further study of LDPC codes in later chapters. We start with the fundamental representations of LDPC codes via parity-check matrices and Tanner graphs. We then learn about the decoding advantages of linear codes that possess sparse parity-check matrices. We will see that this sparseness characteristic makes the code amenable to various iterative decoding algorithms, which in many instances provide near-optimal performance. Gallager [1] of course recognized the decoding advantages of such low-density parity-check codes and he proposed a decoding algorithm for the BI-AWGNC and a few others for the BSC. These algorithms have received much scrutiny in the past decade, and are still being studied. In this chapter, we present these decoding algorithms together with several others, most of which are related to Gallager's original algorithms. We point out that some of the algorithms were independently obtained by other coding researchers (e.g., MacKay and Luby [4, 5]), who were unaware of Gallager's work at the time, as well as by researchers working on graph-based problems unrelated to coding [8].

5.1 Representations of LDPC Codes

5.1.1 Matrix Representation

We shall consider only binary LDPC codes for the sake of simplicity, although LDPC codes can be generalized to nonbinary alphabets as is done in Chapter 14. A *low-density parity-check code* is a linear block code given by the null space

of an $m \times n$ parity-check matrix $\mathbf{H}$ that has a low density of 1s. A *regular LDPC code* is a linear block code whose parity-check matrix $\mathbf{H}$ has column weight g and row weight r, where $r = g(n/m)$ and $g \ll m$. If $\mathbf{H}$ is low density, but its row and column weight are not both constant, then the code is an *irregular LDPC code*. For irregular LDPC codes, the various row and column weights are determined by one of the code-design procedures discussed in subsequent chapters. For reasons that will become apparent later, almost all LDPC code constructions impose the following additional structural property on $\mathbf{H}$: no two rows (or two columns) have more than one position in common that contains a nonzero element. This property is called the *row–column constraint*, or simply, the *RC constraint*.

The descriptor "low density" is unavoidably vague and cannot be precisely quantified, although a density of 0.01 or lower can be called low density (1% or fewer of the entries of $\mathbf{H}$ are 1s). As will be seen later in this chapter, the density need only be sufficiently low to permit effective iterative decoding. This is in fact the key innovation behind the invention of LDPC codes. As is well known, optimum (e.g., maximum-likelihood) decoding of the general linear block code that is useful for applications is not possible due to the vast complexity involved. The low-density aspect of LDPC codes accommodates iterative decoding, which typically has near-maximum-likelihood performance at error rates of interest for many applications. These ideas will be made clearer as the student progresses through this chapter.

As will be seen below and in later chapters, the construction of LDPC codes usually involves the construction of $\mathbf{H}$, which need not be full rank. In this case, the code rate R for a regular LDPC code is bounded as

$$R \geq 1 - \frac{m}{n} = 1 - \frac{g}{r},$$

with equality when $\mathbf{H}$ is full rank.

5.1.2 Graphical Representation

The *Tanner graph* of an LDPC code is analogous to the trellis of a convolutional code in that it provides a complete representation of the code and it aids in the description of decoding algorithms. A Tanner graph is a *bipartite graph* (introduced in Chapter 2), that is, a graph whose nodes may be separated into two types, with edges connecting only nodes of different types. The two types of nodes in a Tanner graph are the *variable nodes* (or *code-bit nodes*) and the *check nodes* (or *constraint nodes*), which we denote by VNs and CNs, respectively. The Tanner graph of a code is drawn as follows: CN i is connected to VN j whenever element h_{ij} in $\mathbf{H}$ is a 1. Observe from this rule that there are m CNs in a Tanner graph, one for each check equation, and n VNs, one for each code bit. Further, the m rows of $\mathbf{H}$ specify the m CN connections, and the n columns of $\mathbf{H}$ specify the n VN connections. Accordingly, the allowable n-bit words represented by the n VNs are precisely the codewords in the code. Throughout this book, we shall

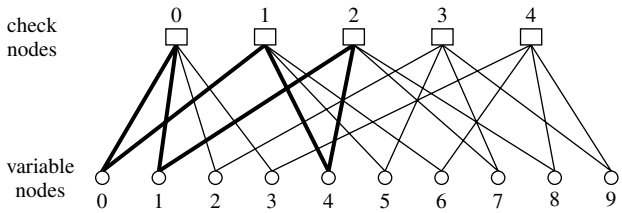


Figure 5.1 The Tanner graph for the code given in Example 5.1.

use both the notation CN i and VN j and the notation CN c_i and VN v_j, where v_j is the jth variable node or the jth code bit, depending on context.

Example 5.1. Consider a (10,5) linear block code with $w_c = 2$ and $w_r = 4$ with the following $\mathbf{H}$ matrix:

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

The Tanner graph corresponding to $\mathbf{H}$ is depicted in Figure 5.1. Observe that VNs 0, 1, 2, and 3 are connected to CN 0 in accordance with the fact that, in the zeroth row of $\mathbf{H}$, $h_{00} = h_{01} = h_{02} = h_{03} = 1$ (all others are zero). Observe that analogous situations hold for CNs 1, 2, 3, and 4, which correspond to rows 1, 2, 3, and 4 of $\mathbf{H}$, respectively. Note, as follows from $\mathbf{v}\mathbf{H}^{\mathrm{T}} = \mathbf{0}$, that the bit values connected to the same check node must sum to zero (mod 2). We may also proceed along columns to construct the Tanner graph. For example, note that VN 0 is connected to CNs 0 and 1 in accordance with the fact that, in the zeroth column of $\mathbf{H}$, $h_{00} = h_{10} = 1$. The sum of rows of $\mathbf{H}$ is the all-zero vector, so $\mathbf{H}$ is not full rank and $R \geq 1 - 5/10$. It is easily seen that the first row is dependent (it is the sum of the other rows). From this, we have $\mathrm{rank}(\mathbf{H}) = 4$ and $R = 1 - 4/10 = 3/5$.

The Tanner graph of an LDPC code acts as a blueprint for the iterative decoder in the following way. Each of the nodes acts as a locally operating processor and each edge acts as a bus that conveys information from a given node to each of its neighbors. The information conveyed is typically probabilistic information, e.g., log-likelihood ratios (LLRs), pertaining to the values of the bits assigned to the variable nodes. The LDPC decoder is initiated by n LLRs from the channel, which are received by the n VN processors. At the beginning of each half-iteration in the basic iterative decoding algorithm, each VN processor takes inputs from

the channel and each of its neighboring CNs, and from these computes outputs for each one of its neighboring CN processors. In the next half-iteration, each CN processor takes inputs from each of its neighboring VNs, and from these computes outputs for each one of its neighboring VN processors. The VN↔CN iterations continue until a codeword is found or until the preset maximum number of iterations has been reached.

The effectiveness of the iterative decoder depends on a number of structural properties of the Tanner graph on which the decoder is based. Observe the six thickened edges in Figure 5.1. A sequence of edges such as these which form a closed path is called a *cycle* (see Chapter 2). We are interested in cycles because short cycles degrade the performance of the iterative decoding algorithms employed by LDPC codes. This fact will be made evident in the discussion of the decoding algorithms later in this chapter, but it can also be seen from the brief algorithm description in the previous paragraph. It should be clear from the description that cycles force the decoder to operate locally in some portions of the graph (e.g., continually around a short cycle) so that a globally optimum solution is impossible. Observe also from the decoder description the necessity of a low-density matrix $\mathbf{H}$: at high densities (about half of the entries are 1s), many short cycles will exist, thus precluding the use of an iterative decoder.

The *length of a cycle* is equal to the number of edges which form the cycle, so the length of the cycle in Figure 5.1 is 6. A cycle of length l is often called an l-cycle. The minimum cycle length in a given bipartite graph is called the graph's *girth*. The girth of the Tanner graph for the example code is clearly 6. The shortest possible cycle in a bipartite graph is clearly a length-4 cycle, and such cycles manifest themselves in the $\mathbf{H}$ matrix as four 1s that lie on the four corners of a rectangular submatrix of $\mathbf{H}$. Observe that the RC constraint eliminates length-4 cycles.

The Tanner graph in the above example is regular: each VN has two edge connections and each CN has four edge connections. We say that the *degree* of each VN is 2 and the degree of each CN is 4. This is in accordance with the fact that $g = 2$ and $r = 4$. It is also clear from this example that $mr = ng$ must hold for all regular LDPC codes since both mr and ng are equal to the number of edges in the graph.

As will be shown in later chapters, it is possible to more closely approach capacity limits with irregular LDPC codes than with regular LDPC codes. For irregular LDPC codes, the parameters g and r vary with the columns and rows, so such notation is not useful in this case. Instead, it is usual in the literature to specify the VN and CN *degree-distribution polynomials*, denoted by $\lambda(X)$ and $\rho(X)$, respectively. In the polynomial

$$\lambda(X) = \sum_{d=1}^{d_v} \lambda_d X^{d-1}, \tag{5.1}$$

λ_d denotes the fraction of all edges connected to degree-d VNs and d_v denotes the maximum VN degree. Similarly, in the polynomial

$$\rho(X) = \sum_{d=1}^{d_c} \rho_d X^{d-1}, \tag{5.2}$$

ρ_d denotes the fraction of all edges connected to degree-d CNs and d_c denotes the maximum CN degree. Note that, for the regular code above, for which $g = 2$ and $r = 4$, we have $\lambda(X) = X$ and $\rho(X) = X^3$.

Let us denote the number of VNs of degree d by $N_v(d)$ and the number of CNs of degree d by $N_c(d)$. Let us further denote by E the number of edges in the graph. Then it can be shown that

$$E = \frac{n}{\int_0^1 \lambda(X)dX} = \frac{m}{\int_0^1 \rho(X)dX}, \tag{5.3}$$

$$N_v(d) = E\lambda_d/d = \frac{n\lambda_d/d}{\int_0^1 \lambda(X)dX}, \tag{5.4}$$

$$N_c(d) = E\rho_d/d = \frac{m\rho_d/d}{\int_0^1 \rho(X)dX}. \tag{5.5}$$

From the two expressions for E we may easily conclude that the code rate is bounded as

$$R \geq 1 - \frac{m}{n} = 1 - \frac{\int_0^1 \rho(X)dX}{\int_0^1 \lambda(X)dX}. \tag{5.6}$$

The polynomials $\lambda(X)$ and $\rho(X)$ represent a Tanner graph's degree distributions from an "edge perspective." The degree distributions may also be represented from a "node perspective" using the notation $\tilde{\lambda}(X)$ and $\tilde{\rho}(X)$, where the coefficient $\tilde{\lambda}_d$ is the fraction of all VNs that have degree d and $\tilde{\rho}_d$ is the fraction of CNs that have degree d. It is easily shown that

$$\tilde{\lambda}_d = \frac{\lambda_d/d}{\int_0^1 \lambda(X)dX}, \tag{5.7}$$

$$\tilde{\rho}_d = \frac{\rho_d/d}{\int_0^1 \rho(X)dX}. \tag{5.8}$$

5.2 Classifications of LDPC Codes

As described in the next chapter, the original LDPC codes are random in the sense that their parity-check matrices possess little structure. This is problematic in that both encoding and decoding become quite complex when the code possesses no structure beyond being a linear code. More recently, LDPC codes with structure have been constructed.

The most obvious type of structure is that of a cyclic code. As discussed in Chapter 3, the encoder of a cyclic code consists of a single length-$(n-k)$ shift register, some binary adders, and a gate. The nominal parity-check matrix $\mathbf{H}$ of a cyclic code is an $n \times n$ circulant; that is, each row is a cyclic-shift of the one above it, with the first row a cyclic-shift of the last row. The implication of a sparse circulant matrix $\mathbf{H}$ for LDPC decoder complexity is substantial: because each check equation is closely related to its predecessor and its successor, implementation can be vastly simplified compared with the case of random sparse $\mathbf{H}$ matrices for which wires are randomly routed. However, beside being regular, a drawback of cyclic LDPC codes is that the nominal $\mathbf{H}$ matrix is $n \times n$, independently of the code rate, implying a more complex decoder. Another drawback is that the known cyclic LDPC codes tend to have large row weights, which makes decoder implementation tricky. On the other hand, as discussed in Chapter 10, cyclic LDPC codes tend to have large minimum distances and very low iteratively decoded error-rate floors. (See a discussion of floors in Section 5.4.6.)

Quasi-cyclic (QC) codes also possess tremendous structure, leading to simplified encoder and decoder designs. Additionally, they permit more flexibility in code design, particularly irregularity, and, hence, lead to improved codes relative to cyclic LDPC codes. The $\mathbf{H}$ matrix of a QC code is generally represented as an array of circulants, e.g.,

$$\mathbf{H} = \begin{bmatrix} A_{11} & \cdots & A_{1N} \\ \vdots & & \vdots \\ A_{M1} & \cdots & A_{MN} \end{bmatrix}, \tag{5.9}$$

where each matrix A_{rc} is a $Q \times Q$ circulant. For LDPC codes, the circulants must be sparse, and in fact weight-1 circulants, which means that the weight of each row and column of the circulants is 1, are common. To effect irregularity, some of the circulants may be the all-zeros $Q \times Q$ matrix using a technique called masking (see Chapters 10 and 11). Chapter 3 discusses how the QC characteristic leads to simplified encoders. Chapter 6 discusses how QC codes permit simplified, modular decoders. The second half of the book presents many code-design techniques that result in QC codes.

In addition to partitioning LDPC codes into three classes – cyclic, quasi-cyclic, and random (but linear) – the LDPC code-construction techniques can be partitioned as well. The first class of construction techniques can be described as algorithmic or computer-based. These techniques will be introduced in Chapter 6 and covered in greater detail in Chapters 8 and 9. The second class of construction techniques consists of those based on finite mathematics, including algebra, combinatorics, and graph theory. These techniques are covered in Chapters 10–14. We note that the computer-based construction techniques can lead to either random or structured LDPC codes. The mathematical construction techniques generally lead to structured LDPC codes, although exceptions exist.

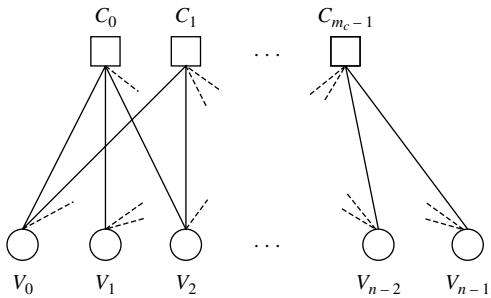


Figure 5.2 A Tanner graph for a G-LDPC code. Circle nodes represent code bits as before, but the square nodes represent generalized constraints.

5.2.1 Generalized LDPC Codes

It is useful in the study of iteratively decodable codes to consider generalizations of LDPC codes. Tanner was the first to generalize Gallager's low-density parity-check code idea in his pioneering work [2], the paper in which Tanner graphs were introduced. In the Tanner graph of a generalized LDPC (G-LDPC) code, depicted in Figure 5.2, constraint nodes (CNs) are more general than single parity-check (SPC) constraints. In fact, the variable nodes (VNs) may represent more complex codes than the repetition codes they represent in standard LDPC codes [7], but we shall not consider such "doubly generalized" LDPC codes here.

In Figure 5.2, the CNs $C_0, C_1, \ldots, C_{m_c-1}$ signify the m_c code constraints placed on the code bits connected to the CNs. Note that, while a bipartite graph gives a complete description of an LDPC code, for a generic G-LDPC code the specifications of the component (linear) block codes are also required. Let us define m as the summation of all the redundancies introduced by the m_c CNs, namely,

$$m = \sum_{i=0}^{m_c-1} m_i,$$

where $m_i = n_i - k_i$ and (n_i, k_i) represent the parameters of the component code for constraint node C_i, whose parity-check matrix is denoted by $\mathbf{H}_i$. Then the rate of a length-n G-LDPC code satisfies

$$R \geq 1 - \frac{m}{n},$$

with equality only if all of the check equations derived from the CNs are linearly independent.

Let $V = \{V_j\}_{j=0}^{n-1}$ be the set of n VNs and $C = \{C_i\}_{i=0}^{m_c-1}$ be the set of m_c CNs in the bipartite graph of a G-LDPC code (Figure 5.2). The connection between the nodes in V and C can be summarized in an $m_c \times n$ adjacency matrix $\mathbf{\Gamma}$. We now introduce the relationship between the adjacency matrix $\mathbf{\Gamma}$ and the parity-check matrix $\mathbf{H}$ for a G-LDPC code.

While $\mathbf{\Gamma}$ for an LDPC code serves as its parity-check matrix, for a G-LDPC code, to obtain $\mathbf{H}$, one requires also knowledge of the parity-check matrices $\mathbf{H}_i$ of the CN component codes. The n_i 1s in the ith binary row of $\mathbf{\Gamma}$ indicate which of the n G-LDPC code bits are constrained by constraint node C_i. Because the parity checks corresponding to C_i are represented by the rows of $\mathbf{H}_i$, $\mathbf{H}$ is easily obtained by replacing the n_i 1s in row i of $\mathbf{\Gamma}$ by the columns of $\mathbf{H}_i$, for all i. The zeros in row i are replaced by a length-m_i all-zero column vector. Note that this procedure allows for the case when C_i is an SPC code, in which case $\mathbf{H}_i$ has only one row. Note also that $\mathbf{H}$ will be of size $m \times n$, where $m = \sum_{i=0}^{m_c-1} m_i$.

5.3 Message Passing and the Turbo Principle

The key innovation behind LDPC codes is the low-density nature of the parity-check matrix, which facilitates iterative decoding. Although the various iterative decoding algorithms are suboptimal, they often provide near-optimal performance, at least for the error rates of interest. In particular, we will see that the so-called *sum–product algorithm* (SPA) is a general algorithm that provides near-optimal performance across a broad class of channels. The remainder of this chapter is dedicated to the development of the SPA decoder and a few other iterative decoding algorithms.

Toward the goal of presenting the near-optimal sum–product algorithm for the iterative decoding of LDPC codes, it is instructive to step back and first look at the larger picture of generic message-passing decoding. *Message-passing decoding* refers to a collection of low-complexity decoders working in a distributed fashion to decode a received codeword in a concatenated coding scheme. Variants of the SPA and other message-passing algorithms have been invented independently in a number of different contexts, including *belief propagation* for inference in Bayesian networks and turbo decoding for parallel concatenated convolutional codes (which are the original "turbo codes"). Researchers in belief propagation liken the algorithm to the distributed counting of soldiers and the inventors of turbo codes were inspired by turbo-charged engines in which the engine is supplied an increased amount of air by exploiting its own exhaust in a feedback fashion. In this section we develop an intuition for message-passing decoding and the so-called turbo principle through a few simple examples. This will be followed by a detailed presentation of the SPA message-passing decoder in the next section for four binary-input channels: the BEC, the BSC, the AWGNC, and the independent Rayleigh channel.

We can consider an LDPC code to be a generalized concatenation of many SPC codes. A message-passing decoder for an LDPC code employs an individual decoder for each SPC code and these decoders operate cooperatively in a distributed fashion to determine the correct code bit values. As indicated in the discussion above, Tanner graphs often serve as decoder models. In the context of a message-passing decoder, the check nodes represent SPC decoders that work cooperatively to decode the received word.

Example 5.2. As our first decoding example, Figure 5.3 shows how an SPC product code may be represented as an LDPC code and iteratively decoded. Figure 5.3(a) gives the product code array and Figure 5.3(b) gives the parity-check matrix. The graph-based decoder model is depicted in Figure 5.3(c). The top check nodes in the graph represent the row SPC decoders and the bottom check nodes represent the column SPC decoders. Figure 5.3(d) presents an example decoding on the BEC, which may be taken to be standard iterative row/column decoding on the BEC, a technique that has been known for several decades. As an example decoding operation, because each row and column of the code array corresponds to an SPC codeword, we know that the middle erasure in the received word must equal 1 since $1 + e + 0 = 0 \pmod 2$. Note in Figure 5.3(d) how certain erasures may be resolved only by the row decoder and other erasures may be resolved only by the column decoder, but by working together all erasures are eventually resolved. Although it is not necessary for this simple example, one can equivalently view this decoding from the point of view of the Tanner graph in which the component SPC decoders iteratively solve for the erasures: first the top decoders and then the bottom decoders, and so on. This decoder is also equivalent to the iterative erasure-filling algorithm to be discussed in Section 5.7.1.

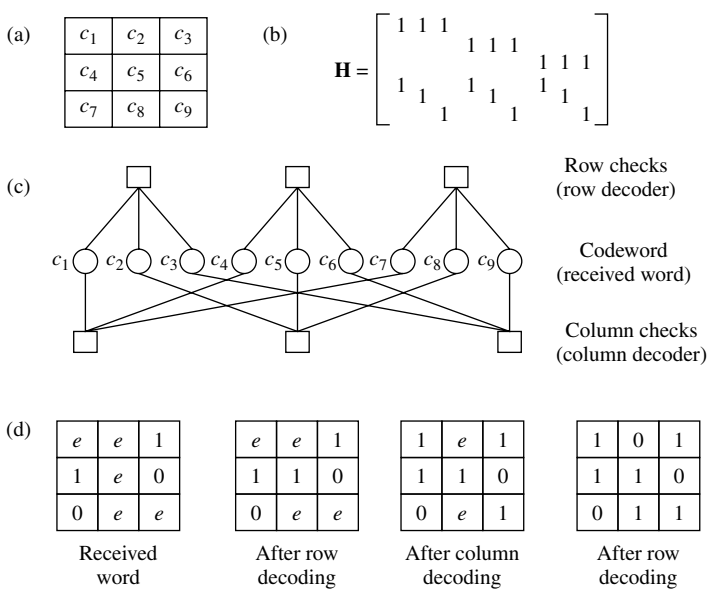


Figure 5.3 (3,2) × (3,2) SPC product code as an LDPC code. (a) The product-code array. (b) The parity-check matrix **H**. (c) The Tanner graph; each CN represents an SPC erasure decoder. The six CNs correspond to the six rows of **H** and the rows and columns of the product code array. (d) An iterative decoding example on the BEC.

We can continue to expand this point of view of message passing with a crossword-puzzle analogy. In solving a crossword puzzle, as one iterates between "Across" words and "Down" words, the answers in one direction tend increasingly to help one to find the answers in the other direction. Certainly it is more difficult to first find all of the "Across" words and then all of the "Down" words than it is to iterate back and forth, receiving clues (messages) from intersecting words.

We consider a final illustrative example of the idea of message passing in a distributed-processor system. This example allows us to introduce the important notion of extrinsic information. It also makes clear (without proof) the fact that message-passing decoding for a collection of *constituent decoders* arranged in a graph is optimal provided that the graph contains no cycles, but it is not optimal for graphs with cycles. The example is the well-known soldier-counting problem in the Bayesian-inference literature [8].

Consider Figure 5.4(a), which depicts six soldiers in a linear formation. The goal is for each of the soldiers to learn the total number of soldiers present by counting in a distributed fashion (imagine that there is a large number of soldiers). The counting rules are as follows. Each soldier receives a number from one side, adds

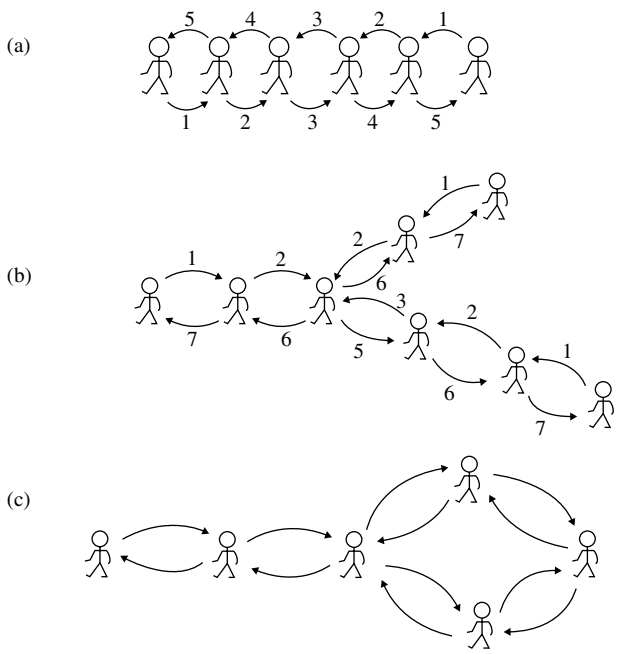


Figure 5.4 Distributed soldier counting. (After Pearl [8].) (a) Soldiers in a line. (b) Soldiers in a tree formation. (c) Soldiers in a formation containing a cycle.

one for himself, and passes the sum to the other side. The soldiers on the ends receive a zero from the side with no neighbor. The sum of the number that a soldier receives from one side and the number the soldier passes to that side is equal to the total number of soldiers, as can be verified in Figure 5.4(a). Of course, this sum is meaningful only on one side for the soldiers at the ends.

Consider now the simple tree formation in Figure 5.4(b). Observe that the third soldier from the left receives messages from three neighboring soldiers and so the rules need to be modified for this more general situation. The modified rule set reduces to the above rule set for the special case of a linear formation. The modified counting rules are as follows. The message that an arbitrary soldier X passes to arbitrary neighboring soldier Y is equal to the sum of all incoming messages, plus one for soldier X, minus the message that soldier Y had just sent to soldier X. The soldiers on the ends receive a zero from the side with no neighbor. The sum of the number that a soldier receives from any one of his neighbors plus the number that the soldier passes to that neighbor is equal to the total number of soldiers, as can be verified in Figure 5.4(b).

This message-passing rule introduces the concept of *extrinsic information*. The idea is that a soldier does not pass to a neighboring soldier any information that the neighboring soldier already has, that is, only extrinsic information is passed. It is for this reason that soldier Y receives the sum total of messages that soldier X has available *minus* the information that soldier Y already has. We say that soldier X passes to soldier Y only extrinsic information, which may be computed as

$$\begin{aligned} I_{X\to Y} &= \sum_{Z\in N(X)} I_{Z\to X} - I_{Y\to X} + I_X \\ &= \sum_{Z\in N(X)-\{Y\}} I_{Z\to X} + I_X, \end{aligned} \tag{5.10}$$

where $N(X)$ is the set of neighbors of soldier X, $I_{X\to Y}$ is the extrinsic information sent from soldier X to soldier Y (and similarly for $I_{Z\to X}$ and $I_{Y\to X}$), and I_X is sometimes called the *intrinsic information*. I_X is the "one" that a soldier counts for himself, so $I_X = 1$ for this example.

Finally, we consider the formation of Figure 5.4(c), which contains a cycle. The reader can easily verify that the situation is untenable: no matter what counting rule one may devise, the cycle represents a type of positive feedback, both in the clockwise and in the counter-clockwise direction, so that the messages passed within the cycle will increase without bound as the trips around the cycle continue. This example demonstrates that message passing on a graph cannot be claimed to be optimal if the graph contains one or more cycles. However, while most practical codes contain cycles, it is well known that message-passing decoding performs very

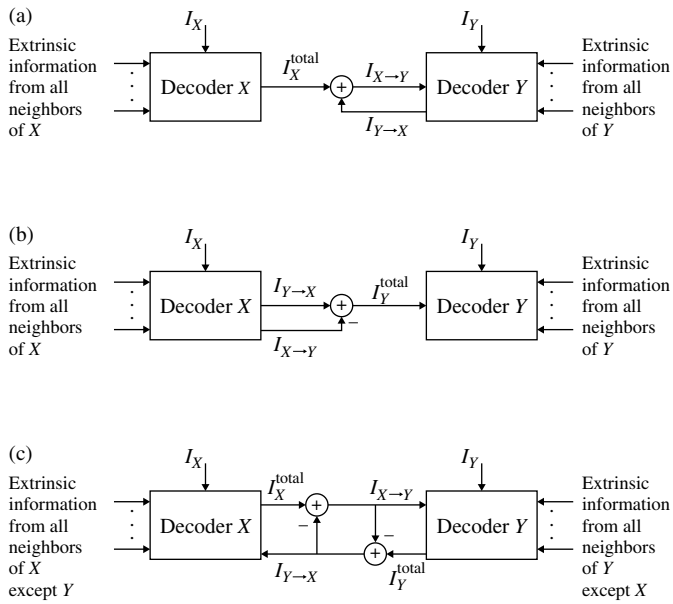


Figure 5.5 A depiction of the turbo principle for concatenated coding schemes. (a) Extrinsic information $I_{X\to Y}$ from decoder X to decoder Y. (b) Extrinsic information $I_{Y\to X}$ from decoder Y to decoder X. (c) A compact symmetric combination of (a) and (b).

well for properly designed codes for most error-rate ranges of interest. It is for this reason that we are interested in message-passing decoding.

The notion of extrinsic-information passing described above has been called the *turbo principle* in the context of the iterative decoding of concatenated codes in communication channels. A depiction of the turbo principle is contained in Figure 5.5. In the figure, $I_X^{\text{total}} = \sum_{Z\in N(X)} I_{Z\to X} + I_X$, and similarly for I_Y^{total}. I_X and I_Y represent intrinsic information received from the channel. Note that Figure 5.5 is simply a block diagram of Equation (5.10), which was discussed in connection with Figure 5.4(b). We remark that addition and subtraction operations in (5.10) can be generalized operations, such as the "box-plus" ($\boxplus$) operation introduced later in this chapter.

For simplicity, our discussion of soldier counting omitted the details of the actual iterative counting procedure: the values given in Figure 5.4 represent the values after the procedure had already reached convergence. The top sequence of five numbers in Figure 5.4(a) would be initialized to all zeros and then be updated as $(1,1,1,1,1) \to (2,2,2,2,1) \to (3,3,3,2,1) \to (4,4,3,2,1) \to (5,4,3,2,1)$; updating stops when no numbers change. The sequence of five numbers at the bottom of Figure 5.4(a) would be computed in a similar fashion (the updated sequences are simply the reverse of those just given). The situation is nearly identical for iterative

decoding schemes with multiple decoders. Thus, the message being passed from decoder X to decoder Y in Figure 5.5(a) will happen multiple times in practice, although the timing depends on the selected *scheduling scheme* which coordinates the message passing among the various decoders. The most commonly used flooding schedule will be discussed in the next section.

5.4 The Sum–Product Algorithm

In this section we derive the sum–product algorithm for general memoryless binary-input channels, applying the turbo principle in our development. As special cases, we consider the BEC, the BSC, the BI-AWGNC, and the independent Rayleigh fading channel. We first present an overview of the SPA, after which the so-called log-domain version of the SPA is developed.

5.4.1 Overview

In addition to introducing LDPC codes in his seminal work in 1960, Gallager also provided a near-optimal decoding algorithm that is now called the sum–product algorithm. This algorithm is also sometimes called the belief-propagation algorithm (BPA), a name taken from the Bayesian-inference literature, where the algorithm was derived independently [8]. The algorithm is identical for all memoryless channels, so our development will be general.

The optimality criterion underlying the development of the SPA decoder is symbol-wise maximum *a posteriori* (MAP). Much like optimal symbol-by-symbol decoding of trellis-based codes (i.e., the BCJR algorithm), we are interested in computing the *a posteriori* probability (APP) that a specific bit in the transmitted codeword $\mathbf{v} = [v_0 \ v_1 \ \ldots \ v_{n-1}]$ equals 1, given the received word $\mathbf{y} = [y_0 \ y_1 \ \ldots \ y_{n-1}]$. Without loss of generality, we focus on the decoding of bit v_j, so that we are interested in computing the APP,

$$\Pr(v_j = 1|\mathbf{y}),$$

the APP ratio (also called the likelihood ratio, LR),

$$l(v_j|\mathbf{y}) \triangleq \frac{\Pr(v_j = 0|\mathbf{y})}{\Pr(v_j = 1|\mathbf{y})},$$

or the more numerically stable log-APP ratio, also called the log-likelihood ratio (LLR),

$$L(v_j|\mathbf{y}) \triangleq \log\left(\frac{\Pr(v_j = 0|\mathbf{y})}{\Pr(v_j = 1|\mathbf{y})}\right).$$

Here and in that which follows, the natural logarithm is assumed for LLRs.

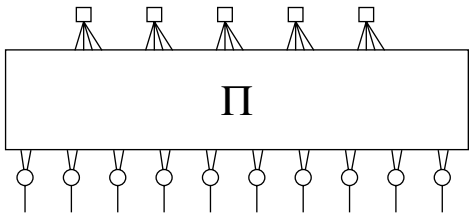


Figure 5.6 Graphical representation of an LDPC code as a concatenation of SPC and REP codes. "π" represents an inter leaver.

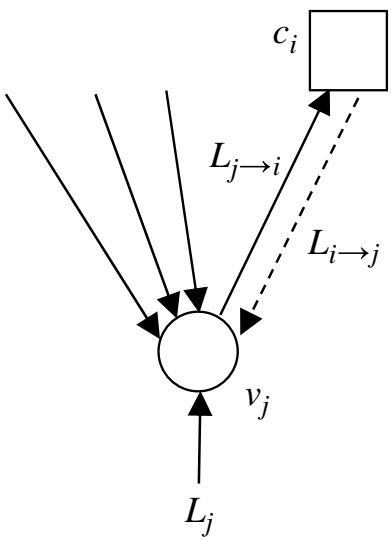


Figure 5.7 A VN decoder (REP code decoder). VN j receives LLR information from the channel and from all of its neighbors, excluding message $L_{i\to j}$ from CN i, from which VN j composes message $L_{j\to i}$ that is sent to CN i.

The SPA for the computation of $\Pr(v_j = 1|\mathbf{y})$, $l(v_j|\mathbf{y})$, or $L(v_j|\mathbf{y})$ is a distributed algorithm that is an application of the turbo principle to a code's Tanner graph. This is more easily seen in Figure 5.6, which presents a different rendition of the Tanner graph of Figure 5.1. Figure 5.6 makes more obvious the notion that an LDPC code can be deemed a collection of SPC codes concatenated through an interleaver to a collection of repetition (REP) codes. Further, the SPC codes are treated as outer codes, that is, they are not connected to the channel. The edges dangling from the VNs in Figure 5.6 are connected to the channel. We can apply the turbo principle of Figure 5.5 to this concatenation of SPC and REP codes as follows.

Figure 5.7 depicts the REP (VN) decoder situation for a VN with degree greater than the degree-2 VNs of Figure 5.6. Note that the VN j decoder receives LLR

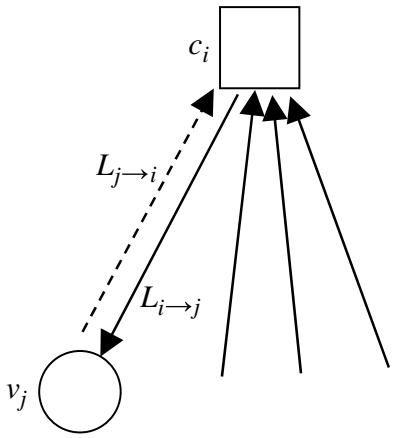


Figure 5.8 A CN decoder (SPC decoder). CN i receives LLR information from all of its neighbors, excluding message $L_{j\to i}$ from VN j, from which CN i composes message $L_{i\to j}$ that is sent to VN j.

information both from the channel and from its neighbors. However, in the computation of the extrinsic information $L_{j\to i}$, VN j need not receive $L_{i\to j}$ from CN i since it would be subtracted out anyway per Equation (5.10) or Figure 5.5. Figure 5.8 depicts the SPC (CN) decoder situation. Similarly to the VN case, in the computation of $L_{i\to j}$ for some specific j, CN i need not receive $L_{j\to i}$, for its impact would be subtracted out anyway.

The VN and CN decoders work cooperatively and iteratively to estimate $L(v_j|\mathbf{y})$ for $j = 0, 1, \ldots, n-1$. Throughout our development, we shall assume that the *flooding schedule* is employed. According to this schedule, all VNs process their inputs and pass extrinsic information up to their neighboring check nodes; the check nodes then process their inputs and pass extrinsic information down to their neighboring variable nodes; and the procedure repeats, starting with the variable nodes. After a preset maximum number of repetitions (or *iterations*) of this VN/CN decoding round, or after some stopping criterion has been met, the decoder computes (estimates) the LLRs $L(v_j|\mathbf{y})$ from which decisions on the bits v_j are made. When the cycles are large, the estimates will be very accurate and the decoder will have near-optimal (MAP) performance. The development of the SPA below relies on the following *independence assumption*: the LLR quantities received at each node from its neighbors are independent. Clearly this breaks down when the number of iterations exceeds half of the Tanner graph's girth.

So far, we have provided in Figures 5.7 and 5.8 conceptual depictions of the functions of the CN and VN decoders, and we have described the distributed decoding schedule in the context of Figure 5.6. At this point, we need to develop the detailed operations within each constituent (CN and VN) decoder. That is, we digress a bit to develop the MAP decoders and APP processors for REP and SPC codes. These decoders are integral to the overall LDPC decoder which will be discussed subsequently.

5.4.2 Repetition Code MAP Decoder and APP Processor

Consider a REP code in which the binary code symbol $c \in \{0, 1\}$ is transmitted over a memoryless channel d times so that the d-vector $\mathbf{r}$ is received. By definition, the MAP decoder computes the log-APP ratio

$$L(c|\mathbf{r}) = \log\left(\frac{\Pr(c = 0|\mathbf{r})}{\Pr(c = 1|\mathbf{r})}\right)$$

or, under an equally likely assumption for the value of c, $L(c|\mathbf{r})$ is the LLR

$$L(c|\mathbf{r}) = \log\left(\frac{\Pr(\mathbf{r}|c = 0)}{\Pr(\mathbf{r}|c = 1)}\right).$$

This simplifies as

$$\begin{aligned} L(c|\mathbf{r}) &= \log\left(\frac{\prod_{l=0}^{d-1} \Pr(r_l|c = 0)}{\prod_{l=0}^{d-1} \Pr(r_l|c = 1)}\right) \\ &= \sum_{l=0}^{d-1} \log\left(\frac{\Pr(r_l|c = 0)}{\Pr(r_l|c = 1)}\right) \\ &= \sum_{l=0}^{d-1} L(r_l|x)\,, \end{aligned}$$

where $L(r_l|x)$ is obviously defined. Thus, the MAP receiver for a REP code computes the LLRs for each channel output r_l and adds them. The MAP decision is $\hat{c} = 0$ if $L(c|\mathbf{r}) \geq 0$ and $\hat{c} = 1$ otherwise.

In the context of LDPC decoding, the above expression is adapted to compute the extrinsic information to be sent from VN j to CN i (cf. Figure 5.7),

$$L_{j\to i} = L_j + \sum_{i' \in N(j)-\{i\}} L_{i'\to j}. \tag{5.11}$$

The quantity L_j in this expression is the LLR value computed from the channel sample y_j,

$$L_j = L(c_j|y_j) = \log\left(\frac{\Pr(c_j = 0|y_j)}{\Pr(c_j = 1|y_j)}\right).$$

Note that the update equation (5.11) follows the format of (5.10). In the context of LDPC decoding, we call the VN an *APP processor* instead of a MAP decoder. At the last iteration, VN j produces a decision based on

$$L_j^{\text{total}} = L_j + \sum_{i \in N(j)} L_{i\to j}. \tag{5.12}$$

5.4.3 Single-Parity-Check Code MAP Decoder and APP Processor

To develop the MAP decoder for an SPC code we first need the following result due to Gallager.

Lemma 5.1. *Consider a vector of d independent binary random variables $\mathbf{a} = [a_0, a_1, \ldots, a_{d-1}]$ in which $\Pr(a_l = 1) = p_1^{(l)}$ and $\Pr(a_l = 0) = p_0^{(l)}$. Then the probability that $\mathbf{a}$ contains an even number of 1s is*

$$\frac{1}{2} + \frac{1}{2}\prod_{l=0}^{d-1}\left(1 - 2p_1^{(l)}\right) \tag{5.13}$$

and the probability that $\mathbf{a}$ contains an odd number of 1s is

$$\frac{1}{2} - \frac{1}{2}\prod_{l=0}^{d-1}\left(1 - 2p_1^{(l)}\right). \tag{5.14}$$

Proof. The proof is by induction on d. (Problem 5.9.) □

Armed with this result, we now derive the MAP decoder for SPC codes. Consider the transmission of a length-d SPC codeword $\mathbf{c}$ over a memoryless channel whose output is $\mathbf{r}$. The bits c_l in the codeword $\mathbf{c}$ have a single constraint: there must be an even number of 1s in $\mathbf{c}$. Without loss of generality, we focus on bit c_0, for which the MAP decision rule is

$$\hat{c}_0 = \arg\max_{b\in\{0,1\}} \Pr(c_0 = b|\mathbf{r},\ \text{SPC}),$$

where the conditioning on SPC is a reminder that there is an SPC constraint imposed on $\mathbf{c}$. Consider now

$$\begin{aligned}\Pr\{c_0 = 0|\mathbf{r},\ \text{SPC}\} &= \Pr\{c_1, c_2, \ldots, c_{d-1} \text{ has an even no. of 1s}|\mathbf{r}\}\\ &= \frac{1}{2} + \frac{1}{2}\prod_{l=1}^{d-1}(1 - 2\Pr(c_l = 1|r_l)),\end{aligned}$$

where the second line follows from (5.13). Rearranging gives

$$1 - 2\Pr(c_0 = 1|\mathbf{r},\ \text{SPC}) = \prod_{l=1}^{d-1}(1 - 2\Pr(c_l = 1|r_l)), \tag{5.15}$$

where we used the fact that $p(c_0 = 0|\mathbf{r},\ \text{SPC}) = 1 - p(c_0 = 1|\mathbf{r},\ \text{SPC})$. We can change this to an LLR representation using the easily proven relation for a generic binary random variable with probabilities p_1 and p_0,

$$1 - 2p_1 = \tanh\left(\frac{1}{2}\log\left(\frac{p_0}{p_1}\right)\right) = \tanh\left(\frac{1}{2}\text{LLR}\right), \tag{5.16}$$

where here LLR $= \log(p_0/p_1)$. Applying this relation to (5.15) gives

$$\tanh\left(\frac{1}{2}L(c_0|\mathbf{r},\ \text{SPC})\right) = \prod_{l=1}^{d-1} \tanh\left(\frac{1}{2}L(c_l|r_l)\right)$$

or

$$L(c_0|\mathbf{r},\ \text{SPC}) = 2\tanh^{-1}\left(\prod_{l=1}^{d-1} \tanh\left(\frac{1}{2}L(c_l|r_l)\right)\right). \tag{5.17}$$

Thus, the MAP decoder for bit c_0 in a length-d SPC code makes the decision $\hat{c}_0 = 0$ if $L(c_0|\mathbf{r},\ \text{SPC}) \geq 0$ and $\hat{c}_0 = 1$ otherwise.

In the context of LDPC decoding, following (5.17), when the CNs function as APP processors instead of MAP decoders, CN i computes the extrinsic information

$$L_{i\to j} = 2\tanh^{-1}\left(\prod_{j'\in N(i)-\{j\}} \tanh\left(\frac{1}{2}L_{j'\to i}\right)\right) \tag{5.18}$$

and transmits it to VN j. Note that, because the product is over the set $N(i) - \{j\}$ (cf. Figure 5.8), the message $L_{j\to i}$ has in effect been subtracted out to obtain the extrinsic information $L_{i\to j}$. Ostensibly, this update equation does not follow the format of (5.10), but we will show in Section 5.4.5 that it can essentially be put in that format for an appropriately defined addition operation.

5.4.4 The Gallager SPA Decoder

Equations (5.11) and (5.18) form the core of the SPA decoder. The information $L_{j\to i}$ that VN j sends to CN i at each iteration is the best (extrinsic) estimate of the value of v_j (the sign bit of $L_{j\to i}$) and the confidence or reliability level of that estimate (the magnitude of $L_{j\to i}$). This information is based on the REP constraint for VN j and all inputs from the neighbors of VN j, excluding CN i. Similarly, the information $L_{i\to j}$ that CN i sends to VN j at each iteration is the best (extrinsic) estimate of the value of v_j (sign bit of $L_{j\to i}$) and the confidence or reliability level of that estimate (magnitude of $L_{j\to i}$). This information is based on the SPC constraint for CN i and all inputs from the neighbors of CN i, excluding VN j.

While (5.11) and (5.18) form the core of the SPA decoder, we still need an iteration-stopping criterion and an initialization step. A standard stopping criterion is to stop iterating when $\hat{\mathbf{v}}\mathbf{H}^{\mathrm{T}} = \mathbf{0}$, where $\hat{\mathbf{v}}$ is a tentatively decoded codeword.

The decoder is initialized by setting all VN messages $L_{j\to i}$ equal to

$$L_j = L(v_j|y_j) = \log\left(\frac{\Pr(v_j = 0|y_j)}{\Pr(v_j = 1|y_j)}\right), \tag{5.19}$$

for all j, i for which $h_{ij} = 1$. Here, y_j represents the channel value that was actually received, that is, it is not a variable here. We consider the following special cases.

BEC. In this case, $y_j \in \{0, 1, e\}$ and we define $p = \Pr(y_j = e|v_j = b)$ to be the erasure probability, where $b \in \{0, 1\}$. Then it is easy to see that

$$\Pr\left(v_j = b|y_j\right) = \begin{cases} 1 & \text{when } y_j = b, \\ 0 & \text{when } y_j = b^c, \\ 1/2 & \text{when } y_j = e, \end{cases}$$

where b^c represents the complement of b. From this, it follows that

$$L(v_j|y_j) = \begin{cases} +\infty, \ y_j = 0, \\ -\infty, \ y_j = 1, \\ \ \ 0, \ y_j = e. \end{cases}$$

BSC. In this case, $y_j \in \{0, 1\}$ and we define $\varepsilon = \Pr(y_j = b^c|v_j = b)$ to be the error probability. Then it is obvious that

$$\Pr(v_j = b|y_j) = \begin{cases} 1 - \varepsilon & \text{when } y_j = b, \\ \varepsilon & \text{when } y_j = b^c. \end{cases}$$

From this, we have

$$L(v_j|y_j) = (-1)^{y_j} \log\left(\frac{1 - \varepsilon}{\varepsilon}\right).$$

Note that knowledge of ε is necessary.

BI-AWGNC. We first let $x_j = (-1)^{v_j}$ be the jth transmitted binary value; note $x_j = +1(-1)$ when $v_j = 0(1)$. We shall use x_j and v_j interchangeably hereafter. The jth received sample is $y_j = x_j + n_j$, where the n_j are independent and normally distributed as $\mathcal{N}\big(0, \sigma^2\big)$. Then it is easy to show that

$$\Pr\left(x_j = x|y_j\right) = \left[1 + \exp\big(-2y_j x/\sigma^2\big)\right]^{-1},$$

where $x \in \{\pm 1\}$ and, from this, that

$$L(v_j|y_j) = 2y_j/\sigma^2.$$

In practice, an estimate of σ^2 is necessary.

Rayleigh. Here we assume an independent Rayleigh fading channel and that the decoder has perfect channel state information. The model is similar to that of the AWGNC: $y_j = \alpha_j x_j + n_j$, where $\{\alpha_j\}$ are independent Rayleigh random variables with unity variance. The channel transition probability in this

case is

$$\Pr(x_j = x|y_j) = \left[1 + \exp\left(-2\alpha_j y_j x/\sigma^2\right)\right]^{-1}.$$

Then the variable nodes are initialized by

$$L(v_j|y_j) = 2\alpha_j y_j/\sigma^2.$$

In practice, estimates of α_j and σ^2 are necessary.

Algorithm 5.1 The Gallager Sum–Product Algorithm

1. **Initialization:** For all j, initialize L_j according to (5.19) for the appropriate channel model. Then, for all i, j for which $h_{ij} = 1$, set $L_{j\to i} = L_j$.
2. **CN update:** Compute outgoing CN messages $L_{i\to j}$ for each CN using (5.18),

$$L_{i\to j} = 2\tanh^{-1}\left(\prod_{j'\in N(i)-\{j\}} \tanh\left(\frac{1}{2}L_{j'\to i}\right)\right),$$

   and then transmit to the VNs. (This step is shown diagrammatically in Figure 5.8.)
3. **VN update:** Compute outgoing VN messages $L_{j\to i}$ for each VN using Equation (5.11),

$$L_{j\to i} = L_j + \sum_{i'\in N(j)-\{i\}} L_{i'\to j},$$

   and then transmit to the CNs. (This step is shown diagrammatically in Figure 5.7.)
4. **LLR total:** For $j = 0, 1, \ldots, n-1$ compute

$$L_j^{\text{total}} = L_j + \sum_{i\in N(j)} L_{i\to j}.$$

5. **Stopping criteria:** For $j = 0, 1, \ldots, n-1$, set

$$\hat{v}_j = \begin{cases} 1 & \text{if } L_j^{\text{total}} < 0, \\ 0 & \text{else,} \end{cases}$$

   to obtain $\hat{\mathbf{v}}$. If $\hat{\mathbf{v}}\mathbf{H}^{\mathrm{T}} = \mathbf{0}$ or the number of iterations equals the maximum limit, stop; else, go to Step 2.

Remarks

1. The origin of the name ‘‘sum–product’’ is not evident from this “log-domain” version of the sum–product algorithm (log-SPA). We refer the reader to [9] for a more general discussion of this algorithm and its applications.
2. For good codes, this algorithm is able to detect an uncorrected codeword with near-unity probability (in Step 5).
3. As mentioned, the SPA assumes that the messages passed are statistically independent throughout the decoding process. When the y_j are independent, this independence assumption would hold true if the Tanner graph possessed no cycles. Further, the SPA would yield exact LLRs in this case. However, for a graph of girth γ, the independence assumption is true only up to the $(\gamma/2)$th iteration, after which messages start to loop back on themselves in the graph's various cycles.
4. This algorithm has been presented for pedagogical clarity, but may be adjusted to optimize the number of computations. For example, fewer operations are required if Step 4 is performed before Step 3, and Step 3 is modified as $L_{j\to i} = L_j^{\text{total}} - L_{i\to j}$.
5. This algorithm can be simplified further for the BEC and BSC channels since the initial LLRs are ternary in the first case and binary in the second case. In fact, the algorithm for the BEC is precisely the iterative erasure filling algorithm of Example 5.2 and Section 5.7.1, as will be shown when discussing the min-sum decoder below.

The update equation (5.18) for Step 2 is numerically challenging due to the presence of the product and the tanh and $\tanh^{-1}$ functions. Following Gallager, we can improve the situation as follows. First, factor $L_{j\to i}$ into its sign and magnitude (or *bit value* and *bit reliability*):

$$\begin{aligned} L_{j\to i} &= \alpha_{ji}\beta_{ji}, \\ \alpha_{ji} &= \operatorname{sign}(L_{j\to i}), \\ \beta_{ji} &= |L_{j\to i}|, \end{aligned}$$

so that (5.18) may be rewritten as

$$\tanh\left(\frac{1}{2}L_{i\to j}\right) = \prod_{j'\in N(i)-\{j\}} \alpha_{j'i} \cdot \prod_{j'\in N(i)-\{j\}} \tanh\left(\frac{1}{2}\beta_{j'i}\right).$$

We then have

$$\begin{aligned} L_{i\to j} &= \prod_{j'} \alpha_{j'i} \cdot 2\tanh^{-1}\left(\prod_{j'} \tanh\left(\frac{1}{2}\beta_{j'i}\right)\right) \\ &= \prod_{j'} \alpha_{j'i} \cdot 2\tanh^{-1}\log^{-1}\log\left(\prod_{j'} \tanh\left(\frac{1}{2}\beta_{j'i}\right)\right) \\ &= \prod_{j'} \alpha_{j'i} \cdot 2\tanh^{-1}\log^{-1}\sum_{j'} \log\left(\tanh\left(\frac{1}{2}\beta_{j'i}\right)\right), \end{aligned}$$

which yields a new form for (5.18), Step 2,

$$\textbf{CN update: } L_{i\to j} = \prod_{j'\in N(i)-\{j\}} \alpha_{j'i} \cdot \phi\left(\sum_{j'\in N(i)-\{j\}} \phi\big(\beta_{j'i}\big)\right), \tag{5.20}$$

where we have defined

$$\phi(x) = -\log[\tanh(x/2)] = \log\left(\frac{e^x+1}{e^x-1}\right)$$

and used the fact that $\phi^{-1}(x) = \phi(x)$ when $x > 0$. Thus, Equation (5.20) may be used instead of (5.18) in Step 2 of the SPA presented above. The function $\phi(x)$, shown in Figure 5.9, may be implemented by use of a look-up table. However, if a very low error rate is a requirement, a table-based decoder can suffer from an "error-rate floor" (see Section 5.4.6 for a discussion of "floors"). One possible alternative to a table is a piecewise linear approximation as discussed in [18] and [19]. Other alternatives are presented in the next subsection and in Section 5.5.

5.4.5 The Box-Plus SPA Decoder

Implementation of the $\phi(x)$ function by use of a look-up table is sufficient for some software simulations, but for most hardware applications it is not. The reason is that, due to dynamic-range issues, it is a difficult function to approximate, even with a large table. Thus, hardware implementations of the $\phi(x)$ function generally suffer from performance degradations, especially in the error-rate floor region. In this section, we approach check-node processing from a slightly different perspective to yield an alternative SPA decoder. This alternative decoder is preferable in some applications, and it leads to certain low-complexity approx-

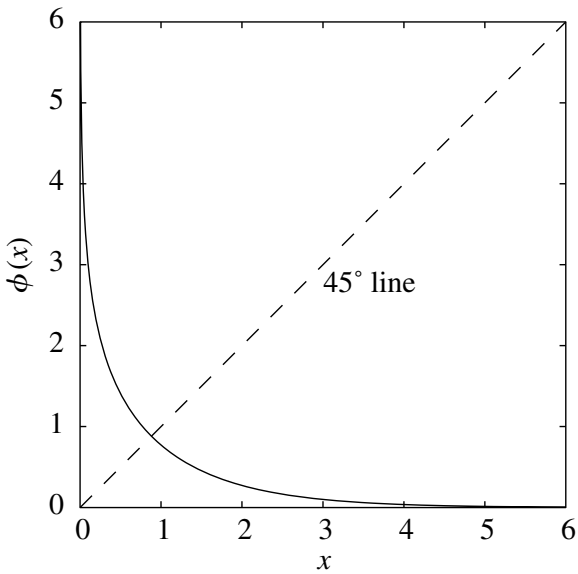


Figure 5.9 A plot of the $\phi(x)$ function.

imations as discussed later in the section. We first need the following general result.

Lemma 5.2. *Consider two independent binary random variables a_1 and a_2 with probabilities $\Pr(a_l = b) = p_b^{(l)}$, $b \in \{0, 1\}$, and LLRs $L_l = L(a_l) = \log\left(p_0^{(l)}/p_1^{(l)}\right)$. The LLR of the binary sum $A_2 = a_1 \oplus a_2$, defined as*

$$L(A_2) = \log\left(\frac{\Pr(A_2 = 0)}{\Pr(A_2 = 1)}\right),$$

is given by

$$L(A_2) = \log\left(\frac{1 + e^{L_1 + L_2}}{e^{L_1} + e^{L_2}}\right). \tag{5.21}$$

Proof. To see this, observe that

$$\begin{aligned} L(A_2) &= \log\left(\frac{\Pr(a_1 \oplus a_2 = 0)}{\Pr(a_1 \oplus a_2 = 1)}\right) \\ &= \log\left(\frac{p_0^{(1)} p_0^{(2)} + p_1^{(1)} p_1^{(2)}}{p_0^{(1)} p_1^{(2)} + p_1^{(1)} p_0^{(2)}}\right) \end{aligned}$$

$$= \log\left(\frac{1 + \frac{p_0^{(1)}}{p_1^{(1)}}\frac{p_0^{(2)}}{p_1^{(2)}}}{\frac{p_0^{(1)}}{p_1^{(1)}} + \frac{p_0^{(2)}}{p_1^{(2)}}}\right)$$
$$= \log\left(\frac{1 + e^{L_1+L_2}}{e^{L_1} + e^{L_2}}\right).$$

□

This result may be directly applied to the computation of a CN output with $d_c - 1 = 2$ inputs because CN outputs are LLRs of mod-2 sums of binary r.v.s. Thus, a CN output L_{out} with inputs L_1 and L_2 is given by the expression on the right-hand side of (5.21). Moreover, to be consistent with (5.18), we must have

$$L_{\text{out}} = L(A_2) = 2\tanh^{-1}\left(\prod_{l=1}^{2}\tanh(L_l/2)\right).$$

If more than two independent binary random variables are involved, as is the case when $d_c > 3$, then the LLR of the sum of these random variables may be computed by repeated application of (5.21). For example, the LLR of $A_3 = a_1 \oplus a_2 \oplus a_3$ may be computed via $A_3 = A_2 \oplus a_3$, yielding

$$L(A_3) = \log\left(\frac{1 + e^{L(A_2)+L_3}}{e^{L(A_2)} + e^{L_3}}\right).$$

As a shorthand, we will write $L_1 \boxplus L_2$ to denote the computation of $L(A_2) = L(a_1 \oplus a_2)$ from L_1 and L_2; that is,

$$L_1 \boxplus L_2 = \log\left(\frac{1 + e^{L_1+L_2}}{e^{L_1} + e^{L_2}}\right). \tag{5.22}$$

Similarly, $L_1 \boxplus L_2 \boxplus L_3$ will denote the computation of $L(A_3) = L(a_1 \oplus a_2 \oplus a_3)$ from L_1, L_2, and L_3; and so on for more variables. $\boxplus$ is called the "box-plus" operator. The box-plus counterpart to (5.18) or (5.20) is

$$\textbf{CN update: } L_{i\to j} = \boxplus_{j'\in N(i)-\{j\}} L_{j'\to i}, \tag{5.23}$$

where $\boxplus$ represents the summation operator for the binary operator $\boxplus$. Observe that this is essentially in the form of (5.10), except that $I_X = 0$ in (5.10) because the CNs are not directly connected to the channel. Observe also the symmetry with the VN update equation (which also follows (5.10)), which we reproduce here to emphasize the point:

$$\textbf{VN update: } L_{j\to i} = L_j + \sum_{i'\in N(j)-\{i\}} L_{i'\to j}.$$

The expression (5.23) may be computed via repeated application of Lemma 5.2. For example, representing the binary operator $\boxplus$ as the two-input function

$$\mathcal{B}(a,b) = 2\tanh^{-1}(\tanh(a/2)\tanh(b/2)), \tag{5.24}$$

$L_1 \boxplus L_2 \boxplus L_3 \boxplus L_4$ would be computed as

$$\mathcal{B}(L_1, \mathcal{B}(L_2, \mathcal{B}(L_3, L_4))).$$

$\mathcal{B}(a,b)$ can be implemented by use of a two-input look-up table.

A degree-d_c CN processor has d_c inputs $\{L_\ell\}$ and d_c outputs $\{L_{out,\ell}\}$, where, ostensibly, each output must be computed using $d_c - 1$ $\boxplus$ operations, for a total of $d_c(d_c-1)$ $\boxplus$ operations. The d_c outputs may be computed more efficiently with a total of only $3(d_c-2)$ $\boxplus$ operations as follows. Let $F_1 = L_1$ and, for $\ell = 2$ to $d_c - 1$, let $F_\ell = F_{\ell-1} \boxplus L_\ell$. Next, let $B_{d_c} = L_{d_c}$ and, for $\ell = d_c - 1$ to 2, let $B_\ell = B_{\ell+1} \boxplus L_\ell$. Now the outputs can be computed from the forward-going and backward-going partial "sums" as follows: $L_{\text{out},1} = B_2$, $L_{\text{out},d_c} = F_{d_c-1}$, and for $\ell = 2$ to $d_c - 1$, $L_{\text{out},\ell} = F_{\ell-1} \boxplus B_{\ell+1}$.

5.4.6 Comments on the Performance of the SPA Decoder

In contrast with the error-rate curves for classical codes – e.g., Reed–Solomon codes with an algebraic decoder or convolutional codes with a Viterbi decoder – the error-rate curves for iteratively decoded codes generally have a region in which the slope *decreases* as the channel SNR increases (or, for a BSC, as ε decreases). An example of such of an error-rate curve for the BI-AWGNC appears in Figure 5.11 in association with Example 5.4 below. The region of the curve just before the slope transition region is called the *waterfall region* of the error-rate curve and the region of the curve with the reduced slope is called the *error-rate floor region*, or simply the *floor region*.

A floor seen in the performance curve of an iteratively decoded code is occasionally attributable to a small minimum distance, particularly when the code is a parallel turbo code (see Chapter 7). However, it is possible to have a floor in the error-rate curve of an LDPC code (or serial turbo code) with a large minimum distance. In this case, the floor is attributable to so-called trapping sets. An (ω, ν) *trapping set* is a set of ω VNs that induce a subgraph with ν odd-degree checks so that, when the ω bits are all in error, there will be ν failed parity checks. An essentially equivalent notion is a near-codeword. An (ω, ν) *near-codeword* is a length-n error pattern of weight ω that results in ν check failures (i.e., the syndrome weight

is ν), where ω and ν are "small." Near-codewords tend to lead to error situations from which the SPA decoder (and its approximations) cannot escape. The implication of a small ω is that the error pattern is more likely; the implication of a small ν is that only a few check equations are affected by the pattern, making it more likely to escape the notice of the iterative decoder. Iterative decoders are susceptible to trapping sets (near-codewords) since an iterative decoder works locally in a distributed-processing fashion, unlike an ML decoder, which finds the globally optimum solution. The floor seen in Example 5.4 is due to a trapping-set-induced graph, as discussed in that example.

Chapter 8 discusses the enumeration of trapping sets in LDPC code ensembles. Chapter 15 discusses the enumeration of trapping sets in specific LDPC codes and estimating the level of trapping-set-induced floors on the BI-AWGN channel and binary-input partial response channels. Chapter 15 also discusses how iterative decoders may be modified to lower the floor by orders of magnitude.

5.5 Reduced-Complexity SPA Approximations

Problem 5.10 presents the so-called probability-domain decoder. It should be clear from the above that the log-SPA decoder has lower complexity and is more numerically stable than the probability-domain SPA decoder. We now present decoders of even lower complexity, which often suffer only a little in terms of performance. As should be clear, the focus of these reduced-complexity approximate-SPA decoders is on the complex check-node processor. The amount of degradation of a given algorithm is a function of the code and the channel, as demonstrated in the examples that follow.

5.5.1 The Min-Sum Decoder

Consider the update equation (5.20) for $L_{i\to j}$ in the log-domain SPA decoder. Note from the shape of $\phi(x)$ that the largest term in the sum corresponds to the smallest β_{ji} so that, assuming that this term dominates the sum,

$$\begin{aligned}\phi\left(\sum_{j'}\phi\big(\beta_{j'i}\big)\right) &\simeq \phi\left(\phi\left(\min_{j'}\beta_{j'i}\right)\right)\\ &= \min_{j'\in N(i)-\{j\}}\beta_{j'i}.\end{aligned}$$

Thus, the min-sum algorithm is simply the log-domain SPA with Step 2 replaced by

$$\textbf{CN update: } L_{i\to j} = \prod_{j'\in N(i)-\{j\}}\alpha_{j'i}\cdot\min_{j'\in N(i)-\{j\}}\beta_{j'i}.$$

It can also be shown that, in the AWGNC case, the initialization $L_{j\to i} = 2y_j/\sigma^2$ may be replaced by $L_{j\to i} = y_j$ when the min-sum algorithm is employed (Problem 5.8). The advantage, of course, is that an estimate of the noise power σ^2 is unnecessary in this case.

Example 5.3. Note that, for the BEC, $\phi\left(\sum_{j'} \phi(\beta_{j'i})\right)$ is 0 or ∞ exactly when $\min_{j'} \beta_{j'i}$ is 0 or ∞, respectively. That is, the log-SPA and min-sum algorithms are identical on the BEC. Figure 5.10 repeats the (3,2) $\times$ (3,2) SPC product-code decoding example of Figure 5.3 using the min-sum decoder. Note that the result of the min operation is either 0 or ∞. Note also that this is equivalent to the iterative erasure filling algorithm of Figure 5.3. This erasure decoding algorithm will be discussed in further generality in Section 5.7.1.

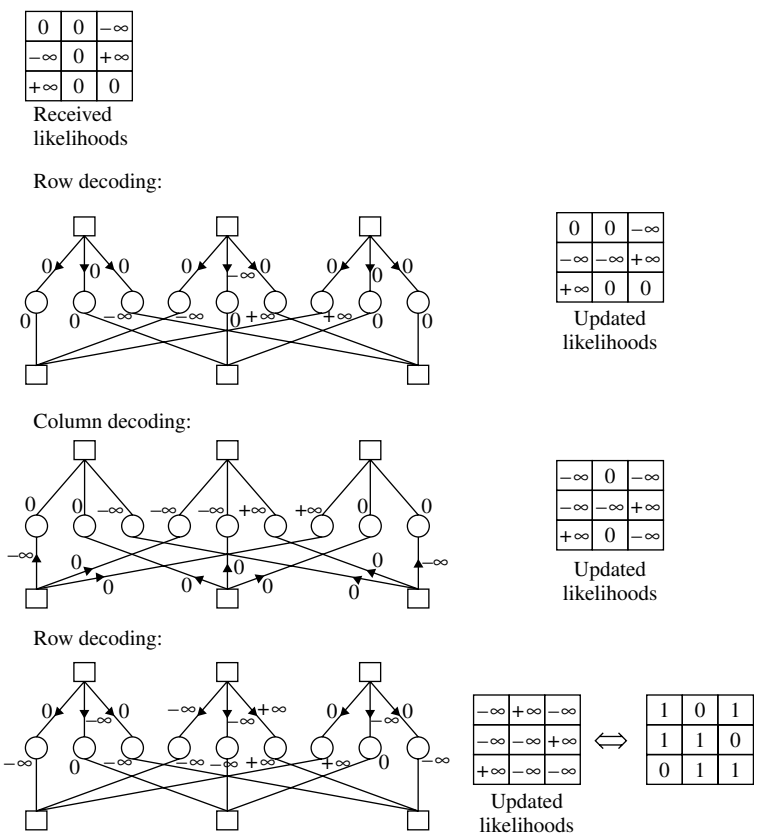


Figure 5.10 An example of min-sum decoding on the BEC using the (3,2) $\times$ (3,2) SPC product code of Figure 5.3.

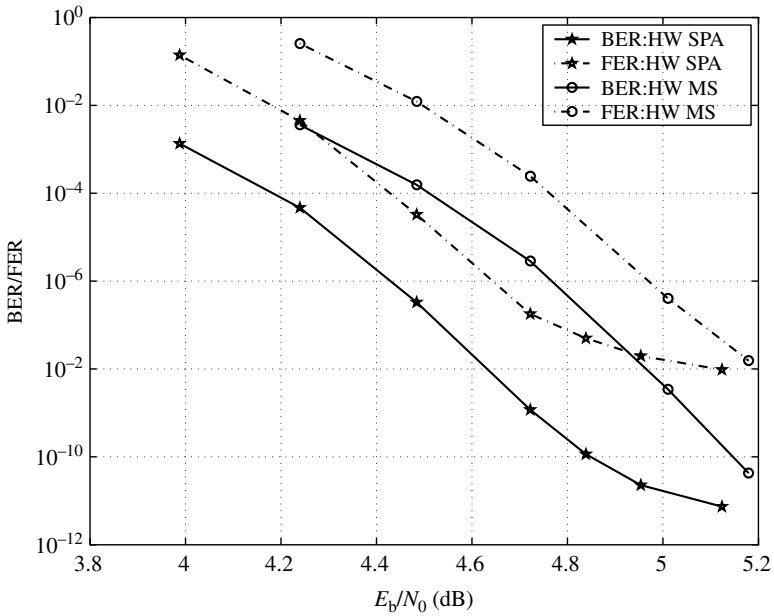


Figure 5.11 Performance of a 0.9(4550,4096) quasi-cyclic IRA code with SPA and min-sum decoders. "HW" represents "hardware," since these simulations were performed on an FPGA-based simulator [10].

Example 5.4. We consider a rate-0.9 quasi-cyclic IRA code with parameters (4544,4096). The bit-error-rate and frame-error-rate performance of this code with the SPA decoder and the min-sum decoder is depicted in Figure 5.11. We observe that, while the SPA decoder is superior in the waterfall region by about 0.3 dB, it suffers from an *error-rate floor*. This floor is attributable to trapping sets seen by the SPA decoder which are apparently transparent to the min-sum decoder. Figure 5.12 displays the induced graphs for the dominant trapping sets for SPA decoding of this code. (Actually, the graphs represent equivalence classes of trapping sets. There are 64 trapping sets within each class because the parity-check matrix for this code is an array of 64×64 circulant permutation matrices.) For some applications, the floor is not an issue. However, for applications requiring extremely low error rates (e.g., magnetic or optical storage), this floor is unacceptable. Decoding techniques for lowering floors, specifically for this code, have been studied in [10–12], where this code was studied. Chapter 15 reviews the decoding techniques of [12] and also a semi-analytic technique for estimating the level of the floor.

We remark that the floor seen in the previous example is not exceptional. In fact it is a characteristic of iteratively decodable codes and was first seen with turbo

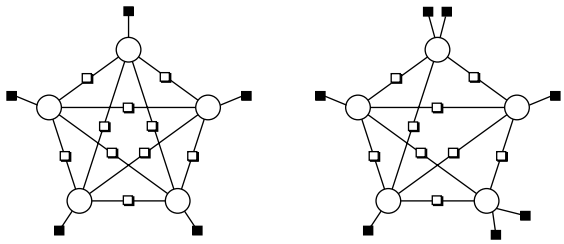


Figure 5.12 Two classes of trapping sets (or graphs induced by trapping sets) for the 0.9(4544,4096) code of Example 5.4. The black squares are unsatisfied check nodes and the white squares are "mis-satisfied" check nodes. The white circles represent erroneous bits (VNs).

codes. Moreover, if one sees an iterative-decoder error-rate curve without a floor, then, in all likelihood, the simulations have not been run at a high enough SNR. That is, the floor likely exists, but it requires more time and effort to find it, since it might be out of the reach of standard computer simulations. The floor can be due to trapping sets (as is usually the case for LDPC codes and serial turbo codes) or to a small minimum distance (as is usually the case for parallel turbo codes).

5.5.2 The Attenuated and Offset Min-Sum Decoders

As demonstrated in Figure 5.13 for the first iteration of a min-sum decoder, the extrinsic information passed from a CN to a VN for a min-sum decoder is on average larger in magnitude than the extrinsic information that would have been passed by an SPA decoder. That is, the min-sum decoder is generally too optimistic in assigning reliabilities. A very simple and effective adjustment that accounts for this fact is the attenuation of such extrinsic information before it is passed to a VN. The resulting update equation is

$$\textbf{CN update: } L_{i\to j} = \prod_{j'\in N(i)-\{j\}} \alpha_{j'i} \cdot c_{\text{atten}} \cdot \min_{j'\in N(i)-\{j\}} \beta_{j'i},$$

where $0 < c_{\text{atten}} < 1$ is the attenuation factor.

A particularly convenient attenuation factor is $c_{\text{atten}} = 0.5$, since it is implementable by a register shift. The impact of a 0.5 attenuator is presented in Figure 5.14, where it can be seen that, for the first iteration, the expected value of a min-sum reliability is about equal to that of an SPA reliability. The performance improvement realized by the attenuated min-sum decoder will be demonstrated by example later in the chapter. We point out that, depending on the code involved, it

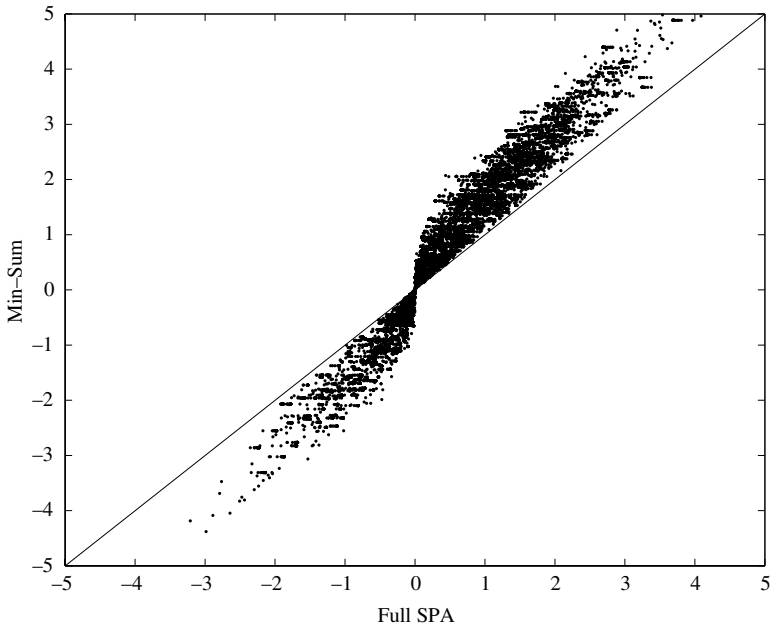


Figure 5.13 Comparison between CN outputs for a min-sum decoder and an SPA decoder.

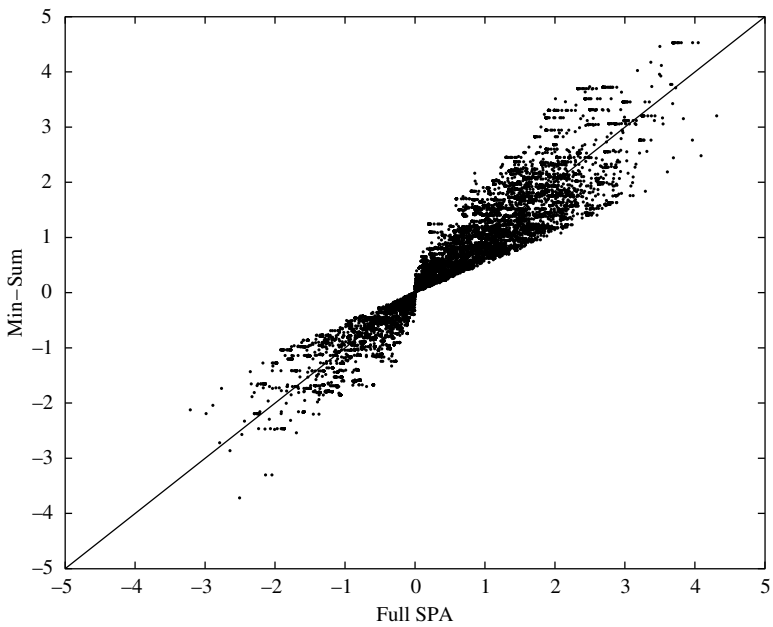


Figure 5.14 Comparison between CN outputs for an attenuated min-sum decoder (with attenuator $c_{\text{atten}} = 0.5$) and an SPA decoder.

may be advantageous to allow the attenuation value to change with each iteration or to periodically turn off the attenuator [10, 11]. It may also be advantageous to attenuate messages entering low-degree VNs less than those entering high-degree VNs [13].

An alternative technique for mitigating the overly optimistic extrinsic information that occurs in a min-sum decoder is to subtract a constant offset $c_{\text{offset}} > 0$ from each CN-to-VN message magnitude $|L_{i\to j}|$, subject to the constraint that the result is non-negative [14, 15]. Thus, the offset-min-sum algorithm computes

$$\textbf{CN update: } L_{i\to j} = \prod_{j'\in N(i)-\{j\}} \alpha_{j'i} \cdot \max\left\{ \min_{j'\in N(i)-\{j\}} \beta_{j'i} - c_{\text{offset}},\ 0 \right\}.$$

This algorithm was implemented in hardware with 4-bit messages in [16] for a (2048,1723) Reed–Solomon-based LDPC code (see Chapter 11), for which its performance was demonstrated to degrade very little in the waterfall region.

5.5.3 The Min-Sum-with-Correction Decoder

The min-sum-with-correction decoder [17] is an approximation to the box-plus decoder described in Section 5.4.5. First, we recall from Chapter 4 that we defined, for any pair of real numbers x, y,

$$\max^*(x,\ y) \triangleq \log(e^x + e^y), \tag{5.25}$$

which was shown to be (see Chapter 4 problems)

$$\max^*(x, y) = \max(x, y) + \log\Big(1 + e^{-|x-y|}\Big). \tag{5.26}$$

Observe from (5.22) and (5.25) that we may write

$$L_1 \boxplus L_2 = \max^*(0, L_1 + L_2) - \max^*(L_1, L_2), \tag{5.27}$$

so that

$$L_1 \boxplus L_2 = \max(0, L_1 + L_2) - \max(L_1, L_2) + s(L_1, L_2), \tag{5.28}$$

where $s(x, y)$ is a so-called *correction term* given by

$$s(x, y) = \log\Big(1 + e^{-|x+y|}\Big) - \log\Big(1 + e^{-|x-y|}\Big).$$

It can be shown (Problem 5.21) that

$$\max(0, L_1 + L_2) - \max(L_1, L_2) = \ \text{sign}(L_1)\text{sign}(L_2)\min(|L_1|, |L_2|)$$

so that

$$\begin{aligned} L_1 \boxplus L_2 &= \operatorname{sign}(L_1)\operatorname{sign}(L_2)\min(|L_1|,|L_2|) + s(L_1,L_2) \\ &= \operatorname{sign}(L_1)\operatorname{sign}(L_2)\left[\min(|L_1|,|L_2|) + s(|L_1|,|L_2|)\right] \\ &\triangleq \operatorname{sign}(L_1)\operatorname{sign}(L_2)\min{}^*(|L_1|,|L_2|), \end{aligned} \tag{5.29}$$

where $\min^*(\cdot,\cdot)$ is defined by the expression in the line above its appearance. From the definition of $\min^*$, $L_1 \boxplus L_2$ may be approximated as

$$L_1 \boxplus L_2 \simeq \operatorname{sign}(L_1)\operatorname{sign}(L_2)\min(|L_1|,|L_2|) \tag{5.30}$$

since $|s(x,y)| \leq 0.693$.

In the computation of $L_{i\to j}$, given by

$$L_{i\to j} = \boxplus_{j'\in N(i)-\{j\}} L_{j'\to i},$$

one can repeatedly apply (5.29). This gives one implementation of the box-plus SPA decoder (e.g., with a look-up table). A lower-complexity alternative substitutes an approximation $\tilde{s}(x,y)$ to the correction term $s(x,y)$ in (5.29), where

$$\tilde{s}(x,y) = \begin{cases} c & \text{if } |x+y| < 2 \text{ and } |x-y| > 2\,|x+y| \\ -c & \text{if } |x-y| < 2 \text{ and } |x+y| > 2\,|x-y| \\ 0 & \text{otherwise} \end{cases}$$

and where c on the order of 0.5 is typical. Observe that, if the approximation (5.30) is used, we have the min-sum algorithm.

Example 5.5. We consider a regular quasi-cyclic rate-0.875 (8176,7156) LDPC code derived from a regular-Euclidean-geometry (EG) code (discussed in Chapter 10). The performance of this code is examined with three of the decoders discussed above: the (log-)SPA, the min-sum, and the min-sum with a correction factor (which we denote by min-sum-c, with c set to 0.5). The $\mathbf{H}$ matrix for this code is 1022×8176 and has column weight 4 and row weight 32. It is a 2×16 array of circulants, each with column and row weight equal to 2. This code has been adopted for NASA near-Earth missions [20, 21]. The performance of this code for the three decoders on a BI-AWGNC is presented in Figure 5.15. With measurements taken at a BER of 10^{-5}, the loss relative to the SPA decoder suffered by the min-sum decoder is 0.3 dB and the loss suffered by the min-sum-c decoder is 0.01 dB. We remark that these losses are a strong function of the code, in particular, the row weight. For codes with a larger row weight, the losses can be more substantial.

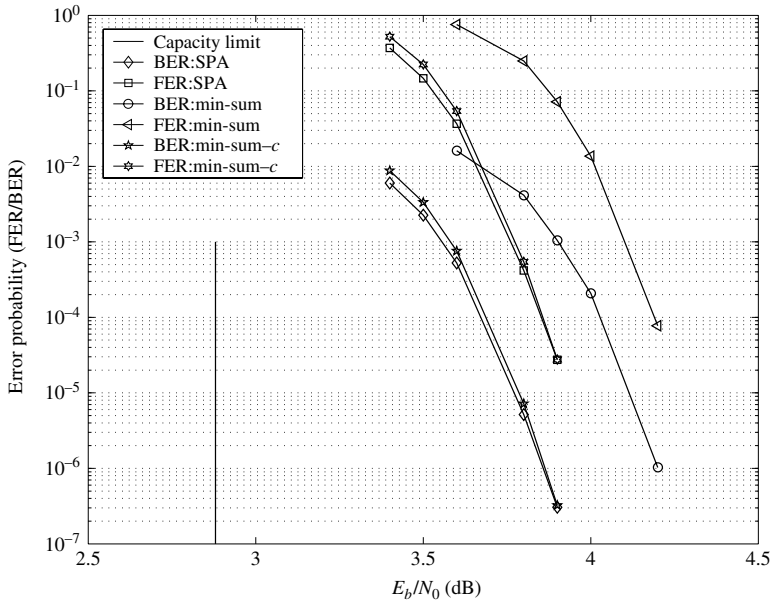


Figure 5.15 Performance of a quasi-cyclic EG (8176,7156) LDPC code on a binary-input AWGN channel and three decoding algorithms, with 50 decoding iterations.

5.5.4 The Approximate min* Decoder

We now consider another approximate SPA decoder, called the approximate min* (a-min*) decoder [18], which reduces the number of operations at each check node. For the sake of simplicity, we focus on the computation of values for $\{L_{i\to j}\}$ using (5.29), although this is not necessary. Note that, for a degree-d_c CN, d_c quantities are computed and $d_c - 1$ applications of the $\boxplus$ operator are required in order to compute each quantity. We can reduce the $d_c(d_c - 1)$ applications of the $\boxplus$ operator at a degree-d_c CN with negligible performance loss as follows.

1. At CN i, find the incoming message of least reliability (i.e., having minimum $|L_{j\to i}|$) and label the source of that message VN $j_{\min}$.
2. The message sent to VN $j_{\min}$ is identically the one used in the standard log-SPA algorithm:

$$L_{i\to j_{\min}} = \boxplus_{j\in N(i)-\{j_{\min}\}} L_{j\to i}.$$

3. For all v in the exclusive neighborhood $N(i) - \{j_{\min}\}$, the message sent to VN v is

$$\begin{aligned} L_{i\to v} &= \left(\prod_{j\in N(i)-\{v\}} \alpha_{ji}\right) \cdot |L_{i\to j_{\min}} \boxplus L_{j_{\min}\to i}| \\ &= \left(\prod_{j\in N(i)-\{v\}} \alpha_{ji}\right) \cdot \left|\boxplus_{j\in N(i)} L_{j\to i}\right|, \end{aligned}$$

where $\alpha_{ji} = \operatorname{sign}(L_{j\to i})$ as before.

Observe that, in this approximation of the SPA, only two magnitudes are computed at each check node, requiring only $d_c - 1$ applications of the $\boxplus$ operator to compute both. That transmitting only one of two magnitudes from a CN node results in negligible performance loss can be intuitively explained via the following consideration of the log-SPA, min-sum, and a-min* decoders. If two or more unreliable LLRs are received at a CN, then all of the outgoing LLRs will be unreliable for all three decoders (this is best seen from the perspective of the min-sum algorithm). If there is only one unreliable LLR received at a CN, then the outgoing likelihood to the unreliable VN will be reliable for all three decoders. However, in both of these situations, the LLR sent to the least reliable bit in the a-min* decoder is exactly that of the log-SPA decoder. This explains the performance improvement of the a-min* decoder over the min-sum decoder and its negligible loss relative to the log-SPA decoder.

5.5.5 The Richardson/Novichkov Decoder

The Richardson/Novichkov (RN) decoder [22] uses an approximation to the CN computation given in (5.20). More specifically, the decoder is a hardware-friendly approximation to the magnitude component of the CN-to-VN message $L_{i\to j}$. With the understanding that the messages being passed in a hardware implementation are integers, we can write this magnitude component as

$$|L_{i\to j}| = \phi_\delta^{-1}\left(\sum_{j'\in N(i)-\{j\}} \phi_\delta(b_{j'i})\right), \tag{5.31}$$

where $\beta_{j'i}$ is represented by an integer $b_{j'i}$ times the scale factor $\delta = \ln(2)$, $\beta_{j'i} = \delta b_{j'i}$, and we define

$$\phi_\delta(b_{j'i}) \triangleq \log\left(\frac{\exp(\delta b_{j'i}) + 1}{\exp(\delta b_{j'i}) - 1}\right) = \phi\big(\beta_{j'i}\big).$$

Now recall that, when x is large, $\log(1 \pm e^{-x}) \simeq \pm e^{-x}$, and from this $\phi_\delta(b_{j'i})$ can be approximated as follows:

$$\begin{aligned}\phi_\delta(b_{j'i}) &= \log\left(\frac{1+\exp(-\delta b_{j'i})}{1-\exp(-\delta b_{j'i})}\right)\\ &= \log(1+\exp(-\delta b_{j'i})) - \log(1-\exp(-\delta b_{j'i}))\\ &\simeq 2\exp(-\delta b_{j'i}).\end{aligned} \tag{5.32}$$

We will use $\hat{\phi}_\delta(b) \triangleq 2\exp(-\delta b)$ to represent this approximation of $\phi_\delta(b)$, which is accurate for $\delta b \gtrsim \ln 2$. (Similarly, $\hat{\phi}(x) = 2\exp(-x)$ is an accurate approximation of $\phi(x)$ for $x \gtrsim \ln 2$.) The inverse of $\hat{\phi}_\delta(x)$ is

$$\hat{\phi}_\delta^{-1}(x) \simeq -\frac{1}{\delta}\ln\left(\frac{x}{2}\right) = -\log_2\left(\frac{x}{2}\right), \tag{5.33}$$

and it is accurate for $x \lesssim \ln 2$. (Similarly, since $\phi^{-1}(x) = \phi(x)$, $\hat{\phi}(x) = -\ln(x/2)$ closely approximates $\phi(x)$ for $x \lesssim \ln 2$.) The accuracy of these approximations is illustrated in Figure 5.16.

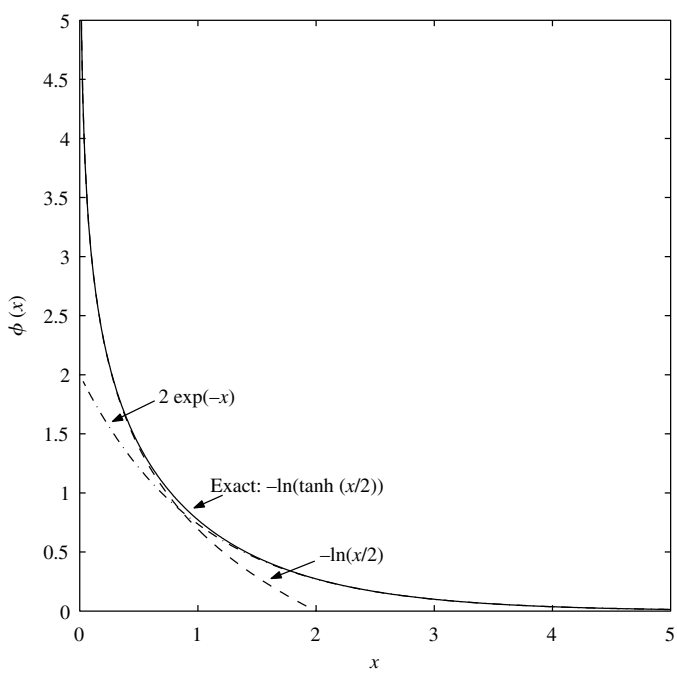


Figure 5.16 A plot of $\phi(x) = -\ln(\tanh(x/2))$ together with the approximations $2e^{-x}$ and $\ln(x/2)$.

Using (5.32) and (5.33) in (5.31), we have

$$\begin{aligned}
|L_{i\to j}| &\simeq -\log_2\left(\sum_{j'\in N(i)-\{j\}} \exp(-\delta b_{j'i})\right) \\
&= -\log_2\left(\sum_{j'\in N(i)-\{j\}} \exp(-\ln(2) b_{j'i})\right) \\
&= -\log_2\left(\sum_{j'\in N(i)-\{j\}} \exp\left(\ln(2^{-b_{j'i}})\right)\right) \\
&= -\log_2\left(\sum_{j'\in N(i)-\{j\}} 2^{-b_{j'i}}\right).
\end{aligned}$$

Because $\hat{\phi}_\delta(b)$ is inaccurate for $\delta b \lesssim \ln 2$, and $\hat{\phi}_\delta^{-1}(x)$ is inaccurate for $x \gtrsim \ln 2$, correction terms must be included in the computation of $|L_{i\to j}|$. Further, $|L_{i\to j}|$ must be converted into an integer message. Thus, the computation of $|L_{i\to j}|$ looks like

$$|L_{i\to j}| = C_1 - \left[\left[\log_2\left(\sum_{j'\in N(i)-\{j\}} \left(2^{-b_{j'i}} + C_2\right)\right)\right]\right],$$

where C_1 and C_2 are appropriately chosen integer constants and $[[x]]$ denotes the integer part of x. See the Flarion patent [22] for additional details.

5.5.6 The Reduced-Complexity Box-Plus Decoder

The reduced-complexity box-plus (RCBP) decoder [23, 24] is based on the combination of the box-plus decoder and the a-min* decoder. The check-node processor design is based on a highly compressed look-up table for the box-plus operator. This compression exploits the symmetry and redundancy of the full table in order to losslessly reduce the 1024-element table to two easily implementable logic equations.

The RCBP decoder starts with a two-input look-up table for the $\mathcal{B}$-function of (5.24). This table, which will be denoted by τ, has the property that entry (r, c) in the table, $\tau(r, c)$, contains the integer representation of the function value $\mathcal{B}(r\Delta + \Delta/2, c\Delta + \Delta/2)$, where Δ is the (uniform) quantization step size. This integer representation corresponds to the same quantization method as that used at the decoder input. Simulations of the table-based decoder with 6-bit messages (including the decoder input) are sufficient to ensure a loss of less than 0.2 dB relative to a floating-point simulator. A step size of $\Delta = 0.5$ was found to give a check-node processor that was robust over a large range of code rates and SNRs. Thus, the 6-bit message values

range from -16 to $+15$. Note that $\mathcal{B}(a,b) = 2\tanh^{-1}(\tanh(a/2)\tanh(b/2)) = \text{sign}(a)\text{sign}(b) \cdot 2\tanh^{-1}(\tanh(|a|/2)\tanh(|b|/2))$ so that we need only work with the magnitudes of the incoming messages $L_{j\to i}$. Thus, the $\mathcal{B}$-table $\tau(r,c)$ need only accept two 5-bit inputs and r and c range from 0 to 31.

We now show how to losslessly compress the $\mathcal{B}$-table $\tau(r,c)$ from the nominal size of 32×32 so that the only loss in performance is due to quantization. The table τ possesses the following properties that facilitate its compression.

1. τ is symmetric: $\tau(r,c) = \tau(c,r)$.
2. The entries of τ are non-decreasing with increasing index. That is, if $r' > r$, then $\tau(r',c) \geq \tau(r,c)$, and if $c' > c$, then $\tau(r,c') \geq \tau(r,c)$.
3. At any "boundary" where an entry increases relative to its predecessor along a row or column, it increases by exactly 1.

Owing to the symmetry of τ, it is necessary only to store half of the table. A judicious choice for the half that will be stored is all elements $\tau(r,c)$ such that $r \geq c$. The reasoning behind this choice is subtle. If one were to imagine τ as a matrix with element index $(0,0)$ in the upper left and element index $(31,31)$ in the lower right, one would notice that, as each individual column is traversed from lowest row number to highest, the values in the columns change increasingly often as the column number is increased. That is, there are at least as many changes in the values of the entries in column $c+1$ as there are in column c. Choosing to keep track of τ column by column, but only for elements of $\tau(r,c)$ such that $r \geq c$ results in keeping track of many elements in small-numbered columns (few or no changes in function value) and very few elements in large-numbered columns (many changes in function value). As will be seen shortly, this result greatly aids in the compression of τ.

To see how the properties 2 and 3 facilitate further compression, consider the following example. Suppose we wish to compress the following 5×5 table,

$$\Omega = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 2 & 2 \\ 0 & 1 & 2 & 2 & 2 \\ 1 & 2 & 2 & 3 & 3 \\ 1 & 2 & 2 & 3 & 4 \end{bmatrix}$$

in the same way as that in which τ is compressed in the RCBP decoder. Notice that Ω has the properties listed above for τ. The first step in the compression is to examine the first element, $\Omega_{0,0}$, along the main diagonal of Ω. The vector $\mathbf{v}_0$, which will represent the first column of Ω, can be written down in the following way.

1. The first entry of $\mathbf{v}_0$ is the element in column 0 that resides on the main diagonal of Ω.

2. The second entry of $\mathbf{v}_0$ is the number of times (including the appearance on the main diagonal) that the element on the main diagonal repeats as the column is traversed downward.
3. Subsequent entries of $\mathbf{v}_0$ occur only when a new number is encountered in a downward column traversal. Each new entry of $\mathbf{v}_0$ equals the number of times that the new element of Ω appears.

Performing this process on the first column of Ω gives $\mathbf{v}_0 = [0, 3, 2]^{\mathrm{T}}$. The columns $\mathbf{v}_1$ through $\mathbf{v}_4$ are computed by analogous processes, each starting with successive elements on the main diagonal. The application of this process to all remaining columns results in the following compressed version of Ω:

$$\begin{aligned}
\mathbf{v}_0 &= [0, 3, 2]^{\mathrm{T}}, \\
\mathbf{v}_1 &= [1, 2, 2]^{\mathrm{T}}, \\
\mathbf{v}_2 &= [2, 3]^{\mathrm{T}}, \\
\mathbf{v}_3 &= [3, 2]^{\mathrm{T}}, \\
\mathbf{v}_4 &= [4, 1]^{\mathrm{T}}.
\end{aligned}$$

When a similar compression is performed on τ, the resulting table τ' contains 92 entries – a compression factor of about 11. The resulting column vectors for τ' are given by

$$\mathbf{v}_c = \begin{cases} [0,\ 32]^{\mathrm{T}} & \text{if } c = 0, \\ [0,\ 2,\ 29]^{\mathrm{T}} & \text{if } c = 1, \\ [c-1,\ 3,\ 29-c]^{\mathrm{T}} & \text{if } 2 \le c \le 29, \\ [c-1,\ 2-\delta(c-31)]^{\mathrm{T}} & \text{if } 30 \le c \le 31, \end{cases}$$

where $\delta(k)$ is the Kronecker delta.

In order to determine the value of any entry of τ by using τ', all that is needed is a measure of the relative distance from the main diagonal in a downward direction. That is, if we desire to retrieve $\tau(r, c)$ with $r \ge c$, it is necessary to know how many elements need to be skipped in the representative column vector $\mathbf{v}_c$ in order to arrive at the value of interest. Thus, define the quantity $d = r - c$. Then, it is possible to rewrite the above look-up method simply as a series of conditional comparisons. The result of this process appears below in pseudo-code as $\widehat{\tau}(r, c)$. It is an exact representation of the 1024-element table τ. In the pseudo-code, the result of the comparison $(d > a)$ returns 1 if true and 0 if false, and c_0 is the least significant bit of the binary representation of c.

Pseudo-code description of the quantized $\mathcal{B}$ table: $\widehat{\tau}(r,c)$ with $r \geq c$
$d = r - c$
if $c > 1$
$\quad \lvert\widehat{\tau}\rvert = (c-1) + (d > 2)$
else % $c = 0$ or $c = 1$
$\quad \lvert\widehat{\tau}\rvert = (d > 1)$ & c_0

Using this pseudo-code to represent $\mathcal{B}(L_1, L_2)$ results in a small quantization error that manifests itself whenever $\widehat{\tau}(r,c)$ returns a value. The magnitude of the relative error introduced by quantization with respect to the exact value of $\mathcal{B}$ averaged over all possible $\widehat{\tau}(r,c)$ is 2.64%. In most cases, errors manifest themselves as a slight over-approximation of $\mathcal{B}$. In an effort to depict the quantization error, the plots of Figure 5.17 show the exact values of $\mathcal{B}$ and the quantized values of $\mathcal{B}$, $\mathcal{B}_{\widehat{\tau}}$, for four columns of τ ($c = 4$, $c = 10$, $c = 18$, and $c = 30$). In each plot, "translated r" on the abscissa represents the real number corresponding to the index of quantization, r. All values of $\mathcal{B}_{\widehat{\tau}}$ are the real numbers corresponding to the integers returned after accessing $\widehat{\tau}(r,c)$.

The RCBP decoder uses a variant of the a-min* algorithm to perform check-node message updates. Recall from Section 5.5.4 that, at every check node c_i, the magnitude of the message sent to the least-reliable variable node, $v_{j\#}$, is precisely

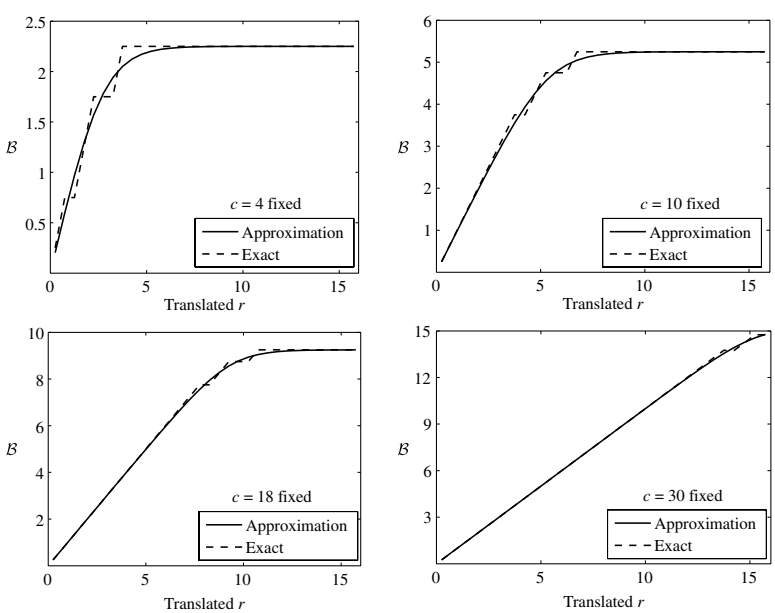


Figure 5.17 Approximation error for various cases of fixed c.

(5.23). Denote this message, $L_{i\to j\#}$, by ζ. Then, the message magnitudes sent to every other neighbor of c_i are set to the same value,

$$\Lambda = \left|\zeta \boxplus L_{j\#\to i}\right|,$$

and their signs are calculated as before. The RCBP algorithm differs from the a-min* algorithm in two regards. The first difference is, of course, that the $\boxplus$ operator is implemented by the $\widehat{\tau}$ function described above. For the second difference, note that the first step in the a-min* algorithm is to find the variable-node message of minimum magnitude, a process that requires $d_c - 1$ comparisons. These comparisons can be avoided by setting a pointer v_γ equal to the current least-reliable variable node as the algorithm loops through the d_c neighbors of c_i, while simultaneously making box-plus computations (via $\widehat{\tau}$) within the loop. Upon completion of the loop, v_γ will reference $v_{j\#}$ and ζ will equal $\left|L_{i\to j\#}\right|$ as in the a-min* algorithm. The RCBP algorithm is presented below.

Algorithm 5.2 The Reduced-Complexity Box-Plus Algorithm

Initialize: $\{|L_{\gamma\to i}| = \infty,\ \zeta = \infty,\ s = 1\}$

for $k = 1$ to d_c
  $s = s \cdot \mathrm{sign}(L_{k\to i})$
  if $(|L_{k\to i}| < |L_{\gamma\to i}|)$
    $\zeta = \widehat{\tau}(\zeta, |L_{\gamma\to i}|)$
    $v_\gamma = v_k$
    $|L_{\gamma\to i}| = |L_{k\to i}|$
  else
    $\zeta = \widehat{\tau}(\zeta, |L_{k\to i}|)$
end
$\Lambda = \widehat{\tau}(\zeta, L_{\gamma\to i})$

outgoing messages:

$$L_{i\to k} = \begin{cases} \mathrm{sign}(v_k)\cdot s\cdot\Lambda & \text{for } v_k \in N(c_i) - \{v_\gamma\}, \\ \mathrm{sign}(v_\gamma)\cdot s\cdot\zeta & \text{for } v_k = v_\gamma, \end{cases}$$

where $N(c_i)$ denotes the set of all variable nodes adjacent to CN c_i.

We now present simulation results in which we compare the performance of the RCBP decoder with that of the RN decoder in terms of bit-error rate (BER) and frame-error rate (FER) for a 0.9(4544,4096) structured IRA code. Since the RN decoder essentially achieves floating-point performance with 5-bit messages [10], it is an excellent benchmark for assessing the performance of other decoder implementations. This IRA code is considered to be a good test code, because it has large check-node degrees (48 is the maximal check-node degree for this code),

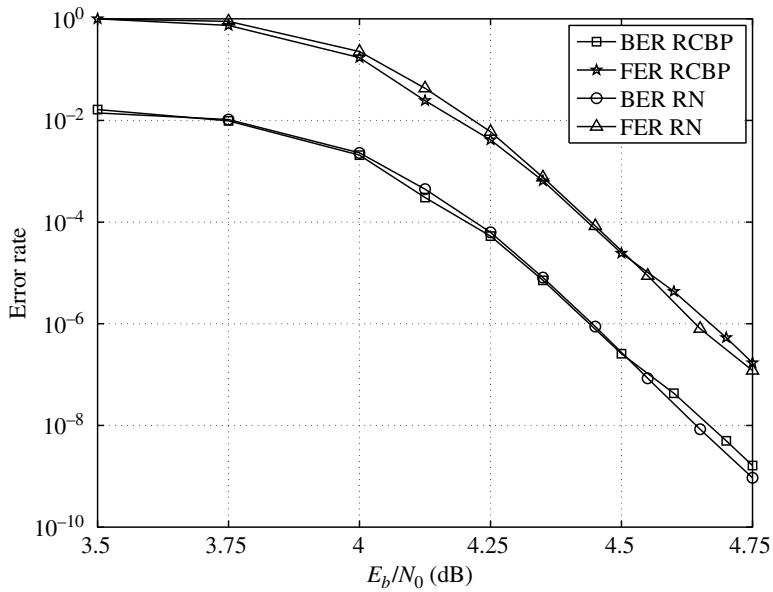


Figure 5.18 RCBP decoder versus RN decoder: (4544,4096) structured IRA code.

requiring many uses of the $\widehat{\tau}$ table in each iteration. As can be seen in Figure 5.18, the RN decoder (5 bits per message) outperforms the RCBP (6 bits per message) decoder by only 0.02 dB after about 4.5 dB. While the RN decoder uses 14.3% fewer bits per internal decoder message than the RCBP decoder, the RCBP decoder is in fact a lower-complexity and higher-speed implementation [23, 24].

Note that the RN decoder effectively computes the LLR and then quantizes it once, whereas the RCBP decoder applies a quantized table $d_c - 1$ times so that quantization error can accumulate. Still, as seen in the above example, the loss is often negligible. Where it might not be negligible is for low code rates ($R \leq 1/2$) that operate in lower SNR (higher noise) such that the CN processor is less likely to have a (quantized) LLR magnitude stand out as the minimum, so that quantization errors are more likely to accumulate. These issues were pointed out by T. Richardson (personal communication).

5.6 Iterative Decoders for Generalized LDPC Codes

The iterative decoder for a generalized LDPC code, introduced in Section 5.2.1, is a straightforward generalization of the iterative decoder for LDPC codes. Recall that, in deriving that decoder, we derived the MAP decoder for the check nodes (which represent SPC codes) and converted the MAP decoder into an APP soft-in/soft-out processor; we then repeated this for the variable nodes (which represent REP codes). For generalized LDPC codes, whose variable nodes are not

generalized (i.e., are REP nodes), we need only concern ourselves with the generalized constraint nodes (G-CNs). Thus, we require a MAP decoder (actually, an APP processor) for an arbitrary G-CN, which is typically a (low-complexity) binary linear code.

The most well-known MAP decoder for binary linear codes is the BCJR algorithm applied to a trellis that represents the linear code. A commonly used trellis, known for its simplicity and its optimality [25, 26], is the so-called *BCJR trellis*. To derive the BCJR trellis for an arbitrary binary linear code, we start with the equation $\mathbf{c}\mathbf{H}^{\mathrm{T}} = \mathbf{0}$, where $\mathbf{H}$ is $m \times n$. If we let $\mathbf{h}_j$ represent the jth column of $\mathbf{H}$, we have

$$c_1\mathbf{h}_1 + c_2\mathbf{h}_2 + \cdots + c_n\mathbf{h}_n = \mathbf{0}.$$

This equation leads directly to a trellis, for its solutions yield the list of codewords just as the paths through a code's trellis gives the codeword list. Thus, the possible states at the ℓth trellis stage can be computed as

$$S_\ell = c_1\mathbf{h}_1 + c_2\mathbf{h}_2 + \cdots + c_\ell\mathbf{h}_\ell.$$

Further, since the $\mathbf{h}_j$ are m-vectors, there are at most 2^m states for trellis stages $\ell = 1, 2, \ldots, n-1$, and of course $S_0 = S_n = \mathbf{0}$.

Example 5.6. Following [25], let

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

Then at stage 1 there are two possible states derived from $S_1 = c_1\mathbf{h}_1$, namely

$$\begin{bmatrix} 0 \\ 0 \end{bmatrix} \text{ and } \begin{bmatrix} 1 \\ 0 \end{bmatrix},$$

corresponding to $c_1 = 0$ and $c_1 = 1$, respectively. At stage 2, there are four possible states derived from

$$S_2 = c_1\mathbf{h}_1 + c_2\mathbf{h}_2;$$

and so on. The resulting trellis is given in Figure 5.19.

In summary, the iterative decoder for the generalized LDPC codes is much like that of LDPC codes, except that the G-CNs require APP processors more general than those for standard SPC CNs. One example is the BCJR-based APP processor designed for the code's BCJR trellis. Because the BCJR algorithm naturally accepts LLRs and produces outputs that are LLRs, it serves well as an APP processor. Of course, alternative, possibly suboptimal, soft-in/soft-out algorithms may be employed in G-CN processors, but we mention the BCJR algorithm because it is optimal and well known. Lastly, irrespective of the G-CN processor used, it

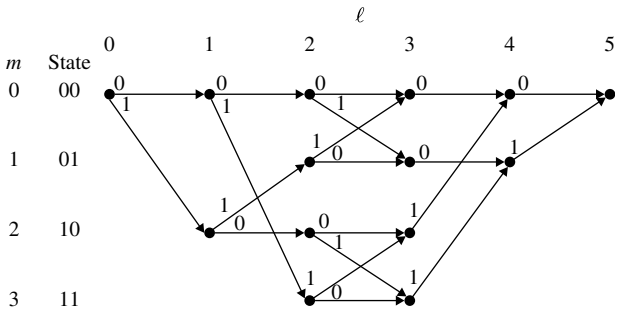


Figure 5.19 The BCJR trellis for example code (after [25]).

is important that *extrinsic* LLR information is passed from the G-CN to each of its neighboring VNs. For example, following Figure 5.5, at the sth iteration, the message from G-CN i to VN j can be computed as

$$L_{i\to j}^{(s)} = \mathrm{LLR}^{(s)} - L_{j\to i}^{(s-1)},$$

where $\mathrm{LLR}^{(s)}$ is the LLR for the G-CN code bit corresponding to edge $j \leftrightarrow i$.

5.7 Decoding Algorithms for the BEC and the BSC

In this section, we consider various decoders for the BEC and the BSC. We consider only those decoders which frequently appear in the literature or will aid our discussion elsewhere in the book. For example, Chapter 13 considers the design of LDPC codes for the BEC.

5.7.1 Iterative Erasure Filling for the BEC

The iterative decoding algorithm for the BEC simply works to fill in the erased bits in such a way that all of the check equations contained in $\mathbf{H}$ are eventually satisfied. We have already seen one example of this algorithm in Example 5.2, but we now consider a simpler example to aid our development of the general iterative erasure-filling algorithm.

Example 5.7. Consider an $\mathbf{H}$ matrix for the (7,4) Hamming code,

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

Suppose the received word is $\mathbf{r} = [e\ 1\ 1\ 1\ 1\ 1]$, where e represents an erasure. Then the first check equation (first row of $\mathbf{H}$) requires that $0 = r_0 + r_2 + r_3 + r_4 = e + 1 + 1 + 1$, leading us to the conclusion that bit r_0 must equal 1. Suppose the situation is instead that two erasures have occurred, with $\mathbf{r} = [e\ 1\ e\ 1\ 1\ 1]$. Clearly, the two erasures cannot be resolved with only the first check equation since there will be two unknowns (erasures) in this single equation. A different check equation is first required in order to resolve one of the two erasures, after which the second erasure may be resolved. The third check equation involves only the second erasure and it may be used to convert this erasure into a 1. After this, the first erasure can be found to be a 1 using either the first or the second check equation.

In summary, the erasure decoding algorithm is as follows.

Algorithm 5.3 Iterative Erasure Decoding Algorithm

1. Find all check equations (rows of $\mathbf{H}$) involving a single erasure and solve for the values of bits corresponding to these erasures.
2. Repeat Step 1 until there exist no more erasures or until all equations are unsolvable.

Note that the above example and algorithm description tell us exactly which erasure patterns will fail to be resolved by this algorithm. Namely, if no parity-check equation that involves only one of the erasures in the erasure set can be found, then none of the equations will be solvable and decoding will stop. Code-bit sets that lead to unsolvable equations when they are erased are called stopping sets. Thus, a subset S of code bits forms a *stopping set* if each equation that involves the bits in S involves two or more of them. In the context of the Tanner graph, S is a stopping set if all of its neighbors are connected to S at least twice. Such a configuration is problematic for an iterative decoder because, if these bits are erased, all equations involved will contain at least two erasures. Notice that the trapping sets depicted in Figure 5.12 are also stopping sets.

Example 5.8. Consider the (7,4) Hamming code again, together with the received pattern $\mathbf{r} = [e\ e\ e\ 1\ 1\ 1\ 1]$. Note that each check equation involves at least two erasures and so a stopping set is $\{c_0, c_1, c_2\}$.

5.7.2 ML Decoder for the BEC

The algorithm described in the previous subsection is not maximum-likelihood, but we can obtain an ML decoding algorithm by applying the above algorithm to

a modified or "adapted" $\mathbf{H}$ matrix as described below. Following the description of this "adapted-$\mathbf{H}$ algorithm," we provide an example in which the stopping set in the previous example is resolvable.

Suppose we transmit any codeword in the code C over a BEC and we receive the word $\mathbf{r}$ whose elements are taken from the set $\{0, 1, e\}$. Let J represent the index set of unerased positions in $\mathbf{r}$ and J' the set of erased positions. It is easy to show (Problem 5.8) that the ML decoder chooses a codeword $\mathbf{c} \in C$ satisfying $c_j = r_j$ for all $j \in J$ and, further, that this is a solution to the equation

$$\mathbf{c}_{J'}\mathbf{H}_{J'}^{\mathrm{T}} = \mathbf{c}_J\mathbf{H}_J^{\mathrm{T}}, \tag{5.34}$$

where $\mathbf{H}_J$ ($\mathbf{H}_{J'}$) is the submatrix of the code's parity-check matrix $\mathbf{H}$ obtained by taking only the columns of $\mathbf{H}$ corresponding to J (J'), and similarly for $\mathbf{c}_J$ ($\mathbf{c}_{J'}$). Equation (5.34) follows from the fact that $\mathbf{0} = \mathbf{c}\mathbf{H}^{\mathrm{T}} = \mathbf{c}_{J'}\mathbf{H}_{J'}^{\mathrm{T}} + \mathbf{c}_J\mathbf{H}_J^{\mathrm{T}}$. There is a unique solution if and only if the rows of $\mathbf{H}_{J'}^{\mathrm{T}}$ are linearly independent, in which case the elements of the unknown $\mathbf{c}_{J'}$ may be determined by Gaussian elimination. Because $\mathbf{H}_{J'}^{\mathrm{T}}$ has $n-k$ columns, its rows can be linearly independent only if $|J'| \leq n-k$, giving us a necessary condition for the uniqueness of the solution to (5.34). Also, because any $d_{\min} - 1$ rows of $\mathbf{H}^{\mathrm{T}}$ (and hence $\mathbf{H}_{J'}^{\mathrm{T}}$) are linearly independent, (5.34) is guaranteed to have a unique solution whenever $|J'| < d_{\min}$.

One may solve for the unknowns $\mathbf{c}_{J'}$ as follows. First apply Gaussian elimination to $\mathbf{H}$, targeting the columns in the index set J' to produce a modified matrix $\tilde{\mathbf{H}}$, which we assume without loss of generality to be of the form $\tilde{\mathbf{H}} = \begin{bmatrix} \tilde{\mathbf{H}}_J & \tilde{\mathbf{H}}_{J'} \end{bmatrix}$. The submatrix $\tilde{\mathbf{H}}_{J'}$ will have the form

$$\tilde{\mathbf{H}}_{J'} = \begin{bmatrix} \mathbf{T} \\ \mathbf{M} \end{bmatrix},$$

where $\mathbf{T}$ is a $|J'| \times |J'|$ lower-triangular matrix with ones along the diagonal and $\mathbf{M}$ is an arbitrary binary matrix (which may be empty).

Thus, one may solve for the unknowns in $\mathbf{c}_{J'}$ by successively solving the $|J'|$ parity-check equations represented by the top $|J'|$ rows of $\tilde{\mathbf{H}}$. This is possible provided that none of the top $|J'|$ rows of $\tilde{\mathbf{H}}_J$ are all zeros. None of these rows can be all zeros since, for example, if row $p \leq |J'|$ of $\tilde{\mathbf{H}}_J$ is all zeros, then the corresponding row of $\tilde{\mathbf{H}}$ has Hamming weight 1. Further, any word with a one in the same position as the single one in this row of $\tilde{\mathbf{H}}$ will have $\mathbf{c}\tilde{\mathbf{H}}^{\mathrm{T}} \neq \mathbf{0}$ and thus is not a valid codeword. However, half of the codewords of any reasonable linear code have a one in any given position. (A "reasonable code" is one whose generator matrix $\mathbf{G}$ has no all-zero columns.) Thus, it follows that none of the top $|J'|$ rows of $\tilde{\mathbf{H}}_J$ are all zeros and the unknowns in $\mathbf{c}_{J'}$ may be determined.

Example 5.9. We continue with the (7,4) Hamming code and the received pattern $\mathbf{r} = [e\ e\ e\ 1\ 1\ 1\ 1]$ which contains a stopping set in the positions $J' = \{0, 1, 2\}$. A few

elementary row operations on $\mathbf{H}$ yield an alternative $\mathbf{H}$ matrix adapted to the erasure positions,

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

which is in the form $\tilde{\mathbf{H}} = \begin{bmatrix} \mathbf{T} & \tilde{\mathbf{H}}_J \end{bmatrix}$. It is evident that, due to the presence of the triangular matrix, the three check equations may be solved in succession and all three erasures found to be 1.

5.7.3 Gallager's Algorithm A and Algorithm B for the BSC

Gallager presented in his seminal work [1] several decoding algorithms for LDPC codes on the BSC. These algorithms have been called Gallager Algorithm A, Gallager Algorithm B, and the bit-flipping algorithm. The first two algorithms will be discussed here, and the bit-flipping algorithm will be discussed in the next subsection.

Gallager Algorithms A and B are message-passing algorithms much like the SPA, so we focus on decoding Steps 2 and 3. Because LLRs are not involved, we use $M_{j\to i}$ instead of $L_{j\to i}$ to denote the message to be passed from VN j to CN i. Similarly, we use $M_{i\to j}$ instead of $L_{i\to j}$. Additionally, M_j will be used in lieu of L_j for the (intrinsic) information coming from the channel. Note that M_j is a binary quantity since we are dealing with the BSC and $M_{j\to i}$, $M_{i\to j}$ are binary messages.

The check-node update equation for Algorithm A is

$$M_{i\to j} = \bigoplus_{j' \in N(i)-\{j\}} M_{j'\to i},$$

where $\oplus$ represents modulo-2 summation. Note that this ensures that the sum of the incoming messages $\{M_{j'\to i}\}$ and the outgoing message $M_{i\to j}$ is zero, modulo 2. At the variable nodes, the outgoing message $M_{j\to i}$ is set to M_j unless all of the incoming extrinsic messages $M_{i'\to j}$ disagree ($i' \neq i$), in which case, $M_{j\to i}$ is set to M_j^{c}, the logical complement of M_j. Notationally, this is

$$M_{j\to i} = \begin{cases} M_j \text{ if } \exists\, i' \in N(j) - \{i\} \text{ s.t. } M_{i'\to j} = M_j, \\ M_j^{\mathrm{c}} \text{ otherwise.} \end{cases}$$

Gallager's decoding Algorithm B is a refinement of Algorithm A in that outgoing VN messages $M_{j\to i}$ do not require that all of the extrinsic messages $M_{i'\to j}$ disagree with M_j in order for the outgoing message to be M_j^{c}. That is, if there are at least t such messages, then $M_{j\to i}$ will be set to M_j^{c}, otherwise it will be set to M_j. Thus, the VN update rule for Algorithm B is

$$M_{j\to i} = \begin{cases} M_j^{\mathrm{c}} \text{ if } \left|\left\{i' \in N(j) - \{i\}\colon M_{i'\to j} = M_j^{\mathrm{c}}\right\}\right| \geq t, \\ M_j \text{ otherwise.} \end{cases}$$

Gallager showed that for (d_v, d_c)-regular LDPC codes, the optimum value of t is the smallest integer t satisfying

$$\frac{1-\varepsilon}{\varepsilon} \leq \left[\frac{1+(1-2p)^{d_c-1}}{1-(1-2p)^{d_c-1}}\right]^{2b-d_v+1},$$

where ε and p are the intrinsic and extrinsic message error rates, respectively. (ε is, of course, the BSC transition probability.) Observe that, when $d_v = 3$, Algorithms A and B are identical.

The foregoing discussion focused on Steps 2 and 3 in the Gallager A and B decoding algorithms. The code bit decisions for these algorithms are majority-based, as follows. If the degree of a variable node is odd, then the decision is set to the majority of the incoming check node messages. If the degree of a variable node is even, then the decision is set to majority of the check node messages and the channel message

Ardakani and Kschischang [27] have shown that Algorithm B is optimal for regular LDPC codes and that it is also optimal for irregular LDPC codes under very loose conditions.

5.7.4 The Bit-Flipping Algorithm for the BSC

Briefly, the bit-flipping algorithm first evaluates all parity-check equations in $\mathbf{H}$ and then "flips" (complements) any bits in the received word that are involved in more than some fixed number of failed parity checks. This step is then repeated with the modified received word until all parity checks are satisfied or until some maximum number of iterations has been executed.

Noting that failed parity-check equations are indicated by the elements of the syndrome, $\mathbf{s} = \mathbf{r}\mathbf{H}^{\mathrm{T}}$, and that the number of failed parity checks for each code bit is contained in the n-vector $\mathbf{f} = \mathbf{s}\mathbf{H}$ (operations over the integers), the algorithm is presented below.

Algorithm 5.4 The Bit-Flipping Algorithm

1. Compute $\mathbf{s} = \mathbf{r}\mathbf{H}^{\mathrm{T}}$ (operations over $\mathbb{F}^2$). If $\mathbf{s} = \mathbf{0}$, stop, since $\mathbf{r}$ is a codeword.
2. Compute $\mathbf{f} = \mathbf{s}\mathbf{H}$ (operations over $\mathbb{Z}$).
3. Identify the elements of $\mathbf{f}$ greater than some preset threshold, and then flip all the bits in $\mathbf{r}$ corresponding to those elements.
4. If you have not reached the maximum number of iterations, go to Step 1 using the updated $\mathbf{r}$.

The threshold will depend on the channel conditions and Gallager has derived the optimum threshold for regular LDPC codes, which we do not repeat here. Instead, we point out a particularly convenient way to obtain a threshold that effectively adapts to the channel quality:

3′ Identify the elements of $\mathbf{f}$ equal to $\max\{f_j\}$ and then flip all the bits in $\mathbf{r}$ corresponding to those elements.

Examples of the performance of this algorithm will be presented in subsequent chapters. Additional discussions of the bit-flipping algorithm, including weighted bit-flipping, may be found in Chapter 10.

5.8 Concluding Remarks

This chapter has presented a number of message-passing decoding algorithms for several important communication channels. As mentioned early on in the chapter, the flooding schedule was adopted in each case. However, it should be emphasized that other schedules are possible, most designed with an eye toward lowering the error floor. Examples can be found in [30, 31]. We also mention that other variations and approximations of the sum–product decoding algorithm exist in the literature [32–35], too many to present here.

Problems

5.1 One version of the $\mathbf{H}$ matrix for the (7,4) Hamming code can be created by listing as its columns all of the nonzero 3-vectors:

$$\mathbf{H}_1 = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}.$$

Draw the Tanner graph for this code and identify all cycles. Repeat for the cyclic version of the (7,4) Hamming code whose generator matrix is given by $g(x) = \left(1 + x + x^3\right)$ and whose $\mathbf{H}$ matrix is

$$\mathbf{H}_2 = \begin{bmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix}.$$

5.2 Re-ordering the rows of an $\mathbf{H}$ matrix does not change the code. Re-ordering the columns of an $\mathbf{H}$ matrix changes the code in the sense that the code bits in each codeword are re-ordered, but the weight spectrum is not changed. Neither of these operations eliminate cycles in the code's Tanner graph. Check whether it is possible to eliminate the 4-cycles in $\mathbf{H}_1$ in the previous problem by replacing rows by a sum of rows. Repeat for $\mathbf{H}_2$.

5.3 Derive Equations (5.3)–(5.6).

5.4 Derive Equations (5.7) and (5.8).

5.5 Via iterative erasure decoding applied to a $(3,2) \times (3,2)$ SPC product code, decode the following codeword received at the output of a BEC (see Example 5.2):

$$\begin{bmatrix} e & 1 & e \\ e & e & 0 \\ e & 0 & 1 \end{bmatrix}$$

Also, find all minimal stopping sets for the graph in Figure 5.3(c).

5.6 Derive $L(v_j|y_j)$ for the BEC, the BSC, the BI-AWGNC, and the Rayleigh channel starting from the channel transition probabilities in each case. Expressions for $L(v_j|y_j)$ together with the models and assumptions are given in Section 5.4.4, but put in all of the missing steps.

5.7 (a) Suppose that, instead of the mapping $x_j = 1 - 2v_j$ defined in this chapter for the BI-AWGNC, we let $x_j = 2v_j - 1$. How does this change the initial messages $L(v_j|y_j)$? (b) Suppose we change the LLR definition so that the zero hypothesis is in the denominator:

$$L(v_j|\mathbf{y}) = \log\left(\frac{\Pr(v_j = 1|\mathbf{y})}{\Pr(v_j = 0|\mathbf{y})}\right).$$

Under this assumption, derive the update equation for $L_{i\to j}$ that would replace (5.18). (c) Now combine parts (a) and (b) so that $x_j = 2v_j - 1$ and both $L(v_j|y_j)$ and $L(v_j|\mathbf{y})$ are defined with the zero hypothesis in the denominator. Find the initial messages $L(v_j|y_j)$ and the update equation for $L_{i\to j}$.

5.8 Consider an LDPC-coded M-ary pulse-position modulation (PPM) free-space optical system configured as follows: LDPC encoder $\to$ PPM modulator $\to$ PPM soft-out demodulator $\to$ LDPC decoder. Under this configuration, each group of $\log_2 M$ LDPC code bits, denoted by $\mathbf{d}$, is mapped to an M-bit channel word, denoted by $\mathbf{x}$, representing the PPM channel symbol. For example, with $M = 4$, $\mathbf{d} = 00 \to \mathbf{x} = 0001$, $\mathbf{d} = 01 \to \mathbf{x} = 0010$, $\mathbf{d} = 10 \to \mathbf{x} = 0100$, and $\mathbf{d} = 11 \to \mathbf{x} = 1000$. With a channel-bit $x_j = 1$ representing a pulse of light and a channel-bit $x_j = 0$ representing the absence of light, the channel with a photon-counting detector at the receiver's front end can be modeled as Poisson so that

$$\Pr(y_j|x_j = 1) = \frac{(n_s + n_b)^{y_j} e^{-(n_s+n_b)}}{y_j!},$$
$$\Pr(y_j|x_j = 0) = \frac{(n_b)^{y_j} e^{-n_b}}{y_j!}.$$

In these expressions, y_j is the number of received photons for the jth PPM slot (corresponding to x_j), n_s is the mean number of signal photons in a PPM slot and n_b is the mean number of noise photons in a PPM slot. Show that the initial LLRs passed to the LDPC decoder from the PPM soft-out demodulator can be

computed as

$$\begin{aligned} L(d_k \mid \mathbf{y}) &= \log\left[\frac{\Pr(d_k = 1|\mathbf{y})}{\Pr(d_k = 0|\mathbf{y})}\right] \\ &= \log\left[\frac{\sum_{\mathbf{x}\in X_1^k} \Pr(\mathbf{x}|\mathbf{y})}{\sum_{\mathbf{x}\in X_0^k} \Pr(\mathbf{x}|\mathbf{y})}\right] \\ &= \log\left[\frac{\sum_{\mathbf{x}\in X_1^k} (1+n_s/n_b)^{y_{l(\mathbf{x})}}}{\sum_{\mathbf{x}\in X_0^k} (1+n_s/n_b)^{y_{l(\mathbf{x})}}}\right], \end{aligned}$$

where $X_1^k \triangleq \{\mathbf{x}\colon d_k = 1\}$, $X_0^k \triangleq \{\mathbf{x}\colon d_k = 0\}$, and the subscript $l(\mathbf{x})$ is the index of the PPM symbol $\mathbf{x}$ at which the 1 is located. (As an example, $l(1000) = 3$, $l(0010) = 1$, and $l(0001) = 0$.) Further, show that this can be rewritten as

$$L(d_k|\mathbf{y}) = \max_{\mathbf{x}\in X_1^k}{}^*\left\{y_{l(\mathbf{x})} \cdot (1+n_s/n_b)\right\} - \max_{\mathbf{x}\in X_0^k}{}^*\left\{y_{l(\mathbf{x})} \cdot (1+n_s/n_b)\right\}.$$

5.9 Prove Equations (5.13) and (5.14) using induction.

5.10 Derive the probability-domain SPA decoder presented below. (The q-ary version is presented in Chapter 14.) Define the messages to be passed, which are probabilities rather than LLRs, as follows:

- $p_{j\to i}(b)$ is the probabilistic message that VN j sends to CN i about the probability that code bit j is $b \in \{0,1\}$, given (i) the decoder input from the channel, (ii) the probabilistic messages, $\{p_{i'\to j}(b)\colon i' \in N(j) - \{i\}\}$, received from the exclusive neighborhood, and (iii) the REP constraint for VN j;
- $p_{i\to j}(b)$ is the probabilistic message that CN i sends to VN j about the probability that code bit j is $b \in \{0,1\}$, given (i) the probabilistic messages, $\{p_{j'\to i}(b)\colon j' \in N(i) - \{j\}\}$, received from the exclusive neighborhood of j, and (ii) the SPC constraint for CN i;
- $P_j(b)$ is the probability that code bit j is $b \in \{0,1\}$, given (i) the decoder input from the channel, (ii) the probabilistic messages, $\{p_{i\to j}(b)\colon i \in N(j)\}$, received from the neighborhood of j, and (iii) the REP constraint for VN j.

Probability-Domain SPA

1. For all i, j for which $h_{ij} = 1$, set $p_{j\to i}(0) = \Pr(c_j = 0|y_j)$ and $p_{j\to i}(1) = \Pr(c_j = 1|y_j)$, where y_j is the jth received channel value.
2. For each $b \in \{0,1\}$, update $\{p_{i\to j}(b)\}$ at each CN using

$$p_{i\to j}(b) = \frac{1}{2} + \frac{(-1)^b}{2} \prod_{j'\in N(i)-\{j\}} (1 - 2p_{j'\to i}(1)).$$

3. For each $b \in \{0, 1\}$, update $\{p_{j\to i}(b)\}$ for each VN using

$$p_{j\to i}(b) = K_{ji} \Pr(c_j = b|y_j) \prod_{i' \in N(j)-\{i\}} p_{i'\to j}(b),$$

where the constants K_{ji} are selected to ensure that $p_{j\to i}(0) + p_{j\to i}(1) = 1$.

4. For each $b \in \{0, 1\}$, and for each $j = 0, 1, \ldots, n-1$, compute

$$P_j(b) = K_j \Pr(c_j = b|y_j) \prod_{i \in N(j)} p_{i\to j}(b),$$

where the constants K_j are chosen to ensure that $P_j(0) + P_j(1) = 1$.

5. For $j = 0, 1, \ldots, n-1$, set

$$\hat{v}_j = \begin{cases} 1 & \text{if } P_j(1) > P_j(0), \\ 0 & \text{else.} \end{cases}$$

If $\hat{\mathbf{v}} H^{\mathrm{T}} = \mathbf{0}$ or the number of iterations equals the maximum limit, stop; else, go to Step 2.

5.11 Show that the min-sum algorithm on the BI-AWGNC requires no knowledge of the noise sample variance σ^2. That is, show that the min-sum decoder will operate identically if the initial LLRs (and hence all subsequent LLRs) are scaled by $\sigma^2/2$, yielding initial LLRs of y_j.

5.12 Show that (5.21) is equivalently

$$L(A_2) = 2 \tanh^{-1}\left(\prod_{l=1}^{2} \tanh(L_l/2)\right).$$

Hint: There is more than one way to show this, but you might find this useful:

$$\tanh^{-1}(z) = \frac{1}{2} \ln\left(\frac{1+z}{1-z}\right).$$

5.13 Consider d binary random variables, $a_1, a_2, \ldots, a_d$ and denote by L_l the LLR of a_l:

$$L_l = L(a_l) = \log\left(\frac{\Pr(a_l = 0)}{\Pr(a_l = 1)}\right).$$

Show, by induction, that

$$\boxplus_{l=1}^{d} L_l = \ln\left(\frac{\prod_{l=1}^{d}\left(e^{L_l}+1\right) + \prod_{l=1}^{d}\left(e^{L_l}-1\right)}{\prod_{l=1}^{d}\left(e^{L_l}+1\right) - \prod_{l=1}^{d}\left(e^{L_l}-1\right)}\right),$$

from which we may write

$$\boxplus_{l=1}^{d} L_l = \ln\left(\frac{1+\prod_{l=1}^{d}\tanh(L_l/2)}{1-\prod_{l=1}^{d}\tanh(L_l/2)}\right)$$
$$= 2\tanh^{-1}\left(\prod_{l=1}^{d}\tanh(L_l/2)\right).$$

Note that

$$\tanh^{-1}(z) = \frac{1}{2}\ln\left(\frac{1+z}{1-z}\right).$$

5.14 Consider d binary random variables, $a_1, a_2, \ldots, a_d$ and denote by L_l the LLR of a_l:

$$L_l = L(a_l) = \log\left(\frac{\Pr(a_l = 0)}{\Pr(a_l = 1)}\right).$$

Consider also a binary-valued constraint χ on these variables. As an example, for the single parity-check constraint, $\chi(\mathbf{a}) = 0$ when the Hamming weight of $\mathbf{a} = (a_1, a_2, \ldots, a_d)$ is even and $\chi(\mathbf{a}) = 1$ otherwise. Show that the LLR for $\chi(\mathbf{a})$ is given by

$$L(\chi(\mathbf{a})) = \log\left(\frac{\Pr(\chi(\mathbf{a}) = 0)}{\Pr(\chi(\mathbf{a}) = 1)}\right)$$
$$= \max_{\mathbf{a}:\chi(\mathbf{a})=0}{}^{*}\left(\sum_{l=1}^{d}\delta_{a_l}L_l\right) - \max_{\mathbf{a}:\chi(\mathbf{a})=1}{}^{*}\left(\sum_{l=1}^{d}\delta_{a_l}L_l\right),$$

where δ_k is the Kronecker delta. Note that this may alternatively be written as

$$L(\chi(\mathbf{a})) = \max_{\mathbf{a}:\chi(\mathbf{a})=0}{}^{*}\left(\sum_{l=1}^{d}a_l^{\mathrm{c}}L_l\right) - \max_{\mathbf{a}:\chi(\mathbf{a})=1}{}^{*}\left(\sum_{l=1}^{d}a_l^{\mathrm{c}}L_l\right),$$

where a_l^{c} is the logical complement of the binary number a_l. Note also, for the special case of $L = 2$ and an SPC constraint, that this becomes

$$L(\chi(\mathbf{a})) = L(a_1 \oplus a_2) = \max{}^{*}(0, L_1 + L_2) - \max{}^{*}(L_1, L_2).$$

5.15 Consider the rate-1/2 product code depicted below, whose rows and columns form (3,2) SPC codewords:

$$\begin{array}{ccc} c_0 & c_1 & c_2 \\ c_3 & c_4 & c_5 \\ c_6 & c_7 & \end{array}$$

(a) Program the probability-domain SPA decoder (see Problem 5.10) and decode the following received word assuming $\sigma^2 = 0.5$: $\mathbf{y} = (0.2, 0.2, -0.9, 0.6, 0.5, -1.1, -0.4, -1.2)$. You should find that the decoder converges to the codeword $\mathbf{c} = (1, 0, 1, 0, 1, 1, 1, 1)$ after seven iterations with the following cumulative probabilities at the eight VNs: $(0.740, 0.338, 0.969,\ 0.409, 0.787,\ 0.957, 0.775, 0.992)$. (b) Program the log-domain SPA decoder and repeat part (a). You should find that the decoder converges to the codeword $\mathbf{c} = (1, 0, 1, 0, 1, 1, 1, 1)$ after seven iterations with the following cumulative LLRs at the eight VNs: $(-1.05, 0.672, -3.45,\ 0.370, -1.306, -3.095, -1.239, -4.863)$. (c) Show that these results are in agreement.

5.16 Simulate SPA decoding of the rate-1/2 product code of the previous problem on the binary-input AWGN channel and plot the bit error probability P_b versus E_b/N_0 (dB). Simulate an exhaustive-search maximum-likelihood decoder (i.e., minimum-Euclidean-distance decoder) on the binary-input AWGN channel and compare its P_b versus E_b/N_0 curve with that of the SPA decoder.

5.17 Carry out the previous problem with the SPA decoder replaced by the min-sum decoder.

5.18 Simulate SPA decoding of the (7,4) Hamming code on the binary-input AWGN channel and plot the bit error probability P_b versus E_b/N_0 (dB). Use the following $\mathbf{H}$ matrix for the (7,4) Hamming code to design your SPA decoder:

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

Simulate an exhaustive-search maximum-likelihood decoder (i.e., a minimum-Euclidean-distance decoder) on the binary-input AWGN channel and compare its P_b versus E_b/N_0 curve with that of the SPA decoder.

5.19 Carry out the previous problem with the SPA decoder replaced by the min-sum decoder.

5.20 Let a be a binary r.v. and let $p_1 = \Pr(a = 1)$. Show that

$$p_1 = \frac{1}{1 + \exp(-L(a))}$$

and

$$p_1 = \frac{1}{2} + \frac{1}{2}\tanh\left(\frac{L(a)}{2}\right),$$

where $L(a) = \ln(p_0/p_1)$.

5.21 Show that

$$\max(0, x + y) - \max(x, y) = \ \mathrm{sign}(x)\mathrm{sign}(y)\min(|x|, |y|)$$

for any pair of real numbers x, y.

5.22 Assume LDPC-coded transmission on the BEC. Show that the iterative erasure-filling decoder is precisely the SPA decoder and, hence, is precisely the min-sum decoder. (Consult Section 5.5.1 and the example therein.)

5.23 Assuming iterative erasure-filling decoding, find all uncorrectable x-erasure patterns, $x = 1, 2, 3$, for the two Hamming code representations of Problem 5.1. Explain why the decoder is unable to resolve these erasure patterns (consider the stopping-set terminology).

5.24 Show that, for the BEC, the ML decoder chooses a codeword $\mathbf{c} \in C$ satisfying $c_j = r_j$ for all $j \in J$ and, further, that this is a solution to the equation

$$\mathbf{c}_{J'}\mathbf{H}_{J'}^{\mathrm{T}} = \mathbf{c}_J\mathbf{H}_J^{\mathrm{T}},$$

where $\mathbf{H}_J$ ($\mathbf{H}_{J'}$) is the submatrix of the code's parity-check matrix $\mathbf{H}$ obtained by taking only the columns of $\mathbf{H}$ corresponding to J (J'), and similarly for $\mathbf{c}_J$ ($\mathbf{c}_{J'}$).

References

[1] R. G. Gallager, *Low-Density Parity-Check Codes*, Cambridge, MA, MIT Press, 1963. (Also, R. G. Gallager, "Low density parity-check codes," *IRE Trans. Information Theory*, vol. 8, no. 1, pp. 21–28, January 1962.)

[2] R. M. Tanner, "A recursive approach to low complexity codes," *IEEE Trans. Information Theory*, vol. 27, no. 9, pp. 533–547, September 1981.

[3] D. MacKay and R. Neal, "Good codes based on very sparse matrices," *Cryptography and Coding, 5th IMA Conf.,* C. Boyd, Ed., Berlin, Springer-Verlag, October 1995.

[4] D. MacKay, "Good error correcting codes based on very sparse matrices," *IEEE Trans. Information Theory*, vol. 45, no. 3, pp. 399–431, March 1999.

[5] N. Alon and M. Luby, "A linear time erasure-resilient code with nearly optimal recovery," *IEEE Trans. Information Theory*, vol. 42, no. 11, pp. 1732–1736, November 1996.

[6] J. Byers, M. Luby, M. Mitzenmacher, and A. Rege, "A digital fountain approach to reliable distribution of bulk data," *ACM SIGCOMM Computer Communication Rev.*, vol. 28, issue 4, October 1998.

[7] Y. Wang and M. Fossorier, "Doubly generalized LDPC codes," *2006 IEEE Int. Symp. on Information Theory*, pp. 669–673, July 2006.

[8] J. Pearl, *Probabilistic Reasoning in Intelligent Systems*, San Mateo, CA, Morgan Kaufmann, 1988.

[9] F. Kschischang, B. Frey, and H.-A. Loeliger, "Factor graphs and the sum-product algorithm," *IEEE Trans. Information Theory*, vol. 47, no. 2, pp. 498–519, February 2001.

[10] Y. Zhang, *Design of Low-Floor Quasi-Cyclic IRA Codes and Their FPGA Decoders*, Ph.D. dissertation, ECE Dept., University of Arizona, May 2007.

[11] Y. Zhang and W. E. Ryan, "Toward low LDPC-code floors: A case study," *IEEE Trans. Communications*, pp. 1566–1573, June 2009.

[12] Y. Han and W. E. Ryan, "Low-floor decoders for LDPC codes," *IEEE Trans. Communications*, pp. 1663–1673, June 2009.

[13] J. Chen, R. M. Tanner, C. Jones, and Y. Li, "Improved min-sum decoding algorithms for irregular LDPC codes," *Proc. Int. Symp. Information Theory*, pp. 449–453, September 2005.

[14] J. Zhao, F. Zarkeshvari, and A. Banihashemi, "On implementation of min-sum algorithm and its modifications for decoding low-density parity-check (LDPC) codes," *IEEE Trans. Communications*, vol. 53, no. 4, pp. 549–554, April 2005.

[15] J. Chen, A. Dholakia, E. Eleftheriou, M. Fossorier, and X.-Y. Hu, "Reduced-complexity decoding of LDPC codes," *IEEE Trans. Communications*, vol. 53, no. 8, pp. 1288–1299, August 2005.

[16] Z. Zhang, L. Dolecek, B. Nikolic, V. Anantharam, and M. Wainwright, "Lowering LDPC error floors by postprocessing," *2008 IEEE GlobeCom Conf.*, pp. 1–6, November–December 2008.

[17] X.-Y. Hu, E. Eleftheriou, D.-M. Arnold, and A. Dholakia, "Efficient implementation of the sum–product algorithm for decoding LDPC codes," *Proc. 2001 IEEE GlobeCom Conf.*, pp. 1036–1036E, November 2001.

[18] C. Jones, E. Valles, M. Smith, and J. Villasenor, "Approximate-min* constraint node updating for LDPC code decoding," *IEEE Military Communications Conf.*, pp. 157–162, October 2003.

[19] C. Jones, S. Dolinar, K. Andrews, D. Divsalar, Y. Zhang, and W. Ryan, "Functions and architectures for LDPC decoding," *2007 Information Theory Workshop*, pp. 577–583, September 2007.

[20] Z.-W. Li, L. Chen, L.-Q. Zeng, S. Lin, and W. Fong, "Efficient encoding of quasi-cyclic low-density parity-check codes," *IEEE Trans. Communications*, vol. 54, no. 1, pp. 71–81, January 2006.

[21] L. H. Miles, J. W. Gambles, G. K. Maki, W. E. Ryan, and S. R. Whitaker, "An 860-Mb/s (8158,7136) low-density parity-check encoder," *IEEE J. Solid-State Circuits*, pp. 1686–1691, August 2006.

[22] T. Richardson and V. Novichikov, "Node processors for use in parity check decoders," United States Patent 6,938,196 B2, August 30, 2005.

[23] M. Viens and W. E. Ryan, "A reduced-complexity box-plus decoder for LDPC codes," *Fifth Int. Symp. on Turbo Codes and Related Topics*, pp. 151–156, September 2008.

[24] M. Viens, *A Reduced-Complexity Iterative Decoder for LDPC Codes*, M. S. Thesis, ECE Dept., University of Arizona, December 2007.

[25] L. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate," *IEEE Trans. Information Theory*, vol. 20, no. 3, pp. 284–287, March 1974.

[26] R. McEliece, "On the BCJR trellis for linear block codes," *IEEE Trans. Information Theory*, vol. 42, no. 7, pp. 1072–1092, July 1996.

[27] M. Ardakani, *Efficient Analysis, Design and Decoding of Low-Density Parity-Check Codes*, Ph.D. dissertation, University of Toronto, 2004.

[28] Y. Kou, S. Lin, and M. Fossorier, "Low-density parity-check codes based on finite geometries: a rediscovery and new results," *IEEE Trans. Information Theory*, vol. 47, no. 11, pp. 2711–2736, November 2001.

[29] S. Lin and D. J. Costello, Jr., *Error Control Coding*, 2nd edn., New Saddle River, NJ, Prentice-Hall, 2004.

[30] H. Xiao and A. H. Banihashemi, "Graph-based message-passing schedules for decoding LDPC codes," *IEEE Trans. Communications*, vol. 52, no. 12, pp. 2098–2105, December 2004.

[31] A. Vila Casado, M. Griot, and R. D. Wesel, "Informed scheduling for belief-propagation decoding of LDPC codes," *2007 Int. Conf. on Communications*, pp. 932–937, June 2007.

[32] M. Mansour and N. Shanbhag, "High-throughput LDPC decoders," *IEEE Trans. VLSI Systems*, pp. 976–996, December 2003.

[33] J. Chen and M. Fossorier, "Near optimum universal belief propagation based decoding of low-density parity check codes," *IEEE Trans. Communications*, vol. 50, no. 3, pp. 406–414, March 2002.

[34] J. Chen, A. Dholakia, E. Eleftheriou, M. Fossorier, and X.-Y Hu. "Reduced-complexity decoding of LDPC codes, *IEEE Trans. Communications*, vol. 53, no. 8, pp. 1288–1299, August 2005.

[35] M. Fossorier, "Iterative reliability-based decoding of low-density parity check codes," *IEEE J. Selected Areas Communications*, vol. 19, no. 5, pp. 908–917, May 2001.

6 Computer-Based Design of LDPC Codes

Unlike BCH and Reed–Solomon codes, for which there exists essentially a single code-design procedure, for LDPC codes there are many code-design approaches. Many of these design approaches, including that for the original LDPC codes, are computer-based algorithms. Many others rely on finite mathematics such as the design techniques of Chapters 10–14. In the present chapter, we first present the original design techniques of Gallager [1] and MacKay [2]. We then provide an overview of two popular computer-based design algorithms: the progressive-edge-growth algorithm [6] and the approximate cycle extrinsic message degree algorithm [7]. Next, we present structured classes of LDPC codes, including protograph-based LDPC codes, accumulator-based LDPC codes, and generalized LDPC codes. The discussion includes selected code-design case studies to provide the student with some useful code-design approaches.

6.1 The Original LDPC Codes

As mentioned in the previous chapter, LDPC codes were originally invented by R. Gallager circa 1960 and were later re-invented by others, including D. MacKay, circa 1995. In this section, we will briefly overview Gallager's and MacKay's LDPC code-design approaches.

6.1.1 Gallager Codes

Gallager's original definition of a (regular) low-density parity-check code was identical to that given in Chapter 5: it is a linear code whose $m \times n$ parity-check matrix $\mathbf{H}$ has $g \ll m$ ones in each column and $r \ll n$ ones in each row. The LDPC code-construction technique he presented is as follows. The matrix $\mathbf{H}$ has the form

$$\mathbf{H} = \begin{bmatrix} \mathbf{H}_1 \\ \mathbf{H}_2 \\ \vdots \\ \mathbf{H}_g \end{bmatrix}, \tag{6.1}$$

where the submatrices $\mathbf{H}_a$, $a = 1, \ldots, g$, have the following structure. For any integers μ and r greater than 1, each submatrix $\mathbf{H}_a$ is $\mu \times \mu r$ with row weight r and column weight 1. The submatrix $\mathbf{H}_1$ has the following specific form: for $i = 0, 1, \ldots, \mu - 1$, the ith row contains all of its r 1s in columns ir to $(i+1)r - 1$. The other submatrices are obtained by column permutations of $\mathbf{H}_1$. It is evident that $\mathbf{H}$ is regular, has dimension $\mu g \times \mu r$, and has row and column weights r and g, respectively. The absence of length-4 cycles in $\mathbf{H}$ is not guaranteed, but they can be avoided via computer design of $\mathbf{H}$. Gallager showed that the ensemble of such codes has excellent distance properties, provided that $g \geq 3$. Further, Gallager pointed out that such codes have low-complexity encoders since parity bits can be solved for as a function of the user bits via the parity-check matrix.

Example 6.1. The following $\mathbf{H}$ matrix is the first example given by Gallager [1]:

$$\mathbf{H} = \left[\begin{array}{cccccccccccccccccccc}
1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&1&1&1&1&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1\\
\hline
1&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&0&0&0&0\\
0&1&0&0&0&1&0&0&0&1&0&0&0&0&0&0&1&0&0&0\\
0&0&1&0&0&0&1&0&0&0&0&0&0&1&0&0&0&1&0&0\\
0&0&0&1&0&0&0&0&0&0&1&0&0&0&1&0&0&0&1&0\\
0&0&0&0&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&1\\
\hline
1&0&0&0&0&1&0&0&0&0&0&1&0&0&0&0&0&1&0&0\\
0&1&0&0&0&0&1&0&0&0&1&0&0&0&0&1&0&0&0&0\\
0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&0&1&0\\
0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&1&0&0&0\\
0&0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&1
\end{array}\right].$$

It corresponds to a (20,5) code with $g = 3$ and $r = 4$. Observe that this matrix has the form of (6.1) and $\mathbf{H}_1$ has the form described above, with $\mu = 5$.

6.1.2 MacKay Codes

Thirty-five years after Gallager had done so, MacKay, unaware of Gallager's work, independently discovered the benefits of designing binary codes with sparse $\mathbf{H}$ matrices and was the first to show by computer simulation the ability of these codes to perform near capacity limits on the BSC and BI-AWGNC. MacKay has archived on a web page [3] a large number of LDPC codes he has designed for application to data communication and storage. A few of the computer-based design algorithms suggested by MacKay are listed below in order of increasing algorithm complexity (but not necessarily improved performance).

1. $\mathbf{H}$ is created by randomly generating weight-g columns and, as nearly as possible, uniform row weight.
2. $\mathbf{H}$ is created by randomly generating weight-g columns, while ensuring weight-r rows, and the ones-overlap of any two columns is at most one (i.e., ensuring the RC constraint to avoid length-4 cycles).
3. $\mathbf{H}$ is generated as in 2, plus additional short cycles are avoided.
4. $\mathbf{H}$ is generated as in 3, plus $\mathbf{H} = \begin{bmatrix} \mathbf{H}_1 & \mathbf{H}_2 \end{bmatrix}$ is constrained so that $\mathbf{H}_2$ is invertible.

One drawback of MacKay codes is that they lack sufficient structure to enable low-complexity encoding. Encoding is performed by putting $\mathbf{H}$ in the form $\begin{bmatrix} \mathbf{P}^{\mathrm{T}} & \mathbf{I} \end{bmatrix}$ via Gauss–Jordan elimination (or by multiplying $\mathbf{H}$ by $\mathbf{H}_2^{-1}$ for algorithm 4), from which the generator matrix can be put into the systematic form $\mathbf{G} = \begin{bmatrix} \mathbf{I} & \mathbf{P} \end{bmatrix}$.[1] The problem with encoding via $\mathbf{G}$ is that the submatrix $\mathbf{P}$ is generally not sparse, so for codes of practical interest the encoding complexity is high. An efficient encoding technique based on the $\mathbf{H}$ matrix was proposed in [4] for arbitrary $\mathbf{H}$ matrices. Later in this chapter and in Chapters 10–14, we give various approaches for introducing structure into parity-check matrices so that encoding is facilitated, and with essentially no performance loss relative to less structured parity-check matrices.

6.2 The PEG and ACE Code-Design Algorithms

6.2.1 The PEG Algorithm

As indicated in the previous chapter, cycles in the Tanner graph of an LDPC code present problems for iterative decoders. Because of this, many papers dealing with the design of Tanner graphs with large girths have been published. While some design techniques have relied on mathematics for achieving large girths (see Chapters 10–14 and [5]), the progressive-edge-growth (PEG) algorithm [6] is very effective and has been used widely for computer-based code design. Although the PEG algorithm is described in detail in Chapter 12, we here provide an overview of this algorithm, for it is a popular computer-based design algorithm.

The PEG algorithm is initialized by the number of variable nodes, n, the number of check nodes, m, and a variable node-degree sequence D_v, which is the list of degrees for each of the n variable nodes. Given these parameters, the goal of the algorithm is to build the graph one edge at a time, and each edge is added to the

[1] We note that Chapter 3 introduces the generator matrix form $\mathbf{G} = \begin{bmatrix} \mathbf{P} & \mathbf{I} \end{bmatrix}$, whereas here we use the form $\mathbf{G} = \begin{bmatrix} \mathbf{I} & \mathbf{P} \end{bmatrix}$. Both are commonly used in the literature and we use both in this book. We have found that the student is comfortable with both forms.

graph in a manner that maximizes the local girth. Thus, the PEG algorithm is a greedy algorithm for creating a Tanner graph with a large girth.

Because the low-degree VNs are the most susceptible to error (they receive the least amount of neighborly help), edge placement begins with the lowest-degree VNs and progresses to VNs of increasing (or non-decreasing) degree. The algorithm does not move to the next VN until all of the edges of the current VN have been attached. The first edge attached to a VN is connected to a lowest-degree CN under the current state of the graph. Subsequent attachments of edges to the VN are done in such a way that the (local) girth for that VN is maximum. Thus, if the current state of the graph is such that one or more CNs cannot be reached from the current VN by traversing the edges connected so far, then the edge should be connected to an unreachable CN so that no cycle is created. Otherwise, if all CNs are reachable from the current VN along some number of edges, the new edge should be connected to a CN of lowest degree that results in the largest girth seen by the current VN. This lowest-degree CN strategy will yield a fairly uniform CN degree distribution.

6.2.2 The ACE Algorithm

The ACE algorithm [7] (acronym defined below) is another computer-based code-design algorithm that accounts for the pitfalls of iterative decoding. Specifically, it accounts for the fact that the iterative decoder not only has difficulties with cycles, but also is hindered by the overlap of multiple cycles. The ACE algorithm was motivated by the following observations made in [7].

1. In a Tanner graph for which each VN degree is at least 2, every stopping set contains multiple cycles, except for the special case in which all VNs in the stopping set are degree-2 VNs, in which case the stopping set is a single cycle. (Recall from Chapter 5 that a stopping set S is a set of VNs whose neighboring CNs are connected to S at least twice. Stopping sets thwart iterative decoding on erasure channels.)
2. For a code with minimum distance $d_{\min}$, each set of $d_{\min}$ columns of $\mathbf{H}$ that sum to the zero vector corresponds to a set of VNs that form a stopping set.
3. The previous result implies that preventing small stopping sets in the design of an LDPC code (i.e., the construction of $\mathbf{H}$) also prevents a small $d_{\min}$.

The upshot of these results is that a code that is well suited for iterative decoding should have a minimum stopping-set size that is as large as possible. One approach for doing this is of course maximizing the girth for the code's Tanner graph, using the PEG algorithm, for example. However, as noted in [7], this is not sufficient. Because short cycles are susceptible to iterative decoding errors, the ACE algorithm seeks to have high connectivity from the VNs within a short cycle to CNs outside of the cycle. In this way, information from outside the cycle flows into the short cycle, under the assumption that such *extrinsic* information is helpful on average.

The description of the ACE algorithm requires the following definitions. An *extrinsic CN* relative to the VNs in a cycle is a CN that is connected to the cycle only once. The *extrinsic message degree* (EMD) for a cycle is the number of such extrinsic CNs connected to it. The *approximate cycle EMD* (ACE) of a length-2δ cycle is the maximum possible EMD for the cycle. Because a degree-d VN within a cycle can be connected to at most $d-2$ extrinsic CNs, the ACE for a length-2δ cycle is given by $\sum_{l=1}^{\delta}(d_l-2)$.

The ACE code-design algorithm builds a Tanner graph ($\mathbf{H}$ matrix) in such a way that the short cycles have large EMD values. Note that this has the effect of increasing the size of the smallest stopping set because the EMD of a stopping set is zero. The ACE algorithm generates a $(\delta_{\text{ACE}}, \epsilon)$ LDPC code, defined as an LDPC code whose cycles of length $2\delta_{\text{ACE}}$ or less have ACE values of ϵ or more. The algorithm begins with the lowest-weight column of $\mathbf{H}$ and progresses to columns of increasing (non-decreasing) weight. Each (randomly generated) candidate column for $\mathbf{H}$ is added only if, in the event its addition creates a cycle of length $2\delta_{\text{ACE}}$ or less, the cycle's ACE value is ϵ or more.

Note that it is possible to combine the PEG and ACE algorithms to obtain a hybrid PEG/ACE algorithm. This was proposed in [8], where the PEG/ACE algorithm was shown to result in codes with good iterative decoding performance. An application of the PEG/ACE algorithm may be found in Section 6.5.2.1. See also Chapter 12 for a description of the PEG/ACE algorithm.

6.3 Protograph LDPC Codes

Soon after the re-invention of LDPC codes in the late 1990s researchers sought improvements in performance and complexity. Performance improvements were achieved via the design of irregular LDPC codes with optimal and near-optimal degree distributions. Such degree distributions may be obtained using so-called density evolution and EXIT chart algorithms, which are discussed in Chapter 9. Notice that the LDPC codes discussed above possess very little structure, i.e., their parity-check matrices are quite random, implying complex encoders and decoders. Improvements in complexity were achieved by considering classes of LDPC codes whose parity-check matrices contain additional structure that facilitates encoding and/or decoding. Some of the more common approaches, each of which may be used in conjunction with the PEG and ACE algorithms, are considered in this and subsequent sections.

With an eye toward simplified design, implementation, and analysis, techniques that rely on the expansion of a smaller matrix or graph prototype (protograph) into a full matrix or graph have been proposed. We shall call such codes protograph-based codes, or simply *protograph codes* [9, 10]. A *protograph* is a relatively small bipartite graph from which a larger graph can be obtained by a copy-and-permute procedure: the protograph is copied Q times, and then

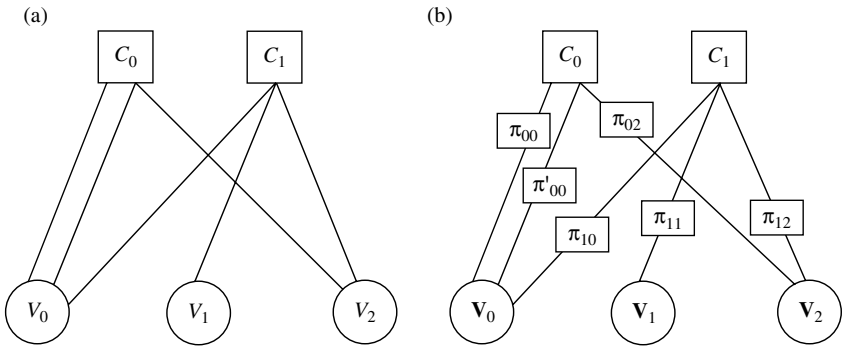


Figure 6.1 (a) An example protograph. (b) The Tanner graph that results after the copy-and-permute procedure is applied to the protograph in (a). Each permutation matrix is $Q \times Q$, each node is interpreted to be a Q-bundle of nodes of the same type, and each edge represents a Q-bundle of edges.

the edges of the individual replicas are permuted among the Q replicas (under restrictions described below) to obtain a single, large bipartite graph. When the protograph possesses n_p variable nodes and m_p constraint nodes, the derived graph will consist of $n = n_p Q$ variable nodes and $m_c = m_p Q$ constraint nodes. This process of expanding a smaller graph into a larger graph by interconnecting copies of the smaller graph is sometimes called *lifting*.

In order to preserve the node degree distributions of the original protograph, the edge permutations among the copies are not arbitrary. In particular, the nodes of the protograph are labeled so that, if variable node j is connected to constraint node i in the protograph, then a "type-j" variable node in a replica can connect only to one of the Q replicated "type-i" constraint nodes. In addition to preserving the degree distributions of the protograph (see Chapters 8 and 9), this also permits the design of quasi-cyclic codes. In particular, if the edge permutations correspond to weight-1 circulants, which are also called *circulant permutation matrices*, then the resulting $\mathbf{H}$ matrix will be an array of such circulants, that is, the code will be quasi-cyclic.

Figure 6.1(a) presents a protograph and Figure 6.1(b) presents the Tanner graph derived from it by replacing each protograph node by a bundle of Q nodes and each protograph edge by a bundle of Q edges. The connections between node bundles via the edge bundles are specified by the $Q \times Q$ matrices $\boldsymbol{\pi}_{ij}$ seen in the figure. Thus, each variable node j becomes a bundle of type-j variable nodes and each check node i becomes a bundle of type-i check nodes. Notice that parallel edges are permissible in a protograph, but the permutation matrices are selected so that the parallel edges no longer exist in the expanded graph.

Example 6.2. Consider the protograph in Figure 6.1(a). The adjacency matrix for this protograph, sometimes called a *base matrix*, is given by

$$\mathbf{B} = \begin{bmatrix} 2 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix},$$

where the "2" entry signifies the two parallel edges between nodes C_0 and V_0, and the "0" entry signifies the absence of an edge between nodes C_0 and V_1. With $Q = 3$, a possible $\mathbf{H}$ matrix derived from $\mathbf{B}$ is

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

where the permutation matrices $\boldsymbol{\pi}_{00}$ and $\boldsymbol{\pi}'_{00}$ were chosen so that $\mathbf{H}$ is a meaningful parity-check matrix and its corresponding Tanner graph contains no parallel edges. Because there are parallel edges between V_0 and C_0, the sum of $\boldsymbol{\pi}_{00}$ and $\boldsymbol{\pi}'_{00}$ appears as a circulant submatrix in the upper-left corner of $\mathbf{H}$. Note that $\mathbf{H}$ is an array of circulants, so the code is quasi-cyclic.

In this example we saw how replacing the nonzero elements in the base matrix of a protograph by circulants gives rise to a quasi-cyclic code. Observe that, conversely, any quasi-cyclic code has a protograph representation. This is easily seen from the previous example because, once $\mathbf{H}$ is given, the base matrix $\mathbf{B}$ is obvious, from which the protograph directly follows. For this reason, we will often speak of QC codes and protograph codes interchangeably. We will see in Chapter 12 that the class of protograph codes is a special case of the class of superposition codes.

We remark also that, when the protograph is small and the desired code length is large, one is tempted to use large circulant permutation matrices. However, code performance can suffer when Q is too large, for the code becomes too structured. Thus, what is typically done in practice is to replace each "1" element of $\mathbf{B}$ by a $Q \times Q$ matrix that is an array of $Q' \times Q'$ circulant permutation matrices, where Q' divides Q. Note that such an array is still a permutation matrix. A "2" element in $\mathbf{B}$ would be replaced by a $Q \times Q$ matrix that is a sum of two arrays of $Q' \times Q'$ circulant permutation matrices (equivalently, an array of $Q' \times Q'$ weight-2 circulants); and so on for values larger than 2. As discussed later, Figure 6.13, in conjunction with Example 6.7, gives an illustrative example.

A brute-force approach to designing a protograph code is as follows. First pick a rate R and the number of protograph VNs n_p. The number m_p of protograph CNs follows from n_p and R. Once R, n_p, and m_p have been selected, the node degree distributions $\lambda(x)$ and $\rho(x)$ may be determined. Using the techniques of Chapter 9, for example, the degree distributions can be chosen (via computer search) to optimize error-rate performance in the waterfall region. Alternatively, using the techniques of Chapter 8, the degree distributions can be chosen to optimize the performance in the floor region. Section 6.6.2.1 presents a technique for obtaining protographs for code ensembles with exceptional performance both in the waterfall region and in the floor region. Once the degree distributions have been determined, the protograph can be expanded into a full Tanner graph via the choice of permutation matrices, one for each protograph edge. The idea is that, if the code ensemble specified by the protograph is good, an arbitrary member of the ensemble can be expected to be good. The selection of permutation matrices (i.e., converting $\mathbf{B}$ to $\mathbf{H}$) can be directed by the PEG and/or ACE algorithms. More systematic protograph code-design approaches involve accumulator-based protograph codes as discussed in [24, 25] and Section 6.6.2.1.

6.3.1 Decoding Architectures for Protograph Codes

As discussed in Chapter 5, a Tanner graph for a code represents a blueprint of its iterative decoder, with the nodes representing processors and the edges representing links between the processors. In principle, an LDPC decoder architecture could mimic the code's Tanner graph, but frequently such a fully parallel architecture exceeds the hardware specifications of an application. The protograph representation leads to two partially parallel alternatives [9, 11–13].

We call the first partially parallel architecture the *circulant-based architecture*. Note that, for a QC code, the set of VNs and the set of CNs are naturally partitioned into subsets of size Q because the $\mathbf{H}$ matrix for QC codes is an $M \times N$ array of $Q \times Q$ circulants. The protograph corresponding to these partitions therefore has M check nodes and N variable nodes. In the circulant-based architecture, there are M check-node processors (CNPs) and N variable-node processors (VNPs). The decoder also requires MN random-access memory (RAM) blocks to store the intermediate messages being passed between the node processors. In the first half-iteration, the N VNPs operate in parallel, but the Q variable nodes assigned to each VNP are updated serially. In the second half-iteration, the M CNPs operate in parallel, but the Q check nodes assigned to each CNP are updated serially. Another way of stating this is that, in the first half-iteration, the circulant-based decoder simultaneously updates all of the VNs within one copy of the protograph, and then does the same for each of the other copies, one copy at a time. Then, in the second half-iteration, the decoder simultaneously updates all of the CNs within one copy of the protograph, and then does so for each of the other copies, one copy at a time.

The second partially parallel architecture is called the *protograph-based architecture*. This decoder architecture uses the fact that the code's Tanner graph was created by Q copies of the protograph. Thus, there are Q "protograph decoders" that run simultaneously, but each protograph decoder is a serial decoder with one VNP and one CNP.

The throughput T for both architectures can be related to other parameters as

$$T \text{ (bits/s)} = \frac{k \text{ (bits)} \cdot f_{\text{clock}} \text{ (cycles/s)}}{I_{\text{ave}} \text{ (iterations)} \cdot S_d \text{ (cycles/iteration)}},$$

where k is the number of information bits per codeword, f_{clock} is the clock frequency, I_{ave} is the average number of decoding iterations, and S_d is the number of clock cycles required per iteration. For typical node processor implementations, S_d is proportional to Q for the circulant-based architecture and is proportional to n_e for the protograph-based architecture, where n_e is the number of edges in the protograph. Thus, we would want Q small for the circulant-based architecture. From the discussion in the previous subsection, n_e increases as Q decreases. Thus, we want Q large in the case of the protograph-based architecture.

6.4 Multi-Edge-Type LDPC Codes

Multi-edge-type LDPC codes [14, 15] represent a generalization of protograph LDPC codes. Multi-edge-type (MET) codes were invented first, but we present them second because they are slightly more involved. Before we give a formal definition of MET LDPC codes, we present an example of a generalized protograph that encapsulates the concepts behind this class of codes.

Example 6.3. Figure 6.2 represents a generalized (expanded) protograph that is much like the graph in Figure 6.1(b). Observe that, in contrast with Figure 6.1(b), in Figure

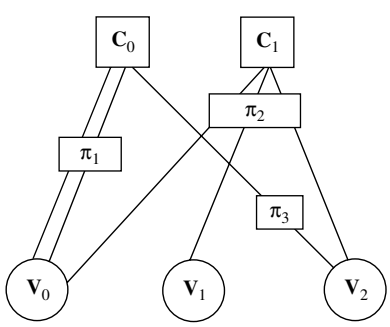


Figure 6.2 A vectorized MET generalized protograph.

6.2 multiple edges may be connected to a permutation matrix. Further, since there is a Q-fold copy-and-permute procedure associated with this graph, the matrix $\boldsymbol{\pi}_1$ is the sum of two distinct $Q \times Q$ permutation matrices since two parallel edges are connected to it. The matrix $\boldsymbol{\pi}_2$ is $Q \times 3Q$ since three skew edges (actually, edge bundles) are connected to it. Finally, $\boldsymbol{\pi}_3$ is a $Q \times Q$ permutation matrix. With $Q = 3$, an example $\mathbf{H}$ matrix is

$$\mathbf{H} = \left[\begin{array}{|ccc|ccc|ccc|}\hline 0&1&1&0&0&0&0&1&0\\ 1&0&1&0&0&0&0&0&1\\ 0&1&1&0&0&0&1&0&0\\ \hline \multicolumn{9}{|c|}{0\ 1\ 0\ 1\ 1\ 0\ 0\ 0\ 0}\\ \multicolumn{9}{|c|}{0\ 0\ 1\ 0\ 0\ 1\ 0\ 1\ 0}\\ \multicolumn{9}{|c|}{1\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 1}\\ \hline \end{array}\right].$$

In the graph of Figure 6.2, there are three *edge types*: the "type-1" edges connected through $\boldsymbol{\pi}_1$, the "type-2" edges connected through $\boldsymbol{\pi}_2$, and the "type-3" edges connected through $\boldsymbol{\pi}_3$. One may see from Figure 6.2 that the MET class of codes contains the protograph codes as a special case. In Figure 6.1(b), there are six edge types.

MET LDPC codes may be formally defined as follows [15]. First, a node of degree d, whether VN or CN, possesses d "sockets" of various types. Further, we associate with each node a *degree vector* $\mathbf{d}$, which enumerates the number of sockets of each type, and whose sum of elements is d. Thus, for example, in a Tanner graph containing three socket types, a degree-8 constraint node with degree vector $\mathbf{d} = [2, 5, 1]$ has two type-1 sockets, five type-2 sockets, and one type-3 socket. An edge of a given type then connects a VN socket to a CN socket, both of which are of the same type as the edge. As in the previous example, the routing of a bundle of edges that connects node sockets of the same type is described by a permutation matrix. Design techniques for MET LDPC codes are covered in [14, 15].

6.5 Single-Accumulator-Based LDPC Codes

The accumulator-based codes that were invented first are the so-called repeat–accumulate (RA) codes [16]. Despite their simple structure, they were shown to provide good performance and, more importantly, they paved a path toward the design of efficiently encodable LDPC codes. RA codes and other accumulator-based codes are LDPC codes that can be decoded as serial turbo codes, but are more commonly treated as LDPC codes. We discuss in this section only the most well known among the accumulator-based LDPC codes.

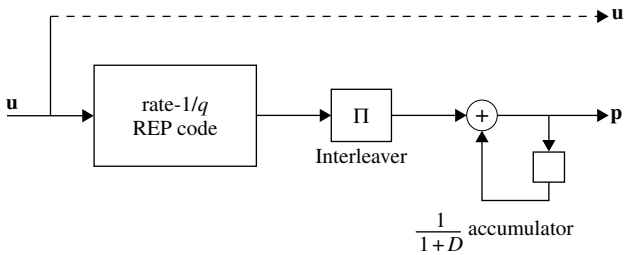


Figure 6.3 A repeat–accumulate code block diagram.

6.5.1 Repeat–Accumulate Codes

As shown in Figure 6.3, an RA code consists of a serial concatenation, through an interleaver, of a single rate-$1/q$ repetition code with an *accumulator* having transfer function $1/(1+D)$. Referring to Figures 4.2 and 4.3, the implementation of the transfer function $1/(1+D)$ is identical to that of an accumulator, although the accumulated value can be only 0 or 1 since operations are over $\mathbb{F}_2$. To ensure a large minimum Hamming distance, the interleaver should be designed so that consecutive 1s at its input are widely separated at its output. To see this, observe that two 1s separated by $s-1$ 0s at the interleaver output (and hence at the accumulator input), will yield a run of s 1s at the accumulator output. Hence, we would like s to be large in order to achieve large Hamming weight at the output of the accumulator.

RA codes can be either non-systematic or systematic. In the first case, the accumulator output, $\mathbf{p}$, is the codeword and the code rate is $1/q$. For systematic RA codes, the information word, $\mathbf{u}$, is combined with $\mathbf{p}$ to yield the codeword $\mathbf{c} = [\mathbf{u}\ \mathbf{p}]$ so that the code rate is $1/(1+q)$. The main limitations of RA codes are the code rate, which cannot be higher than 1/2, and performance at short and medium lengths. They perform surprisingly well on the AWGN channel, but they are not capacity-approaching codes. The following subsections will present a brief overview of the major enhancements to RA codes which permit operation closer to capacity and at high code rates.

6.5.2 Irregular Repeat–Accumulate Codes

The irregular repeat–accumulate (IRA) codes [17, 18] generalize the RA codes in that the repetition rate may differ for each of the k information bits and that linear combinations of the repeated bits are sent through the accumulator. Further, IRA codes are typically systematic. IRA codes provide two important advantages over RA codes. First, they allow flexibility in the choice of the repetition rate for each information bit so that high-rate codes may be designed. Second, their irregularity allows operation closer to the capacity limit.

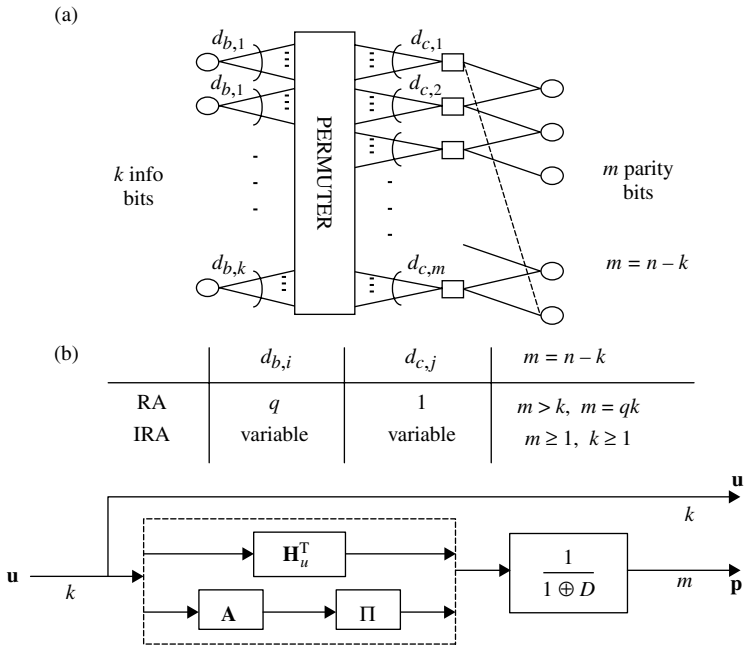


Figure 6.4 A Tanner graph (a) and encoder (b) for irregular repeat–accumulate codes.

The Tanner graph for IRA codes is presented in Figure 6.4(a) and the encoder structure is depicted in Figure 6.4(b). The variable repetition rate is accounted for in the graph by letting the variable node degrees $d_{b,j}$ vary with j. The accumulator is represented by the rightmost part of the graph, where the dashed edge is added to include the possibility of a *tail-biting trellis*, that is, a trellis whose first and last states are identical. Also, we see that $d_{c,i}$ interleaver output bits are added to produce the ith accumulator input. Figure 6.4 also includes the representation for RA codes. As indicated in the table in the figure, for an RA code, each information bit node connects to exactly q check nodes ($d_{b,j} = q$) and each check node connects to exactly one information bit node ($d_{c,i} = 1$).

To determine the code rate for an IRA code, define $\overline{q}$ to be the average repetition rate of the information bits,

$$\overline{q} = \frac{1}{k} \sum_{j=1}^{k} d_{b,j},$$

and $\bar{d}_c$ as the average of the degrees $\{d_{c,i}\}$,

$$\bar{d}_c = \frac{1}{m} \sum_{i=1}^{m} d_{c,i}.$$

Then the code rate for systematic IRA codes is

$$R = \frac{1}{1 + \overline{q}/\bar{d}_c}. \tag{6.2}$$

For non-systematic IRA codes, $R = \bar{d}_c/\overline{q}$.

The parity-check matrix for systematic RA and IRA codes has the form

$$\mathbf{H} = \left[\, \mathbf{H}_u \; \mathbf{H}_p \,\right], \tag{6.3}$$

where $\mathbf{H}_p$ is an $m \times m$ "dual-diagonal" matrix,

$$\mathbf{H}_p = \begin{bmatrix} 1 & & & & (1) \\ 1 & 1 & & & \\ & \ddots & \ddots & & \\ & & 1 & 1 & \\ & & & 1 & 1 \end{bmatrix}, \tag{6.4}$$

where the upper-right 1 is included for tail-biting accumulators. For RA codes, $\mathbf{H}_u$ is a regular matrix having column weight q and row weight 1. For IRA codes, $\mathbf{H}_u$ has column weights $\{d_{b,j}\}$ and row weights $\{d_{c,i}\}$. The encoder of Figure 6.4(b) is obtained by noting that the generator matrix corresponding to $\mathbf{H}$ in (6.3) is

$$\mathbf{G} = \left[\, \mathbf{I} \;\; \mathbf{H}_u^{\mathrm{T}} \mathbf{H}_p^{-\mathrm{T}} \,\right]$$

and writing $\mathbf{H}_u$ as $\mathbf{\Pi}^{\mathrm{T}} \mathbf{A}^{\mathrm{T}}$, where $\mathbf{\Pi}$ is a permutation matrix. Note also that

$$\mathbf{H}_p^{-\mathrm{T}} = \begin{bmatrix} 1 & 1 & \cdots & & 1 \\ & 1 & 1 & \cdots & 1 \\ & & \ddots & & \vdots \\ & & & 1 & 1 \\ & & & & 1 \end{bmatrix}$$

performs the same computation as $1/(1 \oplus D)$ (and $\mathbf{H}_p^{-\mathrm{T}}$ exists only when the "tail-biting 1" is absent). Two other encoding alternatives exist. (1) When the accumulator is not tail-biting, $\mathbf{H}$ may be used to encode since one may solve for the parity bits sequentially from the equation $\mathbf{c}\mathbf{H}^{\mathrm{T}} = \mathbf{0}$, starting with the top row of $\mathbf{H}$ and moving on downward. (2) As discussed in the next section, quasi-cyclic IRA code designs are possible, in which case the QC encoding techniques of Chapter 3 may be used.

Given the code rate, length, and degree distributions, an IRA code is defined entirely by the matrix $\mathbf{H}_u$ (equivalently, by $\mathbf{A}$ and $\mathbf{\Pi}$). From the form of $\mathbf{G}$, a weight-1 encoder input simply selects a row of $\mathbf{H}_u^{\mathrm{T}}$ and sends it through the accumulator (modeled by $\mathbf{H}_p^{-\mathrm{T}}$). Thus, to maximize the weight of the accumulator output for weight-1 inputs, the 1s in the columns of $\mathbf{H}_u$ (rows of $\mathbf{H}_u^{\mathrm{T}}$) should be widely separated. Similarly, weight-2 encoder inputs send the sum of two columns

of $\mathbf{H}_u$ through the accumulator, so the 1s in the sums of all pairs of columns of $\mathbf{H}_u$ should be widely separated. In principle, $\mathbf{H}_u$ could be designed so that even-larger-weight inputs yield sums with widely separated 1s, but the complexity of doing so eventually becomes unwieldy. Further, accounting for only weight-1 and weight-2 inputs generally results in good codes. As a final design guideline, as shown in [20] and Chapter 8, the column weight of $\mathbf{H}_u$ should be at least 4, otherwise there will be a high error floor due to a small minimum distance.

While a random-like $\mathbf{H}_u$ would generally give good performance, it leads to high-complexity decoder implementations. This is because a substantial amount of memory would be required to store the connection information implicit in $\mathbf{H}_u$. In addition, although standard message-passing decoding algorithms for LDPC codes are inherently parallel, the physical interconnections required to realize a code's bipartite graph become an implementation hurdle and prohibit a fully parallel decoder. Using a structured $\mathbf{H}_u$ matrix mitigates these problems.

Tanner [19] was the first to consider structured RA codes, more specifically, quasi-cyclic RA codes, which require tail-biting in the accumulator. Simulation results in [19] demonstrate that the QC-RA codes compete well with random-like RA codes and surpass their performance at high SNR values. We now generalize the result of [19] to IRA codes, following [20].

To attain structure in $\mathbf{H}$ for IRA codes, one cannot simply choose $\mathbf{H}_u$ to be an array of circulant permutation matrices. It is easy to show that doing so will produce a poor LDPC code in the sense of minimum distance. (Consider weight-2 encoder inputs with adjacent 1s assuming such an $\mathbf{H}_u$.) Instead, the following strategy has been used [20]. Let $\mathbf{P}$ be an $L \times J$ array of $Q \times Q$ circulant permutation matrices (for some convenient Q). Then set $\mathbf{A}^{\mathrm{T}} = \mathbf{P}$ so that $\mathbf{H}_u = \mathbf{\Pi}^{\mathrm{T}}\mathbf{P}$ and

$$\mathbf{H}_a = \begin{bmatrix} \mathbf{\Pi}^{\mathrm{T}}\mathbf{P} & \mathbf{H}_p \end{bmatrix}, \tag{6.5}$$

where $\mathbf{H}_p$ represents the tail-biting accumulator. Note that $m = L \times Q$ and $k = J \times Q$.

We now choose $\mathbf{\Pi}$ to be a standard deterministic "row–column" interleaver so that row $lQ + q$ in $\mathbf{P}$ becomes row $qL + l$ in $\mathbf{\Pi}^{\mathrm{T}}\mathbf{P}$, for all $0 \le l < L$ and $0 \le q < Q$. Next, we permute the rows of $\mathbf{H}_a$ by $\mathbf{\Pi}^{-\mathrm{T}}$ to obtain

$$\mathbf{H}_b = \mathbf{\Pi}^{-\mathrm{T}}\mathbf{H} = \begin{bmatrix} \mathbf{P} & \mathbf{\Pi}\mathbf{H}_p \end{bmatrix}, \tag{6.6}$$

where we have used the fact that $\mathbf{\Pi}^{-\mathrm{T}} = \mathbf{\Pi}$. Finally, we permute only the columns corresponding to the parity part of $\mathbf{H}_b$, which gives

$$\mathbf{H}_{\text{QC-IRA}} = \begin{bmatrix} \mathbf{P} & \mathbf{\Pi}\mathbf{H}_p\mathbf{\Pi}^{\mathrm{T}} \end{bmatrix} = \begin{bmatrix} \mathbf{P} & \mathbf{H}_{p,\text{QC}} \end{bmatrix}. \tag{6.7}$$

It is easily shown that the parity part of $\mathbf{H}_{\text{QC-IRA}}$, that is, $\mathbf{H}_{p,\text{QC}} \triangleq \mathbf{\Pi}\mathbf{H}_p\mathbf{\Pi}^{\text{T}}$, is in the quasi-cyclic form

$$\mathbf{H}_{p,\text{QC}} = \begin{bmatrix} I_0 & & & & I_1 \\ I_0 & I_0 & & & \\ & \ddots & \ddots & & \\ & & I_0 & I_0 & \\ & & & I_0 & I_0 \end{bmatrix}, \tag{6.8}$$

where I_0 is the $Q \times Q$ identity matrix and I_1 is obtained from I_0 by cyclically shifting all of its rows leftward once. Therefore, $\mathbf{H}_{\text{QC-IRA}}$ corresponds to a quasi-cyclic IRA code since $\mathbf{P}$ is also an array of $Q \times Q$ circulant permutation matrices. Observe that, except for a re-ordering of the parity bits, $\mathbf{H}_{\text{QC-IRA}}$ describes the same code as $\mathbf{H}_a$ and $\mathbf{H}_b$. If the upper-right "1" in (6.4) is absent, then the upper-right "1" in (6.8) will be absent as well. In this case we refer to the code as simply "structured." For the structured (non-QC) case, encoding may be performed directly from the $\mathbf{H}$ matrix by solving for the parity bits given the data bits.

Given (6.7), fairly good QC-IRA codes are easily designed by choosing the permutation matrices in $\mathbf{P}$ such that 4-cycles are avoided. For enhanced performance, particularly in the error floor region, additional considerations are necessary, such as incorporating the PEG and ACE algorithms into the design. These additional considerations are considered in the following subsection and [20].

6.5.2.1 *Quasi-Cyclic IRA Code Design*

Before presenting the design and performance of QC-IRA codes, we discuss the *potential* of these codes in an ensemble sense. An iterative *decoding threshold* is the theoretical performance limit for the iterative decoding of an ensemble of codes with a given degree distribution assuming infinite code length and an infinite number of decoder iterations. Chapter 9 presents several methods for numerically estimating such thresholds. Table 6.1 compares the binary-input AWGN decoding thresholds (E_b/N_0) for QC-IRA codes with those of regular QC-LDPC codes for selected code rates. The thresholds were calculated using the multidimensional EXIT algorithm presented in Chapter 9 (see also [35]). The QC-IRA codes are semi-regular, with column weight 5 for the systematic part and 2 for the parity part. The regular QC-LDPC codes have constant column weight 5. Observe in Table 6.1 that the QC-IRA codes have better thresholds for all rates, but the advantage decreases with increasing code rate. Also listed in Table 6.1 are the E_b/N_0 capacity limits for each code rate. We note that the gap to capacity for QC-IRA codes is about 0.8 dB for rate 1/2 and about 0.6 dB for rate 8/9. It is possible to achieve performance closer to capacity with non-constant systematic column weights (e.g., weight 3 and higher), but here we target finite code length, in which case a constant column weight of 5 has been shown to yield large minimum distance (see Chapter 8).

Table 6.1. Comparison of thresholds of QC-IRA codes and regular QC-LDPC codes

Code rate	Capacity (dB)	QC-IRA threshold (dB)	Regular QC-LDPC threshold (dB)
1/2	0.187	0.97	2.0
2/3	1.059	1.77	2.25
3/4	2.040	2.66	2.87
4/5	2.834	3.4	3.5
7/8	2.951	3.57	3.66
8/9	3.112	3.72	3.8

Recall that the parity-check matrix for a quasi-cyclic IRA (QC-IRA) code is given by $\mathbf{H} = \begin{bmatrix} \mathbf{P} & \mathbf{H}_{p,\mathrm{QC}} \end{bmatrix}$ per Equations (6.7) and (6.8). Let us first define the submatrix $\mathbf{P}$ to be the following $L \times S$ array of $Q \times Q$ circulant permutation submatrices,

$$\mathbf{P} = \begin{bmatrix} b_{0,0} & b_{0,1} & \dots & b_{0,S-1} \\ b_{1,0} & b_{1,1} & \dots & b_{1,S-1} \\ & & \vdots & \\ b_{L-1,0} & b_{L-1,1} & \dots & b_{L-1,S-1} \end{bmatrix}, \tag{6.9}$$

where $b_{l,s} \in \{\infty, 0, 1, \ldots, Q-1\}$ is the exponent of the circulant permutation matrix π formed by cyclically shifting (mod Q) each row of a $Q \times Q$ identity matrix $\mathbf{I}_Q$ to the right once. Thus, (6.9) is a shorthand notation in which each entry $b_{l,s}$ should be replaced by $\pi^{b_{l,s}}$. We define π^{∞} to be the $Q \times Q$ zero matrix $\mathbf{O}_Q$ which corresponds to the "masked-out" entries of $\mathbf{P}$ (see Chapter 10 for a discussion of masking). Because of the structure of $\mathbf{P}$, rows and columns occur in groups of size Q. We will refer to *row group* l, $0 \le l \le L$, and *column group* s, $0 \le s \le S$.

We now present a design algorithm for finite-length QC-IRA codes and selected performance results for each. The design algorithm is the hybrid PEG/ACE algorithm (see Section 6.2) tailored to QC-IRA codes. It is also a modification for QC-IRA codes of the PEG-like algorithm in [21] proposed for unstructured IRA codes. Recall that the ACE algorithm attempts to maximize the minimum stopping-set size in the Tanner graph by ensuring that cycles of length $2\delta_{\mathrm{ACE}}$ or less have an ACE value no less than ϵ. Clearly a length-ℓ cycle composed of only systematic nodes has a higher ACE value than does a length-ℓ cycle that includes also (degree-2) parity nodes. As in [21], we differentiate between these two cycle types and denote the girths for them by g_{sys} and g_{all}.

By targeting a quasi-cyclic IRA code, the algorithm complexity and speed are improved by a factor of Q compared with those for unstructured IRA codes. This is

because for each Q-group of variable nodes (column group) only one variable node in the group need be condition tested during matrix construction. Further, in contrast with [21], when conducting PEG conditioning, rather than enforcing a single girth value across all columns, we assign an independent target girth value g_{all} to each column group. The advantage of this modification is that, if the girth value for one Q-group of nodes cannot be attained in the design process, only this girth value need be decremented, keeping the girth targets for the other column groups unchanged. Thus, at the end of the design process g_{all} will be a vector of length S. This modification produces a better cycle distribution in the code's graph than would be the case if a single value of g_{all} were used. Finally, in addition to this PEG-like girth conditioning, the ACE algorithm is also included in the condition testing.

Algorithm 6.1 QC-IRA Code-Design Algorithm

Initialization: Initialize the parity part of $\mathbf{H}$ to the quasi-cyclic form of (6.8).
Generation of $\mathbf{P}$:
while P incomplete do

1. randomly select an unmasked position (l, j) in matrix $\mathbf{P}$, $l \in [0, L)$ and $j \in [0, J)$, for which $b_{l,j}$ is as yet undetermined;
2. randomly generate an integer $x \in [0, Q)$ different from others already generated $b_{l',j}$;
3. write the circulant π^x in position (l, j) in matrix $\mathbf{P}$;

if all conditions (g_{sys}, $g_{\text{all}}[j]$, and (d_{ACE}, η)) are satisfied, then continue;
else
  if all xs in $[0, Q)$ have been tried, then decrease $g_{\text{all}}[j]$ by 2 and go to Step 2;
  else clear current circulant π^x, set $x = (x + 1) \bmod Q$ and go to Step 3;
end (while)

Example 6.4. Using the above algorithm, a rate-0.9 (4544,4096) QC-IRA code was designed with the parameters $Q = 64$, $L = 7$, $S = 64$, $g_{\text{sys}} = 6$, initial $g_{\text{all}}[s] = 10$ (for $0 \le s < S$), and ACE parameters $(\delta_{\text{ACE}}, \eta) = (4, 6)$. In Figure 6.5, software simulation results obtained using an SPA decoder on an AWGN channel are displayed for a maximum number of iterations $I_{\max}$ equal to 1, 2, 5, 10, and 50. Observe that the performance of this code at BER $= 10^{-6}$ and $I_{\max} = 50$ is about 1.3 dB from the Shannon limit. Notice also that the gap between $I_{\max} = 10$ and $I_{\max} = 50$ is only about 0.1 dB, so decoder convergence is reasonably fast and only ten iterations (or fewer) would be necessary for most applications. Finally, we point out that the hardware decoder performance of this code was presented in Figure 5.11 of Chapter 5, where it was seen that there exists a floor below BER $= 10^{-10}$.

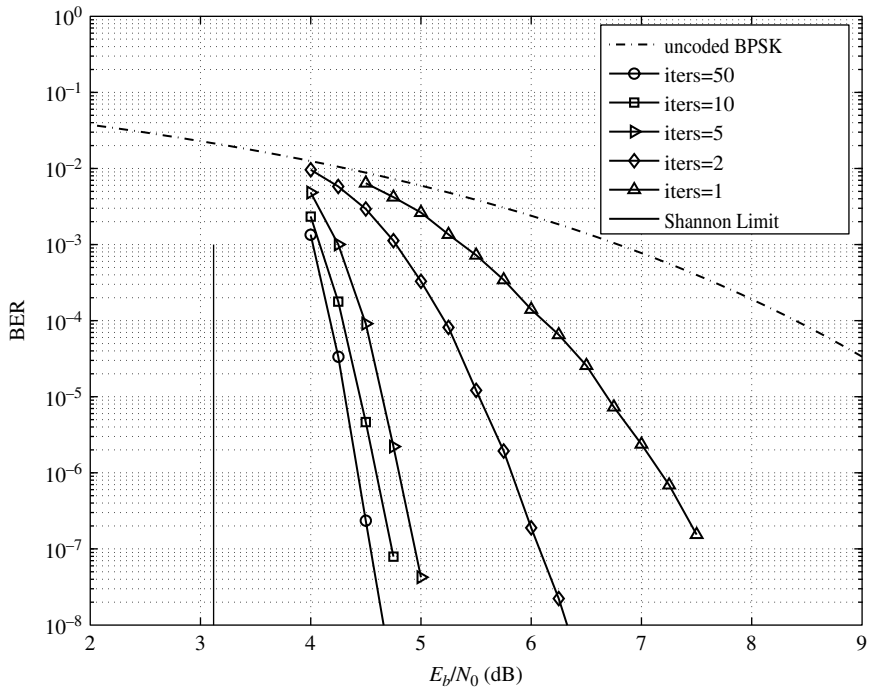


Figure 6.5 The BER and convergence performance of a QC-IRA (4544,4096) code on the AWGN channel.

We now show two approaches for designing a family of QC-IRA codes of constant information word length but varying code rate. Such families have applications in communication systems that are subject to a range of operating SNRs, but require a single encoder/decoder structure. We consider an encoder/decoder-compatible family of codes comprising code rates 1/2, 2/3, and 4/5. In [22], a design method is presented that combines masking and "extension" techniques to design an S-IRA code family with fixed systematic block length k. In this method, the lower-rate codes are constructed by adding rows to the $\mathbf{P}$ matrix of a higher-rate code while masking certain circulant permutation matrices of $\mathbf{P}$ to maintain a targeted column weight. Thus, the codes in the family share some circulants, therefore making the encoder/decoder implementation more efficient. The design of such codes can be enhanced by including girth conditioning (using the PEG algorithm). Also, in contrast with [22], the design order can go from low-rate to high-rate. Specifically, after an initial low-rate design, higher-rate codes inherit circulants from lower-rate codes. The design technique just discussed will be called the "Method I" technique and the performance of such a family for $k = 1024$ will be presented below.

The design of a rate-compatible (RC) [23] family of QC-IRA codes is simple via the use of *puncturing* (i.e., deleting code bits to achieve a higher code rate), an approach we call "Method II." In an RC code family, the parity bits of higher-rate codes are embedded in those of lower-rate codes. The designer must be careful because, when the percentage of check nodes affected by multiple punctured bits is large, decoder message exchange becomes ineffective, leading to poor performance.

As an illustrative design example, we choose as the mother code a rate-1/2 (2044,1024) S-IRA code with parameters $Q = 51$, $L = 20$, $S = 21$, $g = 5$, $g_{\text{sys}} = 6$, and initial $g_{\text{all}}[s] = 16$ (for $0 \leq s < Q$). ACE conditioning turns out to be unnecessary for this design example, so it was omitted in the design of the code presented in the example below. Because $Q \cdot S = 1071$, the rightmost 47 columns of the matrix $\mathbf{P}$ are deleted to make $k = 1024$. The highest rate we seek is 4/5, and, because the rate-4/5 code is the "most vulnerable" in the family, the mother code must be designed such that its own performance and that of its offspring rate-4/5 code are both good. The puncturing pattern is "0001" for the rate-4/5 code, which means that one parity bit out of every four is transmitted, beginning with the fourth. This puncture pattern refers to the (6.6) form of the matrix because it is easiest to discuss puncturing of parity bits before the parity bits are re-ordered as in (6.7).

It can be shown that, equivalently to the "0001" puncture pattern, groups of four check equations can be summed together and replaced by a single equation. Considering the block interleaver applied in (6.5), rows 0, 1, 2, and 3 of $\mathbf{\Pi}^{\mathrm{T}}\mathbf{P}$ are, respectively, the first rows in the first four row groups of $\mathbf{P}$; rows 4, 5, 6, and 7 of $\mathbf{\Pi}^{\mathrm{T}}\mathbf{P}$ are, respectively, the first rows in the second four row groups of $\mathbf{P}$; and so on. Thus, an equivalent parity-check matrix can be derived by summing every four row groups of the matrix given by (6.6). Because $\mathbf{P}$ has 20 row groups, there will be 5 row groups after summation, which can be considered to be the equivalent $\mathbf{P}$ matrix of the rate-4/5 code.

The puncturing pattern for the rate-2/3 code is such that one bit out of every two parity bits is transmitted and similar comments hold regarding the equivalent $\mathbf{P}$ matrix. In order to make the family rate-compatible, the unpunctured bit is selected so that the parity bits in the rate-4/5 code are embedded in those of the rate-2/3 code. A cycle test (CT) can also be applied to the equivalent parity-check matrix of the rate-4/5 code to guarantee that it is free of length-4 cycles. As shown in the example below, this additional CT rule improves the performance of the rate-4/5 code at the higher SNR values without impairing the performance of the mother code. The example also shows that the CT test is not necessary for the rate-2/3 code.

Example 6.5. Figure 6.6 shows the frame-error-rate (FER) performance of the QC-IRA code families designed using Methods I and II (with CT), with $k = 1024$ and rates of 1/2,

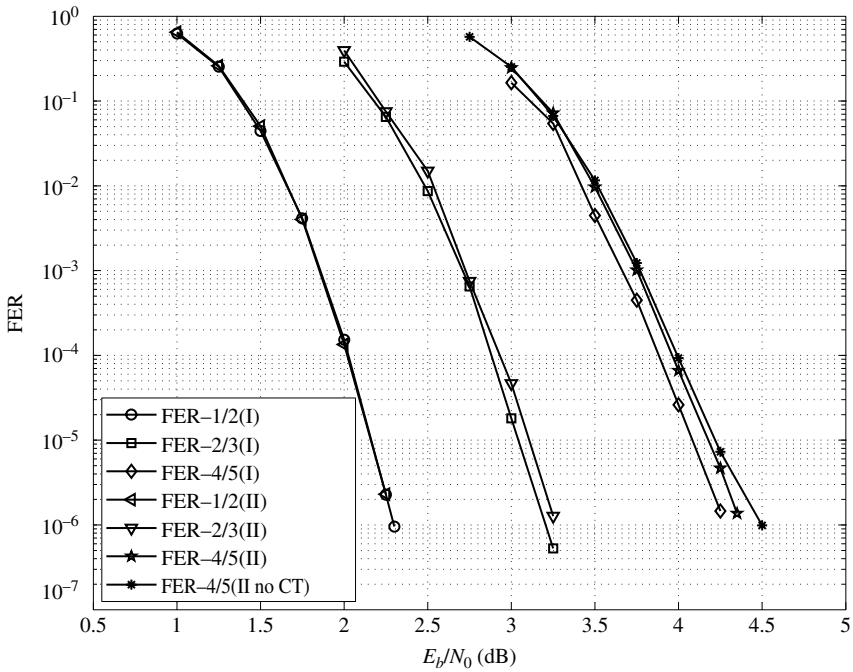


Figure 6.6 The FER performance of QC-IRA code families with $k = 1024$ designed with Methods I and II.

2/3, and 4/5. They are simulated using the approximate-min* decoder with 50 iterations. The rate-2/3 and -4/5 codes in the Method I family are slightly better than those in the Method II family in the waterfall region. However, Method II is much more flexible: (1) the decoder is essentially the same for every member of the family and (2) any rate between 1/2 and 4/5 is easy to obtain by puncturing only, with performance no worse than that of the rate-4/5 code. To verify the improvement by using the cycle test for the equivalent parity-check matrix of the rate-4/5 code, we also plot the curve for a rate-4/5 code obtained without using the CT rule. The results show that the CT rule produces a gain of 0.15 dB at FER $= 10^{-6}$. We did not include the results for the other two rates since the performances for these two rates are the same with and without the CT rule in the region simulated.

6.5.3 Generalized Accumulator LDPC Codes

IRA codes based on *generalized accumulators* (IRGA codes) [26, 27] increase the flexibility in choosing degree distributions relative to IRA codes. The encoding algorithms for IRGA codes are efficient and similar to those of non-QC IRA codes. For IRGA codes, the accumulator $1/(1 \oplus D)$ in Figure 6.4(b) is replaced by a generalized accumulator with transfer function $1/g(D)$, where $g(D) = \sum_{l=0}^{t} g_l D^l$, $t > 1$, and $g_l \in \{0, 1\}$, except $g_0 = 1$. The systematic encoder therefore has the same generator-matrix format, $\mathbf{G} = \begin{bmatrix}\mathbf{I} & \mathbf{H}_u^{\mathrm{T}}\mathbf{H}_p^{-\mathrm{T}}\end{bmatrix}$, but now

$$\mathbf{H}_p = \begin{bmatrix} 1 & & & & & & & \\ g_1 & 1 & & & & & & \\ g_2 & g_1 & \ddots & & & & & \\ \vdots & g_2 & \ddots & \ddots & & & & \\ g_t & \vdots & \ddots & \ddots & \ddots & & & \\ & g_t & \ddots & \ddots & \ddots & \ddots & & \\ & & \ddots & \ddots & \ddots & \ddots & \ddots & \\ & & & g_t & \cdots & g_2 & g_1 & 1 \end{bmatrix}.$$

Further, the parity-check-matrix format is unchanged, $\mathbf{H} = [\mathbf{H}_u \quad \mathbf{H}_p]$.

To design an IRGA code, one must choose $g(D)$ so that the bipartite graph for $\mathbf{H}_p$ contains no length-4 cycles. Once $g(D)$ has been chosen, $\mathbf{H}$ can be completed by constructing the submatrix $\mathbf{H}_u$, according to some prescribed degree distribution, again avoiding short cycles, this time in all of $\mathbf{H}$.

6.6 Double-Accumulator-Based LDPC Codes

In the preceding section, we saw the advantages of LDPC codes that involve single accumulators: low-complexity encoding, simple code designs, and excellent performance. There are, in fact, advantages to adding a second accumulator. If a second (interleaved) accumulator is used to encode the parity word at the output of an IRA encoder, the result is an irregular repeat–accumulate–accumulate (IRAA) code. The impact of the second accumulator is a lower error-rate floor. If the second interleaver is instead used to "precode" selected data bits at the input to an IRA encoder, the result is an accumulate–repeat–accumulate (ARA) code. The impact of the additional accumulator in this case is to improve the waterfall-region performance relative to that for IRA codes. That is, the waterfall portion of the error-rate curve for an ARA code resides in worse channel conditions than does that for the corresponding IRA code.

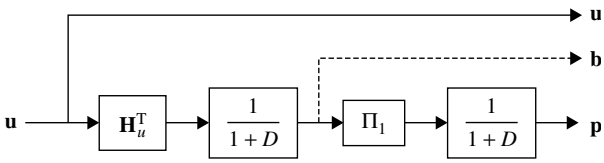


Figure 6.7 An IRAA encoder.

6.6.1 Irregular Repeat–Accumulate–Accumulate Codes

We now consider IRAA codes that are obtained by concatenating the parity arm of the IRA encoder of Figure 6.4(b) with another accumulator, through an interleaver, as shown in Figure 6.7. The IRAA codeword can be either $\mathbf{c} = [\mathbf{u} \quad \mathbf{p}]$ or $\mathbf{c} = [\mathbf{u} \quad \mathbf{b} \quad \mathbf{p}]$, depending on whether the intermediate parity bits $\mathbf{b}$ are punctured or not. The parity-check matrix of the general IRAA code corresponding to Figure 6.7 is

$$\mathbf{H}_{\mathrm{IRAA}} = \begin{bmatrix} \mathbf{H}_u & \mathbf{H}_p & 0 \\ 0 & \mathbf{\Pi}_1^{\mathrm{T}} & \mathbf{H}_p \end{bmatrix}, \tag{6.10}$$

where $\mathbf{\Pi}_1$ is the interleaver between the two accumulators. $\mathbf{H}_u$ for an IRAA code can be designed as for an IRA code. An IRAA code will typically have a lower floor but worse waterfall region than an IRA code of the same length, rate, and complexity, as shown in [12] and what follows.

Example 6.6. Example rate-1/2 protographs for IRA and IRAA codes are presented in Figure 6.8. For the IRA protograph, $d_{b,j} = 5$ for all j, and $d_{c,i} = 5$ for all i. For the IRAA protograph, $d_{b,j} = d_{c,i} = 3$ and the intermediate parity vector $\mathbf{b}$ is not transmitted in order to maintain the code rate at 1/2. Because the decoder complexity is proportional to the number of edges in a code's parity-check matrix, the complexity of the IRAA decoder is about 14% greater than that of the IRA decoder. To see this, note that the IRAA protograph has eight edges whereas the IRA protograph has seven edges.

Analogously to (6.7), the parity-check matrix for a quasi-cyclic IRAA code is given by

$$\mathbf{H}_{\text{QC-IRAA}} = \begin{bmatrix} \mathbf{P} & \mathbf{H}_{p,\mathrm{QC}} & \mathbf{0} \\ \mathbf{0} & \mathbf{\Pi}_{\mathrm{QC}}^{\mathrm{T}} & \mathbf{H}_{p,\mathrm{QC}} \end{bmatrix}, \tag{6.11}$$

where $\mathbf{P}$ and $\mathbf{H}_{p,\mathrm{QC}}$ are as described in (6.9) and (6.8), and $\mathbf{\Pi}_{\mathrm{QC}}$ is a permutation matrix consisting of $Q \times Q$ circulant permutation matrices and $Q \times Q$ zero matrices arranged to ensure that the row and column weights of $\mathbf{\Pi}_{\mathrm{QC}}$ are

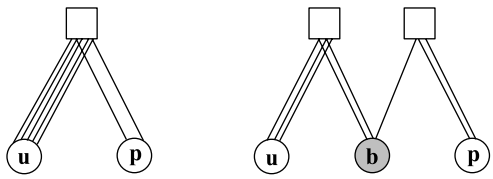


Figure 6.8 Rate-1/2 IRA and IRAA protographs. The shaded node in the IRAA protograph represents punctured bits.

both 1. The design algorithm for QC-IRAA codes is analogous to that for QC-IRA codes. Specifically, (1) the $\mathbf{H}_{p,\mathrm{QC}}$ and $\mathbf{0}$ matrices in (6.11) are fixed components of $\mathbf{H}_{\text{QC-IRAA}}$ and (2) the $\mathbf{P}$ and $\mathbf{\Pi}_{\mathrm{QC}}^{\mathrm{T}}$ submatrices of $\mathbf{H}_{\text{QC-IRAA}}$ may be constructed using a PEG/ACE-like algorithm much like Algorithm 6.1 for QC-IRA codes.

Let us now compare the performance of rate-1/2 (2048,1024) QC-IRA and QC-IRAA codes. For the QC-IRA code $d_{b,j} = 5$ for all j, whereas for the QC-IRAA code $d_{b,j} = 3$ for all j. For the IRAA code, $\mathbf{c} = \begin{bmatrix}\mathbf{u} & \mathbf{p}\end{bmatrix}$, that is, the intermediate parity bits $\mathbf{b}$ are punctured. The QC-IRA code was designed using the algorithm of the previous section. The QC-IRAA code was designed using the algorithm given in the previous paragraph. We observe in Figure 6.9 that, for both codes, there are no error floors in the BER curves down to BER $= 5 \times 10^{-8}$ and in the FER curves down to FER$= 10^{-6}$. While the S-IRAA code is 0.2 dB inferior to the S-IRA code in the waterfall region, it is conjectured that it has a lower floor (which is difficult to measure), which would be due to the second accumulator, whose function is to improve the weight spectrum.

As an example of a situation in which the IRAA class of code is superior, consider the comparison of rate-1/3 (3072,1024) QC-IRA and QC-IRAA codes, with $d_{b,j} = 4$ for the QC-IRA code and $d_{b,j} = 3$ for the QC-IRAA code. In this case, $\mathbf{c} = \begin{bmatrix}\mathbf{u} & \mathbf{b} & \mathbf{p}\end{bmatrix}$, that is, the intermediate parity bits $\mathbf{b}$ are not punctured. Also, the decoder complexities are the same. We see in Figure 6.10 that, in the low-SNR region, the performance of the IRA code is 0.4 dB better than that of the IRAA code. However, as is evident from Figure 6.10, the IRAA code will outperform the IRA code in the high-SNR region due to its lower error floor.

6.6.2 Accumulate–Repeat–Accumulate Codes

For ARA codes, which were introduced in [28, 29], an accumulator is added to precode a subset of the information bits of an IRA code. The primary role of this second accumulator is to improve the decoding threshold of a code (see Chapter 9), that is, to shift the error-rate curve leftward toward the capacity limit. Precoding is generally useful only for relatively low code rates because satisfactory decoding thresholds are easier to achieve for high-rate LDPC codes. Figure 6.11 presents

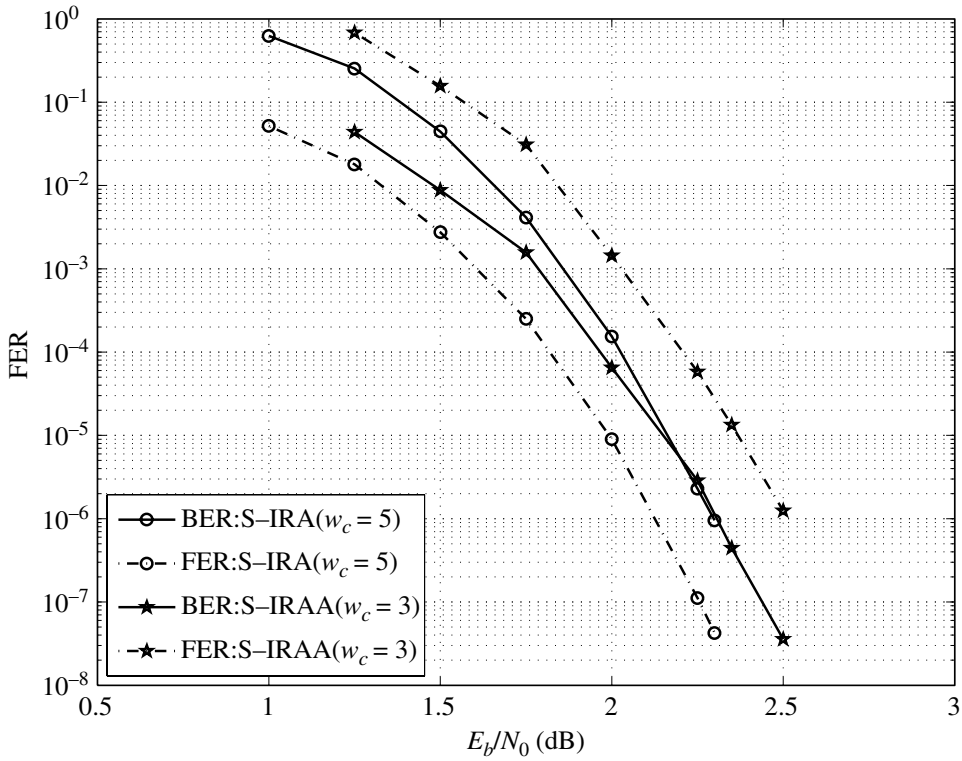


Figure 6.9 Performance of an IRAA code and an IRA code with $n = 2048$ and $k = 1024$ on the AWGN channel ($I_{\max} = 50$). w_c is the column weight of the submatrix $\mathbf{P}$.

a generic ARA Tanner graph in which punctured variable nodes are blackened. The enhanced performance provided by the precoding accumulator is achieved at the expense of these punctured variable nodes which act as auxiliary nodes that enlarge the $\mathbf{H}$ used by the decoder. The iterative graph-based ARA decoder thus has to deal with a redundant representation of the code, implying a larger $\mathbf{H}$ matrix than the nominal $(n-k) \times n$. Note that this is much like the case for IRAA codes.

ARA codes typically rely on very simple protographs. The protograph of a rate-1/2 ARA code ensemble with repetition rate 4, denoted AR4A, is depicted in Figure 6.12(a). This encoding procedure corresponds to a systematic code. The black circle corresponds to a punctured node, and it is associated with the precoded fraction of the information bits. As emphasized in Figure 6.12(a), such a protograph is the serial concatenation of an accumulator protograph and an IRA protograph (with a tail-biting accumulator). Half of the information bits (node 2) are sent directly to the IRA encoder, while the other half (node 5) are first precoded by the outer accumulator. Observe in the IRA sub-protograph that the

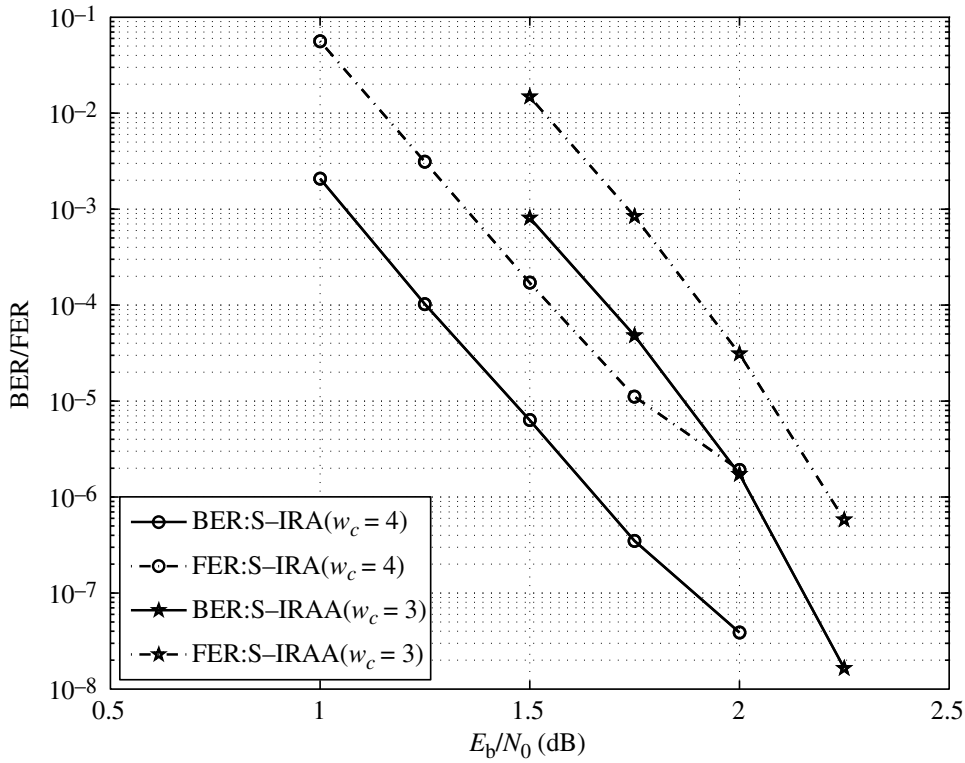


Figure 6.10 Performance of an IRAA code and an IRA code with $n = 3072$ and $k = 1024$ on the AWGN channel ($I_{\max} = 50$). w_c is the column weight of the submatrix $\mathbf{P}$.

VNs and CNs of a minimal protograph (e.g., Figure 6.8) have been doubled to allow for the precoding of half of the input bits.

A non-systematic code structure is represented by the protograph in Figure 6.12(b), which has a parallel-concatenated form. In this case, half of the information bits (node 2) are encoded by the IRA encoder and the other half (node 1) are encoded both by the IRA encoder and by a (3,2) single-parity-check encoder. The node-1 information bits (corresponding to the black circle in the protograph) are punctured, so codes corresponding to this protograph are non-systematic. While the code ensembles specified by the protographs in Figure 6.12(a) are the same in the sense that the same set of codewords is implied, the $\mathbf{u} \to \mathbf{c}$ mappings are different. The advantage of the non-systematic protograph is that, although the node-1 information bits in Figure 6.12(b) are punctured, the node degree is 6, in contrast with the node-5 information bits in Figure 6.12(a), in which the node degree is only 1. In the iterative decoder, the bit log-likelihood values associated with the degree-6 node tend to converge faster and to a larger value than do those associated with the degree-1 node. Hence, these bits will be more reliably decoded.

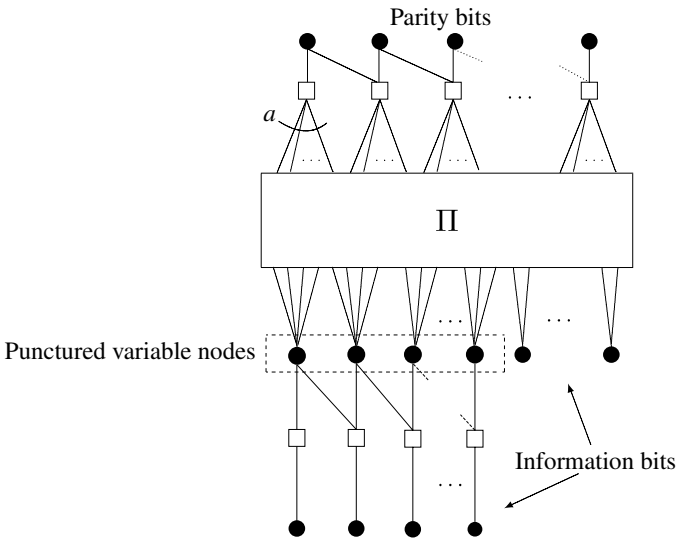


Figure 6.11 A generic bipartite graph for ARA codes.

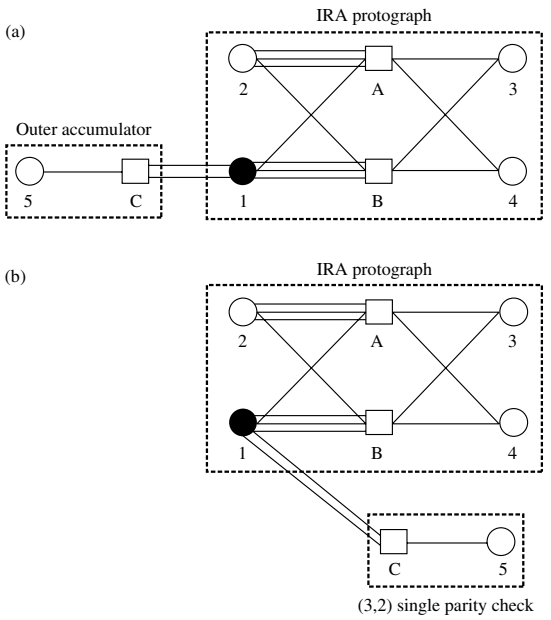


Figure 6.12 AR4A protographs in (a) serial-concatenated form and (b) parallel-concatenated form. The black circle is a punctured variable node.

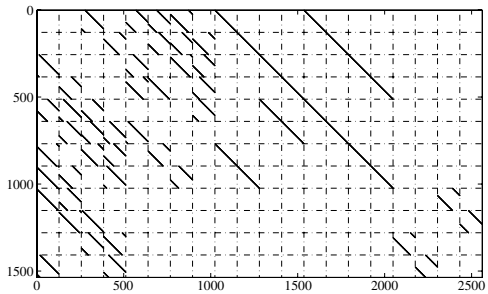


Figure 6.13 The **H** matrix for the (2048,1024) AR4A code.

The design of ARA codes is quite involved and requires the material presented in Chapters 8 and 9. An overview of the design of excellent ARA codes is presented in Section 6.6.2.1.

Example 6.7. A pixelated image of the **H** matrix for a (2048,1024) QC AR4A code is depicted in Figure 6.13. The first group of 512 columns (of weight 6) corresponds to variable-node type 1 of degree 6 (Figure 6.12), whose bits are punctured, and the subsequent four groups of 512 columns correspond, respectively, to node types 2, 3, 4, and 5. The first group of 512 rows (of weight 6) corresponds to check-node type A (of degree 6), and the two subsequent groups of rows correspond to node types B and C, respectively.

6.6.2.1 *Protograph-Based ARA Code Design*

Section 6.3 describes a brute-force technique for designing protograph codes. Here we present two approaches for designing protograph-based ARA codes, following [24, 25]. The goal is to obtain a protograph for an ensemble that has a satisfactory decoding threshold and a minimum distance $d_{\min}$ that grows linearly with the codeword length n. This linear-distance-growth idea was investigated by Gallager, who showed that (1) regular LDPC code ensembles had linear $d_{\min}$ growth, provided that the VN degrees were greater than 2; and (2) LDPC code ensembles with a randomly constructed **H** matrix had linear $d_{\min}$ growth. As an example, for rate-1/2 randomly constructed LDPC codes, $d_{\min}$ grows as $0.11n$ for large n. As another example, for the rate-1/2 (3,6)-regular LDPC code ensemble, $d_{\min}$ grows as $0.023n$.

The first approach to designing protograph-based ARA codes is the result of a bit of LDPC code-design experience and a bit of trial and error. As in the example presented in Figure 6.12, we start with a rate-1/2 IRA protograph and precode 50% of its encoder input bits to improve the decoding threshold. Then, noticing that linear $d_{\min}$ growth is thwarted by a large number of degree-2 VNs, one horizontal branch is added to the accumulator part of the IRA protograph to convert half of the degree-2 VNs to degree-3 VNs. This would result in the protograph

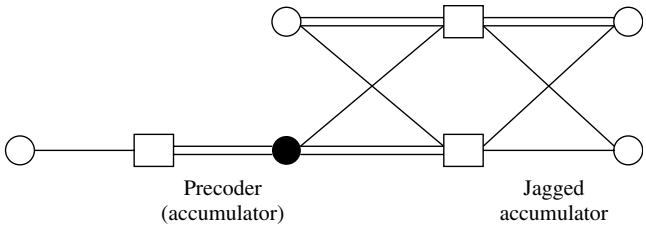


Figure 6.14 An ARJA protograph for rate-1/2 LDPC codes. The black VN is punctured.

in Figure 6.12(a), but with two edges connecting node A to node 1. One could also experiment with the number of edges between nodes 2 and A and nodes 1 and B in that figure. It was found in [24, 25] that performance is improved if the number of edges connecting node 2 to node A is reduced from three to two. The final protograph, called an ARJA protograph for its "jagged" accumulator [24, 25], is shown in Figure 6.14. Given this protograph, one can construct an $\mathbf{H}$ matrix from the base matrix corresponding to the protograph, using a PEG/ACE-like algorithm.

The second approach to arriving at this particular ARJA protograph begins with a rate-2/3 protograph known to have linear $d_{\min}$ growth, that is, a protograph whose VN degrees are 3 or greater. This protograph is shown at the top of Figure 6.15 and the sequence of protographs that we will derive from it lies below that protograph. To lower the code rate to 1/2, an additional CN must be added. To do this, the CN in the top protograph is split into two CNs and the edges connected to the original CN are distributed between the two new CNs. Further, a degree-2 (punctured) VN is connected to the two CNs. Observe that this second protograph is equivalent to the first one: the set of 3-bit words satisfying the constraints of the first protograph is identical to that of the second. If we allow the recently added VN to correspond to transmitted bits instead of punctured bits, we arrive at the first rate-1/2 protograph in Figure 6.15. Further, because the underlying rate-2/3 ensemble has linear $d_{\min}$ growth, the derived rate-1/2 ensemble must have this property as well. Finally, the bottom protograph in Figure 6.15 is obtained with the addition of a precoder (accumulator).

This rate-1/2 protograph has an AWGN channel decoding threshold of $(E_b/N_0)_{\text{thresh}} = 0.64$ dB. Also its asymptotic $d_{\min}$ growth rate goes as $0.015n$. It is possible to further split CNs to obtain protographs with rates lower than 1/2. Alternatively, higher-rate protographs are obtained via the addition of VNs as in Figure 6.16 [24, 25]. Table 6.2 presents the ensemble decoding thresholds for this code family on the binary-input AWGN channel and compares these values with their corresponding capacity limits. Moderate-length codes designed from these protographs were presented in [24, 25], where is it shown that the codes have excellent decoding thresholds and very low error floors, both of which many other

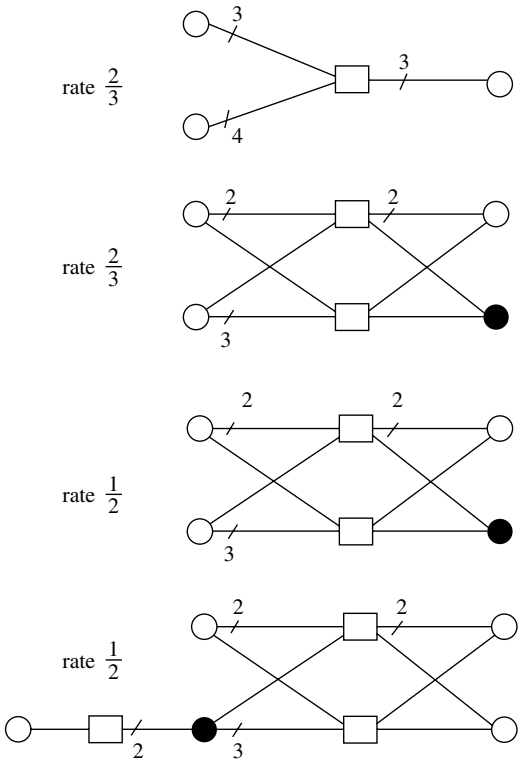


Figure 6.15 Constructing a rate-1/2 ARJA LDPC code from a rate-2/3 protograph to preserve the linear $d_{\min}$ growth property and to achieve a good decoding threshold. Black circles are punctured VNs.

code-design techniques fail to achieve. The drawback, however, is that these codes suffer from slow decoder convergence due to punctured VNs and to degree-1 VNs (precoded bits).

6.7 Accumulator-Based Codes in Standards

Accumulator-based LDPC exist in, or are being considered for, several communication standards. The ETSI DVB S2 standard for digital video broadcast specifies two IRA code families with code block lengths 64 800 and 16 200. The code rates supported by this standard range from 1/4 to 9/10, and a wide range of spectral efficiencies can be achieved by coupling these LDPC codes with QPSK, 8-PSK, 16-APSK, and 32-APSK modulation formats. A further level of protection is afforded by an outer BCH code. For details, see [30, 31].

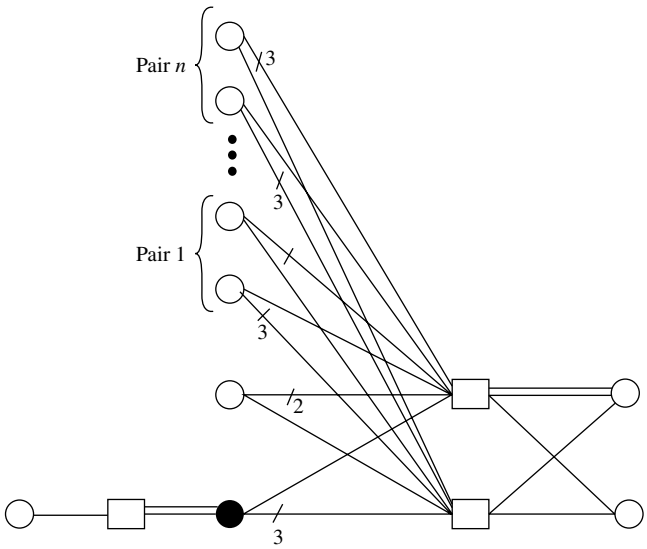


Figure 6.16 An ARJA protograph for a family of LDPC codes with rates $(n+1)/(n+2)$, $n = 0, 1, 2, \ldots$.

Table 6.2. Thresholds of ARJA codes

Rate	Threshold	Capacity limit	Gap
1/2	0.628	0.187	0.441
2/3	1.450	1.059	0.391
3/4	2.005	1.626	0.379
4/5	2.413	2.040	0.373
5/6	2.733	2.362	0.371
6/7	2.993	2.625	0.368
7/8	3.209	2.845	0.364

ARJA codes are being considered by the Consultative Committee for Space Data Systems (CCSDS) for deep-space applications. Details of these codes may be found in [32]. Codes with rates 1/2, 2/3, and 4/5 and data-block lengths of $k =$ 1024, 4096, and 16384 are presented in that document. Also being considered by the CCSDS for standardization for the near-Earth application is the (8176,7156) Euclidean-geometry code presented in Section 10.5. This code, and its offspring (8160,7136) code, are also discussed in [32]. Although this is not an accumulator-based code, it is shown in [33] how accumulators can be added to this code to obtain a rate-compatible family of LDPC codes.

The IEEE standards bodies have also adopted IRA-influenced QC LDPC codes for 802.11n (wireless local-area networks) and 802.16e (wireless metropolitan-area networks). Rather than employing a tail-biting accumulator, or one corresponding to a weight-1 column in $\mathbf{H}$, these standards have replaced the last block-column in (6.8) by a weight-3 block-column and moved it to the first column. An example of such a format is

$$\begin{bmatrix} I_1 & I_0 & & & & & \\ & I_0 & I_0 & & & & \\ & & I_0 & \ddots & & & \\ I_0 & & & \ddots & \ddots & & \\ & & & & \ddots & I_0 & \\ & & & & & I_0 & I_0 \\ I_1 & & & & & & I_0 \end{bmatrix},$$

where I_1 is the matrix that results after cyclically shifting rightward all rows of the identity matrix I_0. Encoding is facilitated by this matrix since the sum of all block-rows gives the block-row $\begin{pmatrix} I_0 & 0 & \cdots & 0 \end{pmatrix}$, so that encoding is initialized by summing all of the block-rows of $\mathbf{H}$ and solving for the first Q parity bits using the resulting block-row.

6.8 Generalized LDPC Codes

Generalized LDPC codes were introduced in Chapter 5. For these codes, the constraint nodes (CNs) are more general than single-parity-check (SPC) constraints. There are m_c code constraints placed on the n code bits connected to the CNs. Let $V = \{V_j\}_{j=0}^{n-1}$ be the set of n VNs and $C = \{C_i\}_{i=0}^{m_c-1}$ be the set of m_c CNs in the bipartite graph of a G-LDPC code. Recall that the connections between the nodes in V and C can be summarized in an $m_c \times n$ adjacency matrix $\mathbf{\Gamma}$. We sometimes call $\mathbf{\Gamma}$ the code's graph because of this correspondence. Chapter 5 explains the relationship between the adjacency matrix $\mathbf{\Gamma}$ and the parity-check matrix $\mathbf{H}$ for a generic G-LDPC code. Here, following [34], we consider the derivation of $\mathbf{H}$ for quasi-cyclic G-LDPC codes and then present a simple QC G-LDPC code design.

To design a quasi-cyclic G-LDPC code, that is, to obtain a matrix $\mathbf{H}$ that is an array of circulant permutation matrices, we exploit the protograph viewpoint. Consider a G-LDPC protograph $\mathbf{\Gamma}_p$ with m_p generalized constraint nodes and n_p variable nodes. The adjacency matrix for the G-LDPC code graph, $\mathbf{\Gamma}$, is constructed by substituting $Q \times Q$ circulant permutation matrices for the 1s in $\mathbf{\Gamma}_p$ in such a manner that short cycles are avoided and parallel edges are eliminated. The 0s in $\mathbf{\Gamma}_p$ are replaced by $Q \times Q$ zero matrices. As demonstrated in Figure 6.1, the substitution of circulant permutation matrices for 1s in $\mathbf{\Gamma}_p$ effectively

applies the copy-and-permute procedure to the protograph corresponding to $\mathbf{\Gamma}_p$. The resulting adjacency matrix for the G-LDPC code will be an $m_p \times n_p$ array of circulant permutation matrices of the form

$$\mathbf{\Gamma} = \begin{bmatrix} \pi_{0,0} & \pi_{0,1} & \cdots & \pi_{0,n_p-1} \\ \pi_{1,0} & \pi_{1,1} & \cdots & \pi_{1,n_p-1} \\ & & \ddots & \\ \pi_{m_p-1,0} & \pi_{m_p-1,1} & \cdots & \pi_{m_p-1,n_p-1} \end{bmatrix},$$

where each $\pi_{\mu,\nu}$ is either a $Q \times Q$ circulant permutation matrix or a $Q \times Q$ zero matrix. Note that $\mathbf{\Gamma}$ is an $m_pQ \times n_pQ = m_c \times n$ binary matrix. Now $\mathbf{H}$ can be obtained from $\mathbf{\Gamma}$ and $\{\mathbf{H}_i\}$ as in the previous paragraph. To see that this procedure leads to a quasi-cyclic code, we consider some examples.

Example 6.8. Consider a protograph with the $m_p \times n_p = 2 \times 3$ adjacency matrix

$$\mathbf{\Gamma}_p = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix}.$$

Let $Q = 2$ and expand $\mathbf{\Gamma}_p$ by substituting 2×2 circulant permutation matrices for each 1 in $\mathbf{\Gamma}_p$. The result is

$$\mathbf{\Gamma} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 0 \end{bmatrix},$$

so that $m_c = m_pQ = 4$ and $n = n_pQ = 6$. Suppose now that the CN constraints are given by

$$\mathbf{H}_1 = \begin{bmatrix} 1 & 1 \end{bmatrix}$$

and

$$\mathbf{H}_2 = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}.$$

Then, upon replacing the 1s in $\mathbf{\Gamma}$ by the corresponding columns of $\mathbf{H}_1$ and $\mathbf{H}_2$, the $\mathbf{H}$ matrix for the G-LDPC code is found to be

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

where the inserted columns of $\mathbf{H}_1$ and $\mathbf{H}_2$ are highlighted in bold. Note that because the μth row of $\mathbf{\Gamma}_p$, $\mu = 0, 1, \ldots, m_p - 1$, corresponds to constraint C_μ (matrix $\mathbf{H}_\mu$) the

μth block-row (the μth group of Q binary rows) within $\mathbf{\Gamma}$ corresponds to constraint C_μ (matrix $\mathbf{H}_\mu$). In the form given above, it is not obvious that $\mathbf{H}$ corresponds to a quasi-cyclic code. However, if we permute the last four rows of $\mathbf{H}$, we obtain the array of circulants

$$\mathbf{H}' = \begin{bmatrix} 1\,0 & 0\,0 & 1\,0 \\ 0\,1 & 0\,0 & 0\,1 \\ \\ 1\,0 & 0\,1 & 0\,0 \\ 0\,1 & 1\,0 & 0\,0 \\ \\ 1\,0 & 0\,0 & 0\,1 \\ 0\,1 & 0\,0 & 1\,0 \end{bmatrix}.$$

The permutation used on the last four rows of $\mathbf{H}$ can be considered to be a mod-m_2 de-interleave of the rows, where $m_2 = 2$ is the number of rows in $\mathbf{H}_2$. No de-interleaving was necessary for the first two rows of $\mathbf{H}$ because $m_1 = 1$.

The fact that in general a code's $\mathbf{H}$ matrix will be an array of permutation matrices (after appropriate row permutation) whenever $\mathbf{\Gamma}$ is an array of permutation matrices should be obvious from the construction of $\mathbf{H}$ from $\mathbf{\Gamma}$ in the above example. We now consider an example in which the protograph contains parallel edges.

Example 6.9. Change the upper-left element of $\mathbf{\Gamma}_p$ of the previous example to "2" so that

$$\mathbf{\Gamma}_p = \begin{bmatrix} 2 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix}.$$

Thus, there are two edges connecting variable node V_0 and constraint node C_0. A possible q-fold expansion of $\mathbf{\Gamma}_p$ is

$$\mathbf{\Gamma} = \begin{bmatrix} \pi_0\ \pi_1 & 0\ \ 0 & 0\ \ \pi_2 \\ \pi_3\ \pi_4 & 0\ \ 0 & \pi_5\ \ 0 \\ \\ \pi_6\ \ 0 & 0\ \ \pi_8 & \pi_{10}\ \ 0 \\ 0\ \ \pi_7 & \pi_9\ \ 0 & 0\ \ \pi_{11} \end{bmatrix},$$

where the permutation matrices are $Q/2 \times Q/2$ and are selected so as to avoid short cycles. $\mathbf{H}$ is then obtained by replacing the 1s in the rows of $\mathbf{\Gamma}$ by the columns of $\mathbf{H}_1$ and $\mathbf{H}_2$; the first Q rows of $\mathbf{\Gamma}$ correspond to $\mathbf{H}_1$ and the second Q rows correspond to $\mathbf{H}_2$.

Note that a G-LDPC code has a parity-check matrix that is 4-cycle-free if its adjacency matrix is 4-cycle-free and the component codes possess parity-check matrices that are 4-cycle-free.

6.8.1 A Rate-1/2 G-LDPC Code

The design of G-LDPC codes is still a developing area and, hence, requires a bit of art. Some successful design approaches may be found in [34]. We now present one of these designs, which leads to an excellent quasi-cyclic rate-1/2 G-LDPC code.

The approach starts with a very simple protograph: 2 CNs, 15 VNs, with both CNs connected to each of the 15 VNs. Thus, the CNs have degree 15 and the VNs have degree 2. Both CNs correspond to the (15,11) Hamming code constraint, but with different code-bit orders. Specifically, protograph CN C_0 is described by the parity-check matrix

$$\mathbf{H}_0 = [\mathbf{M}_1 \quad \mathbf{M}_2] = \begin{bmatrix} 1\,0\,1\,0\,1\,0\,1\,0 & 1\,0\,1\,0\,1\,0\,1 \\ 0\,1\,1\,0\,0\,1\,1\,0 & 0\,1\,1\,0\,0\,1\,1 \\ 0\,0\,0\,1\,1\,1\,1\,0 & 0\,0\,0\,1\,1\,1\,1 \\ 0\,0\,0\,0\,0\,0\,0\,1 & 1\,1\,1\,1\,1\,1\,1 \end{bmatrix} \tag{6.12}$$

and protograph CN C_1 is described by

$$\mathbf{H}_1 = [\mathbf{M}_2 \quad \mathbf{M}_1]. \tag{6.13}$$

Next, the protograph is replicated $Q = 146$ times, yielding a derived graph with 2190 VNs and 292 Hamming CNs, 146 of which are described by (6.12) and 146 of which are described by (6.13). Because the number of parity bits is $m = 292 \cdot (15 - 11) = 1168$, the resulting code has as parameters (2190,1022). The connections between the VNs and the CNs are given by the adjacency matrix $\mathbf{\Gamma}$ at the top of Figure 6.17, which was chosen simply to avoid 4-cycles and 6-cycles in the Tanner graph corresponding to that matrix. Therefore, the girth of the Tanner graph corresponding to $\mathbf{\Gamma}$ is 8. The $\mathbf{H}$ matrix (with appropriately re-ordered rows) is given at the bottom of Figure 6.17. Observe that an alternative approach for obtaining the re-ordered matrix $\mathbf{H}$ is to replace each 1 in the rows of the matrix in (6.12) (the matrix in (6.13)) by the corresponding permutation matrices of the first (second) block row of the adjacency matrix in Figure 6.17 and then stack the first resulting matrix on top of the second.

We may obtain a quasi-cyclic rate-1/2 (2044,1022) G-LDPC code by puncturing the first 146 bits of each (2190,1022) codeword. Observe that this corresponds to puncturing a single VN in the code's protograph and the first column of circulants of $\mathbf{\Gamma}$. The frame-error-rate (FER) performance of this rate-1/2 code on the binary-input AWGN channel is depicted in Figure 6.18. For the simulations, the maximum number of iterations was set to $I_{\max} = 50$. The G-LDPC code does not display a floor down to FER $\simeq 5 \times 10^{-8}$. As shown in Figure 6.18, the code's

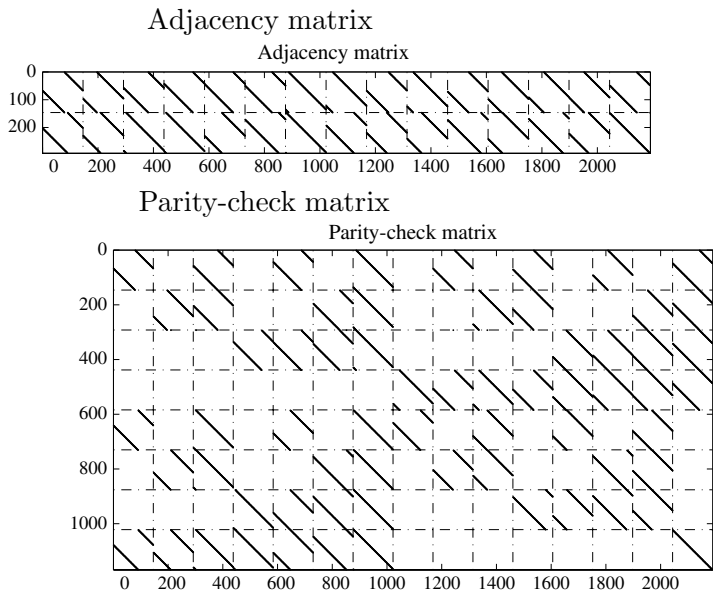


Figure 6.17 The adjacency matrix of the (2190,1022) G-LDPC code and its block-circulant parity-check matrix $\mathbf{H}$.

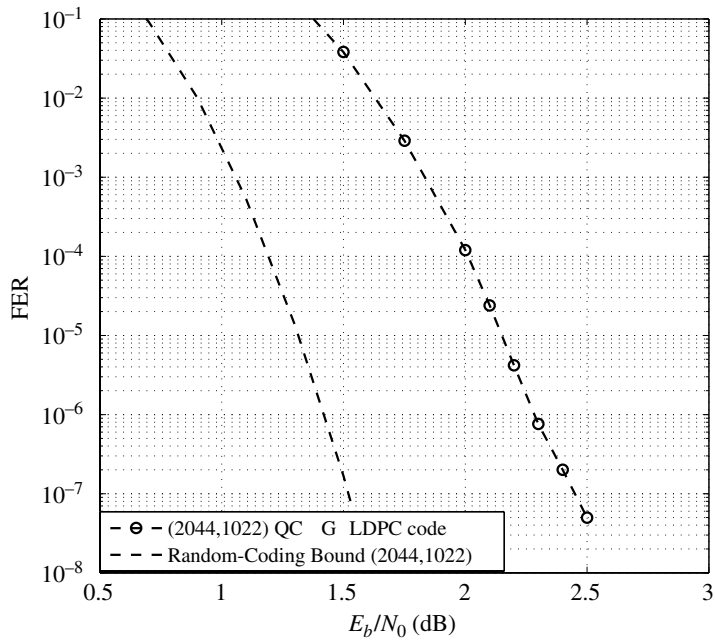


Figure 6.18 The frame-error rate for the (2044,1022) quasi-cyclic G-LDPC code, compared with the random-coding bound. $I_{\max}$ was set to 50.

performance is within 1 dB of Gallager's random-coding bound for (2044,1022) block codes.

Problems

6.1 (Wesel *et al.*) Show that, in a Tanner graph for which the VN degree is at least two, every stopping set contains multiple cycles, except in the special case for which all VNs in the stopping set are degree-2 VNs. For this special case there is a single cycle.

6.2 (Wesel *et al.*) Show that, for code with minimum distance $d_{\min}$, each set of $d_{\min}$ columns of $\mathbf{H}$ that sum to the zero vector corresponds to VNs that form a stopping set.

6.3 Why is the extrinsic message degree of a stopping set equal to zero?

6.4 (a) Consider the parity-check matrix below for the (7,4) Hamming code:

$$\mathbf{H}_1 = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}.$$

Treat matrix $\mathbf{H}_1$ as a base matrix, $\mathbf{H}_{\text{base}} = \mathbf{H}_1$, and find its corresponding protograph. (b) Find (possibly by computer) a parity-check matrix $\mathbf{H}$ for a length-21 code obtained by selecting 3×3 circulant permutation matrices $\boldsymbol{\pi}_{ij}$ between VN j and CN i in the protograph, for all j and i. Equivalently, replace the 1s in $\mathbf{H}_{\text{base}}$ by the matrices $\boldsymbol{\pi}_{ij}$. The permutation matrices should be selected to avoid 4-cycles to the extent possible.

6.5 A quasi-cyclic code has the parity-check matrix

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

Find its corresponding protograph.

6.6 Sketch the multi-edge-type generalized protograph that corresponds to the parity-check matrix

$$\mathbf{H} = \left[\begin{array}{|ccc|cc|cc|}\hline 0&0&0&0&1&1&0&1\\ 0&0&0&1&0&1&0&0\\ 0&0&0&1&1&0&1&0\\ \hline \hline 1&0&1&0&0&0&0&1\\ 0&1&0&0&0&1&1&0\\ 0&0&0&1&1&0&1&1\\ \hline \end{array}\right]$$

and indicate which permutation matrix corresponds to each edge (or pair of edges).

6.7 You are given the following parity-check matrix for an IRA code:

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

Use Equation (6.2) to determine the code rate and compare your answer with the code rate that is determined from the dimensions of $\mathbf{H}$. Find the codeword corresponding to the data word $[1 \quad 0 \quad 0 \quad 1 \quad 1]$.

6.8 The $m \times n$ parity-check matrix for systematic RA and IRA codes has the form $\mathbf{H} = [\mathbf{H}_u \quad \mathbf{H}_p]$, where $\mathbf{H}_p$ is $m \times m$ and is given by (6.4). Let each codeword have the form $\mathbf{c} = [\mathbf{u} \quad \mathbf{p}]$, where $\mathbf{u}$ is the data word and $\mathbf{p}$ is the length-m parity word. Show that, depending on the data word $\mathbf{u}$, there exist either two solutions or no solutions for $\mathbf{p}$ when the "1" in the upper-right position in $\mathbf{H}_p$ in (6.4) is present. This statement is true as well for the quasi-cyclic form in (6.8). (The impediment is that $\text{rank}(\mathbf{H}_p) = m - 1$. Given this, it would be wise to choose one of the parity-bit positions to correspond to one of the columns of $\mathbf{H}_u$, assuming that $\mathbf{H}$ is full rank. (With acknowledgment of Y. Han.)

6.9 Find the generator matrix $\mathbf{G}$ corresponding to the parity-check matrix $\mathbf{H}$ given in (6.10) assuming that no puncturing occurs. Figure 6.7 might be helpful. Show that $\mathbf{G}\mathbf{H}^{\mathrm{T}} = \mathbf{0}$. Repeat for the quasi-cyclic form of the IRAA parity-check matrix given in (6.11).

6.10 Design a rate-1/2 (2048,1024) LDPC code based on the ARJA protograph of Figure 6.14 by constructing an $\mathbf{H}$ matrix. It is not necessary to include PEG/ACE conditioning, but your design must be free of 4-cycles. Consider values of Q from the set $\{16, 32, 64\}$. Present your $\mathbf{H}$ matrix both as a matrix of exponents of π (the identity cyclically shifted rightward once) and as displayed in Figure 6.13. Simulate your designed code down to an error rate of 10^{-8} on the AWGN channel using a maximum of 100 decoder iterations.

6.11 Repeat the previous problem for a rate-2/3 code of dimension $k = 1024$ using the ARJA protograph of Figure 6.16.

6.12 Prove the following. In a given Tanner graph (equivalently, $\mathbf{H}$ matrix) for an (n,k) LDPC code, the maximum number of degree-2 variable nodes possible before a cycle is created involving only these degree-2 nodes is $n - k - 1 = m - 1$. Furthermore, for codes free of such "degree-2 cycles" and possessing this maximum, the submatrix of H composed of only its weight-2 columns is simply a permutation of the following $m \times (m-1)$ parent matrix:

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

6.13 Suppose we would like to design a rate-1/3 protograph-based RA code. Find the protograph that describes this collection of codes. Note that there will be one input VN that will be punctured, three output VNs that will be transmitted, and three CNs. Find a formula for the code rate as a function of $N_{v,p}$, the number of punctured VNs, $N_{v,t}$, the number of transmitted VNs, and N_c, the number of CNs. (With acknowledgment of D. Divsalar *et al.*)

6.14 (a) Draw the protograph for the ensemble of rate-1/2 (6,3) LDPC codes, that is, LDPC codes with degree-3 VNs and degree-6 CNs. (b) Suppose we would like to precode half of the codeword bits of a rate-1/2 (6,3) LDPC code using an accumulator. Draw a protograph for this ensemble of rate-1/2 codes that maximizes symmetry. Since half of the code bits are precoded, you will need to double the number of VNs and CNs in your protograph of part (a) prior to adding the precoder. Your protograph should have three degree-3 VNs, one degree-5 VN (that will be punctured), and one degree-1 VN (this one is precoded). (With acknowledgment of D. Divsalar *et al.*)

6.15 Instead of an accumulate–repeat–accumulate code, consider an accumulate–repeat–generalized-accumulate code with generalized accumulator transfer function $1/(1 + D + D^2)$. Find the rate-1/2 protograph for such a code. Set the column weight for the systematic part of the parity-check matrix equal to 5 (i.e., $d_{b,j} = 5$ for all j). Construct an example parity-check matrix for a (200,100) realization of such a code and display a pixelated version of it as in Figure 6.13.

6.16 Consider a generalized protograph-based LDPC with protograph adjacency matrix

$$\mathbf{\Gamma}_p = \begin{bmatrix} 1 & 0 & 1 & 1 \\ 1 & 2 & 0 & 1 \end{bmatrix}.$$

(a) Sketch the protograph. (b) With $Q = 3$, find an adjacency matrix $\mathbf{\Gamma}$ for the G-LDPC code. Use circulant permutation matrices. (c) Now suppose the first constraint in the protograph is the (3,2) SPC code and the second constraint is the (4,1) code with parity-check matrix

$$\begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix}.$$

Find the parity-check matrix $\mathbf{H}$ for the G-LDPC code and from this give its quasi-cyclic form $\mathbf{H}'$.

6.17 Consider a product code as a G-LDPC code with two constraints, the row constraint (row code) and the column constraint (column code). Let the row code have length N_r and the column code have length N_c. (a) Show that the adjacency matrix is of the form

$$\mathbf{\Gamma} = \begin{bmatrix} V & & & \\ & V & & \\ & & \ddots & \\ & & & V \\ I & I & \cdots & I \end{bmatrix},$$

where V is the length-N_r all-ones vector and I is the $N_r \times N_r$ identity matrix. V and I each appear N_c times in the above matrix. (b) Sketch the Tanner graph for this G-LDPC code. (c) Determine the parity-check matrix $\mathbf{H}$ for such a product code if both the row and the column code are the (7,4) Hamming code with parity-check matrix given by $\mathbf{H}_1$ in Problem 6.4.

References

[1] R. G. Gallager, *Low-Density Parity-Check Codes*, Cambridge, MA, MIT. Press, 1963. (Also, R. G. Gallager, "Low density parity-check codes," *IRE Trans. Information Theory*, vol. 8, no. 1, pp. 21–28, January 1962.)

[2] D. MacKay, "Good error correcting codes based on very sparse matrices," *IEEE Trans. Information Theory*, vol. 45, no. 3, pp. 399–431, March 1999.

[3] http://www.inference.phy.cam.ac.uk/mackay/CodesFiles.html

[4] T. J. Richardson and R. Urbanke, "Efficient encoding of low-density parity-check codes," *IEEE Trans. Information Theory*, vol. 47, no. 2, pp. 638–656, February 2001.

[5] B. Vasic and O. Milenkovic, "Combinatorial constructions of low-density parity-check codes for iterative decoding," *IEEE Trans. Information Theory*, vol. 50, no. 6, pp. 1156–1176, June 2004.

[6] X.-Y. Hu, E. Eleftheriou, and D.-M. Arnold, "Progressive edge-growth Tanner graphs," *2001 IEEE Global Telecommunications Conf.*, pp. 995–1001, November 2001.

[7] T. Tian, C. Jones, J. Villasenor, and R. Wesel, "Construction of irregular LDPC codes with low error floors," *2003 IEEE Int. Conf. on Communications*, pp. 3125–3129, May 2003.

[8] H. Xiao and A. Banihashemi, "Improved progressive-edge-growth (PEG) construction of irregular LDPC codes," *2004 IEEE Global Telecommunications Conf.*, pp. 489–492, November/December 2004.

[9] T. Richardson and V. Novichkov, "Methods and apparatus for decoding LDPC codes," U.S. Patent 6,633,856, October 14, 2003.

[10] J. Thorpe, "Low density parity check (LDPC) codes constructed from protographs," JPL INP Progress Report 42-154, August 15, 2003.

[11] T. Richardson and V. Novichkov, "Method and apparatus for decoding LDPC codes," U.S. Patent 7,133,853, November 7, 2006.

[12] Y. Zhang, *Design of Low-Floor Quasi-Cyclic IRA Codes and Their FPGA Decoders,* Ph.D. Dissertation, ECE Dept., University of Arizona, May 2007.

[13] C. Jones, S. Dolinar, K. Andrews, D. Divsalar, Y. Zhang, and W. Ryan, "Functions and Architectures for LDPC Decoding," *2007 IEEE Information Theory Workshop*, pp. 577–583, September 2–6, 2007.

[14] T. Richardson, "Multi-edge type LDPC codes," presented at the Workshop honoring Professor R. McEliece on his 60th birthday, California Institute of Technology, Pasadena, CA, May 24–25, 2002.

[15] T. Richardson and R. Urbanke, "Multi-edge type LDPC codes," submitted to *IEEE Trans. Information Theory.*

[16] D. Divsalar, H. Jin, and R. McEliece, "Coding theorems for turbo-like codes," *Proc. 36th Annual Allerton Conf. on Communication, Control, and Computing*, pp. 201–210, September 1998.

[17] H. Jin, A. Khandekar, and R. McEliece, "Irregular repeat–accumulate codes," *Proc. 2nd. Int. Symp. on Turbo Codes and Related Topics*, Brest, France, pp. 1–8, September 4, 2000.

[18] M. Yang, W. E. Ryan, and Y. Li, "Design of efficiently encodable moderate-length high-rate irregular LDPC codes," *IEEE Trans. Communications*, vol. 52, no. 4, pp. 564–571, April 2004.

[19] R. Michael Tanner, "On quasi-cyclic repeat–accumulate codes," in *Proc. 37th Allerton Conf. on Communication, Control, and Computing*, September 1999.

[20] Y. Zhang and W. E. Ryan, "Structured IRA codes: Performance analysis and construction," *IEEE Trans. Communications*, vol. 55, no. 5, pp. 837–844, May 2007.

[21] L. Dinoi, F. Sottile, and S. Benedetto, "Design of variable-rate irregular LDPC codes with low error floor," *2005 IEEE Int. Conf. on Communications*, pp. 647–651, May 2005.

[22] Y. Zhang, W. E. Ryan, and Y. Li, "Structured eIRA codes," *Proc. 38th Asilomar Conf. on Signals, Systems, and Computing*, Pacific Grove, CA, November 2004, pp. 7–10.

[23] J. Hagenauer, "Rate-compatible punctured convolutional codes and their applications," *IEEE Trans. Communications*, vol. 36, no. 4, pp. 389–400, April 1988.

[24] D. Divsalar, C. Jones, S. Dolinar, and J. Thorpe, "Protograph based LDPC codes with minimum distance linearly growing with block size," *2005 IEEE Global Telecomunications Conf.*

[25] D. Divsalar, S. Dolinar, and C. Jones, "Construction of protograph LDPC codes with linear minimum distance," *2006 Int. Symp. on Information Theory.*

[26] G. Liva, E. Paolini, and M. Chiani, "Simple reconfigurable low-density parity-check codes," *IEEE Commununications Lett.*, vol. 9, no. 3, pp. 258–260, March, 2005

[27] S. J. Johnson and S. R. Weller, "Constructions for irregular repeat–accumulate codes," in *Proc. IEEE Int. Symp. on Information Theory*, Adelaide, September 2005.

[28] A. Abbasfar, D. Divsalar, and K. Yao, "Accumulate repeat accumulate codes," in *Proc. 2004 IEEE GlobeCom Conf.*, Dallas, Texas, November 2004.

[29] A. Abbasfar, D. Divsalar, and K. Yao, "Accumulate–repeat–accumulate codes," *IEEE Trans. Communications*, vol. 55, no. 4, pp. 692–702, April 2007.

[30] ETSI Standard TR 102 376 V1.1.1: Digital Video Broadcasting (DVB) User Guidelines for the Second Generation System for Broadcasting, Interactive Services, News Gathering and Other Broadband Satellite Applications (DVB-S2). http://webapp.etsi.org/workprogram/Report_WorkItem.asp?WKI_ID=21402

[31] M. C. Valenti, S. Cheng, and R. Iyer Seshadri, "Turbo and LDPC codes for digital video broadcasting," Chapter 12 of *Turbo Code Applications: A Journey from a Paper to Realization*, Berlin, Springer-Verlag, 2005.

[32] *Low Density Parity Check Codes for Use in Near-Earth and Deep Space*. Orange Book. Issue 2. September 2007. http://public.ccsds.org/publications/OrangeBooks.aspx

[33] S. Dolinar, "A rate-compatible family of protograph-based LDPC codes built by expurgation and lengthening," *Proc. 2005 Int. Symp. on Information Theory*, pp. 1627–1631, September 2005.

[34] G. Liva, W. E. Ryan, and M. Chiani, "Quasi-cyclic generalized LDPC codes with low error floors," IEEE Trans. Communications, pp. 49–57, January 2008.

[35] G. Liva and M. Chiani, "Protograph LDPC codes design based on EXIT analysis," *Proc. 2007 IEEE GlobeCom Conf.*, pp. 3250–3254, November 2007.

7 Turbo Codes

Turbo codes, which were first presented to the coding community in 1993 [1, 2], represent one of the most important breakthroughs in coding since Ungerboeck introduced trellis codes in 1982 [3]. In fact, the invention of turbo codes and their iterative "turbo" decoders started a revolution in iteratively decodable codes and iterative receiver processing (such as "turbo equalization"). Most frequently, a turbo code refers to a concatenation of two (or more) convolutional encoders separated by interleavers. The turbo decoder consists of two (or more) soft-in/soft-out convolutional decoders which iteratively feed probabilistic information back and forth to each other in a manner that is reminiscent of a turbo engine. In this chapter we introduce the most important classes of turbo codes, provide some heuristic justification as to why they should perform well, and present their iterative decoders. Our focus will be on parallel- and serial-concatenated convolutional codes (PCCCs and SCCCs) on the binary-input AWGN channel, but we also include the important class of turbo product codes. These codes involve block codes arranged as rows and columns in a rectangular array of bits. The decoder is similar to that for PCCCs and SCCCs, except that the constituent decoders are typically suboptimal soft-in/soft-out list decoders. The reader is advised to consult the leading papers in the field for additional information on the codes considered in this chapter: [4–19]. Our focus is on the binary-input AWGN channel. Nonbinary turbo codes for the noncoherent reception of orthogonal signals are considered in [20].

7.1 Parallel-Concatenated Convolutional Codes

Figure 7.1 depicts the encoder for a standard parallel-concatenated convolutional code (PCCC), the first class of turbo code invented. As seen in the figure, a PCCC encoder consists of two binary rate-1/2 recursive systematic convolutional (RSC) encoders separated by a K-bit interleaver or permuter, together with an optional puncturing mechanism. Clearly, without the puncturer, the encoder is rate 1/3, mapping K data bits to $3K$ code bits. We observe that the encoders are configured in a manner reminiscent of classical concatenated codes. However, instead of cascading the encoders in the usual *serial* fashion, the encoders are arranged in a so-called *parallel concatenation*. As will be argued below, recursive convolutional encoders are necessary in order to attain the exceptional performance for which

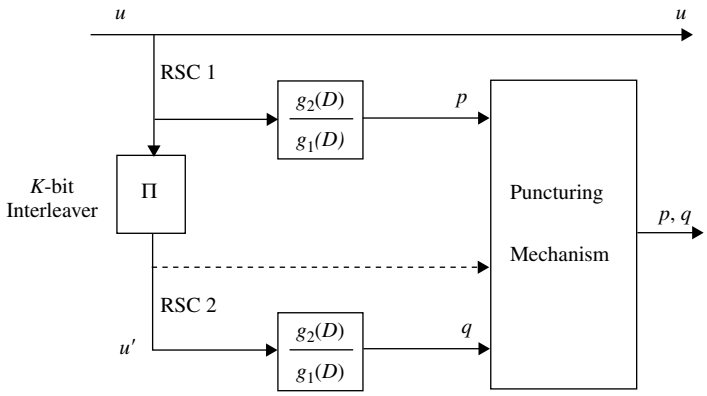


Figure 7.1 The encoder for a standard parallel-concatenated convolutional code.

turbo codes are known. Before describing further details of the PCCC encoder in its entirety, we shall first discuss its individual components.

7.1.1 Critical Properties of RSC Codes

Without any essential loss of generality, we assume that the constituent RSC codes are identical, with generator matrix

$$\mathbf{G}(D) = \begin{bmatrix} 1 & \dfrac{g^{(2)}(D)}{g^{(1)}(D)} \end{bmatrix}.$$

Observe that the code sequence $u(D)\mathbf{G}(D)$ will be of finite weight if and only if the input sequence is divisible by $g^{(1)}(D)$. We have from this fact the following immediate results which we shall use later.

Proposition 7.1. *A weight-1 input into an RSC encoder will produce an infinite-weight output, for such an input is never divisible by a (non-trivial) polynomial $g^{(1)}(D)$. (In practice, "infinite" should be replaced by "large" since the input length would be finite.)*

Proposition 7.2. *For any non-trivial $g^{(1)}(D)$, there exists a family of weight-2 inputs into an RSC encoder of the form $D^j(1 + D^P)$, $j \geq 0$, which produce finite-weight outputs, i.e., which are divisible by $g^{(1)}(D)$. When $g^{(1)}(D)$ is a primitive polynomial of degree m, then $P = 2^m - 1$. More generally, P is the length of the pseudo-random sequence generated by $g^{(1)}(D)$.*

Proof. Because the encoder is linear, its output due to a weight-2 input $D^j(1 + D^t)$ is equal to the sum of its outputs due to D^j and D^jD^t. The output due to D^j will

be periodic with period P since the encoder is a finite-state machine: the state at time j must be reached again in a finite number of steps P, after which the state sequence is repeated indefinitely with period P. Now, letting $t = P$, the output due to $D^j D^P$ is just the output due to D^j shifted by P bits. Thus, the output due to $D^j(1 + D^P)$ is the sum of the outputs due to D^j and $D^j D^P$, which must be of finite length and weight since all but one period will cancel out in the sum. □

In the context of the code's trellis, Proposition 7.1 says that a weight-1 input will create a path that diverges from the all-zeros path, but never remerges. Proposition 7.2 says that there will always exist a trellis path that diverges and remerges later, which corresponds to a weight-2 data sequence.

Example 7.1. Consider the RSC code with generator matrix

$$\mathbf{G}(D) = \begin{bmatrix} 1 & \dfrac{1 + D + D^3 + D^4}{1 + D^3 + D^4} \end{bmatrix}.$$

Thus, $g^{(1)}(D) = 1 + D^3 + D^4$ and $g^{(2)}(D) = 1 + D + D^3 + D^4$ or, in octal form, the generators are $(31, 33)$. Observe that $g^{(1)}(D)$ is primitive of order 15, so that, for example, $u(D) = 1 + D^{15}$ produces the finite-length code sequence $(1 + D^{15}, 1 + D + D^4 + D^5 + D^7 + D^9 + D^{10} + D^{11} + D^{12} + D^{15})$. Of course, any delayed version of this input, say $D^7(1 + D^{15})$, will simply produce a delayed version of this code sequence.

7.1.2 Critical Properties of the Interleaver

The function of the interleaver is to take each incoming block of bits and rearrange them in a pseudo-random fashion prior to encoding by the second RSC encoder. It is crucial that this interleaver permute the bits in a manner that ostensibly lacks any apparent order, although it should be tailored in a certain way for weight-2 and weight-3 inputs, as will be made clearer below. The S-random interleaver [17] is quite effective in this regard. This particular interleaver ensures that any two input bits whose positions are within S of each other are separated by an amount greater than S at the interleaver output. S should be selected to be as large as possible for a given value of K. Observe that the implication of this design is that, if the input $u(D)$ is of the form $D^j(1 + D^P)$, where $P = 2^m - 1$ (see Proposition 7.2), then the interleaver output $u'(D)$ will be of the form $D^{j'}(1 + D^{P'})$, where $P' > 2^m - 1$ and is somewhat large. This means that the parity weight at the second encoder output is not likely to also be low. Lastly, as we shall see, performance increases with K, so $K \geq 1000$ is typical.

7.1.3 The Puncturer

The role of the turbo-code puncturer is identical to that of any other code, that is, to delete selected bits to reduce coding overhead. It is most convenient to delete only parity bits, otherwise the decoder (described in Section 7.2) will have to be substantially re-designed. For example, to achieve a rate of 1/2, one might delete all even-parity bits from the top encoder and all odd-parity bits from the bottom one. However, there is no guarantee that deleting only parity bits (and not data bits) will yield the maximum possible minimum codeword distance for the code rate and length of interest. Approaches to puncturing may be found in the literature, such as [18].

7.1.4 Performance Estimate on the BI-AWGNC

A maximum-likelihood (ML) sequence decoder would be far too complex for a PCCC due to the presence of the interleaver. However, the suboptimum iterative decoding algorithm to be described in Section 7.2 offers near-ML performance. Hence, we shall now estimate the performance of an ML decoder for a PCCC code on the BI-AWGNC. A more careful analysis of turbo codes and iteratively decodable codes in general may be found in Chapter 8.

Armed with the above descriptions of the components of the PCCC encoder of Figure 7.1, it is easy to conclude that it is linear since its components are linear. (We ignore the nuisance issue of terminating the trellises of the two constituent encoders.) The constituent codes are certainly linear, and the permuter is linear since it may be modeled by a permutation matrix. Further, the puncturer does not affect linearity since all codewords share the same puncture locations. As usual, the importance of linearity is that, in considering the performance of a code, one may choose the all-zeros sequence as a reference. Thus, hereafter we shall assume that the all-zeros codeword was transmitted.

Now consider the all-zeros codeword (the 0th codeword) and the kth codeword, for some $k \in \{1, 2, \ldots, 2^K - 1\}$. The ML decoder will choose the kth codeword over the 0th codeword with probability $Q\left(\sqrt{2d_k R E_b/N_0}\right)$, where R is the code rate and d_k is the weight of the kth codeword. The bit error rate for this two-codeword situation would then be

$$\begin{aligned} P_b(k|0) &= w_k \text{ (bit errors/cw error)} \\ &\quad \times \frac{1}{K} \text{ (cw/data bits)} \\ &\quad \times Q\left(\sqrt{2Rd_k E_b/N_0}\right) \text{ (cw errors/cw)} \\ &= \frac{w_k}{K} Q\left(\sqrt{\frac{2Rd_k E_b}{N_0}}\right) \text{ (bit errors/data bit)}, \end{aligned}$$

where w_k is the weight of the kth data word. Now, including all of the codewords and invoking the usual union bounding argument, we may write

$$\begin{aligned} P_b &= P_b(\text{choose any } k \in \{1, 2, \ldots, 2^K - 1\}|0) \\ &\leq \sum_{k=1}^{2^K-1} P_b(k|0) \\ &= \sum_{k=1}^{2^K-1} \frac{w_k}{K} Q\left(\sqrt{\frac{2Rd_k E_b}{N_0}}\right). \end{aligned}$$

Note that every nonzero codeword is included in the above summation. Let us now reorganize the summation as

$$P_b \leq \sum_{w=1}^{K} \sum_{v=1}^{\binom{K}{w}} \frac{w}{K} Q\left(\sqrt{\frac{2Rd_{wv}E_b}{N_0}}\right), \tag{7.1}$$

where the first sum is over the weight-w inputs, the second sum is over the $\binom{K}{w}$ different weight-w inputs, and d_{wv} is the weight of the codeword produced by the vth weight-w input.

Consideration of the first few terms in the outer summation of (7.1) leads to a certain characteristic of the code's weight spectrum, which is called *spectral thinning* [9], explained in the following.

$\underline{w = 1}$: From Proposition 7.1 and associated discussion above, weight-1 inputs will produce only large-weight codewords at both constituent encoder outputs since the trellis paths created never remerge with the all-zeros path. Thus, each d_{1v} is significantly greater than the minimum codeword weight so that the $w = 1$ terms in (7.1) will be negligible.

$\underline{w = 2}$: Of the $\binom{K}{2}$ weight-2 encoder inputs, only a fraction will be divisible by $g_1(D)$ (i.e., yield remergent paths) and, of these, only certain ones will yield the smallest weight, $d_{2,\min}^{\text{CC}}$, at a constituent encoder output (here, CC denotes "constituent code"). Further, with the permuter present, if an input $u(D)$ of weight-2 yields a weight-$d_{2,\min}^{\text{CC}}$ codeword at the first encoder's output, it is unlikely that the permuted input, $u'(D)$, seen by the second encoder will also correspond to a weight-$d_{2,\min}^{\text{CC}}$ codeword (much less be divisible by $g_1(D)$). We can be sure, however, that there will be some minimum-weight turbo codeword produced by a $w = 2$ input, and that this minimum weight can be bounded as

$$d_{2,\min}^{\text{PCCC}} \geq 2d_{2,\min}^{\text{CC}} - 2,$$

with equality when both of the constituent encoders produce weight-$d_{2,\min}^{\text{CC}}$ codewords (minus 2 for the bottom encoder). The exact value of $d_{2,\min}^{\text{PCCC}}$ is permuter-dependent. We will denote the number of weight-2 inputs that produce weight-$d_{2,\min}^{\text{PCCC}}$ turbo codewords by n_2 so that, for $w = 2$, the inner sum in (7.1)

may be approximated as

$$\sum_{v=1}^{\binom{K}{2}} \frac{2}{K} Q\left(\sqrt{\frac{2Rd_{2v}E_b}{N_0}}\right) \simeq \frac{2n_2}{K} Q\left(\sqrt{\frac{2Rd_{2,\min}^{\mathrm{PCCC}}E_b}{N_0}}\right). \tag{7.2}$$

From the foregoing discussion, we can expect n_2 to be small relative to K. This fact is due to the spectral-thinning effect brought about by the combination of the recursive encoder and the interleaver.

$\underline{w = 3}$: Following an argument similar to the $w = 2$ case, we can approximate the inner sum in (7.1) for $w = 3$ as

$$\sum_{v=1}^{\binom{K}{3}} \frac{3}{K} Q\left(\sqrt{\frac{2Rd_{3v}E_b}{N_0}}\right) \simeq \frac{3n_3}{K} Q\left(\sqrt{\frac{2Rd_{3,\min}^{\mathrm{PCCC}}E_b}{N_0}}\right), \tag{7.3}$$

where n_3 and $d_{3,\min}^{\mathrm{PCCC}}$ are obviously defined. While n_3 is clearly dependent on the interleaver, we can make some comments on its size relative to n_2 for a randomly generated interleaver. Although there are $(K-2)/3$ times as many $w = 3$ terms in the inner summation of (7.1) as there are $w = 2$ terms, we can expect the number of weight-3 terms divisible by $g_1(D)$ to be on the order of the number of weight-2 terms divisible by $g_1(D)$. Thus, most of the $\binom{K}{3}$ terms in (7.1) can be removed from consideration for this reason. Moreover, given a weight-3 encoder input $u(D)$ divisible by $g_1(D)$ (e.g., $g_1(D)$ itself in the above example), it becomes very unlikely that the permuted input $u'(D)$ seen by the second encoder will also be divisible by $g_1(D)$. For example, suppose $u(D) = g_1(D) = 1 + D + D^4$. Then the permuter output will be a multiple of $g_1(D)$ if the three input 1s become the jth, $(j+1)$th, and $(j+4)$th bits out of the permuter, for some j. If we imagine that the permuter acts in a purely random fashion so that the probability that one of the 1s lands in a given position is $1/K$, the permuter output will be $D^j g_1(D) = D^j\left(1 + D + D^4\right)$ with probability $3!/K^3$. This is not the only weight-3 pattern divisible by $g_1(D)$ ($g_1^2(D) = 1 + D^2 + D^8$ is another one, but this too has probability $3!/K^3$ of occurring). For comparison, for $w = 2$ inputs, a given permuter output pattern occurs with probability $2!/K^2$. Thus, we would expect the number of weight-3 inputs, n_3, resulting in remergent paths in both encoders to be much less than n_2, $n_3 \ll n_2$, with the result being that the inner sum in (7.1) for $w = 3$ is negligible relative to that for $w = 2$. Because our argument assumes a "purely random" permuter, the inequality $n_3 \ll n_2$ has to be interpreted probabilistically. Thus, it is more accurate to write $E\{n_3\} \ll E\{n_2\}$, where the expectation is over all interleavers. Alternatively, for the *average* interleaver, we would expect $n_3 \ll n_2$.

$\underline{w \geq 4}$: Again, we can approximate the inner sum in (7.1) for $w = 4$ in the same manner as in (7.2) and (7.3). Still, we would like to make some comments on its size for the "random" interleaver. A weight-4 input might appear to the first encoder as a weight-3 input concatenated some time later with a weight-1 input, leading to

a non-remergent path in the trellis and, hence, a negligible term in the inner sum in (7.1). It might also appear as a concatenation of two weight-2 inputs, in which case the turbo codeword weight is at least $2d_{2,\min}^{\mathrm{PCCC}}$, again leading to a negligible term in (7.1). Finally, if it happens to be some other pattern divisible by $g_1(D)$ at the first encoder, with probability on the order of $1/K^3$, it will be simultaneously divisible by $g_1(D)$ at the second encoder. (The value of $1/K^3$ derives from the fact that ideally a particular divisible output pattern occurs with probability $4!/K^4$, but there will be approximately K shifted versions of that pattern, each divisible by $g_1(D)$.) Thus, we may expect $n_4 \ll n_2$ so that the $w \geq 4$ terms are negligible in (7.1). The cases for $w > 4$ are argued similarly.

To summarize, the bound in (7.1) can be approximated as

$$\begin{aligned} P_b &\simeq \sum_{w\geq 2} \frac{wn_w}{K} Q\left(\sqrt{\frac{2Rd_{w,\min}^{\mathrm{PCCC}} E_b}{N_0}}\right) \\ &\simeq \max_{w\geq 2} \left\{ \frac{wn_w}{K} Q\left(\sqrt{\frac{2Rd_{w,\min}^{\mathrm{PCCC}} E_b}{N_0}}\right) \right\}, \end{aligned} \tag{7.4}$$

where n_w and $d_{w,\min}^{\mathrm{PCCC}}$ are functions of the particular interleaver employed. From our discussion above, we would expect that the $w = 2$ term dominates for a randomly generated interleaver, although it is easy to find interleavers for which this is not true. One example is the S-random interleaver mentioned earlier. In any case, we observe that P_b decreases with K, so that the error rate can be reduced simply by increasing the interleaver length (w and n_w do not change appreciably with K). This effect is called *interleaver gain* and demonstrates the necessity of large interleavers. Finally, we emphasize that recursive encoders are crucial elements of a turbo code since, for non-recursive encoders, division by $g_1(D)$ (non-remergent trellis paths) would not be an issue and (7.4) would not hold with small values of n_w.

Example 7.2. We consider the P_b performance of a rate-1/2 (23,33) parallel turbo code for two different interleavers of size $K = 1000$. We start first with an interleaver that was randomly generated. We found, for this particular interleaver, $n_2 = 0$ and $n_3 = 1$, with $d_{3,\min}^{\mathrm{PC}} = 9$, so that the $w = 3$ term dominates in (7.4). The interleaver input corresponding to this dominant error event was $D^{168}(1 + D^5 + D^{10})$, which produces the interleaver output $D^{88}(1 + D^{15} + D^{848})$, where of course both polynomials are divisible by $g_1(D) = 1 + D + D^4$. Figure 7.2 gives the simulated P_b performance of this code for 15 iterations of the iterative decoding algorithm detailed in the next section. Also included in Figure 7.2 is the estimate of (7.4) for the same interleaver, which is observed to be very close to the simulated values. The interleaver was then modified to improve the weight spectrum of the code. It was a simple matter to attain $n_2 = 1$ with $d_{2,\min}^{\mathrm{PC}} = 12$ and $n_3 = 4$ with

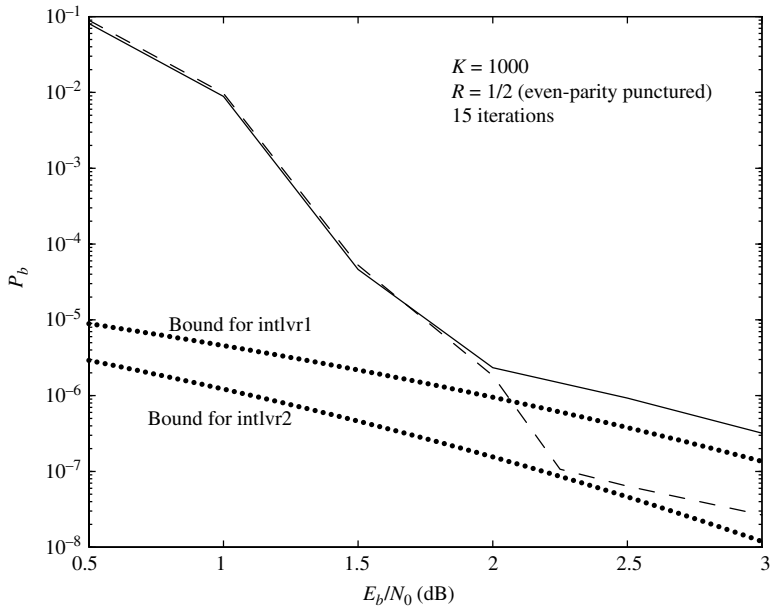


Figure 7.2 Simulated performance of a rate-1/2 (23,33) PCCC for two different interleavers (intlvr1 and intlvr2) ($K = 1000$) together with the asymptotic performance estimates of each given by (7.4).

$d^{\mathrm{PC}}_{3,\min} = 15$ for this second interleaver so that the $w = 2$ term now dominates in (7.4). The simulated and estimated performance curves for this second interleaver are also included in Figure 7.2.

In addition to illustrating the use of the estimate (7.4), this example helps explain the unusual shape of the error-rate curve: it may be interpreted as the usual Q-function shape for a signaling scheme in AWGN with a modest $d_{\min}$, "pushed down" by the interleaver gain $w^* n_{w^*}/K$, where w^* is the maximizing value of w in (7.4) and $w^* n_{w^*}$ is small relative to K.

The spectral-thinning effect that leads to a small $w^* n_{w^*}$ factor is clearly a consequence of the combination of RSC encoders and pseudo-random interleavers. "Thinning" refers to the small multiplicities of low-weight codewords relative to classical codes. For example, consider a cyclic code of length N. Because each cyclic-shift is a codeword, the multiplicity $n_{d_{\min}}$ of the minimum-weight codewords will be a multiple of N. Similarly, consider a convolutional code (recursive or not) with length-K inputs. If the input $u(D)$ creates a minimum-weight codeword, so does $D^j u(D)$ for $j = 1, 2, \ldots$. Thus, the multiplicity $n_{d_{\min}}$ will be a multiple of K. By contrast, for a turbo code, $D^j u(D)$ and $D^k u(D)$, $j \neq k$, will almost surely lead

to codewords of different weights because the presence of the interleaver makes the turbo code time varying.

Occasionally, the codeword error rate, $P_{cw} = \Pr\{\text{decoder output contains one or more residual errors}\}$, is the preferred performance metric. (As mentioned in Chapter 1, P_{cw} is also denoted by FER or WER in the literature, representing "frame" or "word" error rate, respectively.) An estimate of P_{cw} may be derived using steps almost identical to those used to derive P_b. The resulting expressions are nearly identical, except that the factor w/K is removed from the expressions. The result is

$$\begin{aligned} P_{cw} &\simeq \sum_{w\geq 2} n_w Q\left(\sqrt{\frac{2Rd_{w,\min}^{\text{PCCC}}E_b}{N_0}}\right) \\ &\simeq \max_{w\geq 2}\left\{n_w Q\left(\sqrt{\frac{2Rd_{w,\min}^{\text{PCCC}}E_b}{N_0}}\right)\right\}. \end{aligned} \tag{7.5}$$

The interleaver has an effect on P_{cw} as well and its impact involves the values of n_w for small values of w: relative to conventional unconcatenated codes, the values for n_w are small.

7.2 The PCCC Iterative Decoder

Given the knowledge gained from Chapters 4 and 5 on BCJR decoding and the turbo principle, respectively, one should be able to derive the iterative (turbo) decoder for a PCCC. The turbo decoder which follows directly from the turbo principle discussed in Chapter 5 is presented in Figure 7.3 (where, as in

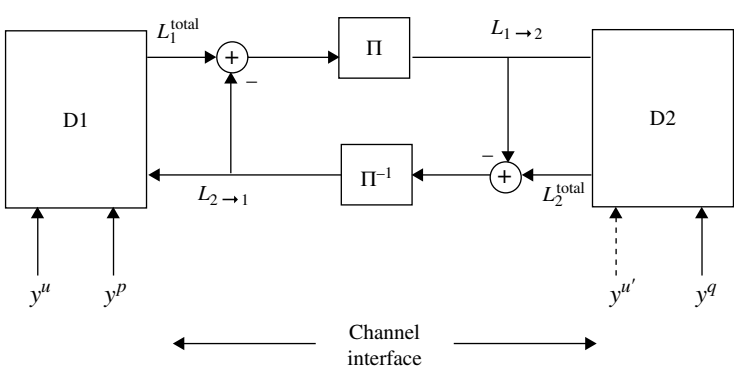


Figure 7.3 A turbo decoder obtained directly from the turbo principle (Chapter 5). L_1^{total} is the total log-likelihood information at the output of RSC decoder D1, and similarly for L_2^{total}. The log-likelihood quantity $L_{1\to 2}$ is the extrinsic information sent from RSC decoder 1 (D1) to RSC decoder 2 (D2), and similarly for $L_{2\to 1}$.

Chapter 5, necessary buffering is omitted). The two decoders are soft-in/soft-out (SISO) decoders, typically BCJR decoders, matched to the top and bottom RSC encoders of Figure 7.1. SISO decoder 1 (D1) receives noisy data (y_k^u) and parity (y_k^p) bits from the channel, corresponding to the bits u_k and p_k transmitted by RSC 1 in Figure 7.1. Decoder D2 receives only the noisy parity bits y_k^q, corresponding to the output q_k of RSC 2 in Figure 7.1. Per the turbo principle, only extrinsic log-likelihood-ratio (LLR) information is sent from one decoder to the other, with appropriate interleaving/de-interleaving in accordance with the PCCC encoder. We will learn below how a BCJR decoder processes information from both the channel and from a companion decoder. After a certain number of iterations, either decoder can sum the LLRs from the channel, from its companion decoder, and from its own computations to produce the total LLR values for the data bits u_k. Decisions are made from the total LLR values: $\hat{u}_k = \text{sign}[LLR(u_k)]$.

An alternative interpretation of the decoder can be appreciated by considering the two graphical representations of a PCCC in Figure 7.4. The top graph

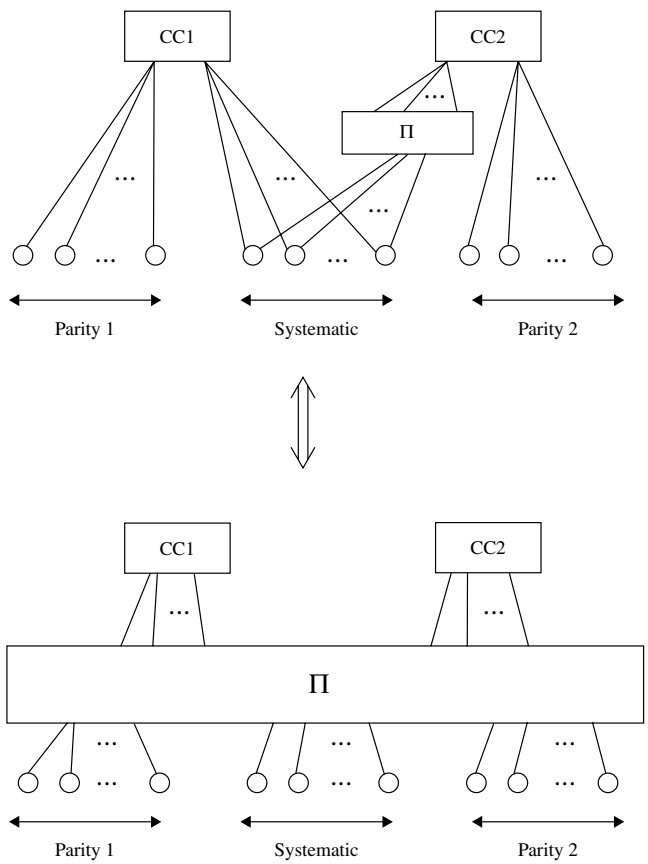


Figure 7.4 Interpretation of a PCCC code's graph as the graph of a generalized LDPC code.

follows straightforwardly from Figure 7.1 and the bottom graph follows straightforwardly from the top graph (where the bottom interleaver permutes only the edges between the second convolutional code constraint node (CC2) and the systematic bit nodes). Now observe that the bottom graph is essentially that of a generalized LDPC code whose decoder was discussed in Chapter 5. Thus, the decoder for a PCCC is very much like that of a generalized LDPC code.

In spite of the foregoing discussion, we still find it useful and instructive to present the details of the iterative PCCC decoder ("turbo decoder"). In particular, it is helpful to illuminate the computation of extrinsic information within the constituent BCJR decoders and how these BJCR decoders process incoming extrinsic information together with incoming channel information.

7.2.1 Overview of the Iterative Decoder

The goal of the iterative decoder is to iteratively estimate the *a posteriori* probabilities (APPs) $\Pr(u_k|\mathbf{y})$, where u_k is the kth data bit, $k = 1, 2, \ldots, K$, and $\mathbf{y}$ is the received codeword in AWGN, $\mathbf{y} = \mathbf{c} + \mathbf{n}$. In this equation, we assume for convenience that the components of $\mathbf{c}$ take values in the set $\{\pm 1\}$ (and similarly for $\mathbf{u}$) and that $\mathbf{n}$ is a noise word whose components are AWGN samples. Knowledge of the APPs allows one to make optimal decisions on the bits u_k via the maximum *a posteriori* (MAP) rule[1]

$$\frac{\Pr(u_k = +1|\mathbf{y})}{\Pr(u_k = -1|\mathbf{y})} \underset{-1}{\overset{+1}{\gtrless}} 1$$

or, more conveniently,

$$\hat{u}_k = \operatorname{sign}[L(u_k)]\,, \tag{7.6}$$

where $L(u_k)$ is the log *a posteriori* probability (log-APP) ratio defined as

$$L(u_k) \triangleq \log\left(\frac{\Pr(u_k = +1|\mathbf{y})}{\Pr(u_k = -1|\mathbf{y})}\right).$$

We shall use the term log-likelihood ratio (LLR) in place of log-APP ratio for consistency with the literature.

From Bayes' rule, the LLR for an arbitrary SISO decoder can be written as

$$L(u_k) = \log\left(\frac{p(\mathbf{y}|u_k = +1)}{p(\mathbf{y}|u_k = -1)}\right) + \log\left(\frac{\Pr(u_k = +1)}{\Pr(u_k = -1)}\right) \tag{7.7}$$

[1] It is well known that the MAP rule minimizes the probability of bit error. For comparison, the ML rule, which maximizes the likelihoods $P(\mathbf{y}|\mathbf{c})$ over the codewords $\mathbf{c}$, minimizes the probability of codeword error. See Chapter 1.

with the second term representing *a priori* information. Since typically $\Pr(u_k = +1) = \Pr(u_k = -1)$, the *a priori* term is usually zero for conventional decoders. However, for *iterative* decoders, each component decoder receives extrinsic information for each u_k from its companion decoder, which serves as *a priori* information. We adopt the convention that the top RSC encoder in Figure 7.1 is encoder 1, denoted E1, and the bottom encoder is encoder 2, denoted E2. The SISO component decoders matched to E1 and E2 will be denoted by D1 and D2, respectively. The idea behind extrinsic information is that D2 provides soft information to D1 for each u_k using only information not available to D1, and D1 does likewise for D2. The iterative decoding proceeds as D1 → D2 → D1 → D2 → D1 → ..., with the previous decoder passing soft information along to the next decoder at each half-iteration. Either decoder may initiate the chain of component decodings or, for hardware implementations, D1 and D2 may operate simultaneously as we shall see. This type of iterative algorithm is known to converge to the true value of the LLR $L(u_k)$ for the concatenated code, provided that the graphical representation of this code contains no loops [23–25]. The graph of a turbo code does in fact contain loops [25], but the algorithm nevertheless generally provides near-ML performance.

7.2.2 Decoder Details

We assume no puncturing in the PCCC encoder of Figure 7.1 so that the overall code rate is 1/3. The transmitted codeword $\mathbf{c}$ will have the form $\mathbf{c} = [c_1, c_2, \ldots, c_K] = [u_1, p_1, q_1, \ldots, u_K, p_K, q_K]$, where $c_k \triangleq [u_k, p_k, q_k]$. The received word $\mathbf{y} = \mathbf{c} + \mathbf{n}$ will have the form $\mathbf{y} = [y_1, y_2, \ldots, y_K] = [y_1^u, y_1^p, y_1^q, \ldots, y_K^u, y_K^p, y_K^q]$, where $y_k \triangleq [y_k^u, y_k^p, y_k^q]$, and similarly for $\mathbf{n}$. We denote the codewords produced by E1 and E2 by, respectively, $\mathbf{c}_1 = [c_1^1, c_2^1, \ldots, c_K^1]$, where $c_k^1 \triangleq [u_k, p_k]$, and $\mathbf{c}_2 = [c_1^2, c_2^2, \ldots, c_K^2]$, where $c_k^2 \triangleq [u'_k, q_k]$. Note that $\{u'_k\}$ is a permuted version of $\{u_k\}$ and is not actually transmitted (see Figure 7.1). We define the noisy received versions of $\mathbf{c}_1$ and $\mathbf{c}_2$ to be $\mathbf{y}_1$ and $\mathbf{y}_2$, respectively, having components $y_k^1 \triangleq [y_k^u, y_k^p]$ and $y_k^2 \triangleq [y_k^{u'}, y_k^q]$, respectively. Note that $\mathbf{y}_1$ and $\mathbf{y}_2$ can be assembled from $\mathbf{y}$ in an obvious fashion: using an interleaver to obtain $\left\{y_k^{u'}\right\}$ from $\{y_k^u\}$. For doing so, the component decoder inputs are the two vectors $\mathbf{y}_1$ and $\mathbf{y}_2$.

Each SISO decoder is essentially a BCJR decoder (developed in Chapter 4), except that each BCJR decoder is modified so that it may accept extrinsic information from a companion decoder and produce soft outputs. Thus, rather than using the LLRs to obtain hard decisions, the LLRs become the SISO BCJR decoder's outputs. We shall often use SISO and BCJR interchangeably. To initiate the discussion, we present the BCJR algorithm summary for a single RSC code, which has been adapted from Chapter 4. (For the time being, we will discuss a generic SISO decoder so that we may avoid using cumbersome superscripts for the two constituent decoders until it is necessary to do so.)

Algorithm 7.1 BCJR Algorithm Summary

Initialize $\tilde{\alpha}_0(s)$ and $\tilde{\beta}_K(s)$ according to

$$\tilde{\alpha}_0(s) = \begin{cases} 0, & s = 0 \\ -\infty, & s \neq 0 \end{cases}$$

$$\tilde{\beta}_K(s) = \begin{cases} 0, & s = 0 \\ -\infty, & s \neq 0 \end{cases}$$

for $k = 1$ to K

- get $y_k = [y_k^u, y_k^p]$
- compute $\tilde{\gamma}_k(s', s)$ for all allowable state transitions $s' \to s$ ($c_k = c_k(s', s)$)
- compute $\tilde{\alpha}_k(s)$ for all s using the recursion

$$\tilde{\alpha}_k(s) = \max_{s'}{}^* [\tilde{\alpha}_{k-1}(s') + \tilde{\gamma}_k(s', s)], \tag{7.8}$$

end

for $k = K$ to 2 step -1

compute $\tilde{\beta}_{k-1}(s')$ for all s' using the recursion

$$\tilde{\beta}_{k-1}(s') = \max_{s}{}^* \left[\tilde{\beta}_k(s) + \tilde{\gamma}_k(s', s)\right], \tag{7.9}$$

end

for $k = 1$ to K

- compute $L(u_k)$ using

$$\begin{aligned} L(u_k) = & \max_{U^+}{}^* \left[\tilde{\alpha}_{\ k-1}(s') + \tilde{\gamma}_k(s', s) + \tilde{\beta}_k(s)\right] \\ & - \max_{U^-}{}^* \left[\tilde{\alpha}_{\ k-1}(s') + \tilde{\gamma}_k(s', s) + \tilde{\beta}_k(s)\right]. \end{aligned} \tag{7.10}$$

end

In the algorithm above, the branch metric $\tilde{\gamma}_k(s', s) = \log \gamma_k(s', s)$, where, from Chapter 4,

$$\gamma_k(s', s) = \frac{P(u_k)}{2\pi\sigma^2} \exp\left[-\frac{\|y_k - c_k\|^2}{2\sigma^2}\right]. \tag{7.11}$$

$P(u_k)$ is the probability that the data bit $u_k(s', s)$ corresponding to the transition $s' \to s$ is equal to the specific value $u_k \in \{\pm 1\}$, i.e., $P(u_k) = \Pr(u_k(s', s) = u_k)$. In Chapter 4, assuming equiprobable encoder inputs, we promptly set $P(u_k) = 1/2$

for either value of u_k. As indicated earlier, the extrinsic information received from a companion decoder takes the role of *a priori* information in the iterative decoding algorithm (cf. (7.7) and surrounding discussion), so that

$$L^{\mathrm{e}}(u_k) \Leftrightarrow \log\left(\frac{\Pr(u_k = +1)}{\Pr(u_k = -1)}\right). \tag{7.12}$$

To incorporate extrinsic information into the BCJR algorithm, we consider the log-domain version of the branch metric,

$$\tilde{\gamma}_k(s', s) = \log(P(u_k)) - \log\bigl(2\pi\sigma^2\bigr) - \frac{\|y_k - c_k\|^2}{2\sigma^2}. \tag{7.13}$$

Now observe that we may write

$$\begin{aligned} P(u_k) &= \left(\frac{\exp[-L^{\mathrm{e}}(u_k)/2]}{1 + \exp[-L^{\mathrm{e}}(u_k)]}\right) \cdot \exp[u_k L^{\mathrm{e}}(u_k)/2] \\ &= A_k \exp[u_k L^{\mathrm{e}}(u_k)/2], \end{aligned} \tag{7.14}$$

where the first equality follows since the right-hand side equals

$$\left(\frac{\sqrt{P_-/P_+}}{1 + P_-/P_+}\right)\sqrt{P_+/P_-} = P_+$$

when $u_k = +1$ and

$$\left(\frac{\sqrt{P_-/P_+}}{1 + P_-/P_+}\right)\sqrt{P_-/P_+} = P_-$$

when $u_k = -1$, and we have defined $P_+ \triangleq \Pr(u_k = +1)$ and $P_- \triangleq \Pr(u_k = -1)$ for convenience. Substitution of (7.14) into (7.13) yields

$$\tilde{\gamma}_k(s', s) = \log\bigl[A_k/(2\pi\sigma^2)\bigr] + u_k L^{\mathrm{e}}(u_k)/2 - \frac{\|y_k - c_k\|^2}{2\sigma^2}, \tag{7.15}$$

where the first term may be ignored since it is independent of u_k (equivalently, of $s' \to s$). (It may appear from the notation that A_k is a function of u_k since it is a function of $L^{\mathrm{e}}(u_k)$, but from (7.12) the latter quantity is a function of the probability mass function for u_k, not u_k.) In summary, the extrinsic information received from a companion decoder is included in the computation through the modified branch metric

$$\tilde{\gamma}_k(s', s) = u_k L^{\mathrm{e}}(u_k)/2 - \frac{\|y_k - c_k\|^2}{2\sigma^2}. \tag{7.16}$$

The rest of the SISO BCJR algorithm proceeds exactly as before.

Upon substitution of (7.16) into (7.10), we have

$$\begin{aligned} L(u_k) = \; & L^{\mathrm{e}}(u_k) + \max_{U^+}{}^* \left[\tilde{\alpha}_{k-1}(s') + u_k y_k^u/\sigma^2 + p_k y_k^p/\sigma^2 + \tilde{\beta}_k(s)\right] \\ & - \max_{U^-}{}^* \left[\tilde{\alpha}_{k-1}(s') + u_k y_k^u/\sigma^2 + p_k y_k^p/\sigma^2 + \tilde{\beta}_k(s)\right], \end{aligned} \tag{7.17}$$

where we have used the fact that

$$\begin{aligned} \|y_k - c_k\|^2 &= (y_k^u - u_k)^2 + (y_k^p - p_k)^2 \\ &= (y_k^u)^2 - 2u_k y_k^u + u_k^2 + \left(y_k^p\right)^2 - 2p_k y_k^p + p_k^2 \end{aligned}$$

and only the terms in this expression dependent on U^+ or U^-, namely $u_k y_k^u/\sigma^2$ and $p_k y_k^p/\sigma^2$, survive after the subtraction. Now note that $u_k y_k^u/\sigma^2 = y_k^u/\sigma^2$ under the first $\max^*(\cdot)$ operation in (7.17) (U^+ is the set of state transitions for which $u_k = +1$) and $u_k y_k^u/\sigma^2 = -y_k^u/\sigma^2$ under the second $\max^*(\cdot)$ operation. Using the definition for $\max^*(\cdot)$, it is easy to see that these terms may be isolated out so that

$$\begin{aligned} L(u_k) = 2y_k^u/\sigma^2 + L^{\mathrm{e}}(u_k) + \; & \max_{U^+}{}^* \left[\tilde{\alpha}_{k-1}(s') + p_k y_k^p/\sigma^2 + \tilde{\beta}_k(s)\right] \\ & - \max_{U^-}{}^* \left[\tilde{\alpha}_{k-1}(s') + p_k y_k^p/\sigma^2 + \tilde{\beta}_k(s)\right]. \end{aligned} \tag{7.18}$$

The interpretation of this new expression for $L(u_k)$ is that the first term is likelihood information received directly from the channel, the second term is extrinsic likelihood information received from a companion decoder, and the third "term" ($\max_{U^+}{}^* - \max_{U^-}{}^*$) is extrinsic likelihood information to be passed to a companion decoder. Note that this third term is likelihood information gleaned from received parity that is not available to the companion decoder. Thus, specializing to decoder D1, for example, on any given iteration, D1 computes

$$L_1(u_k) = 2y_k^u/\sigma^2 + L_{21}^{\mathrm{e}}(u_k) + L_{12}^{\mathrm{e}}(u_k), \tag{7.19}$$

where $L_{21}^{\mathrm{e}}(u_k)$ is extrinsic information received from D2, and $L_{12}^{\mathrm{e}}(u_k)$ is the ($\max_{U^+}{}^* - \max_{U^-}{}^*$) term in (7.18) which is to be used as extrinsic information from D1 to D2.

To recapitulate, the turbo decoder for a PCCC includes two SISO-BCJR decoders matched to the two constituent RSC encoders within the PCCC encoder. As summarized in (7.19) for D1, each SISO decoder accepts scaled soft channel information ($2y_k^u/\sigma^2$) as well as soft extrinsic information from its counterpart ($L_{21}^{\mathrm{e}}(u_k)$). Each SISO decoder also computes soft extrinsic information to send to its counterpart ($L_{12}^{\mathrm{e}}(u_k)$). The two SISO decoders would be configured in a turbo decoder as in Figure 7.5. Although this more symmetric decoder configuration appears to be different from the decoder configuration of Figure 7.3, the main difference being that $\left\{y_k^{u'}\right\}$ are not fed to D2 in Figure 7.3, they are essentially equivalent. To see this, suppose $L_{2\to 1}$ in Figure 7.3 is equal to $L_{12}^{\mathrm{e}}(u_k)$ so that the inputs to D1 in that figure are y_k^u, y_k^p, and $L_{12}^{\mathrm{e}}(u_k)$. Then, from our

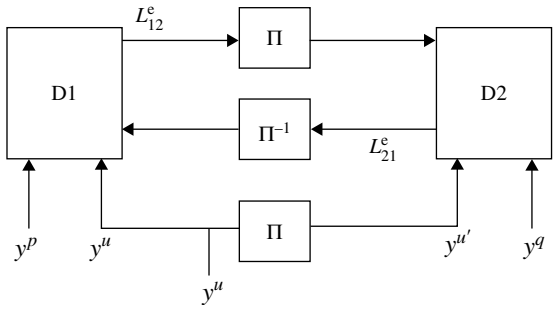


Figure 7.5 The turbo decoder for a PCCC code.

development that led to (7.19), the output of D1 in Figure 7.3 is clearly $L_1^{\text{total}} = L_1(u_k) = 2y_k^u/\sigma^2 + L_{21}^{\text{e}}(u_k) + L_{12}^{\text{e}}(u_k)$. From this, $L_{1\to 2}$ must be $L_1^{\text{total}} - L_{2\to 1} = 2y_k^u/\sigma^2 + L_{12}^{\text{e}}(u_k)$ so that $L_2^{\text{total}} = L_2(u_k) = 2y_k^u/\sigma^2 + L_{12}^{\text{e}}(u_k) + L_{21}^{\text{e}}(u_k)$. Lastly, $L_{2\to 1} = L_2^{\text{total}} - L_{1\to 2} = L_{12}^{\text{e}}(u_k)$, which agrees with our initial assumption.

7.2.3 Summary of the PCCC Iterative Decoder

We now present pseudo-code for the turbo decoder for PCCCs. The algorithm given below for the iterative decoding of a parallel turbo code follows directly from the development above. The constituent decoder order is D1, D2, D1, D2, etc. The interleaver and de-interleavers are represented by the arrays $P[\cdot]$ and $P\text{inv}[\cdot]$, respectively. For example, the permuted word $\mathbf{u}'$ is obtained from the original word $\mathbf{u}$ via the following pseudo-code statement: `for` $k = 1 : K$, $u'_k = u_{P[k]}$, `end`. We point out that knowledge of the noise variance $\sigma^2 = N_0/2$ by each SISO BCJR decoder is necessary. Also, a simple way to obtain higher code rates via puncturing is, in the computation of $\gamma_k(s', s)$, to set to zero the received parity samples, y_k^p or y_k^q, corresponding to the punctured parity bits, p_k or q_k. (This will set to zero the term in the branch metric corresponding to the punctured bit.) Thus, puncturing need not be performed at the encoder for computer simulations.

When discussing the BCJR algorithm, it is usually assumed that the trellis starts in the zero state and terminates in the zero state. This is accomplished for a single convolutional code by appending μ appropriately chosen "termination bits" at the end of the data word (μ is the RSC memory size), and it is accomplished in the same way for E1 in the PCCC encoder. However, termination of encoder E2 to the zero state can be problematic due to the presence of the interleaver. (Various solutions exist in the literature.) Fortunately, there is generally a small performance loss when E2 is not terminated. In this case, $\beta_K(s)$ for D2 may be set to $\alpha_K(s)$ for all s, or it may be set to a nonzero constant (e.g., $1/S_2$, where S_2 is the number of E2 states).

We remark that some sort of iteration-stopping criterion is necessary. The most straightforward criterion is to set a maximum number of iterations. However, this

can be inefficient since the correct codeword is often found after only a few iterations. An efficient technique utilizes a carefully chosen outer error-detection code. After each iteration, a parity check is carried out and the iterations stop whenever no error is detected. Other stopping criteria are presented in the literature. Observe that it is not possible to use the parity-check matrix $\mathbf{H}$ for the PCCC together with the check equation $\mathbf{c}\mathbf{H}^{\mathrm{T}} = \mathbf{0}$ because the decoder outputs are decisions on the systematic bits, not on the entire codeword.

We first present an outline of the turbo decoding algorithm assuming D1 decodes first. Then we present the algorithm in detail.

Outline

1. Initialize all state metrics appropriately and set all extrinsic information to zero.
2. D1 decoder: Run the SISO BJCR algorithm with inputs $y_k^1 = [y_k^u, y_k^p]$ and $L_{21}^{\mathrm{e}}(u_{P\mathrm{inv}[k]})$ to obtain $L_{12}^{\mathrm{e}}(u_k)$. Send extrinsic information $L_{12}^{\mathrm{e}}(u_{P[k]})$ to the D2 decoder.
3. D2 decoder: Run the SISO BJCR algorithm with inputs $y_k^2 = [y_{P[k]}^u, y_k^q]$ and $L_{12}^{\mathrm{e}}(u_{P[k]})$ to obtain $L_{21}^{\mathrm{e}}(u_k)$. Send extrinsic information $L_{21}^{\mathrm{e}}(u_{P\mathrm{inv}[k]})$ to the D1 decoder.
4. Repeat Steps 2 and 3 until the preset maximum number of iterations is reached or some other stopping criterion is satisfied. Make decisions on bits according to $\mathrm{sign}[L_1(u_k)]$, where $L_1(u_k) = 2y_k^u/\sigma^2 + L_{21}^{\mathrm{e}}(u_{P\mathrm{inv}[k]}) + L_{12}^{\mathrm{e}}(u_k)$. (One can instead use $\mathrm{sign}[L_2(u_k)]$.)

Algorithm 7.2 PCCC Iterative Decoder

Initialization

D1:

$$\tilde{\alpha}_0^{(1)}(s) = \begin{cases} 0 & \text{for } s = 0 \\ -\infty & \text{for } s \neq 0 \end{cases}$$

$$\tilde{\beta}_K^{(1)}(s) = \begin{cases} 0 & \text{for } s = 0 \\ -\infty & \text{for } s \neq 0 \end{cases}$$

$$L_{21}^{\mathrm{e}}(u_k) = 0 \text{ for } k = 1, 2, \ldots, K$$

D2:

$$\tilde{\alpha}_0^{(2)}(s) = \begin{cases} 0 & \text{for } s = 0 \\ -\infty & \text{for } s \neq 0 \end{cases}$$

$\tilde{\beta}_K^{(2)}(s) = \tilde{\alpha}_K^{(2)}(s)$ for all s (set once after computation of $\{\tilde{\alpha}_K^{(2)}(s)\}$ in the *first* iteration)

$L_{12}^{\mathrm{e}}(u_k)$ is to be determined from D1 after the first half-iteration and so need not be initialized

The nth iteration

D1:

for $k = 1$ to K

- get $y_k^1 = [y_k^u, y_k^p]$
- compute $\tilde{\gamma}_k(s', s)$ for all allowable state transitions $s' \to s$ from (7.16) or the simplified form (see the discussion following (7.16))

$$\tilde{\gamma}_k(s', s) = u_k L_{21}^{\mathrm{e}}(u_{P\mathrm{inv}[k]})/2 + u_k y_k^u/\sigma^2 + p_k y_k^p/\sigma^2$$

  (u_k (p_k) in this expression is set to the value of the encoder input (output) corresponding to the transition $s' \to s$)
- compute $\tilde{\alpha}_k^{(1)}(s)$ for all s using (7.8)

end

for $k = K$ to 2 step -1

- compute $\tilde{\beta}_{k-1}^{(1)}(s)$ for all s using (7.9)

end

for $k = 1$ to K

- compute $L_{12}^{\mathrm{e}}(u_k)$ using

$$\begin{aligned} L_{12}^{\mathrm{e}}(u_k) = & \max_{U^+}{}^* \left[\tilde{\alpha}_{k-1}^{(1)}(s') + p_k y_k^p/\sigma^2 + \tilde{\beta}_k^{(1)}(s)\right] \\ & - \max_{U^-}{}^* \left[\tilde{\alpha}_{k-1}^{(1)}(s') + p_k y_k^p/\sigma^2 + \tilde{\beta}_k^{(1)}(s)\right] \end{aligned}$$

end

D2:

for $k = 1$ to K

- get $y_k^2 = [y_{P[k]}^u, y_k^q]$
- compute $\tilde{\gamma}_k(s', s)$ for all allowable state transitions $s' \to s$ from

$$\tilde{\gamma}_k(s', s) = u_k L_{12}^{\mathrm{e}}(u_{P[k]})/2 + u_k y_{P[k]}^u/\sigma^2 + q_k y_k^q/\sigma^2$$

  (u_k (q_k) in this expression is set to the value of the encoder input (output) corresponding to the transition $s' \to s$)
- compute $\tilde{\alpha}_k^{(2)}(s)$ for all s using (7.8)

end

for $k = K$ to 2 step -1

- compute $\tilde{\beta}_{k-1}^{(2)}(s)$ for all s using (7.9)

end

for $k = 1$ to K

- compute $L_{21}^{\mathrm{e}}(u_k)$ using

$$L_{21}^{\mathrm{e}}(u_k) = \max_{U^+}{}^* \left[\tilde{\alpha}_{k-1}^{(2)}(s') + q_k y_k^q/\sigma^2 + \tilde{\beta}_k^{(2)}(s)\right] - \max_{U^-}{}^* \left[\tilde{\alpha}_{k-1}^{(2)}(s') + q_k y_k^q/\sigma^2 + \tilde{\beta}_k^{(2)}(s)\right]$$

end

Decision after the last iteration
for $k = 1$ to K

- compute

$$L_1(u_k) = 2y_k^u/\sigma^2 + L_{21}^{\mathrm{e}}(u_{P\mathrm{inv}[k]}) + L_{12}^{\mathrm{e}}(u_k)$$

$\hat{u}_k = \mathrm{sign}[L(u_k)]$

end

7.2.4 Lower-Complexity Approximations

The core algorithm within the turbo decoder, the BCJR algorithm, uses the max* function which (from Chapter 4) has the following representations:

$$\begin{aligned} \max{}^*(x, y) &\triangleq \log(e^x + e^y) \\ &= \max(x, y) + \log\left(1 + e^{-|x-y|}\right). \end{aligned} \tag{7.20}$$

From the second expression, we can see that this function may be implemented by a max function followed by use of a look-up table for the term $\log\left(1 + e^{-|x-y|}\right)$. It has been shown that such a table can be quite small (e.g., of size 8), but its size depends on the turbo code in question and the performance requirements. Alternatively, since $\log\left(1 + e^{-|x-y|}\right) \leq \ln(2) = 0.693$, this term can be dropped in (7.20), with the max function used in place of the max* function. This will, of course, be at the expense of some performance loss, which depends on turbo-code parameters. Note that, irrespective of whether it is max* or max that is used, when more than two quantities are involved, the computation may take place pair-wise. For example, for the three quantities x, y, and z, $\max{}^*(x, y, z) = \max{}^*[\max{}^*(x, y), z]$.

An alternative to the BCJR algorithm is the *soft-output Viterbi algorithm* (SOVA) [7, 12, 13], which is a modification of the Viterbi algorithm that produces soft (reliability) outputs for use in iterative decoders and also accepts extrinsic information. The BJCR algorithm is an algorithm that has two Viterbi-like algorithms, one forward and one backward. Thus, the complexity of the SOVA is about half that of the BCJR algorithm. Moreover, as we will see, unlike the BCJR decoder, the SOVA decoder does not require knowledge of the noise variance. Of course, the advantages of the SOVA are gained at the expense of a performance degradation. For most situations, this degradation can be made to be quite small by attenuating the SOVA output (its reliability estimates are too large, on average).

We assume a rate-$1/n$ convolutional code so that the cumulative (squared Euclidean distance) metric $\Gamma_k(s)$ for state s at time k is updated according to

$$\Gamma_k(s) = \min\{\Gamma_{k-1}(s') + \lambda_k(s', s), \Gamma_{k-1}(s'') + \lambda_k(s'', s)\}, \tag{7.21}$$

where s' and s'' are the two states leading to state s, $\lambda_k(s', s)$ is the branch metric for the transition from state s' to state s, and the rest of the quantities are obviously defined. When the Viterbi decoder chooses a survivor according to (7.21), it chooses in favor of the smaller of the two metrics. The difference between these two metrics is a measure of the reliability of that decision and, hence, of the reliability of the corresponding bit decision. This is intuitively so, but we can demonstrate it as follows.

Define the reliability of a decision to be the LLR of the correct/error binary hypothesis:

$$\rho = \ln\left(\frac{1 - \Pr(\text{error})}{\Pr(\text{error})}\right).$$

To put this in terms of the Viterbi algorithm, consider the decision made at one node in a trellis according to the add–compare–select operation of (7.21). Owing to code linearity and channel symmetry, the Pr(error) derivation is independent of the correct path. Thus, let S_1 be the event that the correct path comes from state s' and let D_2 be the event that the path from s'' is chosen. Define also

$$M'_k = \Gamma_{k-1}(s') + \lambda_k(s', s)$$

and

$$M''_k = \Gamma_{k-1}(s'') + \lambda_k(s'', s)$$

so that $M_k'' \leq M_k'$ when event D_2 occurs. Then, several applications of Bayes' rule and cancellation of equal probabilities yield the following:

$$\begin{aligned}
\Pr(\text{error}) &= \Pr(D_2|\mathbf{y}, S_1) \\
&= \frac{\Pr(D_2)}{\Pr(S_1)} \Pr(S_1|\mathbf{y}, D_2) \\
&= \Pr(S_1|\mathbf{y}) \\
&= \frac{p(\mathbf{y}|S_1)\Pr(S_1)}{p(\mathbf{y})} \\
&= \frac{p(\mathbf{y}|S_1)}{p(\mathbf{y}|S_1) + p(\mathbf{y}|S_2)} \\
&= \frac{\exp\left[-M_k'/(2\sigma^2)\right]}{\exp\left[-M_k'/(2\sigma^2)\right] + \exp\left[-M_k''/(2\sigma^2)\right]} \\
&= \frac{1}{1 + \exp\left[\left(M_k' - M_k''\right)/(2\sigma^2)\right]} \\
&= \frac{1}{1 + \exp[\Delta_k/(2\sigma^2)]},
\end{aligned} \tag{7.22}$$

where Δ_k is the tentative metric-difference magnitude at time k:

$$\Delta_k = |M_k' - M_k''| \,. \tag{7.23}$$

Rearranging (7.22) yields

$$\frac{\Delta_k}{2\sigma^2} = \ln\left(\frac{1 - \Pr(\text{error})}{\Pr(\text{error})}\right) = \rho, \tag{7.24}$$

which coincides with our earlier intuition that the metric difference is a measure of reliability.

Note that ρ includes the noise variance σ^2, whereas the standard Viterbi algorithm does not, that is, the branch metrics $\lambda_k(.,.)$ in (7.21) are unscaled Euclidean distances. If we use the same branch metrics as used by the BCJR algorithm, then the noise variance will be automatically included in the metric-difference computation. That is, if we replace the squared Euclidean-distance branch metric $\lambda_k(s', s)$ by the branch metric (we now include the incoming extrinsic information term)

$$\tilde{\gamma}_k(s', s) = u_k L^{\mathrm{e}}(u_k)/2 + u_k y_k^u/\sigma^2 + p_k y_k^p/\sigma^2, \tag{7.25}$$

we have the alternative normalized metrics

$$\tilde{M}_k' = \Gamma_{k-1}(s') + \tilde{\gamma}_k(s', s) \tag{7.26}$$

and

$$\tilde{M}_k'' = \Gamma_{k-1}(s'') + \tilde{\gamma}_k(s'', s). \tag{7.27}$$

For the correlation metric (7.25), the cumulative metrics are computed as

$$\Gamma_k(s) = \max\{M'_k, M''_k\}.$$

The metric difference Δ_k is now defined in terms of these normalized metrics as

$$\Delta_k = \left|\tilde{M}'_k - \tilde{M}''_k\right| /2$$

and, under this new definition, the metric reliability is given by $\rho = \Delta_k/2$.

Observe that Δ_k gives a path-wise reliability, whereas we need to compute bit-wise reliabilities to be used as soft outputs. To obtain the soft output for a given bit u_k, we first obtain the hard decision $\hat{u}_k$ after a delay δ (i.e., at time $k+\delta$), where δ is the decoding depth. At time $k+\delta$, we select the surviving path with the largest metric. We trace back the largest-metric path to obtain the hard decision $\hat{u}_k$. Along this path, there are $\delta+1$ non-surviving paths that have been discarded (due to the add–compare–select algorithm), and each non-surviving path has a certain difference metric Δ_j, where $k \le j \le k+\delta$. The *bit-wise reliability* for the decision $\hat{u}_k$ is defined as

$$\Delta^*_k = \min\{\Delta_k, \Delta_{k+1}, \ldots, \Delta_{k+\delta}\},$$

where the minimum is taken only over the non-surviving paths along the largest-metric path within the time window $[k \le j \le k+\delta]$ that would have led to a different decision for $\hat{u}_k$. We may now write that the soft output for bit u_k is given by

$$L_{\text{SOVA}}(u_k) = \hat{u}_k \Delta^*_k.$$

Thus, the turbo decoder of Figure 7.3 would have as constituent decoders two SOVA decoders with $\{L_{\text{SOVA}}(u_k)\}$ acting as the total LLR outputs. We mention some additional details that are often incorporated in practice. First, the scaling by σ^2 is often dropped in practice, making knowledge of the noise variance unnecessary. For this version of the SOVA, the computed extrinsic information tends to be larger on average than that of a BCJR. Thus, attenuation of $L_{1\to 2}$ and $L_{2\to 1}$ in Figure 7.3 generally improves performance. See [21, 22] for additional details.

Next, since the two RSC encoders start at the zero state, and E1 is terminated at the zero state, the SOVA-based turbo decoder should exploit this as follows. For times $k = 1, 2, \ldots, K$, the SOVA decoders determine the survivors and their difference metrics, but no decisions are made until $k = K$. At this time, select for D1 the single surviving path that ends at state zero. For D2, select as the single surviving path at time $k = K$ the path with the highest metric. For each constituent decoder, the decisions may be obtained by tracing back along their respective sole survivors, computing $\{\hat{u}_k\}$, $\{\Delta_k\}$, and $\{\Delta^*_k\}$ along the way.

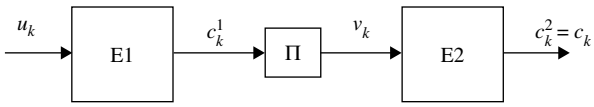


Figure 7.6 The encoder for a standard serial-concatenated convolutional code.

7.3 Serial-Concatenated Convolutional Codes

Figure 7.6 depicts the encoder for a standard serial-concatenated convolutional code (SCCC). An SCCC encoder consists of two binary convolutional encoders (E1 and E2) separated by a (K/R_1)input word length. The outer code has rate R_1, the inner code has rate R_2, with possible puncturing occurring at either encoder, and the overall code rate is $R = R_1R_2$. As will be made evident below, an SCCC can achieve interleaving gain only if the inner code is a recursive convolutional code (usually systematic), although the outer code need not be recursive. Much like in the PCCC case, the interleaver is important in that it ensures that if the inner encoder produces a low-weight codeword it is unlikely that the outer code does so too.

Example 7.3. Suppose the outer and inner codes have identical generator matrices

$$\mathbf{G}_1(D) = \mathbf{G}_2(D) = \begin{bmatrix} 1 & \dfrac{1+D^2}{1+D+D^2} \end{bmatrix}.$$

Now suppose the input is $u(D) = 1 + D + D^2$. Then the output of the outer encoder, E1, is $\begin{bmatrix} 1+D+D^2 & 1+D^2 \end{bmatrix}$ or, as a multiplexed binary sequence, the output is 11111000.... The output of the interleaver (e.g., an S-random interleaver) will be five scattered 1s. But, ignoring delays (or leading zeros), the response of encoder E2 to each of the five isolated 1s is

$$\begin{bmatrix} 1 & \dfrac{1+D^2}{1+D+D^2} \end{bmatrix} = \begin{bmatrix} 1 & 1+D+D^2+D^4+D^5+D^7+D^8+\cdots \end{bmatrix}$$

or, as a multiplexed binary sequence, 1101010001010001010.... Thus, the combined effect of the interleaver and the recursive inner code is the amplification of inner codeword weight so that the serial-concatenated code will have a large minimum distance (when the interleaver is properly designed).

7.3.1 Performance Estimate on the BI-AWGNC

Because of their similarity, and their linearity, the derivation of an ML performance estimate for SCCCs would be very similar to the development given in Section 7.1.4

for the PCCCs. Thus, from Section 7.1.4 we may immediately write

$$P_b \leq \sum_{k=1}^{2^K-1} \frac{w_k}{K} Q\left(\sqrt{\frac{2Rd_k E_b}{N_0}}\right)$$
$$= \sum_{w=1}^{K} \sum_{v=1}^{\binom{K}{w}} \frac{w}{K} Q\left(\sqrt{\frac{2Rd_{wv} E_b}{N_0}}\right)$$

and examine the impact of input weights $w = 1, 2, 3 \ldots$ on the SCCC encoder output weight. As before, weight-1 encoder inputs have negligible impact on performance because at least one of the encoders is recursive. Also, as before weight-2, weight-3, and weight-4 inputs can be problematic. Thus, we may write

$$P_b \simeq \sum_{w \geq 2} \frac{w n_w}{K} Q\left(\sqrt{\frac{2Rd_{w,\min}^{\text{SCCC}} E_b}{N_0}}\right)$$
$$\simeq \max_{w \geq 2} \left\{ \frac{w n_w}{K} Q\left(\sqrt{\frac{2Rd_{w,\min}^{\text{SCCC}} E_b}{N_0}}\right) \right\}, \tag{7.28}$$

where n_w and $d_{w,\min}^{\text{SCCC}}$ are functions of the particular interleaver employed. Analogous to $d_{w,\min}^{\text{PCCC}}$, $d_{w,\min}^{\text{SCCC}}$ is the minimum SCCC codeword weight among weight-w inputs. Again, there exists an interleaver gain because wn_w/K tends to be much less than unity (for sufficiently large K). An interleaver gain of the form wn_w/K^f for some integer $f > 1$ is often quoted for SCCC codes, but this form arises in the average performance analysis for an ensemble of SCCC codes. The ensemble result for large K and $f > 1$ implies that, among the codes in the ensemble, a given SCCC is unlikely to have the worst-case performance (which depends on the worst-case $d_{w,\min}^{\text{SCCC}}$). This is discussed in detail in Chapter 8.

When codeword error rate, $P_{cw} = \Pr\{\text{decoder output contains one or more residual errors}\}$, is the preferred performance metric, the result is

$$P_{cw} \simeq \sum_{w \geq 2} n_w Q\left(\sqrt{\frac{2Rd_{w,\min}^{\text{SCCC}} E_b}{N_0}}\right)$$
$$\simeq \max_{w \geq 2} \left\{ n_w Q\left(\sqrt{\frac{2Rd_{w,\min}^{\text{SCCC}} E_b}{N_0}}\right) \right\}. \tag{7.29}$$

Example 7.4. We consider in this example a PCCC and an SCCC, both rate 8/9 with parameters $(N,K) = (1152,1024)$. The PCCC encoder uses two identical four-state RSC encoders whose generator polynomials are $(\mathbf{g}_{\text{octal}}^{(1)}, \mathbf{g}_{\text{octal}}^{(2)}) = (7,5)$. To achieve a code rate of 8/9, only one bit is saved in every 16-bit block of parity bits at each RSC encoder

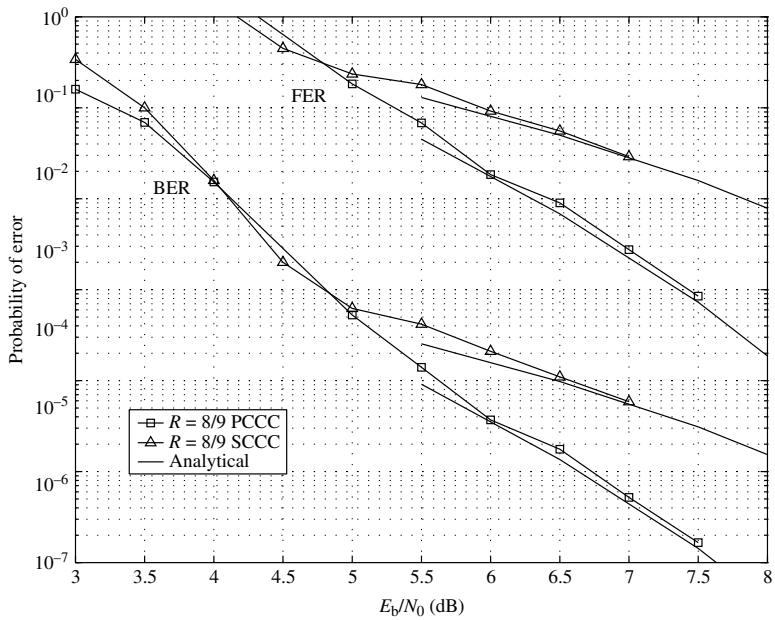


Figure 7.7 PCCC and SCCC bit-error-rate (BER) and frame-error-rate (FER) simulation results for rate-8/9 (1152,1024) codes together with analytical results in (7.4) and (7.28).

output. As for the SCCC encoder, the outer constituent encoder is the same four-state RSC encoder, and the inner code is a rate-1 differential encoder with transfer function $1/(1 \oplus D)$. A rate of 8/9 is achieved in this case by saving one bit in every 8-bit block of parity bits at the RSC encoder output. The PCCC interleaver is a 1024-bit pseudo-random interleaver with no constraints added (e.g., no S-random constraint). The SCCC interleaver is a 1152-bit pseudo-random interleaver with no constraints added. Figure 7.7 presents simulated performance results for these codes using iterative decoders (the SCCC iterative decoder will be discussed in the next section). Simulation results both for bit error rate P_b (BER in the figure) and for the frame or codeword error rate P_{cw} (FER in the figure) are presented. Also included in Figure 7.7 are analytic performance estimates for ML decoding using (7.4), (7.5), (7.28), and (7.29). We can see the close agreement between the analytical and simulated results in this figure.

We comment on the fact that the PCCC in Figure 7.7 is substantially better than the SCCC, whereas it is known that SCCCs generally have lower floors [11]. We attribute this to the fact that the outer RSC code in the SCCC has been punctured so severely that $d_{\min} = 2$ for this outer code (although $d_{\min}$ for the SCCC is certainly larger than 2). The RSC encoders for the PCCC are punctured

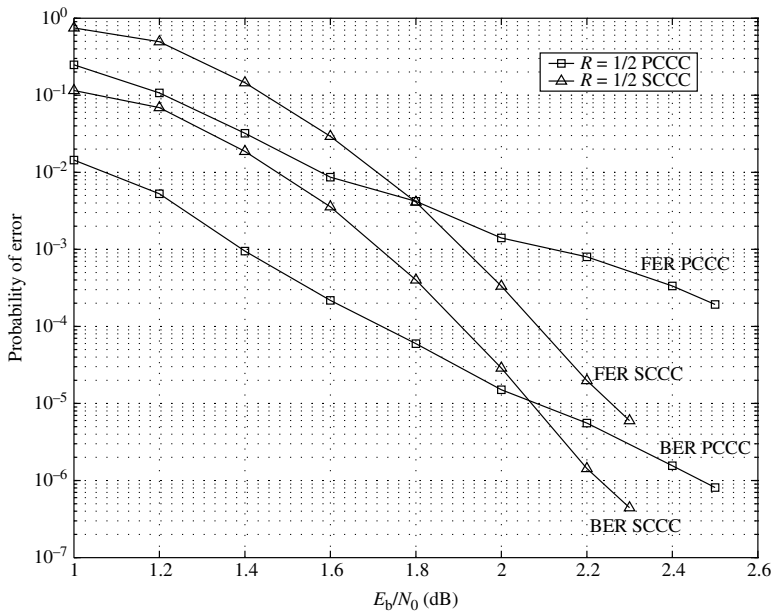


Figure 7.8 PCCC and SCCC bit-error-rate (BER) and frame-error-rate (FER) simulation results for rate-1/2 (2048,1024) codes.

only half as much, so $d_{\min} > 2$ for each of these encoders. We also attribute this to the fact that we have not used an optimized interleaver for this example. In support of these comments, we present the following example.

Example 7.5. We have simulated rate-1/2 versions of the same code structure as in the previous example, but no puncturing occurs for the SCCC and much less occurs for the PCCC. In this case, $(N,K) = (2048,1024)$ and S-random interleavers were used ($S = 16$ for PCCC and $S = 20$ for SCCC). The results are presented in Figure 7.8, where we observe that the SCCC has a much lower error-rate floor, particularly for the FER curves. Finally, we remark that the $w \geq 4$ terms in (7.28) and (7.29) are necessary for an accurate estimate of the floor level of the SCCC case in Figure 7.8.

7.3.2 The SCCC Iterative Decoder

We present in this section the iterative decoder for an SCCC consisting of two constituent rate-1/2 RSC encoders. We assume no puncturing so that the overall code rate is 1/4. Higher code rates are achievable via puncturing and/or by replacing the inner encoder by a rate-1 differential encoder with transfer function $1/(1 \oplus D)$.

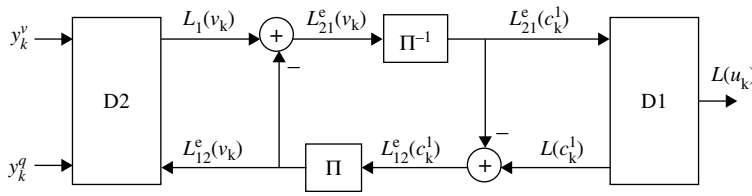


Figure 7.9 The SCCC iterative decoder.

It is straightforward to derive the iterative decoding algorithm for other SCCC codes from the special case that we consider here.

A block diagram of the SCCC iterative decoder with component SISO decoders is presented in Figure 7.9. We denote by $\mathbf{c}_1 = [c_1^1, c_2^1, \ldots, c_{2K}^1] = [u_1, p_1, u_2, p_2, \ldots, u_K, p_K]$ the codeword produced by E1 whose input is $\mathbf{u} = [u_1, u_2, \ldots, u_K]$. We denote by $\mathbf{c}_2 = [c_1^2, c_2^2, \ldots, c_{2K}^2] = [v_1, q_1, v_2, q_2, \ldots, v_{2K}, q_{2K}]$ (with $c_k^2 \triangleq [v_k, q_k]$) the codeword produced by E2 whose input $\mathbf{v} = [v_1, v_2, \ldots, v_{2K}]$ is the interleaved version of $\mathbf{c}_1$, that is, $\mathbf{v} = \mathbf{c}_1'$. As indicated in Figure 7.6, the transmitted codeword $\mathbf{c}$ is the codeword $\mathbf{c}_2$. The received word $\mathbf{y} = \mathbf{c} + \mathbf{n}$ will have the form $\mathbf{y} = [y_1, y_2, \ldots, y_{2K}] = [y_1^v, y_1^q, \ldots, y_{2K}^v, y_{2K}^q]$, where $y_k \triangleq [y_k^v, y_k^q]$.

The iterative SCCC decoder in Figure 7.9 employs two SISO decoding modules. Note that, unlike the PCCC case which focuses on the systematic bits, these SISO decoders share extrinsic information on the E1 code bits $\{c_k^1\}$ (equivalently, on the E2 input bits $\{v_k\}$) in accordance with the fact that these are the bits known to both encoders. A consequence of this is that D1 must provide likelihood information on E1 *output* bits, whereas D2 produces likelihood information on E2 *input* bits, as indicated in Figure 7.9. Further, because LLRs must be obtained on the original data bits u_k so that final decisions may be made, D1 must also compute likelihood information on E1 input bits. Note also that, because E1 feeds no bits directly to the channel, D1 receives no samples directly from the channel. Instead, the only input to D1 is the extrinsic information it receives from D2.

In light of the foregoing discussion, the SISO module D1 requires two features that we have not discussed when discussing the BCJR algorithm or the PCCC decoder. The first feature is the requirement for likelihood information on a constituent encoder's input *and* output. In prior discussions, such LLR values were sought only for a constituent encoder's inputs (the u_ks). Since we assume the SCCC constituent codes are systematic, computing LLRs on an encoder's output bits gives LLRs for both input and output bits.

If we retrace the development of the BCJR algorithm, it will become clear that the LLR for an encoder's output c_k can be computed by modifying the sets in (7.10) over which the max* operations are taken. For a given constituent code, let C^+ equal the set of trellis transitions at time k for which $c_k = +1$ and let C^- equal

the set of trellis transitions at time k for which $c_k = -1$. Then, with all other steps in the BCJR algorithm the same, the LLR $L(c_k)$ can be computed as

$$L(c_k) = \max_{C^+}{}^* \left[\tilde{\alpha}_{k-1}(s') + \tilde{\gamma}_k(s', s) + \tilde{\beta}_k(s) \right] - \max_{C^-}{}^* \left[\tilde{\alpha}_{k-1}(s') + \tilde{\gamma}_k(s', s) + \tilde{\beta}_k(s) \right]. \tag{7.30}$$

We note also that a trellis-based BCJR/SISO decoder is generally capable of decoding either the encoder's input or its output, irrespective of whether the code is systematic. This is evident since the trellis branches are labeled both by inputs and by outputs, and again one need only perform the max* operation over the appropriate sets of trellis transitions.

The second feature required by the SCCC iterative decoder that was not required by the PCCC decoder is that constituent decoder D1 has only extrinsic information as input. In this case the branch metric (7.16) is simply modified as

$$\tilde{\gamma}_k(s', s) = u_k L_{21}^{\mathrm{e}}(u_k)/2 + p_k L_{21}^{\mathrm{e}}(p_k)/2. \tag{7.31}$$

Other than these modifications, the iterative SCCC decoder proceeds much like the PCCC iterative decoder and as indicated in Figure 7.9.

7.3.3 Summary of the SCCC Iterative Decoder

Essentially all of the comments mentioned for the PCCC decoder hold also for the SCCC, so we do not repeat them. The only difference is that the decoding order is D2 → D1 → D2 → D1 → $\cdots$. We first present an outline of the SCCC decoding algorithm, and then we present the algorithm in detail.

Outline

1. Initialize all state metrics appropriately and set all extrinsic information to zero.
2. D2 decoder: Run the SISO BJCR algorithm with inputs $y_k = [y_k^v, y_k^q]$ and $L_{12}^{\mathrm{e}}(v_k)$ to obtain $L_{21}^{\mathrm{e}}(v_k)$. Send extrinsic information $L_{21}^{\mathrm{e}}(v_k)$ to the D1 decoder.
3. D1 decoder: Run the SISO BJCR algorithm with inputs $L_{21}^{\mathrm{e}}(v_k)$ to obtain $L_{12}^{\mathrm{e}}(v_k)$. Send extrinsic information $L_{12}^{\mathrm{e}}(v_k)$ to the D2 decoder.
4. Repeat Steps 2 and 3 until the preset maximum number of iterations is reached (or some other stopping criterion is satisfied). Make decisions on bits according to sign$[L(u_k)]$, where $L(u_k)$ is computed by D1.

Algorithm 7.3 SCCC Iterative Decoder

Initialization

D1:

$$\tilde{\alpha}_0^{(1)}(s) = 0 \text{ for } s = 0$$
$$= -\infty \text{ for } s \neq 0$$
$$\tilde{\beta}_K^{(1)}(s) = 0 \text{ for } s = 0$$
$$= -\infty \text{ for } s \neq 0$$

$L_{21}^{\mathrm{e}}(c_k^1)$ is to be determined from D2 after the first half-iteration and so need not be initialized

D2:

$$\tilde{\alpha}_0^{(2)}(s) = 0 \text{ for } s = 0$$
$$= -\infty \text{ for } s \neq 0$$

$\tilde{\beta}_{2K}^{(2)}(s) = \tilde{\alpha}_{2K}^{(2)}(s)$ for all s (set after computation of $\{\tilde{\alpha}_{2K}^{(2)}(s)\}$ in the *first* iteration)

$L_{12}^{\mathrm{e}}(v_k) = 0$ for $k = 1, 2, \ldots, 2K$

The nth iteration

D2:
for $k = 1$ to $2K$

- get $y_k = [y_k^v, y_k^q]$
- compute $\tilde{\gamma}_k(s', s)$ for all allowable state transitions $s' \to s$ from

$$\tilde{\gamma}_k(s', s) = v_k L_{12}^{\mathrm{e}}(v_k)/2 + v_k y_k^v/\sigma^2 + q_k y_k^q/\sigma^2$$

  (v_k (q_k) in the above expression is set to the value of the encoder input (output) corresponding to the transition $s' \to s$; $L_{12}^{\mathrm{e}}(v_k)$ is $L_{12}^{\mathrm{e}}(c_{P[k]}^1)$, the interleaved extrinsic information from the previous D1 iteration)
- compute $\tilde{\alpha}_k^{(2)}(s)$ for all s using (7.8)

end

for $k = 2K$ to 2 step -1

- compute $\tilde{\beta}_{k-1}^{(2)}(s)$ for all s using (7.9)

end

for $k = 1$ to $2K$

- compute $L_{21}^{\mathrm{e}}(v_k)$ using

$$\begin{aligned} L_{21}^{\mathrm{e}}(v_k) = & \max_{V^+}{}^* \left[\tilde{\alpha}_{k-1}^{(2)}(s') + \tilde{\gamma}_k(s', s) + \tilde{\beta}_k^{(2)}(s)\right] \\ & - \max_{V^-}{}^* \left[\tilde{\alpha}_{k-1}^{(2)}(s') + \tilde{\gamma}_k(s', s) + \tilde{\beta}_k^{(2)}(s)\right] - L_{12}^{\mathrm{e}}(v_k) \\ = & \max_{V^+}{}^* \left[\tilde{\alpha}_{k-1}^{(2)}(s') + v_k y_k^v/\sigma^2 + q_k y_k^q/\sigma^2 + \tilde{\beta}_k^{(2)}(s)\right] \\ & - \max_{V^-}{}^* \left[\tilde{\alpha}_{k-1}^{(2)}(s') + v_k y_k^v/\sigma^2 + q_k y_k^q/\sigma^2 + \tilde{\beta}_k^{(2)}(s)\right], \end{aligned}$$

  where V^+ is set of state-transition pairs (s', s) corresponding to the event $v_k = +1$, and V^- is similarly defined.

end

D1:
for $k = 1$ to K

- for all allowable state transitions $s' \to s$ set $\tilde{\gamma}_k(s', s)$ via

$$\begin{aligned} \tilde{\gamma}_k(s', s) &= u_k L_{21}^{\mathrm{e}}(u_k)/2 + p_k L_{21}^{\mathrm{e}}(p_k)/2 \\ &= u_k L_{21}^{\mathrm{e}}(c_{2k-1}^1)/2 + p_k L_{21}^{\mathrm{e}}(c_{2k}^1)/2 \end{aligned}$$

  (u_k (p_k) in the above expression is set to the value of the encoder input (output) corresponding to the transition $s' \to s$; $L_{21}^{\mathrm{e}}(c_{2k-1}^1)$ is $L_{21}^{\mathrm{e}}(v_{Pinv[2k-1]})$, the de-interleaved extrinsic information from the previous D2 iteration, and similarly for $L_{21}^{\mathrm{e}}(c_{2k}^1)$)
- compute $\tilde{\alpha}_k^{(1)}(s)$ for all s using (7.8)

end

for $k = K$ to 2 step -1

- compute $\tilde{\beta}_{k-1}^{(1)}(s)$ for all s using (7.9)

end

for $k = 1$ to K

- compute $L_{12}^{\mathrm{e}}(u_k) = L_{12}^{\mathrm{e}}(c_{2k-1}^1)$ using

$$\begin{aligned} L_{12}^{\mathrm{e}}(u_k) = & \max_{U^+}{}^* \left[\tilde{\alpha}_{k-1}^{(1)}(s') + \tilde{\gamma}_k(s', s) + \tilde{\beta}_k^{(1)}(s)\right] \\ & - \max_{U^-}{}^* \left[\tilde{\alpha}_{k-1}^{(1)}(s') + \tilde{\gamma}_k(s', s) + \tilde{\beta}_k^{(1)}(s)\right] - L_{21}^{\mathrm{e}}(c_{2k-1}^1) \\ = & \max_{U^+}{}^* \left[\tilde{\alpha}_{k-1}^{(1)}(s') + p_k L_{21}^{\mathrm{e}}(p_k)/2 + \tilde{\beta}_k^{(1)}(s)\right] \\ & - \max_{U^-}{}^* \left[\tilde{\alpha}_{k-1}^{(1)}(s') + p_k L_{21}^{\mathrm{e}}(p_k)/2 + \tilde{\beta}_k^{(1)}(s)\right] \end{aligned}$$

- compute $L_{12}^{\mathrm{e}}(p_k) = L_{12}^{\mathrm{e}}(c_{2k}^1)$ using

$$\begin{aligned} L_{12}^{\mathrm{e}}(p_k) = & \max_{P^+}{}^*\left[\tilde{\alpha}_{k-1}^{(1)}(s') + \tilde{\gamma}_k(s',s) + \tilde{\beta}_k^{(1)}(s)\right] \\ & - \max_{P^-}{}^*\left[\tilde{\alpha}_{k-1}^{(1)}(s') + \tilde{\gamma}_k(s',s) + \tilde{\beta}_k^{(1)}(s)\right] - L_{21}^{\mathrm{e}}(c_{2k}^1) \\ = & \max_{P^+}{}^*\left[\tilde{\alpha}_{k-1}^{(1)}(s') + u_k L_{21}^{\mathrm{e}}(u_k)/2 + \tilde{\beta}_k^{(1)}(s)\right] \\ & - \max_{P^-}{}^*\left[\tilde{\alpha}_{k-1}^{(1)}(s') + u_k L_{21}^{\mathrm{e}}(u_k)/2 + \tilde{\beta}_k^{(1)}(s)\right] \end{aligned}$$

end

After the last iteration
for $k = 1$ to K

- for all allowable state transitions $s' \to s$ set $\tilde{\gamma}_k(s',s)$ via

$$\tilde{\gamma}_k(s',s) = u_k L_{21}^{\mathrm{e}}(c_{2k-1}^1)/2 + p_k L_{21}^{\mathrm{e}}(c_{2k}^1)/2$$

- compute $L(u_k)$ using

$$\begin{aligned} L(u_k) = & \max_{U^+}{}^*\left[\tilde{\alpha}_{k-1}^{(1)}(s') + \tilde{\gamma}_k(s',s) + \tilde{\beta}_k^{(1)}(s)\right] \\ & - \max_{U^-}{}^*\left[\tilde{\alpha}_{k-1}^{(1)}(s') + \tilde{\gamma}_k(s',s) + \tilde{\beta}_k^{(1)}(s)\right] \end{aligned}$$

$\hat{u}_k = \mathrm{sign}[L(u_k)]$

end

7.4 Turbo Product Codes

A *turbo product code* (TPC), also called *block turbo code* (BTC) [15], is best explained via the diagram of Figure 7.10 which depicts a two-dimensional codeword. The codeword is an element of a product code (see Chapter 3), where the first constituent code has parameters (n_1,k_1) and the second constituent code has parameters (n_2,k_2). The $k_1 \times k_2$ submatrix in Figure 7.10 contains the k_1k_2-bit data word. The columns of this submatrix are encoded by the "column code," after which the rows of the resulting $n_1 \times k_2$ matrix are encoded by the "row code." Alternatively, row encoding may occur first, followed by column encoding. Because the codes are linear, the resulting codeword is independent of the encoding order. In particular, the "checks-on-checks" submatrix will be unchanged.

The aggregate code rate for this product code is $R = R_1R_2 = (k_1k_2)/(n_1n_2)$, where R_1 and R_2 are the code rates of the individual codes. The minimum distance of the product code is $d_{\min} = d_{\min,1}d_{\min,2}$, where $d_{\min,1}$ and $d_{\min,2}$ are the minimum distances of the individual codes. The constituent codes are typically extended

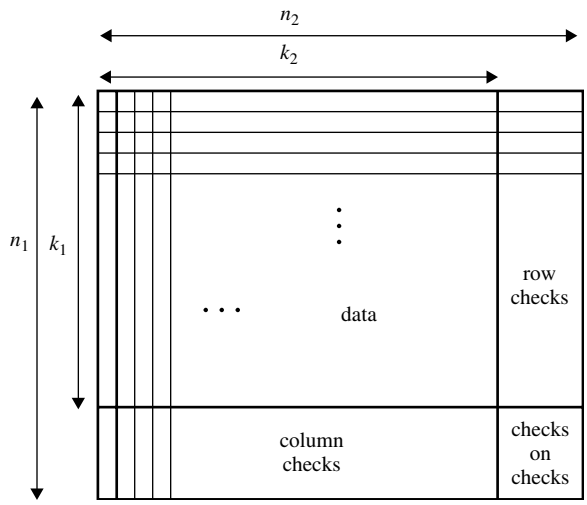


Figure 7.10 The product code of two constituent block codes with parameters (n_1, k_1) and (n_2, k_2).

BCH codes (including extended Hamming codes). The extended BCH codes are particularly advantageous because the extension beyond the nominal BCH code increases the (design) minimum distance of each constituent code by one, at the expense of only one extra parity bit, while achieving an increase in aggregate minimum distance by $d_{\min,1} + d_{\min,2} + 1$.

Example 7.6. Let the constituent block codes be two (15,11) Hamming codes, for which $d_{\min,1} = d_{\min,2} = 3$. The minimum distance of the product code is then $d_{\min} = 9$. For comparison, let the constituent block codes be two (16,11) Hamming codes, for which $d_{\min,1} = d_{\min,2} = 4$. The minimum distance of the resulting TPC is then $d_{\min} = 16$. In the first case the product code is a rate-0.538 (225,121) code and in the second case the product code is a rate-0.473 (256,121) code.

We remark that the P_b and P_{cw} expressions for ML decoding performance of a TPC on the AWGN channel are no different from those of any other code:

$$P_b \sim \frac{w_{\min}}{k_1 k_2} Q\left(\sqrt{\frac{2Rd_{\min}E_b}{N_0}}\right),$$

$$P_{cw} \sim A_{\min} Q\left(\sqrt{\frac{2Rd_{\min}E_b}{N_0}}\right),$$

where $w_{\min}$ is the total information weight corresponding to all of the $A_{\min}$ TPC codewords at the minimum distance $d_{\min}$. We note that, unlike for PCCCs and SCCCs, $w_{\min}$ and $A_{\min}$ are quite large for TPCs. That is, there exists no spectral thinning because the interleaver is deterministic and the constituent codes are not recursive.

7.4.1 Turbo Decoding of Product Codes

Product codes were invented (in 1954 [16]) long before turbo product codes, but the qualifier "turbo" refers to the iterative decoder which comprises two SISO constituent decoders [15]. Such a turbo decoder is easy to derive if we recast a product code as a serial concatenation of block codes. Under this formulation, the codes in Figure 7.6 are block codes and the interleaver is a deterministic "column–row" interleaver. A column–row interleaver can be represented as a rectangular array whereby the array is written column-wise and the bits are read out row-wise. The corresponding iterative (turbo) decoder is that of Figure 7.9. The challenge then would be to design the constituent SISO block decoders necessary for the iterative decoder. One obvious approach would be constituent BCJR decoders based on the BCJR trellises of each block code. However, except for very short codes, this approach leads to a high-complexity decoder because the maximum number of states in the time-varying BCJR trellis of an (n,k) block code is 2^{n-k}. An alternative approach is to use constituent soft-output Viterbi decoders instead of BCJR decoders. Yet another involves a SISO Chase decoder [15], which has substantially lower implementation complexity, at the expense of some performance loss.

We shall now describe the turbo-product-code decoder based on the SISO Chase decoder. As before, we assume knowledge of the turbo principle and turbo decoding, as in Figure 7.9, so our main focus need only be on the constituent soft-output Chase decoders. We will put in details of the turbo product decoder later. Each SISO Chase decoder performs the following steps, which we will discuss in the following.

1. Produce a list of candidate codewords.
2. From that list, make a decision on which codeword was transmitted.
3. For each code bit in the selected codeword, compute soft outputs and identify extrinsic information.

7.4.1.1 Obtaining the List of Candidate Codewords

Let $\mathbf{y} = \mathbf{c} + \mathbf{n}$ be the received word, where the components of the transmitted codeword $\mathbf{c}$ take values in the set $\{\pm 1\}$ and $\mathbf{n}$ is a noise word whose components are AWGN samples. Next, make hard decisions h_k on the elements y_k of $\mathbf{y}$ according to $h_k = \text{sign}(y_k)$. Next, we rely on the fact that, at least for high-rate codes for which the SNR is relatively high, the transmitted word $\mathbf{c}$ is likely to be within a radius $\delta - 1$ of the hard-decision word $\mathbf{h}$, where δ is the constituent code's minimum distance. To find such a list of candidate words, we require the reliabilities of each

sample y_k. We assign reliability r_k to sample y_k according to

$$r_k = \left| \ln\left(\frac{\Pr(y_k|c_k = +1)}{\Pr(y_k|c_k = -1)} \right) \right| .$$

The list $\mathcal{L}$ of candidate codewords is then constructed as follows.

1. Determine the positions of the $p = \lfloor \delta/2 \rfloor$ least reliable elements of $\mathbf{y}$ (and hence of the decisions in $\mathbf{h}$).
2. Form $2^p - 1$ test words obtained from $\mathbf{h}$ by adding a single 1 among the p least reliable positions, then two 1s among the p positions, ..., and finally p 1s among the p positions.
3. Decode the test words using an algebraic decoder (which has correction capability $\lfloor (\delta - 1)/2 \rfloor$). The candidate list $\mathcal{L}$ is the set of codewords at the decoder output.

7.4.1.2 *The Codeword Decision*

For an arbitrary code $\mathcal{C}$, the optimum (ML) decision on the binary-input AWGN channel is given by

$$\hat{\mathbf{c}}_{\mathrm{ML}} = \arg\min_{\mathbf{c} \in \mathcal{C}} \|\mathbf{y} - \mathbf{c}\|^2 .$$

Unless the code is described by a relatively low-complexity trellis, performing this operation requires unacceptable complexity. The utility of the Chase decoder is that the minimization is applied over a smaller set of codewords, those in $\mathcal{L}$:

$$\hat{\mathbf{c}} = \arg\min_{\mathbf{c} \in \mathcal{L}} \|\mathbf{y} - \mathbf{c}\|^2 .$$

Observe that, since the test words add at most $\lfloor \delta/2 \rfloor$ 1s to $\mathbf{h}$, and since the algebraic decoder adds at most $\lfloor (\delta - 1)/2 \rfloor$ 1s to a test pattern, the members of the list $\mathcal{L}$ are approximately within a radius $\delta - 1$ of $\mathbf{h}$.

7.4.1.3 *Computing Soft Outputs and Extrinsic Information*

Given the codeword decision $\hat{\mathbf{c}}$, we now must obtain reliability values for each of the bits in $\hat{\mathbf{c}}$. To do this, we start with the standard log-APP ratio involving the transmitted bits c_k,

$$\ln\left(\frac{\Pr(c_k = +1|\mathbf{y})}{\Pr(c_k = -1|\mathbf{y})} \right),$$

and then we add conditioning on the neighborhood of $\hat{\mathbf{c}}$, to obtain the log-APP ratio:

$$L(c_k) = \ln\left(\frac{\Pr(c_k = +1|\mathbf{y}, \mathcal{L})}{\Pr(c_k = -1|\mathbf{y}, \mathcal{L})} \right) . \tag{7.32}$$

Now let $\mathcal{L}_k^+$ represent the set of codewords in $\mathcal{L}$ for which $c_k = +1$ and let $\mathcal{L}_k^-$ represent the set of codewords in $\mathcal{L}$ for which $c_k = -1$ (recalling the correspondence

$0 \leftrightarrow +1$ and $1 \leftrightarrow -1$). Then, after applying Bayes' rule to (7.32) (under a uniform-codeword-distribution assumption), we may write

$$L(c_k) = \ln\left(\frac{\sum_{\mathbf{c}\in\mathcal{L}_k^+} p(\mathbf{y}|\mathbf{c})}{\sum_{\mathbf{c}\in\mathcal{L}_k^-} p(\mathbf{y}|\mathbf{c})}\right), \tag{7.33}$$

where

$$p\left(\mathbf{y}|\mathbf{c}\right) = \left(\frac{1}{\sqrt{2\pi}\sigma}\right)^n \exp\left(-\frac{\|\mathbf{y}-\mathbf{c}\|^2}{2\sigma^2}\right) \tag{7.34}$$

and σ^2 is the variance of each AWGN sample. Now, ignoring the scale factor in (7.34), (7.33) contains two logarithms of sums of exponentials, a form we have found convenient to express using the max* function. Thus, (7.33) becomes

$$L(c_k) = \max_{\mathbf{c}\in\mathcal{L}_k^+}{}^* \left(-\|\mathbf{y}-\mathbf{c}\|^2/(2\sigma^2)\right) - \max_{\mathbf{c}\in\mathcal{L}_k^-}{}^* \left(-\|\mathbf{y}-\mathbf{c}\|^2/(2\sigma^2)\right). \tag{7.35}$$

The preceding expression may be used to compute the reliabilities $|L(c_k)|$, but a simplified, approximate expression is possible that typically leads to a small performance loss. This is done by replacing the max* functions in (7.35) by max functions so that

$$\begin{aligned}\tilde{L}(c_k) &= \frac{1}{2\sigma^2}\left(\left\|\mathbf{y}-\mathbf{c}_k^-\right\|^2 - \left\|\mathbf{y}-\mathbf{c}_k^+\right\|^2\right)\\ &= \frac{1}{\sigma^2}\mathbf{y}\cdot\left(\mathbf{c}_k^+ - \mathbf{c}_k^-\right),\end{aligned} \tag{7.36}$$

where $\mathbf{c}_k^+$ is the codeword in $\mathcal{L}_k^+$ that is closest to $\mathbf{y}$ and $\mathbf{c}_k^-$ is the codeword in $\mathcal{L}_k^-$ that is closest to $\mathbf{y}$. Clearly, the decision $\hat{\mathbf{c}}$ must either be $\mathbf{c}_k^+$ or $\mathbf{c}_k^-$, so we must find its counterpart, which we will denote by $\mathbf{c}'$. Given this, we may write (7.36) as

$$\tilde{L}(c_k) = \frac{1}{\sigma^2}\mathbf{y}\cdot(\hat{\mathbf{c}}-\mathbf{c}')\,\hat{c}_k \tag{7.37}$$

because $\hat{\mathbf{c}} = \mathbf{c}_k^+$ when $\hat{c}_k = +1$ and $\hat{\mathbf{c}} = \mathbf{c}_k^-$ when $\hat{c}_k = -1$ (compare (7.37) with (7.36)).

At this point, it is not clear where extrinsic information arises. To clarify this we expand (7.37) as

$$\begin{aligned}\tilde{L}(c_k) &= \frac{1}{\sigma^2}\left(2y_k\hat{c}_k + \sum_{s\neq k} y_s\left(\hat{c}_s - c_s'\right)\right)\hat{c}_k\\ &= \frac{2y_k}{\sigma^2} + \frac{\hat{c}_k}{\sigma^2}\left(\sum_{s\neq k} y_s\hat{c}_s - \sum_{s\neq k} y_s c_s'\right).\end{aligned} \tag{7.38}$$

Observe that $\tilde{L}(c_k)$ has a form very similar to (7.19) in that it includes a channel likelihood information term $\left(2y_k/\sigma^2\right)$ and an extrinsic information term (but not yet an extrinsic information term from the SISO decoder counterpart). We call the second term extrinsic information because it first correlates the received word $\mathbf{y}$, exclusive of y_k, with the words $\hat{\mathbf{c}}$ and $\mathbf{c}'$, exclusive of $\hat{c}_k$ and c'_k, respectively. Then the correlation difference is used to bias $\tilde{L}(c_k)$ in the direction of $\hat{c}_k$ if the $\hat{\mathbf{c}}\backslash\{\hat{c}_k\}$ correlation is larger, or in the opposite direction of $\hat{c}_k$ if the $\mathbf{c}'\backslash\{c'_k\}$ correlation is larger.

It is important to point out that $\mathbf{c}'$, the counterpart to $\hat{\mathbf{c}}$, might not exist. Recall that $\hat{\mathbf{c}} \in \{\mathbf{c}_k^+, \mathbf{c}_k^-\}$ and $\mathbf{c}' = \{\mathbf{c}_k^+, \mathbf{c}_k^-\}\backslash\{\hat{\mathbf{c}}\}$ only if $\{\mathbf{c}_k^+, \mathbf{c}_k^-\}\backslash\{\hat{\mathbf{c}}\}$ exists in $\mathcal{L}$. If $\{\mathbf{c}_k^+, \mathbf{c}_k^-\}\backslash\{\hat{\mathbf{c}}\} \notin \mathcal{L}$, then one might increase the Chase algorithm parameter p to increase the size of $\mathcal{L}$, but this would be at the expense of increased complexity. A very effective low-complexity alternative is to let

$$\tilde{L}(c_k) = \hat{c}_k \rho \tag{7.39}$$

for some constant ρ that can be optimized experimentally. In the context of iterative decoding, ρ changes with each iteration.

7.4.1.4 *The Turbo Decoder*

Given the above details on the SISO constituent Chase algorithm block decoders, we are now ready to present the turbo-product-code decoder. We first point out that the scale factor $2/\sigma^2$ in (7.38) is irrelevant so that we may instead use

$$\tilde{L}(c_k) = y_k + L_k^{\mathrm{e}},$$

where L^{e} is the extrinsic information term given by

$$L_k^{\mathrm{e}} = 0.5\hat{c}_k \left(\sum_{s \neq k} y_s \hat{c}_s - \sum_{s \neq k} y_s c'_s \right). \tag{7.40}$$

Further, we point out that the extrinsic term may be thought of as the difference between the LLR and the sample y_k. So in the event that $\mathbf{c}'$ does not exist and (7.39) must be used, the extrinsic information may be computed as the difference $L_k^{\mathrm{e}} = \tilde{L}(c_k) - y_k = \hat{c}_k \rho - y_k$.

The decoder is depicted in Figure 7.11, which we note is very similar to the PCCC and SCCC decoders in Figures 7.3 and 7.9. Moreover, the update equations for each decoder are very familiar. That is, at the lth iteration, the row decoder computes for each c_k

$$L_{\mathrm{row}}(c_k) = y_k + a_l L_{rc,k}^{\mathrm{e}} + a_l L_{cr,k}^{\mathrm{e}} \tag{7.41}$$

and the column decoder computes

$$L_{\mathrm{col}}(c_k) = y_k + a_l L_{cr,k}^{\mathrm{e}} + a_l L_{rc,k}^{\mathrm{e}}. \tag{7.42}$$

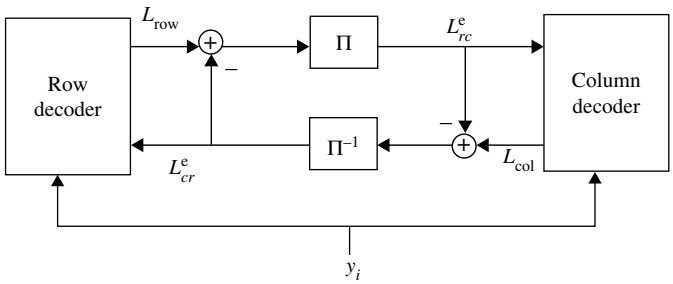


Figure 7.11 The TPC iterative decoder.

In (7.41), $L^{\text{e}}_{rc,k}$ is the extrinsic information computed by the row decoder per Equation (7.40) to be passed on to the column decoder. $L^{\text{e}}_{cr,k}$ is the extrinsic information received from the column decoder. Similarly, in (7.42), $L^{\text{e}}_{cr,k}$ is the extrinsic information computed by the column decoder to be passed on to the row decoder, and $L^{\text{e}}_{rc,k}$ is the extrinsic information received from the row decoder. The scale factors $a_l \in [0, 1]$ are chosen to attenuate the extrinsic LLRs, a necessity that follows from the approximations we have made in deriving the decoder. As is the case for the min-sum and SOVA decoders, the extrinsic magnitudes are too large on average relative to the true magnitudes, so attenuation improves performance.

Not shown in Figure 7.11 are the Chase decoders within the row and column decoders, which are necessary in view of (7.40). Focusing first on the row decoder, in the first iteration the row Chase decoder takes as inputs $\{y_k\}$. In subsequent iterations, the row Chase decoder inputs are $\{y_k + a_l L^{\text{e}}_{cr,k}\}$, that is, the sum of the input from the channel and the extrinsic information from the column decoder. Similarly, in the first iteration the column Chase decoder has inputs $\{y_k\}$ and in subsequent iterations its inputs are $\{y_k + a_l L^{\text{e}}_{rc,k}\}$.

In summary, the TPC decoder performs the following steps (assuming that the row decoder decodes first).

1. Initialize $L^{\text{e}}_{cr,k} = L^{\text{e}}_{rc,k} = 0$ for all k.
2. Row decoder: Run the SISO Chase algorithm with inputs $y_k + a_l L^{\text{e}}_{cr,k}$ to obtain $\{L_{\text{row}}(c_k)\}$ and $\{L^{\text{e}}_{rc,k}\}$. Send extrinsic information $\{L^{\text{e}}_{rc,k}\}$ to the column decoder.
3. Column decoder: Run the SISO Chase algorithm with inputs $y_k + a_l L^{\text{e}}_{rc,k}$ to obtain $\{L_{\text{col}}(c_k)\}$ and $\{L^{\text{e}}_{cr,k}\}$. Send extrinsic information $\{L^{\text{e}}_{cr,k}\}$ to the row decoder.
4. Repeat Steps 2 and 3 until the preset maximum number of iterations is reached or some other stopping criterion is satisfied. Make decisions on bits according to $\text{sign}[L_{\text{row}}(c_k)]$ or $\text{sign}[L_{\text{col}}(c_k)]$, where $L_{\text{row}}(c_k)$ and $L_{\text{col}}(c_k)$ are given by (7.41) and (7.42).

Problems

7.1 Consider an RSC encoder with octal generators (7,5). Find the output of the weight-1 input sequence $1000\ldots$ and note that it is of infinite length. (The length is measured by the location of the last '1,' or the degree-plus-1 of the polynomial representation of the sequence.) Find the output of the weight-2 input sequence $1001000\ldots$ and note that it is of finite length. Why does the weight-2 sequence $10001000\ldots$ produce an infinite-length sequence?

7.2 The chapter describes the PCCC decoder for the BI-AWGNC. How would the decoder be modified for the binary erasure channel? How would it be modified for the binary symmetric channel? Provide all of the details necessary for one to be able to write decoder-simulation programs for these channels.

7.3 Consider a rate-1/3 PCCC whose constituent encoders are both RSC codes with octal generators (7,5). Let the interleaver be specified by the linear congruential relationship $P[k] = (13P[k-1] + 7) \bmod 32$ for $k = 1, 2, \ldots, 31$ and $P[0] = 11$. Thus, $P[0], P[1], \ldots, P[31]$ is $11, 22, \ldots, 20$. This means that the first bit out of the interleaver will be the 11th bit in, and the last bit out of the interleaver will be the 20th bit in. Find the codeword corresponding to the input sequences $1001000\ldots0$ (32 bits) and $01001000\ldots0$ (32 bits). Notice that, unlike its constituent codes, the PCCC is not time-invariant. That is, the codeword for the second input sequence is not a shift of the codeword for the first input sequence.

7.4 Consider the rate-1/3 PCCC of the previous problem, except now increase the interleaver length to 128 so that $P[k] = (13P[k-1] + 7) \bmod 128$ for $k = 1, 2, \ldots, 127$ and $P[0] = 11$. By computer search, find $d_{2,\min}^{\text{PCCC}}$, the minimum codeword weight corresponding to weight-2 PCCC encoder inputs, and n_2, the number of such codewords. Repeat for $d_{3,\min}^{\text{PCCC}}$, the minimum codeword weight corresponding to weight-3 PCCC encoder inputs, and n_3, the number of such codewords. Which is dominant, the codewords due to weight-2 inputs or those due to weight-3 inputs? Can you improve any of these four parameters ($d_{2,\min}^{PCCC}$, n_2, $d_{3,\min}^{\text{PCCC}}$, and n_3) with an improved interleaver design? Consider an S-random interleaver.

7.5 Do the previous problem, except use constituent RSC encoders with octal generators (5,7). Note that, for the (7,5) RSC code, the generator polynomial $g^{(1)}(D) = 1 + D + D^2$ is primitive, whereas for the (5,7) RSC code $g^{(1)}(D) = 1 + D^2$ is not primitive. This affects $d_{2,\min}^{\text{PCCC}}$. See the discussion in Chapter 8 regarding these two PCCCs.

7.6 Consider a *dicode* partial-response channel with transfer function $1 - D$ and AWGN. Thus, for inputs $u_k \in \{\pm 1\}$, the outputs of this channel are given by $r_k = c_k + n_k$, where $c_k = u_k - u_{k-1} \in \{0, \pm 2\}$ and the noise samples n_k are distributed as $\eta(0, \sigma^2)$. (a) Draw the two-state trellis for this channel. (b) Suppose the channel is known to start and end in channel state $+1$. Perform the BCJR algorithm by hand to detect the received sequence $(r_1, r_2, r_3, r_4) = (0.5, -1.3, 1.5, 0.2)$. You should find that the detected bits are $(\hat{u}_1, \hat{u}_2, \hat{u}_3, \hat{u}_4) = (+1, -1, +1, -1)$.

7.7 Simulate the rate-1/3 PCCC with RSC octal generators (7,5) on the binary-input AWGN channel. Use the length-128 interleaver given by $P[k] = (13P[k-1]+7) \bmod 128$ for $k = 1, 2, \ldots, 127$ and $P[0] = 11$ (the first bit out is the 11th bit in). You should find that, for $E_b/N_0 = 3, 4$, and 5 dB, the bit error rate P_b is in the vicinity of 3×10^{-5}, 4×10^{-6}, and 5×10^{-7}, respectively. Repeat using an S-random interleaver in an attempt to lower the "floor" of the P_b versus E_b/N_0 curve.

7.8 Do the previous problem using the max function instead of the max* function in the constituent BCJR decoders.

7.9 This chapter describes the SCCC decoder for the BI-AWGNC. How would the decoder be modified for the binary erasure channel? How would it be modified for the binary symmetric channel? Provide all of the details necessary for one to be able to write decoder-simulation programs for these channels.

7.10 Consider a rate-1/2 SCCC whose outer code is the rate-1/2 $\bar{\text{R}}\bar{\text{S}}$C code with octal generators (7,5) and whose inner code is the rate-1 accumulator with transfer function $1/(1 \oplus D)$. Let the interleaver be specified by the linear congruential relationship $P[k] = (13P[k-1]+7) \bmod 32$ for $k = 1, 2, \ldots, 31$ and $P[0] = 11$. Thus, $P[0], P[1], \ldots, P[31]$ is $11, 22, \ldots, 20$. This means that the first bit out of the interleaver will be the 11th bit in, and the last bit out of the interleaver will be the 20th bit in. Find the codeword corresponding to the input sequences $100\ldots0$ (32 bits) and $0100\ldots0$ (32 bits). Notice that, unlike its constituent codes, the SCCC is not time-invariant. That is, the codeword for the second input sequence is not a shift of the codeword for the first input sequence.

7.11 Consider the rate-1/2 SCCC of the previous problem, except now increase the interleaver length to 128 so that $P[k] = (13P[k-1]+7) \bmod 128$ for $k = 1, 2, \ldots, 127$ and $P[0] = 11$. By computer search, find $d_{1,\min}^{\text{SCCC}}$, the minimum codeword weight corresponding to weight-1 PCCC encoder inputs, and n_1, the number of such codewords. Repeat for $d_{2,\min}^{\text{PCCC}}$, the minimum codeword weight corresponding to weight-2 PCCC encoder inputs, and n_2, the number of such codewords. Which is dominant, the codewords due to weight-1 inputs or those due to weight-2 inputs? Can you improve any of these four parameters ($d_{1,\min}^{\text{PCCC}}$, n_1, $d_{2,\min}^{\text{PCCC}}$, and n_2) with an improved interleaver design? Consider an S-random interleaver.

7.12 Do the previous problem, except use an RSC encoder with octal generators (7,5) instead of an $\bar{\text{R}}\bar{\text{S}}$C encoder. Comment on your findings.

7.13 Simulate the rate-1/2 SCCC whose outer code is the rate-1/2 $\bar{\text{R}}\bar{\text{S}}$C code with octal generators (7,5) and whose inner code is the rate-1 accumulator with transfer function $1/(1 \oplus D)$. Use the length-128 interleaver specified by the linear congruential relationship $P[k] = (13P[k-1]+7) \bmod 128$ for $k = 1, 2, \ldots, 127$ and $P[0] = 11$. Repeat your simulation using an S-random interleaver in an attempt to lower the "floor" of the P_b versus E_b/N_0 curve.

7.14 Do the previous problem using the max function instead of the max* function in the constituent BCJR decoders.

7.15 Show that the minimum distance of a product code is given by $d_{\min} = d_{\min,1} d_{\min,2}$, where $d_{\min,1}$ and $d_{\min,2}$ are the minimum distances of the row and column codes.

7.16 Would the turbo-product-code iterative-decoding algorithm described in this chapter work if the row code and the column code were both SPC codes? Explain your answer in appropriate detail. Can the sum–product algorithm used for LDPC codes be used to decode turbo product codes? Explain your answer in appropriate detail.

References

[1] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon limit error-correcting coding and decoding: Turbo codes," *Proc. 1993 Int. Conf. on Communications*, 1993, pp. 1064–1070.

[2] C. Berrou and A. Glavieux, "Near optimum error correcting coding and decoding: turbo-codes," *IEEE Trans. Communications*, pp. 1261–1271, October 1996.

[3] G. Ungerboeck, "Channel coding with multilevel/phase signals," *IEEE Trans. Information Theory*, vol. 28, no. 1, pp. 55–67, January 1982.

[4] P. Robertson, "Illuminating the structure of code and decoder of parallel concatenated recursive systematic (turbo) codes," *Proc. GlobeCom 1994*, 1994, pp. 1298–1303.

[5] S. Benedetto and G. Montorsi, "Unveiling turbo codes: some results on parallel concatenated coding schemes," *IEEE Trans. Information Theory*, vol. 40, no. 3, pp. 409–428, March 1996.

[6] S. Benedetto and G. Montorsi, "Design of parallel concatenated codes," *IEEE Trans. Communications*, pp. 591–600, May 1996.

[7] J. Hagenauer, E. Offer, and L. Papke, "Iterative decoding of binary block and convolutional codes," *IEEE Trans. Information Theory*, vol. 42, no. 3, pp. 429–445, March 1996.

[8] D. Arnold and G. Meyerhans, "The realization of the turbo-coding system," Semester Project Report, ETH Zürich, July 1995.

[9] L. Perez, J. Seghers, and D. Costello, "A distance spectrum interpretation of turbo codes," *IEEE Trans. Information Theory*, vol. 42, no. 11, pp. 1698–1709, November 1996.

[10] L. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate," *IEEE Trans. Information Theory*, vol. 20, no. 3, pp. 284–287, March 1974.

[11] S. Benedetto, D. Divsalar, G. Montorsi, and F. Pollara, "Serial concatenation of interleaved codes: performance analysis, design, and iterative decoding," *IEEE Trans. Information Theory*, vol. 44, no. 5, pp. 909–926, May 1998.

[12] J. Hagenauer and P. Hoeher, "A Viterbi algorithm with soft-decision outputs and its applications," *1989 IEEE Global Telecommunications Conf.*, pp. 1680–1686, November 1989.

[13] P. Robertson, E. Villebrun, and P. Hoeher, "A comparison of optimal and suboptimal MAP decoding algorithms operating in the log domain," *Proc. 1995 Int. Conf. on Communications*, pp. 1009–1013.

[14] A. Viterbi, "An intuitive justification and a simplified implementation of the MAP decoder for convolutional codes," *IEEE J. Selected Areas in Communications*, vol. 18, no. 2, pp. 260–264, February 1998.

[15] R. Pyndiah, "Near-optimum decoding of product codes: block turbo codes," *IEEE Trans. Communications*, vol. 46, no. 8, pp. 1003–1010, August 1998.

[16] P. Elias, "Error-free coding," *IEEE Trans. Information Theory*, vol. 1, no. 9, pp. 29–37, September 1954.

[17] D. Divsalar and F. Pollara, "Multiple turbo codes for deep-space communications," JPL TDA Progress Report, 42-121, May 15, 1995.

[18] O. Acikel and W. Ryan, "Punctured turbo codes for BPSK/QPSK channels," *IEEE Trans. Communications*, vol. 47, no. 9, pp. 1315–1323, September 1999.

[19] O. Acikel and W. Ryan, "Punctured high rate SCCCs for BPSK/QPSK channels," *Proc. 2000 IEEE Int. Conf. on Communications*, pp. 434–439, June 2000.

[20] M.C. Valenti and S. Cheng, "Iterative demodulation and decoding of turbo coded M-ary noncoherent orthogonal modulation," *IEEE J. Selected Areas in Communications (Special Issue on Differential and Noncoherent Wireless Communications)*, vol. 23, no. 9, pp. 1738–1747, September 2005.

[21] C. X. Huang and A. Ghrayeb, "A simple remedy for the exaggerated extrinsic information produced by the SOVA algorithm," *IEEE Trans. Wireless Communications*, vol. 5, no. 5, pp. 996–1002, May 2006.

[22] A. Ghrayeb and C. X. Huang, "Improvements in SOVA-based decoding for turbo-coded storage channels," *IEEE Trans. Magnetics*, vol. 41, no. 12, pp. 4435–4442, December 2005.

[23] J. Pearl, *Probabilistic Reasoning in Intelligent Systems*, San Mateo, CA, Morgan Kaufmann, 1988.

[24] B. Frey, *Graphical Models for Machine Learning and Digital Communication*, Cambridge, MA, MIT Press, 1998.

[25] N. Wiberg, *Codes and Decoding on General Graphs*, Ph.D. dissertation, University of Linköping, Sweden, 1996.

8 Ensemble Enumerators for Turbo and LDPC Codes

Weight enumerators or *weight-enumerating functions* are polynomials that represent in a compact way the input and/or output weight characteristics of the encoder for a code. The utility of weight enumerators is that they allow us to easily estimate, via the union bound, the performance of a maximum-likelihood (ML) decoder for the code. Given that turbo and LDPC codes employ suboptimal iterative decoders this may appear meaningless, but it is actually quite sensible for at least two reasons. One reason is that knowledge of ML-decoder performance bounds allows us to weed out weak codes. That is, if a code performs poorly for the ML decoder, we can expect it to perform poorly for an iterative decoder. Another reason for the ML-decoder approach is that the performance of an iterative decoder is generally approximately equal to that of its counterpart ML decoder, at least over a restricted range of SNRs. We saw this in Chapter 7 and we will see it again in Figure 8.5 in this chapter.

Another drawback to the union-bound/ML-decoder approach is that the bound diverges in the low-SNR region, which is precisely the region of interest when one is attempting to design codes that perform very close to the capacity limit. Thus, when attempting to design codes that are simultaneously effective in the floor region (high SNRs) and the waterfall region (low SNRs), the techniques introduced in this chapter should be supplemented with the techniques in the next chapter which are applicable to the low-SNR region. We also introduce in this chapter trapping-set and stopping-set enumerators because trapping sets and stopping sets are also responsible for floors in the high-SNR region. The presentation in this chapter follows [1–12].

We emphasize that the weight enumerators derived in this chapter are for code ensembles rather than for a specific code. Thus, the performance estimates correspond to the *average* performance over a code ensemble rather than for a specific code, although the formulas can be applied to a specific code if its weight enumerator is known. The motivation for the ensemble approach is that finding the weight enumerator for a specific code is generally much more difficult than finding an ensemble enumerator. Further, if one finds an ensemble with excellent performance, then one can pick a code at random from the ensemble and expect excellent performance.

8.1 Notation

The weight distribution $\{A_0, A_1, \ldots, A_N\}$ of a length-N linear code $\mathcal{C}$ was first introduced in Chapter 3. When the values $A_1, \ldots, A_N$ are used as coefficients of a degree-n polynomial, the resulting polynomial is called a weight enumerator. Thus, we define the *weight enumerator* (WE) $A(W)$ for an (N, K) linear block code to be

$$A(W) = \sum_{w=1}^{N} A_w W^w,$$

where A_w is the number of codewords of weight w. Note that the weight-0 codeword is not included in the summation. Various weight enumerators were discussed in detail in Chapter 4 when discussing the performance of the ML decoder. In this chapter, we introduce various other enumerators necessary for evaluating the performance of turbo and LDPC codes.

To develop the notation for the various enumerating functions, we use as a running example the cyclic (7,4) Hamming code generated in systematic form by the binary polynomial $g(x) = 1 + x + x^3$. Below we list the codewords in three columns, where the first column lists the weight-0 and the weight-7 codewords, the second column lists the seven weight-3 codewords, and the third column lists the seven weight-4 codewords:

0000 000	1000 110	0010 111
1111 111	0100 011	1001 011
	1010 001	1100 101
	1101 000	1110 010
	0110 100	0111 001
	0011 010	1011 100
	0001 101	0101 110

The WE for the (7,4) Hamming code is, then,

$$A(W) = 7W^3 + 7W^4 + W^7.$$

We also define the *information-parity weight enumerator* (IP-WE) $A(I, P)$ for systematic encoders as

$$A(I, P) = \sum_{i=1}^{K} \sum_{p=0}^{N-K} A_{i,p} I^i P^p,$$

where $A_{i,p}$ is the number of codewords with information-weight i and parity-weight p. (Throughout our discussions of enumerating functions, "transform variables" such as W, I, and P are simply indeterminates.) Again, the zero codeword is excluded. The IP-WE for the Hamming code is

$$A(I, P) = I\big(3P^2 + P^3\big) + I^2\big(3P + 3P^2\big) + I^3(1 + 3P) + I^4 P^3.$$

Thus, there are three codewords with information-weight 1 and parity-weight 2, there is one codeword with information-weight 1 and parity-weight 3, and so on. We will also find it useful to define the *conditional parity-weight enumerator* (C-PWE), written as

$$A_i(P) = \sum_{p=0}^{N-K} A_{i,p} P^p,$$

which enumerates parity weight for a given information-weight i. Thus, for the Hamming code,

$$\begin{aligned} A_1(P) &= 3P^2 + P^3, \\ A_2(P) &= 3P + 3P^2, \\ A_3(P) &= 1 + 3P, \\ A_4(P) &= P^3. \end{aligned}$$

We next define the *input–output weight enumerator* (IO-WE) as follows:

$$A(I, W) = \sum_{i=1}^{K} \sum_{w=1}^{N} A_{i,w} I^i W^w,$$

where $A_{i,w}$ is the number of weight-w codewords produced by weight-i encoder inputs. The IO-WE for the (7,4) Hamming code is

$$A(I, W) = I\big(3W^3 + W^4\big) + I^2\big(3W^3 + 3W^4\big) + I^3\big(W^3 + 3W^4\big) + I^4 W^7.$$

The IO-WE gives rise to the *conditional output-weight enumerator* (C-OWE) expressed as

$$A_i(W) = \sum_{w=1}^{N} A_{i,w} W^w,$$

which enumerates codeword weight for a given information-weight i. For the Hamming code, the following is clear:

$$\begin{aligned} A_1(W) &= 3W^3 + W^4, \\ A_2(W) &= 3W^3 + 3W^4, \\ A_3(W) &= W^3 + 3W^4, \\ A_4(W) &= W^7. \end{aligned}$$

Observe the following straightforward relationships:

$$\begin{aligned} A(W) &= A(I, P)|_{I=P=W} = A(I, W)|_{I=1}, \\ A(I, P) &= \sum_{i=1}^{K} I^i A_i(P). \end{aligned}$$

The above weight-enumerating functions are useful for estimating a code's codeword error rate P_{cw} (sometimes called the frame error rate, FER). When P_b, the bit error rate (BER), is the metric of interest, bit-wise enumerators are appropriate. For information-word length K, the *cumulative information-weight enumerator* (CI-WE) is given by

$$B(W) = \sum_{i=1}^{K} iI^i A_i(P)\Big|_{I=P=W},$$

the *scaled C-PWE* is given by

$$B_i(P) = iA_i(P) = i\sum_{p=0}^{N-K} A_{i,p}P^p,$$

and the *cumulative IP-WE* is given by

$$B(I,P) = \sum_{i=1}^{K} I^i B_i(P) = \sum_{i=1}^{K}\sum_{p=0}^{N-K} B_{i,p}I^iP^p,$$

where $B_{i,p} = iA_{i,p}$. Observe that we may write

$$B(W) = B(I,P)|_{I=P=W} = \sum_{w=1}^{N} B_w W^w,$$

where $B_w = \sum_{i,p:i+p=w} B_{i,p}$ is the total information-weight for the weight-w codewords. For the (7,4) Hamming code, it is easy to show that

$$B(I,P) = I\big(3P^2 + P^3\big) + 2I^2\big(3P + 3P^2\big) + 3I^3(1+3P) + 4I^4P^3$$

and

$$B(W) = B(I,P)|_{I=P=W} = 12W^3 + 16W^4 + 4W^7.$$

Given the WE $A(W)$ for a length-N code, we may upper bound its FER on a binary-input AWGN channel with two-sided power-spectral density $N_0/2$ as

$$P_{cw} \le \sum_{w=1}^{N} A_w Q\Big(\sqrt{2wRE_b/N_0}\Big), \tag{8.1}$$

where

$$Q(\alpha) = \int_{\alpha}^{\infty} \frac{1}{\sqrt{2\pi}} \exp\big(-\lambda^2/2\big)d\lambda,$$

R is the code rate, and E_b/N_0 is the well-known SNR measure. A more compact, albeit looser, bound is given by

$$P_{cw} < A(W)|_{W=\exp(-RE_b/N_0)}, \tag{8.2}$$

which follows from the bound $Q(\alpha) < \exp(-\alpha^2/2)$. Similarly, we can impose an upper bound on the BER as

$$P_b \leq \frac{1}{K} \sum_{w=1}^{N} B_w Q\left(\sqrt{2wRE_b/N_0}\right), \tag{8.3}$$

and the corresponding looser bound is given by

$$P_b < \frac{1}{K} B(W)\Big|_{W=\exp(-RE_b/N_0)} = \frac{1}{K} \sum_{i=1}^{K} iI^i A_i(P)\Big|_{I=P=\exp(-RE_b/N_0)}. \tag{8.4}$$

8.2 Ensemble Enumerators for Parallel-Concatenated Codes

8.2.1 Preliminaries

In the previous section we discussed enumerators for specific codes. While finding such enumerators is possible for relatively short codes, it is not possible for moderate-length or long codes, even with the help of a fast computer. For this reason, code designers typically study the average performance of ensembles of codes and then pick a code from a good ensemble. In this section, we consider enumerators for ensembles of parallel-concatenated codes (PCCs), configured as in Figure 8.1, where the constituent codes can be either block codes or convolutional codes. The ensembles we consider are obtained by fixing the constituent codes and then varying the length-K interleaver over all $K!$ possibilities. It is possible to consider interleaving multiple input blocks using a length-pK interleaver, $p > 1$, but we shall not do so here.

Prior to studying ensemble enumerators, it is helpful to first develop an understanding of how an enumerator for a PCC is related to the corresponding enumerators of its constituent codes. This also further motivates the ensemble approach.

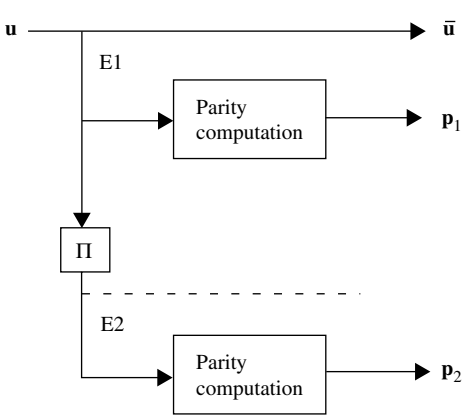


Figure 8.1 An encoder for generic parallel-concatenated code.

Thus, consider the simple PCC whose constituent codes, C_1 and C_2, are both (7,4) Hamming codes and fix the interleaver $\mathbf{\Pi}$ to one of its $4! = 24$ possibilities. Now consider weight-1 encoder inputs. Then the C-PWEs are $A_1^{C_1}(P) = A_1^{C_2}(P) = P^2 + P^2 + P^2 + P^3$, where we have intentionally written four separate terms for the four possible weight-1 inputs. That is, for the four possible weight-1 inputs into either encoder, E1 or E2, three will yield parity-weight-2 outputs and one will yield a parity-weight-3 output. As an example of a C-PWE computation for the PCC, suppose a particular weight-1 input to E1 produces parity-weight-3, producing a P^3 term, and, after the input has passed through the interleaver, E2 produces parity-weight 2, producing a P^2 term. The corresponding term for the PCC is clearly $P^3 \cdot P^2 = P^5$ since the PCC parity-weight for that input will be 5. Note how each of the four terms in $A_1^{C_1}(P)$ pairs with each of the four terms in $A_1^{C_2}(P)$, depending on $\mathbf{\Pi}$, as in the $P^3 \cdot P^2$ calculation. Except for when K is small, keeping track of such pairings is impossible in general. This fact suggests the ensemble approach whereby we compute the average C-PWE for a PCC ensemble, where the average is over all $K!$ interleavers (C_1 and C_2 are fixed).

To see how this is done, note that the set of all length-K interleavers permutes a given weight-i input into all $\binom{K}{i}$ possible permutations with equal probability. Thus, in the present example for which $K = 4$, when averaged over the 4! interleavers, a given term in $A_1^{C_1}(P)$ will be paired with a given term in $A_1^{C_2}(P)$ with probability $1/\binom{4}{1} = 1/4$ in the computation of the C-PWE $A_1^{\mathrm{PCC}}(P)$ for the PCC. Thus, averaged over the ensemble of such parallel-concatenated codes, the average C-PWE is given by

$$\begin{aligned} A_1^{\mathrm{PCC}}(P) &= \left(P^2 + P^2 + P^2 + P^3\right) \cdot \frac{1}{4} \cdot \left(P^2 + P^2 + P^2 + P^3\right) \\ &= \frac{A_1^{C_1}(P) \cdot A_1^{C_2}(P)}{\binom{4}{1}}. \end{aligned}$$

More generally, for arbitrary constituent codes, for a length-K interleaver and weight-i inputs, it can be shown that the ensemble enumerator is

$$A_i^{\mathrm{PCC}}(P) = \frac{A_i^{C_1}(P) \cdot A_i^{C_2}(P)}{\binom{K}{i}}. \tag{8.5}$$

Note that, given the ensemble C-PWE in (8.5), one can determine any of the other ensemble enumerators, including the bit-wise enumerators. For example,

$$A^{\mathrm{PCC}}(I, P) = \sum_{i=1}^{K} I^i A_i^{\mathrm{PCC}}(P)$$

and

$$A^{\mathrm{PCC}}(W) = A^{\mathrm{PCC}}(I, P)\big|_{I=P=W} \cdot$$

We remark that this approach for obtaining ensemble enumerators for parallel concatenated codes has been called the uniform-interleaver approach. A length-K *uniform interleaver* is a probabilistic contrivance that maps a weight-i input binary word into each of its $\binom{K}{i}$ permutations with uniform probability $1/\binom{K}{i}$. Thus, we can repeat the development of (8.5) assuming a uniform interleaver is in play and arrive at the same result.

8.2.2 PCCC Ensemble Enumerators

We now consider ensemble enumerators for parallel-concatenated convolutional codes (PCCCs) in some detail. We will consider the less interesting parallel-concatenated block-code case along the way. We shall assume that the constituent convolutional codes are identical, terminated, and, for convenience, rate 1/2. Termination of C_2 is, of course, problematic in practice, but the results developed closely match practice when K is large, $K \gtrsim 100$, which is generally the case. Our development will follow that of [1, 2, 4].

In view of (8.5), to derive the C-PWE for a PCCC, we require $A_i^{C_1}(P)$ and $A_i^{C_2}(P)$ for the constituent convolutional codes considered as block codes with length-K inputs (including the termination bits). This statement is not as innocent as it might appear. Recall that the standard enumeration technique for a convolutional code (Chapter 4) considers all paths in a split state graph that depart from state 0 at time zero and return to state 0 some time later. The enumerator is obtained by calculating the transfer function for that split-state graph. Because the all-zero code sequence is usually chosen as a reference in such analyses, the paths which leave and then return to state 0 are typically called *error events*. Such paths also correspond to nonzero codewords that are to be included in our enumeration. However, the transfer-function technique for computing enumerators *does not include paths that return to state 0 more than once*, that is, that contain multiple error events. On the other hand, for an accurate computation of the enumerators for the equivalent block codes of convolutional codes C_1 and C_2, we require all paths, including those that contain multiple error events. We need to consider such paths because they are included in the list of nonzero codewords, knowledge of which is necessary in order to compute performance bounds.

Although there is a clear distinction between a convolutional code (with semi-infinite codewords) and its equivalent block code, we point out that the weight enumerator for the former may be used to find the weight enumerator for the latter. This is so because the nonzero codewords of the equivalent block code may be thought of as concatenations of single error events of the convolutional code. Further, error-event enumerators for convolutional codes are easily obtained via the transfer-function technique or computer search. For now, assume that we know the parity-weight enumerator $A_{i,n,\ell}(P)$ conditioned on convolutional encoder input weight i, which results in n concatenated error events whose lengths total ℓ trellis stages (or ℓ input bits). As stated, $A_{i,n,\ell}(P)$ can be derived from the

convolutional-code single-error-event transfer-function-based enumerator, but the counting procedure includes only one form of concatenation of the n convolutional-code error events. An example would be n consecutive error events, the first of which starts at time zero, and their permutations. That is, the counting procedure which yields $A_{i,n,\ell}(P)$ does not include the number of locations within the K-block in which the n error events may occur. Thus, we introduce the notation $N(K,n,\ell)$, which is the number of ways n error events having total length ℓ may be situated in a K-block in a particular order. We may now write for the enumerator for the equivalent block code (EBC)

$$A_i^{\mathrm{EBC}}(P) = \sum_{n=1}^{n_{\max}} \sum_{\ell=1}^{K} N(K,n,\ell) A_{i,n,\ell}(P). \tag{8.6}$$

It can be shown that

$$N(K,n,\ell) = \binom{K-\ell+n}{n},$$

independently of the individual error-event lengths. This follows because $N(K,n,\ell)$ is equivalent to the number of ways $K-\ell$ can be partitioned into numbers that sum to $K-\ell$. (There must be a total of $K-\ell$ zeros in the zero-strings that separate the n error events.) Figure 8.2 provides an example showing that the total number of ways in which the $n=2$ error events of total length $\ell=5$ may be situated in a block of length $K=8$ is

$$\frac{4\cdot 3 + 4\cdot 2}{2!} = \frac{5\cdot 4}{2!} = \binom{8-5+2}{2} = N(8,2,5).$$

We divide by 2!, or by $n!$ in the general case, because $A_{i,n,\ell}(P)$ already accounts for the $n!$ different permutations of the n error events.

Under the assumptions $K \gg \ell$ and $K \gg n$, as is generally the case, we may write

$$N(K,n,\ell) \simeq \binom{K}{n},$$

so that (8.6) becomes

$$A_i^{\mathrm{EBC}}(P) \simeq \sum_{n=1}^{n_{\max}} \binom{K}{n} A_{i,n}(P), \tag{8.7}$$

where

$$A_{i,n}(P) = \sum_{\ell=1}^{K} A_{i,n,\ell}(P). \tag{8.8}$$

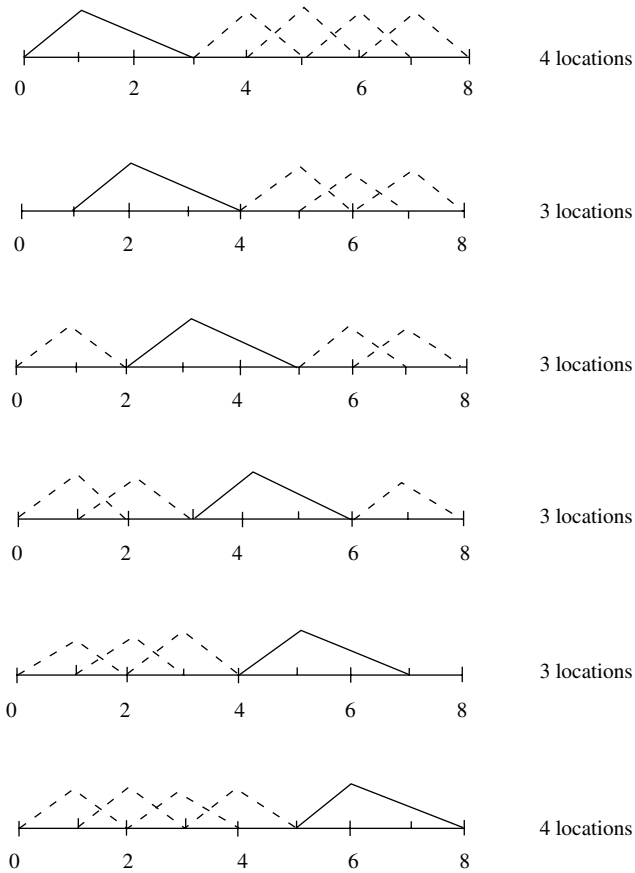


Figure 8.2 An error-event location-counting example for $K = 8$, $n = 2$, and $\ell = 5$. The horizontal lines represent the all-zero path and the triangular figures represent error events.

Substitution of (8.7) into (8.5) yields

$$A_i^{\mathrm{PCCC}}(P) \simeq \sum_{n_1=1}^{n_{\max}} \sum_{n_2=1}^{n_{\max}} \frac{\binom{K}{n_1}\binom{K}{n_2}}{\binom{K}{i}} A_{i,n_1}(P) A_{i,n_2}(P).$$

Three applications of the approximation

$$\binom{K}{s} \simeq \frac{K^s}{s!},$$

which assumes $s \ll K$, gives

$$A_i^{\mathrm{PCCC}}(P) \simeq \sum_{n_1=1}^{n_{\max}} \sum_{n_2=1}^{n_{\max}} \frac{i!}{n_1! n_2!} K^{n_1+n_2-i} A_{i,n_1}(P) A_{i,n_2}(P).$$

Also, since K is large, this last expression can be approximated by the term in the double summation with the highest power of K ($n_1 = n_2 = n_{\max}$), so that

$$A_i^{PCCC}(P) \simeq \frac{i!}{(n_{\max}!)^2} K^{2n_{\max}-i} \left[A_{i,n_{\max}}(P)\right]^2 .$$

We may now use this expression together with (8.4) to estimate PCCC ensemble BER performance on the binary-input AWGN channel as

$$\begin{aligned} P_b^{\text{PCCC}} &< \sum_{i=1}^{K} \frac{i}{K} I^i A_i^{\text{PCCC}}(P) \Bigg|_{I=P=\exp(-RE_b/N_0)} \\ &\simeq \sum_{i=i_{\min}}^{K} i \frac{i!}{(n_{\max}!)^2} K^{2n_{\max}-i-1} I^i [A_{i,n_{\max}}(P)]^2 \Bigg|_{I=P=\exp(-RE_b/N_0)} , \end{aligned} \quad (8.9)$$

where $i_{\min}$ is the minimum information-weight leading to non-negligible terms in (8.9). For example, when the constituent encoders for the PCCC are non-recursive $i_{\min} = 1$, and when they are recursive $i_{\min} = 2$. We now examine these two cases individually.

8.2.2.1 *PCCCs with Non-Recursive Constituent Encoders*

We will examine (8.9) more closely, assuming a non-recursive memory-μ constituent encoder, in which case $i_{\min} = 1$. Further, for a weight-i input, the greatest number of error events possible is $n_{\max} = i$ (consider isolated 1s). In this case,

$$A_{i,n_{\max}}(P) = A_{i,i}(P) = \left[A_{1,1}(P)\right]^i ,$$

that is, the error events due to the i isolated 1's are identical and their weights simply add since $A_{1,1}(P)$ has a single term. Applying this to (8.9) with $n_{\max} = i$ yields

$$P_b^{\text{PCCC}} \simeq \sum_{i=1}^{K} \frac{K^{i-1}}{(i-1)!} I^i [A_{1,1}(P)]^{2i} \Bigg|_{I=P=\exp(-RE_b/N_0)} .$$

Observe that the dominant term ($i = 1$) is independent of the interleaver size K. Thus, as argued more superficially in Chapter 7, interleaver gain does not exist for non-recursive constituent encoders. This is also the case for block constituent codes, for which $i_{\min} = 1$ and $n_{\max} = i$ are also true. By contrast, we will see that interleaver gain is realized for PCCCs with recursive constituent encoders.

8.2.2.2 *PCCCs with Recursive Constituent Encoders*

As we saw in Chapter 7, the minimum PCCC encoder input weight leading to non-negligible terms in the union bound on error probability is $i_{\min} = 2$. Further, for input weight i, the largest number of error events is $n_{\max} = \lfloor i/2 \rfloor$. Given this,

consider now the ith term in (8.9) for i odd:

$$i\frac{i!}{(\lfloor i/2\rfloor!)^2}K^{2\lfloor i/2\rfloor-i-1}I^i\left[A_{i,\lfloor i/2\rfloor}(P)\right]^2 = i\frac{i!}{(\lfloor i/2\rfloor!)^2}K^{-2}I^i\left[A_{i,\lfloor i/2\rfloor}(P)\right]^2.$$

When i is even, $n_{\max} = i/2$ and the ith term in (8.9) is

$$i\frac{i!}{((i/2)!)^2}K^{-1}I^i\left[A_{i,i/2}(P)\right]^2. \tag{8.10}$$

Thus, odd terms go as K^{-2} and even terms go as K^{-1}, so the odd terms may be ignored. Moreover, the factor K^{-1} in (8.10) represents interleaver gain. This result, which indicates that interleaver gain is realized when the constituent convolutional encoders are recursive, was seen in Chapter 7, albeit at a more superficial level.

Keeping only the even terms in (8.9) which have the form (8.10), we have, with $i = 2k$,

$$\begin{aligned} P_b^{\mathrm{PCCC}} &\simeq \sum_{k=1}^{\lfloor K/2\rfloor} 2k\frac{(2k)!}{(k!)^2}K^{-1}I^{2k}[A_{2k,k}(P)]^2\Bigg|_{I=P=\exp(-RE_b/N_0)} \\ &= \sum_{k=1}^{\lfloor K/2\rfloor} 2k\binom{2k}{k}K^{-1}I^{2k}[A_{2,1}(P)]^{2k}\Bigg|_{I=P=\exp(-RE_b/N_0)}, \end{aligned} \tag{8.11}$$

where the second line follows since

$$\binom{2k}{k} = \frac{(2k)!}{(k!)^2}$$

and

$$A_{2k,k}(P) = [A_{2,1}(P)]^k. \tag{8.12}$$

Equation (8.12) holds because the codewords corresponding to $A_{2k,k}(P)$ must be the concatenation of k error events, each of which is produced by weight-2 recursive convolutional-encoder inputs, each of which produces a single error event.

$A_{2,1}(P)$ has the simple form

$$\begin{aligned} A_{2,1}(P) &= P^{p_{\min}} + P^{2p_{\min}-2} + P^{3p_{\min}-4} + \cdots \\ &= \frac{P^{p_{\min}}}{1 - P^{p_{\min}-2}}. \end{aligned} \tag{8.13}$$

To show this, observe that the first 1 into the parity computation circuit with transfer function $g^{(2)}(D)/g^{(1)}(D)$ will yield the impulse response which consists of a transient $t(D)$ of length τ (and degree $\tau - 1$) followed by a periodic binary sequence having period τ, where τ is dictated by $g^{(1)}(D)$. If the second 1 comes along at the start of the second period, it will in effect "squelch" the output so that the subsequent output of the parity circuit is all zeros. Thus, the response to the

two 1s separated by $\tau - 2$ zeros is the transient followed by a 1, i.e., $t(D) + D^\tau$, and the weight of this response will be denoted by $p_{\min}$. If the second 1 instead comes along at the start of the third period, by virtue of linearity the parity output will be $[t(D) + D^\tau] + D^\tau[t(D) + D^\tau]$, which has weight $2p_{\min} - 2$ since the 1 which trails the first transient will cancel out the 1 that starts the second transient. If the second 1 comes along at the start of the fourth period, the parity weight will be $3p_{\min} - 4$; and so on. From this, we conclude (8.13).

As an example, with $g^{(2)}(D)/g^{(1)}(D) = (1 + D^2)/(1 + D + D^2)$, the impulse response is 111 011 011 011 ..., where $\tau = 3$ is obvious. The response to 10010... is clearly 111 011 011 011 ... + 000 111 011 011 ... = 11110..., and $p_{\min} = 4$. The response to 10000010... is 111 011 011 011,... + 000 000 111 011 011 ... = 11101110..., which has weight $2p_{\min} - 2$.

Substitution of (8.13) into (8.11) yields

$$P_b^{\mathrm{PCCC}} \simeq \sum_{k=1}^{\lfloor K/2 \rfloor} 2k \binom{2k}{k} K^{-1} I^{2k} \left[\frac{P^{p_{\min}}}{1 - P^{p_{\min}-2}} \right]^{2k} \Bigg|_{I=P=\exp(-RE_b/N_0)} . \tag{8.14}$$

When $\exp(-RE_b/N_0)$ is relatively small, the dominant ($k = 1$) term has the approximation

$$4K^{-1}[IP^{p_{\min}}]^2 \Big|_{I=P=\exp(-RE_b/N_0)} = \frac{4}{K} \exp[-(2 + 2p_{\min})RE_b/N_0], \tag{8.15}$$

from which we may make the following observations. First, the interleaver gain is evident from the factor $4/K$. Second, the exponent has the factor $2 + 2p_{\min}$ which is due to the worst-case inputs for both constituent encoders: two 1s separated by $\tau - 2$ zeros. Such a situation will yield information-weight 2 and weight $p_{\min}$ from each parity circuit. Of course, a properly designed interleaver can avoid such a misfortune for a specific code, but our ensemble result must include such worst-case instances.

8.2.2.3 *PCCC Design Principles*

We would now like to consider how the code polynomials, $g^{(1)}(D)$ and $g^{(2)}(D)$, employed at both constituent encoders should be selected to determine an ensemble with (near-)optimum performance. The optimization criterion is usually one of two criteria: the polynomials can be selected to minimize the level of the floor of the error-rate curve or they can be selected to minimize the SNR at which the waterfall region occurs. The latter topic will be considered in Chapter 9, where the waterfall region is shown to be connected to the idea of a "decoding threshold." We will now consider the former topic.

In view of the previous discussion, it is clear that the ensemble floor will be lowered (although not necessarily minimized) if $p_{\min}$ can be made as large as possible. This statement assumes a given complexity, i.e., a given memory size μ. However, $p_{\min}$ is the weight of $t(D) + D^\tau$, where $t(D)$ is the length-τ transient part of the impulse response of $g^{(2)}(D)/g^{(1)}(D)$. Assuming that about half of the

bits in $t(D)$ are 1s, one obvious way to ensure that the weight of $t(D)$, and hence of $p_{\min}$, is (nearly) maximal is to choose $g^{(1)}(D)$ to be a primitive polynomial so that $t(D)$ is of maximal length, i.e., length $2^{\mu} - 1$.

8.2.2.4 *Example Performance Bounds*

We now consider the ensemble performance bounds for two PCCCs, one for which $g^{(1)}(D)$ is primitive and one for which $g^{(1)}(D)$ is not primitive. Specifically, for PCCC1, both constituent codes have the generator matrix

$$G_1(D) = \begin{bmatrix} 1 & \dfrac{1+D^2}{1+D+D^2} \end{bmatrix} \text{(PCCC1)}$$

and for PCCC2 both constituent codes have the generator matrix

$$G_2(D) = \begin{bmatrix} 1 & \dfrac{1+D+D^2}{1+D^2} \end{bmatrix} \text{(PCCC2)}.$$

Note that $g^{(1)}(D) = 1 + D + D^2$ for $G_1(D)$ is primitive, whereas $g^{(1)}(D) = 1 + D^2$ for $G_2(D)$ is not primitive. Earlier we showed for $G_1(D)$ that the input $1 + D^3$ yields the parity output $1 + D + D^2 + D^3$ so that $p_{\min} = 4$ for this code. Correspondingly, the worst-case PCCC1 codeword weight for weight-2 inputs is $d_{2,\min}^{\text{PCCC1}} = 2 + 4 + 4 = 10$. The overall minimum distance for the PCCC1 ensemble is produced by the input $1 + D + D^2$, which yields the output $\begin{bmatrix} 1+D+D^2 & 1+D^2 & 1+D^2 \end{bmatrix}$, so that $d_{\min}^{\text{PCCC1}} = 3 + 2 + 2 = 7$. As for PCCC2, it is easy to see that the input $1 + D^2$ yields the parity output $1 + D + D^2$ so that $p_{\min} = 3$. The worst-case PCCC2 codeword weight for weight-2 inputs is then $d_{2,\min}^{\text{PCCC2}} = 2 + 3 + 3 = 8$. The overall minimum distance for the PCCC2 ensemble is also produced by the input $1 + D^2$ so that $d_{\min}^{\text{PCCC2}} = d_{2,\min}^{\text{PCCC2}} = 8$.

Thus, while PCCC1 has the smaller minimum distance $d_{\min}^{\text{PCCC}}$, it has the larger *effective minimum distance* $d_{2,\min}^{\text{PCCC}}$ and, thus, superior performance. The impact of its smaller minimum distance is negligible since it is due to an odd-weight encoder input (weight-3) as discussed in Section 8.2.2.2. This can also be easily seen by plotting (8.14) with $p_{\min} = 4$ for PCCC1 and $p_{\min} = 3$ for PCCC2. However, we shall work toward a tighter bound starting with (8.9) and using $Q\left(\sqrt{2iRE_b/N_0}\right)$ in place of its upper bound, $\exp(-iRE_b/N_0)$. The looser bound (8.14) is useful for code-ensemble performance comparisons, but we pursue the tighter result which has closer agreement with simulations.

To use (8.9), we require $\{A_{i,n_{\max}}(P)\}$ for the constituent codes for each PCCC. We find these enumerators first for $G_1(D)$, whose state diagram (split at state 0) is displayed in Figure 8.3. From this diagram, we may obtain the augmented

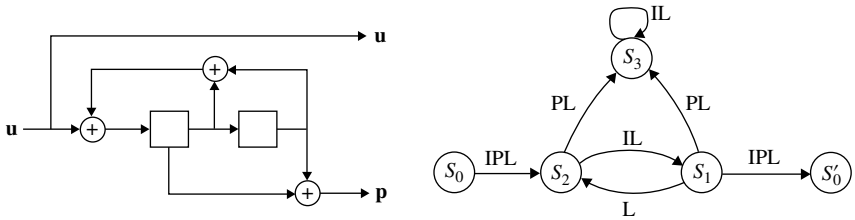


Figure 8.3 The encoder and split state diagram for a constituent encoder for PCCC1 with $G_1(D) = \begin{bmatrix} 1 & \dfrac{1+D^2}{1+D+D^2} \end{bmatrix}$.

information-parity WE,

$$\begin{aligned} A(I,P,B) &= \frac{I^3P^2B^3 + I^2P^4B^4 - I^4P^2B^4}{1 - P^2B^3 - I(B+B^2) + I^2B^3} \\ &= I^3P^2B^3 + I^2P^4B^4 + \left(I^3P^4 + I^4P^2\right)B^5 \\ &\quad + \left(2I^3P^4 + I^4P^4\right)B^6 + \left(I^2P^6 + 2I^4P^4 + I^5P^2 + I^5P^4\right)B^7 \\ &\quad + \left(2I^3P^6 + 3I^4P^4 + 2I^5P^4 + I^6P^4\right)B^8 \\ &\quad + \left(3I^3P^6 + 3I^4P^6 + 3I^5P^4 + I^6P^2 + 2I^6P^4 + I^7P^4\right)B^9 + \cdots, \end{aligned}$$

where the exponent of the augmented indeterminate B in a given term indicates the length of the error event(s) for that term. From this, we may identify the single-error-event ($n = 1$) enumerators $A_{i,1,\ell}(P)$ as follows (for $\ell \leq 9$):

$$\begin{array}{llll} A_{2,1,4}(P) = P^4, & A_{2,1,7}(P) = P^6, & & \\ A_{3,1,3}(P) = P^2, & A_{3,1,5}(P) = P^4, & A_{3,1,6}(P) = 2P^4, & A_{3,1,8}(P) = 2P^6, \\ A_{3,1,9}(P) = 3P^6 & & & \\ A_{4,1,5}(P) = P^2, & A_{4,1,6}(P) = P^4, & A_{4,1,7}(P) = 2P^4, & A_{4,1,8}(P) = 3P^4, \\ A_{4,1,9}(P) = 3P^6 & & & \\ & & \cdots & \end{array}$$

Note that $A_{1,n,\ell}(P) = 0$ for all n and ℓ because weight-1 inputs produce trellis paths that leave the zero path and never remerge. From the list above and (8.8), we can write down the first few terms of $A_{i,1}(P)$:

$$\begin{aligned} A_{2,1}(P) &= P^4 + P^6 + \cdots, \\ A_{3,1}(P) &= P^2 + 3P^4 + 5P^6 + \cdots, \\ A_{4,1}(P) &= P^2 + 6P^4 + 3P^6 + \cdots, \\ &\cdots \end{aligned}$$

To determine the double-error-event ($n = 2$) enumerators $A_{i,2,\ell}(P)$, we first compute

$$\begin{aligned} [A(I,P,B)]^2 = I^6P^4B^6 &+ 2I^5P^6B^7 + (I^4P^8 + 2I^6P^6 + 2I^7P^4)B^8 \\ &+ (2I^5P^8 + 6I^6P^6 + 2I^7P^6)B^9 + \cdots. \end{aligned}$$

From this, we may write

$$\begin{aligned}
&A_{4,2,8}(P) = P^8,\\
&A_{5,2,7}(P) = 2P^6, \quad A_{5,2,9}(P) = 2P^8,\\
&A_{6,2,6}(P) = P^4, \quad A_{6,2,8}(P) = 2P^6, \quad A_{6,2,9}(P) = 6P^6,\\
&\qquad\qquad \cdots
\end{aligned}$$

Again using (8.8), we can write down the first few terms of $A_{i,2}(P)$:

$$\begin{aligned}
A_{4,2}(P) &= P^8 + \cdots,\\
A_{5,2}(P) &= 2P^6 + 2P^8 + \cdots,\\
A_{6,2}(P) &= P^4 + 8P^6 + \cdots,\\
&\cdots
\end{aligned}$$

We consider only the first few terms of (8.9), which are dominant. For input weights $i = 2$ and 3, $n_{\max} = 1$, and for $i = 4$ and 5, $n_{\max} = 2$. Thus, with $I = P = \exp(-RE_b/N_0)$, we have for PCCC1

$$\begin{aligned}
P_b^{\text{PCCC1}} &< \frac{4}{K} I^2 [A_{2,1}(P)]^2 + \frac{18}{K^2} I^3 [A_{3,1}(P)]^2 + \frac{24}{K} I^4 [A_{4,2}(P)]^2\\
&\quad + \frac{150}{K^2} I^5 [A_{5,2}(P)]^2 + \cdots\\
&\approx \frac{4}{K} I^2 \left[P^4 + P^6\right]^2 + \frac{18}{K^2} I^3 \left[P^2 + 3P^4 + 5P^6\right]^2 + \frac{24}{K} I^4 \left[P^8\right]^2\\
&\quad + \frac{150}{K^2} I^5 \left[P^4 + 8P^6\right]^2 + \cdots\\
&= \frac{18}{K^2} W^7 + \frac{108}{K^2} W^9 + \frac{4}{K} W^{10} + \frac{342}{K^2} W^{11} + \frac{8}{K} W^{12} + \frac{540}{K^2} W^{13}\\
&\quad + \frac{4}{K} W^{14} + \cdots,
\end{aligned}$$

where in the last line we set $W = I = P$. The bit-error-rate estimate can be obtained by setting $W = \exp(-RE_b/N_0)$ in the above expression, but we prefer the tighter expression

$$\begin{aligned}
P_b^{\text{PCCC1}} &< \frac{18}{K^2} f(7) + \frac{108}{K^2} f(9) + \frac{4}{K} f(10) + \frac{342}{K^2} f(11) + \frac{8}{K} f(12) + \frac{540}{K^2} f(13)\\
&\quad + \frac{4}{K} f(14) + \cdots, \qquad (8.16)
\end{aligned}$$

where $f(w) = Q(\sqrt{2wRE_b/N_0})$ and $R = 1/3$. It is expression (8.16) that we shall plot, but we shall first develop the analogous expression for PCCC2.

To find the enumerators $\{A_{i,n_{\max}}(P)\}$ for $G_2(D)$, we start with the state diagram displayed in Figure 8.4. Upon comparison with the state diagram for $G_1(D)$ in Figure 8.3, we see that one may be obtained from the other by exchanging I and P. In fact, the enumerator $A_{G_2}(I, P, B)$ for $G_2(D)$ may be obtained by exchanging I and P in the enumerator $A_{G_1}(I, P, B)$ for $G_1(D)$. That is, $A_{G_2}(I, P, B) = A_{G_1}(P, I, B)$,

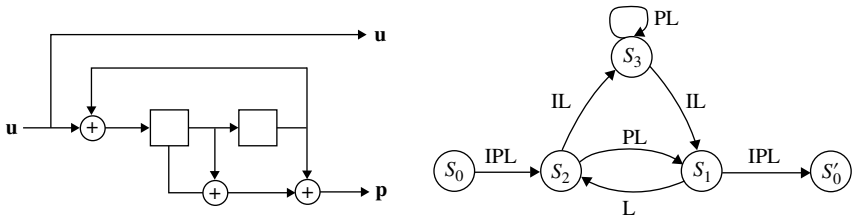


Figure 8.4 The encoder and split state diagram for a constituent encoder for PCCC2 with $G_2(D) = \begin{bmatrix} 1 & \dfrac{1+D+D^2}{1+D^2} \end{bmatrix}$.

yielding for $G_2(D)$

$$\begin{aligned} A(I,P,B) &= \frac{I^2P^3B^3 + I^4P^2B^4 - I^2P^4B^4}{1 - I^2B^3 - P(B+B^2) + P^2B^3} \\ &= I^2P^3B^3 + I^4P^2B^4 + \big(I^4P^3 + I^2P^4\big)B^5 \\ &\quad + \big(2I^4P^3 + I^4P^4\big)B^6 + \big(I^6P^2 + 2I^4P^4 + I^2P^5 + I^4P^5\big)B^7 \\ &\quad + \big(2I^6P^3 + 3I^4P^4 + 2I^4P^5 + I^4P^6\big)B^8 \\ &\quad + \big(3I^6P^3 + 3I^6P^4 + 3I^4P^5 + I^2P^6 + 2I^4P^6 + I^4P^7\big)B^9 + \cdots . \end{aligned}$$

By following steps similar to those in the $G_1(D)$ case, we may identify the single-error-event ($n = 1$) enumerators $A_{i,1,\ell}(P)$ (for $\ell \le 9$) for $G_2(D)$, from which we obtain the enumerators $A_{i,1}(P)$,

$$\begin{aligned} A_{2,1}(P) &= P^3 + P^4 + P^5 + P^6 + \cdots , \\ A_{3,1}(P) &= 0, \\ A_{4,1}(P) &= P^2 + 3P^3 + 6P^4 + 6P^5 + 3P^6 + P^7 + \cdots , \\ &\cdots \end{aligned}$$

Similarly, we can obtain the double-error-event ($n = 2$) enumerators $A_{i,2}(P)$,

$$\begin{aligned} A_{4,2}(P) &= P^6 + P^7 + \cdots , \\ A_{5,2}(P) &= 0, \\ A_{6,2}(P) &= P^5 + 2P^6 + \cdots , \\ &\cdots \end{aligned}$$

Again, we consider only the first few terms of (8.9). For input weight $i = 2$, $n_{\max} = 1$; for $i = 4$, $n_{\max} = 2$; and for $i = 3$ and 5, $A_{i,1}(P) = 0$. Thus, we have

for PCCC2

$$\begin{aligned}
P_b^{\text{PCCC2}} &< \frac{4}{K} I^2 [A_{2,1}(P)]^2 + \frac{24}{K} I^4 [A_{4,2}(P)]^2 + \cdots \\
&\approx \frac{4}{K} I^2 \left[P^3 + P^4 + P^5 + P^6\right]^2 + \frac{24}{K} I^4 \left[P^6 + P^7\right]^2 + \cdots \\
&= \frac{4}{K} \left[W^8 + 2W^9 + 3W^{10} + 4W^{11} + 3W^{12} + 2W^{13} + W^{14}\right] \\
&\quad + \frac{24}{K} \left[W^{16} + 2W^{17} + W^{18}\right] + \cdots ,
\end{aligned}$$

from which

$$\begin{aligned}
P_b^{\text{PCCC2}} \approx \frac{4}{K} &[f(8) + 2f(9) + 3f(10) + 4f(11) + 3f(12) + 2f(13) + f(14)] \\
&+ \frac{24}{K} [f(16) + 2f(17) + f(18)], \qquad (8.17)
\end{aligned}$$

where $f(w) = Q(\sqrt{2wRE_b/N_0})$ and $R = 1/3$.

We have plotted (8.16) and (8.17) in Figure 8.5 for $K = 100$, together with simulation results for a randomly selected interleaver (the same interleaver for both codes) and 20 decoding iterations. As predicted, PCCC1, for which $g^{(1)}(D)$ is primitive, has the superior performance. We also see the somewhat surprising close agreement between the ensemble performance bounds and the simulated

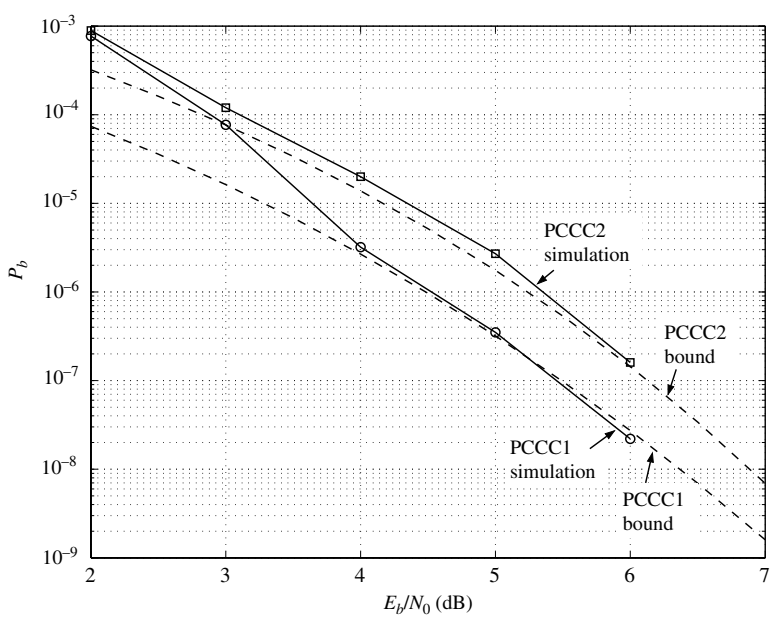


Figure 8.5 A performance comparison between the PCCC1 and PCCC2 ensembles for $K = 100$. Simulation results are for one randomly selected instance in each ensemble, with 20 decoder iterations.

performance curves for the two specific codes. Beside the fact that we are comparing an ensemble result with a specific code result, this plot is "surprising" because the bounds are estimates for maximum-likelihood decoders, whereas the simulations use the turbo decoder of the previous chapter. We remark that, in each case, increasing K by a factor of α will lower the curves by the factor α, as is evident from the error-rate expressions for P_b^{PCCC1} and P_b^{PCCC2}, Equations (8.16) and (8.17).

8.3 Ensemble Enumerators for Serial-Concatenated Codes

8.3.1 Preliminaries

As we did for the PCC case, we first examine how an enumerator for an SCC is related to that of its constituent codes, which may be either block or convolutional. The configuration assumed is pictured in Figure 8.6. The outer code, C_1, has parameters (N_1,K_1) and code rate $R_1 = K_1/N_1$ and the inner code, C_2, has parameters (N_2,K_2) and code rate $R_2 = K_2/N_2$. The overall SCC code rate is therefore $R = R_1R_2$. We shall assume that the size of the interleaver is N_1, although extension to the case pN_1, $p > 1$, is possible. As we shall see, in contrast with the PCC case for which the C-PWE $A_i(P)$ (or IP-WE $A(I, P)$) is appropriate (since parity weights are added), for SCCs the relevant WE is the C-OWE $A_i(W)$ (or IO-WE $A(I, W)$).

Consider an example for which C_1 is the (4,3) SPC code and C_2 is the (7,4) Hamming code so that the interleaver size is 4. Consider also weight-1 SCC encoder inputs. For this code, $A_1^{C_1}(W) = 3W^2$ is obvious. Thus, a weight-1 SCC encoder input will yield a weight-2 C_1 output and, hence, a weight-2 C_2 input. Given that $A_2^{C_2}(W) = 3W^3 + 3W^4$ (from Section 8.1), we cannot know whether the C_2 (hence SCC) output has weight 3 or 4, since this depends on which of the $\binom{4}{2} = 6$ inputs occurs at the C_2 encoder input, which in turn depends on the interleaver chosen and its input. Thus, as was done for the PCC case, to avoid unmanageable combinatorics, we seek the average C-OWE, where the average is over all interleavers. This is easily done by letting the interleaver act as the uniform interleaver. Thus, in the present example, a given term in $A_1^{C_1}(W) = W^2 + W^2 + W^2$ will, in effect, select a term in $A_2^{C_2}(P) = W^3 + W^3 + W^3 + W^4 + W^4 + W^4$ with probability $1/\binom{4}{2} = 1/6$ in the computation of the C-OWE $A_1^{\text{SCC}}(W)$ for the SCC. That is, a given weight-2 C_1 output will select a given C_2 output with probability 1/6. Thus,

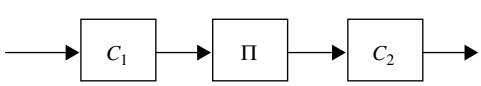


Figure 8.6 The SCCC encoder: codes C_1 and C_2 separated by interleaver Π.

we have

$$A_{1,3}^{\mathrm{SCC}} = \frac{A_{1,2}^{C_1} \cdot A_{2,3}^{C_2}}{\binom{4}{2}} = \frac{3 \cdot 3}{6} = 3/2.$$

As we will see in the next paragraph, in general, to obtain $A_{i,w}^{\mathrm{SCC}}$ one would have to sum over all C_1 output weights possible for the given SCC input and output weights i and w, but for this example there was only a single output weight: 2.

We now consider deriving $A_i^{\mathrm{SCC}}(W)$ for the general SCC which employs the uniform interleaver. For constituent code C_1, the $\binom{K_1}{i}$ possible weight-i inputs produce $A_{i,d}^{C_1}$ weight-d outputs. For constituent code C_2, the $\binom{K_2}{d}$ weight-d inputs produce $A_{d,w}^{C_2}$ weight-w outputs. Now consider concatenating C_1 with C_2 through a length-K_2 uniform interleaver. Each of the $A_{i,d}^{C_1}$ C_1 codewords will select one of the $A_{d,w}^{C_2}$ codewords of C_2 with probability $1/\binom{K_2}{d}$. Thus, the number of weight-w codewords produced by weight-i encoder inputs for the SCC with a length-K_2 uniform interleaver (equivalently, for the ensemble average) is given by the IO-WE coefficients $A_{i,w}^{\mathrm{SCC}}$ computed as

$$A_{i,w}^{\mathrm{SCC}} = \sum_{d=d_{\min,1}}^{K_2} \frac{A_{i,d}^{C_1} \cdot A_{d,w}^{C_2}}{\binom{K_2}{d}}, \tag{8.18}$$

where $d_{\min,1}$ is the minimum distance for the convolutional code C_1 considered as a block code. The (average) C-OWE for an SCC ensemble is thus

$$A_i^{\mathrm{SCC}}(W) = \sum_{w=w_{\min}}^{N_2} A_{i,w}^{\mathrm{SCC}} W^w = \sum_{d=d_{\min,1}}^{K_2} \frac{A_{i,d}^{C_1} \cdot A_d^{C_2}(W)}{\binom{K_2}{d}}, \tag{8.19}$$

where $A_d^{C_2}(W) = \sum_{w=w_{\min}}^{N_2} A_{d,w}^{C_2} W^w$ and $w_{\min}$ is the minimum SCC codeword weight. The IO-WE for an SCC ensemble is

$$A^{\mathrm{SCC}}(I, W) = \sum_{i=i_{\min}}^{K_1} I^i A_i^{\mathrm{SCC}}(W) = \sum_{d=d_{\min,1}}^{K_2} \frac{A_d^{C_1}(I) \cdot A_d^{C_2}(W)}{\binom{K_2}{d}}, \tag{8.20}$$

where $A_d^{C_1}(I) = \sum_{i=i_{\min}}^{K_1} A_{i,d}^{C_1} I^i$ enumerates the weights of the C_1 encoder inputs corresponding to weight-d C_1 encoder outputs and $i_{\min}$ is the minimum input weight that creates a non-negligible error event in the SCC.

8.3.2 SCCC Ensemble Enumerators

We now consider SCCs whose constituent codes are convolutional codes, that is, serial-concatenated convolutional codes (SCCCs). SCCs with constituent block codes are possible, but, as we will see, those with constituent convolutional codes generally enjoy greater interleaver gain. Further, since convolutional-code trellises are less complex, their BCJR-based decoders are too. Our development follows that of [3].

The outer convolutional code C_1 has rate $R_1 = k/p = K_1/N_1$ and the inner convolutional code C_2 has rate $R_2 = p/m = K_2/N_2$ so that the SCCC rate is $R = k/m$. In these expressions, $K_1 = Sk$ is the SCCC (and hence C_1) input block length and $K_2 = Sp$ is the C_2 input block length, where $S = K_1/k = K_2/p$ is the number of trellis stages for both $C_1$and C_2. (One trellis stage in C_1 corresponds to p output bits, which gives rise to one trellis stage in C_2, hence the equality.) The interleaver size is K_2 as before.

We now start with the upper bound on P_b, expressed as

$$\begin{aligned} P_b &< \frac{1}{K_1} \sum_{i=i_{\min}}^{K_1} i A_i^{\mathrm{SCCC}}(W) \Bigg|_{W=\exp(-RE_b/N_0)} \\ &= \frac{1}{K_1} \sum_{w=w_{\min}}^{K_2} \sum_{i=i_{\min}}^{K_1} i A_{i,w}^{\mathrm{SCCC}} \exp(-wRE_b/N_0), \end{aligned} \tag{8.21}$$

where $A_{i,w}^{\mathrm{SCCC}}$ may be found for a given SCCC ensemble via (8.18). This implies that we will require knowledge of the IO-WE coefficients $A_{i,d}^{C_1}$ and $A_{d,w}^{C_2}$ for the constituent codes C_1 and C_2. Following the development of the PCCC case, which focused on parity weight, see (8.7), we can write for the equivalent block code of a constituent convolutional code within an SCCC with generic input length K

$$A_i^{\mathrm{EBC}}(W) \lesssim \sum_{n=1}^{n_{\max}} \binom{S}{n} A_{i,n}(W). \tag{8.22}$$

We note that, in adapting (8.7) to obtain (8.22), K is set to S because the code rate assumed in (8.7) is 1/2. In (8.22),

$$A_{i,n}(W) = \sum_{\ell=1}^{K} A_{i,n,\ell}(W),$$

where $A_{i,n,\ell}(W)$ is the constituent convolutional-encoder output-weight enumerator conditioned on input weight i, which results in n concatenated error events whose lengths total ℓ trellis stages. From (8.22), we may write for the wth coefficient of $A_i^{\mathrm{EBC}}(W)$

$$A_{i,w}^{\mathrm{EBC}} \lesssim \sum_{n=1}^{n_{\max}} \binom{S}{n} A_{i,n,w}. \tag{8.23}$$

Substitution of (8.23) into (8.18) (twice, once for each constituent code) yields

$$A_{i,w}^{\mathrm{SCCC}} \lesssim \sum_{d=d_{\min,1}}^{K_2} \sum_{n_1=1}^{n_{1,\max}} \sum_{n_2=1}^{n_{2,\max}} \frac{\binom{S}{n_1}\binom{S}{n_2}}{\binom{K_2}{d}} A_{i,n_1,d} A_{d,n_2,w}. \tag{8.24}$$

As we did for the PCCC case, we can arrive at a simpler expression by further bounding $A_{i,w}^{\mathrm{SCCC}}$. Specifically, in the numerator we may (twice) use the bound

$$\binom{S}{n} < \frac{S^n}{n!}$$

and in the denominator we may use the bound

$$\binom{K_2}{d} > \frac{(K_2 - d + 1)^d}{d!} > \frac{(K_2)^d}{d^d d!},$$

where the second inequality follows from $(K_2 - d + 1) > K_2/d$ for all $d > 1$. Substitution of these bounds together with $S = K_2/p$ into (8.24) yields

$$A_{i,w}^{\mathrm{SCCC}} \lesssim \sum_{d=d_{\min,1}}^{K_2} \sum_{n_1=1}^{n_{1,\max}} \sum_{n_2=1}^{n_{2,\max}} (K_2)^{n_1+n_2-d} \frac{d^d d!}{p^{n_1+n_2} \cdot n_1! n_2!} A_{i,n_1,d} A_{d,n_2,w}. \tag{8.25}$$

Finally, substitution of (8.25) into (8.21) yields

$$\begin{aligned} P_b < & \sum_{w=w_{\min}}^{K_2} \exp(-wRE_b/N_0) \\ & \times \sum_{i=i_{\min}}^{K_1} \sum_{d=d_{\min,1}}^{K_2} \sum_{n_1=1}^{n_{1,\max}} \sum_{n_2=1}^{n_{2,\max}} \frac{i}{k} (K_2)^{n_1+n_2-d-1} \\ & \times \frac{d^d d!}{p^{n_1+n_2-1} \cdot n_1! n_2!} A_{i,n_1,d} A_{d,n_2,w}, \end{aligned} \tag{8.26}$$

where we used $K_1 = (K_2/p)k$.

If we consider the quadruple sum in the second and third lines of (8.26) to be a coefficient of the exponential, then, for a given w, this coefficient is maximum (dominant) when the exponent of K_2, namely $n_1 + n_2 - d - 1$, is maximum. Noting that n_1 depends on the C_1 encoder input weight i and n_2 depends on the C_2 encoder input weight d, we may write for this maximum exponent (which is a function of the SCCC codeword weight w)

$$\kappa_w = \max_{i,d} \{n_1 + n_2 - d - 1\}. \tag{8.27}$$

We now examine the worst-case exponent κ_w for two important special cases.

8.3.2.1 *High-SNR Region: The Exponent κ_w for $w = w_{\min}$*

When E_b/N_0 is large, the $w = w_{\min}$ term in (8.26) can be expected to be dominant. Thus, we examine the worst-case exponent, κ_w, of K_2 for $w = w_{\min}$. Given $w = w_{\min}$, and recalling that $n_{2,\max}$ is the maximum number of error events possible for the given w, we may write

$$n_{2,\max} \leq \left\lfloor \frac{w_{\min}}{d_{\min,2}} \right\rfloor,$$

where $d_{\min,2}$ is the minimum weight for an error event of C_2. Similarly, for a given C_1 encoder output weight d,

$$n_{1,\max} \leq \left\lfloor \frac{d}{d_{\min,1}} \right\rfloor.$$

We may now rewrite (8.27) with $w = w_{\min}$ as

$$\begin{aligned} \kappa_{w_{\min}} &\leq \max_d \left\{ \left\lfloor \frac{d}{d_{\min,1}} \right\rfloor + \left\lfloor \frac{w_{\min}}{d_{\min,2}} \right\rfloor - d - 1 \right\} \\ &= \left\lfloor \frac{d(w_{\min})}{d_{\min,1}} \right\rfloor + \left\lfloor \frac{w_{\min}}{d_{\min,2}} \right\rfloor - d(w_{\min}) - 1, \end{aligned} \tag{8.28}$$

where $d(w_{\min})$ is the minimum weight d of C_1 codewords that produces a weight-$w_{\min}$ C_2 codeword. It is generally the case that $d(w_{\min}) < 2d_{\min,1}$ and $w_{\min} < 2d_{\min,2}$ so that (8.28) becomes

$$\begin{aligned} \kappa_{w_{\min}} &\leq 1 - d(w_{\min}) \\ &\leq 1 - d_{\min,1}, \end{aligned}$$

where the second inequality follows from the fact that $d(w_{\min}) \geq d_{\min,1}$ by virtue of the definition of $d(w_{\min})$.

Thus, focusing on the $w = w_{\min}$ term in (8.26), we have (for large values of E_b/N_0)

$$P_b \sim B_{\min} \left(K_2\right)^{1-d_{\min,1}} \exp(-w_{\min} R E_b/N_0), \tag{8.29}$$

where $B_{\min}$ is a constant corresponding to the four inner summations of (8.26) with $w = w_{\min}$. We see that, for relatively large values of E_b/N_0 (i.e., in the floor region), P_b is dominated by the minimum distance of the SCCC, $w_{\min}$, and that interleaver gain is possible under the requirement that the outer code is non-trivial, i.e., that $d_{\min,1} > 1$. Further, the larger $w_{\min}$ and $d_{\min,1}$ are, the lower we can expect the floor to be. Note that an interleaver gain factor of $1/(K_2)^2$ is possible with only $d_{\min,1} = 3$, which is far better than is possible for PCCCs, which have an interleaver gain factor of $1/K$.

We remind the reader that (8.29) is a performance estimate for an SCCC *ensemble*. So an interleaver gain factor of $1/(K_2)^2$ in front of the exponential $\exp(-w_{\min} R E_b/N_0)$ means that, within the ensemble, a minimum distance of $w_{\min}$ for a given member of the ensemble is unlikely (since $1/(K_2)^2$

is so small) and a minimum distance greater than $w_{\min}$ is more common within the ensemble. That is, the performance for a *specific* SCCC will never go as $\left[B_{\min}/(K_2)^2\right]\exp(-w_{\min}RE_b/N_0)$; rather, taking the dominant ($w = w_{\min}$) term in (8.21), the performance of a specific SCCC will go as $[B_{\min}/K_1]\exp(-w_{\min}RE_b/N_0)$, as for PCCCs. However, for specific codes, we can expect $w_{\min,\mathrm{SCCC}}$ to be larger than $w_{\min,\mathrm{PCCC}}$ because of the much smaller ensemble interleaver gain factors possessed by SCCCs. Lastly, we emphasize that these high-SNR results are true irrespective of the code type, namely block or convolutional, recursive or non-recursive.

8.3.2.2 *Low-SNR Region: The Maximum Exponent of K_2*

The foregoing subsection examined P_b for SCCC ensembles for large values of E_b/N_0, i.e., for SCCC codeword weight fixed at $w = w_{\min}$. We now examine the low-SNR region, the so-called waterfall region which determines how closely a code's (or code ensemble's) performance approaches the capacity limit. In this case, the worst-case (over all w, i, and d) exponent $(n_1 + n_2 - d - 1)$ of K_2 in (8.26) comes into play. We denote this worst-case exponent by

$$\kappa^* = \max_{i,d,w}\{n_1 + n_2 - d - 1\}.$$

When the inner convolutional code is non-recursive, the maximum number of inner code error events, $n_{2,\max}$, is equal to d so that

$$\kappa^* = n_{1,\max} - 1,$$

which is non-negative. Thus, low-SNR interleaver gain is not possible for non-recursive inner codes, so let us consider recursive inner convolutional codes.

For a recursive inner encoder input weight of d, the maximum number of error events is $n_{2,\max} = \lfloor d/2 \rfloor$ so that

$$\begin{aligned}\kappa^* &= \max_{i,d}\{n_{1,\max} + \lfloor d/2 \rfloor - d - 1\}\\ &= \max_{i,d}\left\{n_{1,\max} - \left\lfloor \frac{d+1}{2} \right\rfloor - 1\right\}.\end{aligned}$$

It can be shown [3] that κ^* reduces to

$$\kappa^* = -\left\lfloor \frac{d_{\min,1} + 1}{2} \right\rfloor$$

so that interleaver gain exists for a recursive inner code. Note that this result holds also for a rate-1 recursive inner code, such as an accumulator, as was studied in an example in the previous chapter.

We now summarize the SCCC design rules we have obtained from our ensemble performance analysis.

- At high SNR values, independently of the constituent encoder types, the ensemble interleaver gain factor is $(K_2)^{1-d_{\min,1}}$, so an outer code with a large value for $d_{\min,1}$ is mandatory if one seeks a code with a low error-rate floor.
- At low SNR values, no interleaver gain is realized when the inner code is non-recursive. When the inner code is recursive, the interleaver gain is $(K_2)^{-\lfloor (d_{\min,1}+1)/2 \rfloor}$, so we seek an outer code with a large $d_{\min,1}$ together with the inner recursive convolutional code.

8.4 Enumerators for Selected Accumulator-Based Codes

In this section we study weight enumerators for repeat–accumulate (RA) code ensembles and irregular repeat–accumulate (IRA) code ensembles. Both code types were introduced as a subclass of LDPC codes in Chapter 6, where it was shown that their parity-matrix structure is such that encoding may be accomplished using the parity-check matrix.

8.4.1 Enumerators for Repeat–Accumulate Codes

A repeat–accumulate code is a serially concatenated code whose outer constituent code is a rate-$1/q$ repetition code and whose inner constituent code is a rate-1 accumulator with transfer function $1/(1+D)$. Thus, the encoder takes a length-K data block and repeats it q times before sending it through a length-qK interleaver, whose output is then sent through the accumulator. Although the code may be made systematic by transmitting the length-K data block together with the length-qK accumulator output block, we consider only non-systematic (qK,K) RA code ensembles. Application of (8.18) to RA code ensembles gives for the IO-WE coefficients

$$A_{i,w}^{\mathrm{RA}} = \sum_{d=q}^{qK} \frac{A_{i,d}^{\mathrm{rep}} \cdot A_{d,w}^{\mathrm{acc}}}{\binom{qK}{qi}}, \tag{8.30}$$

where $A_{i,d}^{\mathrm{rep}}$ is the number of weight-d repetition-code codewords corresponding to weight-i inputs and $A_{d,w}^{\mathrm{acc}}$ is the number of weight-w accumulator codewords corresponding to weight-d accumulator inputs. Note that, in comparison with (8.18), we have set $d_{\min,1} = q$ and $K_2 = qK$. The IO-WE coefficients for the repetition code with length-K inputs are clearly given by

$$A_{i,d}^{\mathrm{rep}} = \begin{cases} \binom{K}{i}, & d = qi, \\ 0, & d \neq qi. \end{cases} \tag{8.31}$$

The IO-WE coefficients for the accumulator $1/(1+D)$ with a length-m, weight-d input and length-m, weight-w output are given [5] by

$$A_{d,w}^{\text{acc}} = \binom{m-w}{\lfloor d/2 \rfloor}\binom{w-1}{\lceil d/2 \rceil - 1}. \tag{8.32}$$

We give a partial proof of this result as follows. First observe that error events are determined by all pairs of consecutive 1s at the accumulator input, and that the output due to two 1s separated by $h-1$ zeros is a string of h consecutive 1s. Thus, $A_{d,w}^{\text{acc}}$ should be determined in this light, that is, considering pairs of consecutive 1s among the d 1s at the accumulator input (together with the "leftover" 1 when d is odd). We consider a few values for d.

$\underline{d=1}$: $A_{1,w}^{\text{acc}} = 1 = \binom{m-w}{0}\binom{w-1}{0}$. To see that there is only a single weight-1 input word that yields a weight-w output word, observe that the output from a single 1 followed by 0s is a string of 1s. This output string will have weight w if and only if that single 1 at the accumulator input is followed by exactly $w-1$ zeros that make up the end of the input block.

$\underline{d=2}$: $A_{2,w}^{\text{acc}} = m-w = \binom{m-w}{1}\binom{w-1}{0}$. To see that there are $m-w$ weight-2 input words that yield weight-w output words, note that the pair of 1s at the accumulator input will yield w 1s at the output only if they are separated by $w-1$ 0s. Further, there are exactly $m-w$ ways in which a pair of 1s separated by $w-1$ zeros can fit in a block of m bits.

$\underline{d=3}$: $A_{3,w}^{\text{acc}} = (m-w)(w-1) = \binom{m-w}{1}\binom{w-1}{1}$. In this case, the first two 1s create an error event and the third 1 yields at the accumulator output a string of 1s whose length s equals n_z+1, where n_z is the number of zeros following the third 1. The length is $1 \le s \le w-1$, for, if $s \ge w$, the weight of the output would exceed w. Thus, $0 \le n_z \le w-2$ since $s = n_z+1$. Consequently, for a given value of n_z, the output will have weight w, provided that the first two 1s are separated by $w-n_z-1$ zeros. For each of the $w-1$ values of n_z, the two 1s separated by $w-n_z-1$ zeros can be in exactly $m-w$ places; hence $A_{3,w}^{\text{acc}} = (m-w)(w-1)$.

The situation for $d=4$ involves two error events, the situation for $d=5$ involves two error events followed by a single 1, and so on for $d>5$.

Substitution of (8.31) and (8.32) into (8.30) yields the following coefficients for the IO-WE for RA code ensembles:

$$A_{i,w}^{\text{RA}} = \frac{\binom{K}{i}\binom{qK-w}{\lfloor qi/2 \rfloor}\binom{w-1}{\lceil qi/2 \rceil - 1}}{\binom{qK}{qi}}.$$

These results may be used to plot union-bound performance curves, which would show that RA codes perform very well. This is somewhat surprising because neither the repetition code alone nor the accumulator alone provides coding gain, but, when concatenated together through an interleaver, the resulting RA code

provides substantial gain. For large values of K, the coding gain is within 1 dB of the capacity limit for $q \geq 4$ and within about 0.25 dB for $q \geq 8$. We will not examine RA code ensembles any further. Instead, we now examine the more practical IRA codes which allow more flexibility in code design and code-rate selection. The flexibility provided by irregularity leads to superior performance.

8.4.2 Enumerators for Irregular Repeat–Accumulate Codes

The techniques of the next section permit one to compute enumerators for IRA codes based on protographs, which are the most practical form of IRA code. However, it is illuminating to consider another subset of the IRA codes. This subset is characterized by the parameter triple $\langle N, K, w_c \rangle$, where N is the codeword length, K is the information word length, and w_c is the column weight of $\mathbf{H}_u$, the systematic portion of the IRA code parity-check matrix, $\mathbf{H} = [\mathbf{H}_u \quad \mathbf{H}_p]$. We leave the row weight distribution of $\mathbf{H}_u$ unspecified. To simplify the analysis, we assume that the w_c 1s in each column of $\mathbf{H}_u$ are uniformly distributed and that the 1s in any two rows have at most one position in common, i.e., there are no 4-cycles in the graph of $\mathbf{H}_u$.

It is convenient to use the following form of union bound on the bit error rate for the IRA code ensemble,

$$P_b < \frac{1}{K} \sum_{w>0} \sum_{i:i+p=w} i A_{i,p} W^w \Bigg|_{W=e^{-RE_b/N_0}}, \tag{8.33}$$

where $A_{i,p}$ is the coefficient for the conditional parity-weight enumerator; that is, $A_{i,p}$ is the number of weight-p parity words for weight-i inputs. To facilitate the analysis, we use the encoder structure in Figure 8.7, which was introduced in Chapter 6, composed of the matrix $\mathbf{A} \triangleq \mathbf{H}_u^{\mathrm{T}} \mathbf{\Pi}^{-1}$, the uniform interleaver $\mathbf{\Pi}$, and the accumulator. If we treat the bottom branch of the encoder as a

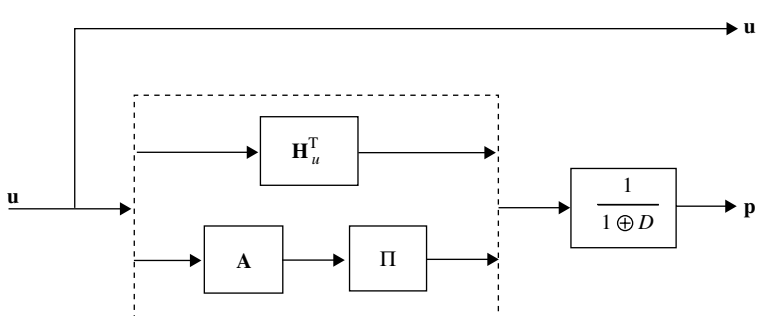


Figure 8.7 An encoder for irregular repeat–accumulate code.

serial-concatenated code, then we may write

$$A_{i,p} = \sum_{\ell} \frac{A_{i,\ell}^{(\mathbf{A})} \cdot A_{\ell,p}^{(\mathrm{acc})}}{\binom{M}{\ell}}, \tag{8.34}$$

where $A_{i,\ell}^{(\mathbf{A})}$ is the IO-WE of the outer "code" corresponding to $\mathbf{A}$, $A_{\ell,p}^{(\mathrm{acc})}$ is the IO-WE of the inner accumulator, and $M = N - K$. From the previous section, the IO-WE of the accumulator is given by

$$A_{\ell,p}^{(\mathrm{acc})} = \binom{M-p}{\lfloor \ell/2 \rfloor}\binom{p-1}{\lceil \ell/2 \rceil - 1}. \tag{8.35}$$

The IO-WE $A_{i,\ell}^{(\mathbf{A})}$ of the outer "code" can be written as the number of weight-i inputs, $\binom{K}{i}$, times the fraction, $F_{i,\ell}$, of those inputs that create weight ℓ outputs:

$$A_{i,\ell}^{(\mathbf{A})} = \binom{K}{i} F_{i,\ell}.$$

The computation of the fractions $F_{i,\ell}$ for all possible i and ℓ is complex in general. However, in the high-SNR region P_b is dominated by the low-weight codewords corresponding to low-weight inputs. Hence, we focus on $F_{i,\ell}$ for only small values of i. Now, when the input word $\mathbf{u}$ has weight $i = 1$, the output of "encoder" $\mathbf{A}$, given by $\mathbf{uA} = \mathbf{uH}_u^{\mathrm{T}}\mathbf{\Pi}^{-1}$, is simply a row of $\mathbf{H}_u^{\mathrm{T}}$ permuted by $\mathbf{\Pi}^{-1}$. But each row of $\mathbf{H}_u^{\mathrm{T}}$ has weight w_c per our assumption above. Thus, we have

$$F_{1,\ell} = \begin{cases} 1, & \text{for } \ell = w_c, \\ 0, & \text{for } \ell \neq w_c. \end{cases}$$

For weight-2 inputs ($i = 2$), the weight of the output of "encoder" $\mathbf{A}$ will be either $\ell = 2w_c$ or $\ell = 2w_c - 2$, since each weight-2 input adds two rows in $\mathbf{H}_u^{\mathrm{T}}$ and, by assumption, there is at most one overlap of 1s in the rows. Then, we may write, for $i = 2$, $\ell = 2w_c$, and $\ell = 2w_c - 2$,

$$F_{2,\ell} = \frac{N_{2,\ell}}{N_{2,2w_c} + N_{2,2w_c-2}}, \tag{8.36}$$

where $N_{i,\ell}$ is the number of weight-ℓ outputs from "encoder" $\mathbf{A}$, given weight-i inputs. The two terms $N_{i,\ell}$ that appear in (8.36) can be shown [7, 8] to be

$$N_{2,2w_c} = \frac{1}{2}\binom{M}{w_c}\binom{M-w_c}{w_c}$$

and

$$N_{2,2w_c-2} = \frac{1}{2}\binom{M}{w_c} w_c \binom{M-w_c}{w_c-1}.$$

$F_{1,\ell}$ and $F_{2,\ell}$ are generally dominant in the error-floor region of an IRA code's error-rate curve, so $i \geq 3$ terms may be omitted. This is demonstrated in the following example.

Example 8.1. We applied our performance estimate to the following IRA code $\langle N, K, w_c \rangle$ ensembles, grouped so that codes of like parameters may be compared:

1. $\langle 200, 100, 3 \rangle$ and $\langle 200, 100, 5 \rangle$
2. $\langle 2000, 1000, 3 \rangle$ and $\langle 2000, 1600, 3 \rangle$
3. $\langle 200, 100, 3 \rangle$ and $\langle 2000, 1000, 3 \rangle$
4. $\langle 4161, 3431, 3 \rangle$, $\langle 4161, 3431, 4 \rangle$, and $\langle 4161, 3431, 5 \rangle$.

For each ensemble in the first three groups, two codes were randomly selected (designed) from the ensemble. For the last group, only one code is designed for each ensemble. The simulation results of the above four comparison groups are shown in Figure 8.8 together with the floor estimate (8.33). We observe that the floor estimates are consistent with the simulation results for the randomly selected codes. Moreover, we notice that increasing w_c (to at least 5) tends to lower the floor dramatically.

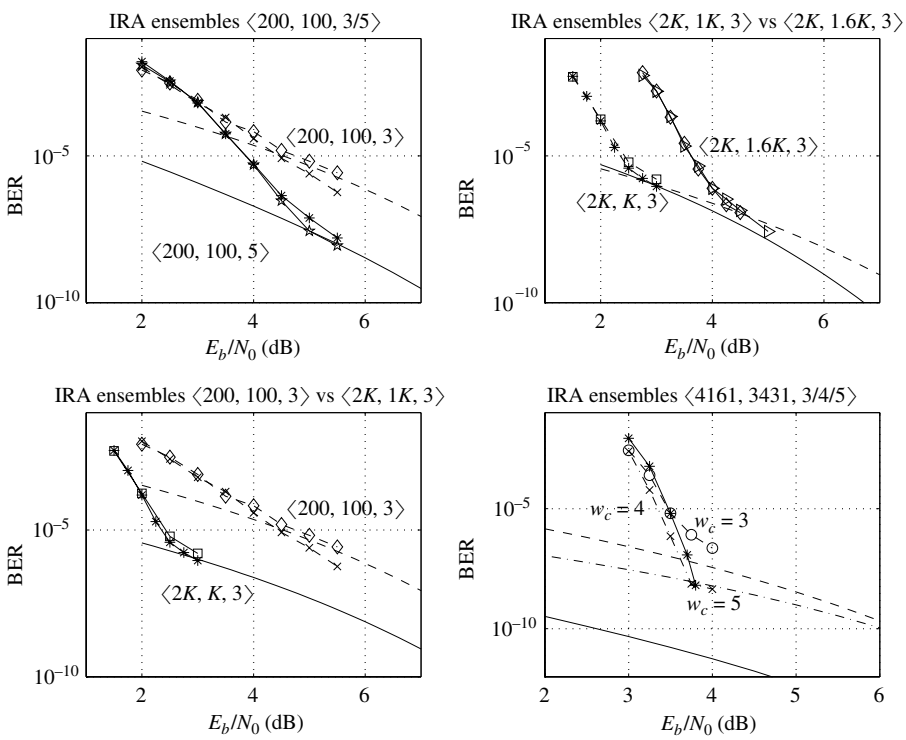


Figure 8.8 Comparison of the simulated performance curves of selected specific IRA codes and their corresponding ensemble performance curves.

8.5 Enumerators for Protograph-Based LDPC Codes

The preceding sections in this chapter on enumerators first dealt with parallel turbo codes, then serial turbo codes, and then accumulator-based codes, which can be deemed as a hybrid of a turbo code and an LDPC code. The next step in this progression is, naturally, LDPC codes. There exists a good number of results on enumerators for LDPC codes, but we will focus on LDPC codes based on protographs. The reason for this is that virtually all practical LDPC codes are quasi-cyclic and all quasi-cyclic codes are protograph codes. To see this, recall that, when the underlying permutation matrices involved in the protograph copy-and-permute procedure are circulant, the LDPC code is quasi-cyclic (see Chapter 6). Conversely, any quasi-cyclic code has an underlying protograph. Because essentially no extra work is involved, the results we present are for generalized LDPC (G-LDPC) codes, that is, codes whose Tanner graphs are sparse and contain constraint nodes beyond SPC nodes.

We first derive ensemble codeword weight enumerators for protograph-based G-LDPC codes for finite-length ensembles. We then use this result and extend it to infinite-length ensembles, and show how to determine when an ensemble has the property that its minimum distance grows with code length. We then show how the codeword weight-enumerator approach may be adapted to obtain trapping-set enumerators and stopping-set enumerators. Our development closely follows [9–12]. For other important expositions on LDPC code ensemble enumerators, see [13–15].

A note of caution: as mentioned in the introduction to this chapter, if one finds an ensemble with excellent performance, then one can pick a "typical" code at random from the ensemble and expect excellent performance. However, code designers often choose a quasi-cyclic member from the ensemble. In the context of this section, this means that the various interleavers (see Figure 8.9) are chosen to be circulant permutation matrices. However, the fraction of quasi-cyclic codes goes to zero as the code length tends to infinity. That is, a quasi-cyclic code is not a typical member of the code ensemble and one cannot expect such a code to share the attractive properties seen in the ensemble average. (This was pointed out by D. Costello.) On the other hand, many excellent quasi-cyclic codes have been designed in the way suggested here [16–21].

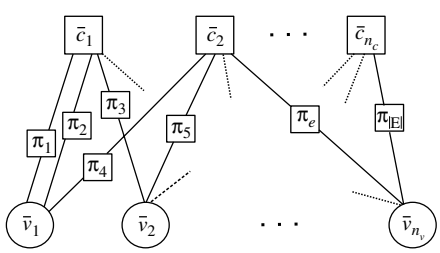


Figure 8.9 A vectorized protograph, a shorthand for the copy-and-permute procedure.

8.5.1 Finite-Length Ensemble Weight Enumerators

Consider a G-LDPC protograph, $G = (V,C,E)$, where $V = \{v_1,v_2, \ldots, v_{n_v}\}$ is the set of n_v variable nodes (VNs), $C = \{c_1,c_2, \ldots, c_{n_c}\}$ is the set of n_c constraint nodes (CNs), and E is the set of edges. Denote by q_{v_j} the degree of variable node v_j and by q_{c_i} the degree of constraint node c_i. Now consider the G-LDPC code constructed from a protograph G by making Q replicas of G and using uniform interleavers, each of size Q, to permute the edges among the replicas of the protograph. As described in Chapter 6, this procedure is conveniently represented by an "expanded" protograph as in Figure 8.9. We refer to each of the Q copies of protograph node v_j as a "type v_j node" and similarly for copies of protograph node c_i.

Recall that a length-ℓ uniform interleaver is a probabilistic device that maps each weight-l input into the $\binom{\ell}{l}$ distinct permutations of it with equal probability, $1/\binom{\ell}{l}$. This approach allows ensemble-average weight enumerators to be derived for various types of concatenated codes. By exploiting the uniform-interleaver concept, the average weight enumerator for a protograph-based LDPC ensemble can be obtained. To do so, the VNs and CNs are treated as constituent codes in a serial-concatenated code (SCC) scheme as explained further below.

In the case of two serially concatenated constituent codes, C_1 and C_2, separated by a uniform interleaver, the average number of weight-w codewords created by weight-l inputs in the SCC ensemble is given by (8.18). To apply this result to the G-LDPC code case, the group of Q VNs of type v_j is considered to be a constituent (repetition) code with a weight-d_j input of length Q and q_{v_j} length-Q outputs. Further, the group of Q CNs of type c_i is considered to be a constituent code with q_{c_i} inputs, each of length Q, and a fictitious output of weight zero. Now let $A(\mathbf{w})$ be the average (over the ensemble) number of codewords having *weight vector* $\mathbf{w} = [w_1, w_2, \ldots, w_{n_v}]$ corresponding to the n_v length-Q inputs satisfying the protograph constraints. $A(\mathbf{w})$ is called the *weight-vector enumerator* for the ensemble of codes of length $Q \cdot n_v$ described by the protograph. Let us further define the following:

- $A^{v_j}(w_j, \mathbf{d}_j) = \binom{Q}{w_j}\delta_{w_j,d_{j,1}}\delta_{w_j,d_{j,2}}\cdots\delta_{w_j,d_{j,q_{v_j}}}$ is the weight-vector enumerator for the type-v_j constituent code for a weight-w_j input, where $\mathbf{d}_j = \left[d_{j,1}, d_{j,2}, \ldots, d_{j,q_{v_j}}\right]$ is a weight vector describing the constituent code's output (note that, while the elements of $\mathbf{w} = [w_1, w_2, \ldots, w_{n_v}]$ correspond to VN input weights, they may be deemed also as G-LDPC encoder output weights); and
- $A^{c_i}(\mathbf{z}_i)$ is the weight-vector enumerator for the type-c_i constituent code and $\mathbf{z}_i = \left[z_{i,1}, z_{i,2}, \ldots, z_{i,q_{c_i}}\right]$, where $z_{i,l} = w_{j,k}$ if the lth edge of CN c_i is the kth edge of VN v_j.

A generalization of the development that led to (8.18) will yield for the G-LDPC protograph the average protograph weight-vector enumerator,

$$A(\mathbf{w}) = \sum_{w_{m,u}} \frac{\prod_{j=1}^{n_v} A^{v_j}(w_j, \mathbf{d}_j) \prod_{i=1}^{n_c} A^{c_i}(\mathbf{z}_i)}{\prod_{s=1}^{n_v} \prod_{r=1}^{q_{v_s}} \binom{Q}{w_{s,r}}} \tag{8.37}$$

$$= \frac{\prod_{i=1}^{n_c} A^{c_i}(\mathbf{w}_i)}{\prod_{j=1}^{n_v} \binom{Q}{d_j}^{q_{v_j}-1}}, \tag{8.38}$$

where the summation in the first line is over all weights $w_{m,u}$, where $w_{m,u}$ is the weight along the uth edge of VN v_m, where $m = 1, \ldots, n_v$ and $u = 1, \ldots, q_{v_m}$. The second line follows from the expression given above for $A^{v_j}(w_j, \mathbf{d}_j)$, a scaled product of Kronecker delta functions. The weight vector $\mathbf{w}_i = \left[w_{i_1}, w_{i_2}, \ldots, w_{i_{q_{c_i}}}\right]$ describes the weights of the Q-bit words on the edges connected to CN c_i, which are produced by the VNs neighboring c_i. The elements of $\mathbf{w}_i$ comprise a subset of the elements of $\mathbf{w}$.

Let S_t be the set of transmitted VNs and let S_p be the set of punctured VNs. Then the average number of codewords of weight w in the ensemble, denoted by A_w, equals the sum of $A(\mathbf{w})$ over all $\mathbf{w}$ for which $\sum_{\{w_j : v_j \in S_t\}} w_j = w$. Notationally,

$$A_w = \sum_{\{w_j : v_j \in S_t\}} \sum_{\{w_k : v_k \in S_p\}} A(\mathbf{w}) \tag{8.39}$$

under the constraint $\sum_{\{w_j : v_j \in S_t\}} w_j = w$. Thus, to evaluate A_w in (8.39), one first needs to compute the weight-vector enumerators, $A^{c_i}(\mathbf{w}_i)$, for the constraint nodes c_i, as seen in (8.38).

Consider the generic constituent (μ,κ) linear block code $\mathcal{C}$ in an expanded protograph. We need to find its weight-vector enumerator $A^{\mathcal{C}}(\mathbf{w})$, where $\mathbf{w} = [w_1, w_2, \ldots, w_\mu]$ is a constituent-code weight vector. The $\{A^{\mathcal{C}}(\mathbf{w})\}$ may be easily found as the coefficients of the multidimensional "W-transform" of $\{A^{\mathcal{C}}(\mathbf{w})\}$ as follows. Exploiting the uniform-interleaver property and the fact that the multidimensional W-transform of a single constraint node is $\sum_{\mathbf{x} \in \mathcal{C}} W_1^{x_1} W_2^{x_2} \ldots W_\mu^{x_\mu}$, the multidimensional W-transform for Q copies of the protograph is

$$A^{\mathcal{C}}(W_1, W_2, \ldots, W_\mu) = \left(\sum_{\mathbf{x} \in \mathcal{C}} W_1^{x_1} W_2^{x_2} \ldots W_\mu^{x_\mu} \right)^Q, \tag{8.40}$$

where the W_ls are indeterminate bookkeeping variables and $\mathbf{x} = [x_1, x_2, \ldots, x_\mu]$, $x_l \in \{0, 1\}$, is a codeword in $\mathcal{C}$. Expanding the right-hand side of (8.40) will yield

the form

$$A^{\mathcal{C}}(W_1, W_2, \ldots, W_\mu) = \sum_{\mathbf{w}} A^{\mathcal{C}}(\mathbf{w}) W_1^{w_1} W_2^{w_2} \ldots W_\mu^{w_\mu}, \tag{8.41}$$

from which we may obtain $A^{\mathcal{C}}(\mathbf{w})$. The direct application of the multinomial theorem on the right-hand side of (8.40) gives

$$A^{\mathcal{C}}(W_1, W_2, \ldots, W_\mu) = \sum_{\substack{n_1, n_2, \ldots, n_K \geq 0 \\ n_1 + n_2 + \cdots + n_K = Q}} C(Q; n_1, n_2, \ldots, n_K) \prod_{\mathbf{x} \in \mathcal{C}} \left(W_1^{x_1} W_2^{x_2} \ldots W_\mu^{x_\mu} \right)^{n_K}, \tag{8.42}$$

where $K = 2^\kappa$ is the number of codewords in $\mathcal{C}$, n_K is the number of occurrences of the Kth codeword, and $C(Q; n_1, n_2, \ldots, n_K)$ is the multinomial coefficient, given by

$$C(Q; n_1, n_2, \ldots, n_K) = \frac{Q!}{n_1! n_2! \ldots n_K!}. \tag{8.43}$$

Let $\mathbf{M}^{\mathcal{C}}$ be the $K \times \mu$ matrix with the codewords of $\mathcal{C}$ as its rows and $\mathbf{n} = [n_1, n_2, \ldots, n_K]$. Then (8.42) can be written as

$$A^{\mathcal{C}}(W_1, W_2, \ldots, W_\mu) = \sum_{\mathbf{w}} \sum_{\{\mathbf{n}\}} C(Q; n_1, n_2, \ldots, n_K) W_1^{w_1} W_2^{w_2} \ldots W_\mu^{w_\mu}, \tag{8.44}$$

where $\{\mathbf{n}\}$ is the set of integer solutions to $\mathbf{w} = \mathbf{n} \cdot \mathbf{M}^{\mathcal{C}}$, under the constraints $n_1, n_2, \ldots, n_K \geq 0$ and $\sum_{k=1}^{K} n_k = Q$. To see the last step, note that the product in (8.42) can be manipulated as follows:

$$\prod_{\mathbf{x} \in \mathcal{C}} \left(W_1^{x_1} W_2^{x_2} \ldots W_\mu^{x_\mu} \right)^{n_k} = W_1^{w_1} W_2^{w_2} \ldots W_\mu^{w_\mu}, \tag{8.45}$$

where $w_i = \sum_{\mathbf{x} \in \mathcal{C}} x_l n_k$, $l = \{1, 2, \ldots, \mu\}$. Also, if $\mathbf{w} = \mathbf{n} \cdot \mathbf{M}^{\mathcal{C}}$ has more than one solution of $\mathbf{n}$, the term $W_1^{x_1} W_2^{x_2} \ldots W_\mu^{x_\mu}$ will appear as a common factor in all of the terms that are associated with these solutions. This explains the presence of the second summation in (8.44). Finally, comparing (8.41) and (8.44) leads to the expression for the weight-vector enumerator,

$$A^{\mathcal{C}}(\mathbf{w}) = \sum_{\{\mathbf{n}\}} C(Q; n_1, n_2, \ldots, n_K), \tag{8.46}$$

where $\{\mathbf{n}\}$ is the set of integer solutions to $\mathbf{w} = \mathbf{n} \cdot \mathbf{M}^{\mathcal{C}}$, with $n_1, n_2, \ldots, n_K \geq 0$ and $\sum_{k=1}^{K} n_k = Q$.

Example 8.2. Consider the degree-4 SPC constraint node. The codeword set is $\mathcal{C} = \{0000, 1001, 0101, 1100, 0011, 1010, 0110, 1111\}$, so $K = 8$.

(a) Consider $Q = 3$ protograph copies and the weight vector $\mathbf{w} = [2, 2, 2, 2]$. From $\mathbf{w} = \mathbf{n} \cdot \mathbf{M}^{\mathcal{C}}$ and the associated constraints on $\mathbf{n}$, it is easy to see that $\{\mathbf{n}\} = \{[0, 0, 0, 1, 1,$

0, 0, 1], [0, 0, 1, 0, 0, 1, 0, 1], [0, 1, 0, 0, 0, 0, 1, 1], [1, 0, 0, 0, 0, 0, 0, 2]}. From this, $A^{\mathcal{C}}(\mathbf{w}) = 21$ (via (8.46)).
(b) When $N = 4$ and $\mathbf{w} = [4, 2, 2, 2]$, $\{\mathbf{n}\} = \{[0, 1, 0, 1, 0, 1, 0, 1]\}$ and so $A^{\mathcal{C}}(\mathbf{w}) = 24$.
(c) When $N = 4$ and $\mathbf{w} = [3, 2, 2, 2]$, $\{\mathbf{n}\}$ is empty and so $A^{\mathcal{C}}(\mathbf{w}) = 0$.

Example 8.3. Consider the protograph with a single (7,4) Hamming constraint node and seven degree-1 VNs, all transmitted. Noting that the denominator in (8.38) is unity (since $q_{v_i} = 1$ for all i) and the numerator is $A^{c_1}(\mathbf{w}_1)$ (since $n_c = 1$), we will compute A_w for $w = 0, 1, 2, 3$, assuming $Q = 4$ copies of the protograph. The Hamming code is generated by

$$\mathbf{G} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix},$$

from which the matrix $\mathbf{M}^{\mathcal{C}}$ may be obtained; we assume that the codewords are listed in the natural binary order with respect to the 16 input words. From (8.38) and (8.39), with $n_v = 7$ and $n_c = 1$, $A_w = \sum_{\mathbf{w}} A^{c_1}(\mathbf{w})$, where $\sum w_j = w$. Thus, $A_0 = A_{[0,0,0,0,0,0,0]} = C(4; 4, 0, \ldots, 0) = 1$. $A_1 = \sum_{\mathbf{w}} A^{c_1}(\mathbf{w})$ with $\sum w_j = 1$, but any such $\mathbf{w}$ must result in $\{\mathbf{n}\}$ being empty. Consequently, $A_1 = 0$. Similarly, one finds $A_2 = 0$. With $A_3 = \sum_{\mathbf{w}} A^{c_1}(\mathbf{w})$ such that $\sum w_j = 3$, $\mathbf{w} = \{[1, 0, 0, 0, 0, 1, 1], [0, 1, 0, 0, 1, 1, 0], [1, 0, 1, 0, 1, 0, 0], [0, 1, 1, 0, 1, 0, 0], [0, 0, 0, 1, 1, 0, 1], [1, 1, 0, 1, 1, 0, 0, 0], [0, 0, 1, 1, 0, 1, 0]\}$. Each $\mathbf{w}$ yields only one solution $\mathbf{n}$ to the equation $\mathbf{w} = \mathbf{n} \cdot \mathbf{M}^{\mathcal{C}}$. Each solution has $n_1 = 3$ together with $n_k = 1$, where k corresponds to a row in $\mathbf{M}^{\mathcal{C}}$ containing one of the seven weight-3 codewords. Note that the other $\mathbf{w}$s that achieve $\sum w_j = 3$ result in $\{\mathbf{n}\}$ being empty. Using (8.38), (8.39), and (8.46), $A_3 = \sum_{\mathbf{w}} A^{c_1}(\mathbf{w}) = \sum_{\{\mathbf{n}\}} C(4; 3, 0, \ldots, 1, \ldots, 0) = 28$.

8.5.2 Asymptotic Ensemble Weight Enumerators

For the asymptotic case, we define the normalized logarithmic asymptotic weight enumerator (we will simply call it the *asymptotic weight enumerator*) as

$$r(\delta) = \lim_{n\to\infty} \sup \frac{\ln A_w}{n}, \tag{8.47}$$

where $\delta \triangleq w/n$ (recall that n is the number of transmitted variable nodes in the code). As an example, it is easily shown that the weight enumerator for a rate-R, length-n random linear code ensemble is

$$A_w = \binom{n}{w} 2^{-n(1-R)}. \tag{8.48}$$

From this and *Stirling's approximation* of $n!$ for large n, namely, $n! \simeq \sqrt{2\pi n} n^n / e^n$, it can be shown that, for random linear codes,

$$r(\delta) = H(\delta) - (1 - R)\ln(2),$$

where $H(\delta) = -(1-\delta)\ln(1-\delta) - \delta \ln \delta$.

It can be shown (with non-negligible effort) both for Gallager codes [22] and for protograph-based codes [23] that

$$A_w \leq f(n) e^{nr(\delta)}, \tag{8.49}$$

where $f(n)$ is a sub-exponential function of n. (Compare this with a rearrangement of (8.47) with the lim sup operation removed: $A_w = e^{nr(\delta)}$.) The implication of (8.49) is that $A_w \simeq 0$ for large n when $r(\delta) < 0$. This allows us to make statements about the minimum distance of a code ensemble, as follows. Observing that $r(0) = 0$, let $\delta_{\min}$ be the second zero-crossing of $r(\delta)$, if it exists. Assuming that $\delta_{\min}$ exists and $r(\delta) < 0$ for all $0 < \delta < \delta_{\min}$, $\delta_{\min}$ is called the *typical (normalized) minimum distance*. $\delta_{\min}$ is so called because, from its definition, we may conclude for large n that $A_{\lfloor n\delta_{\min} \rfloor} \simeq 0$ and $A_{\lfloor n\delta_{\min} \rfloor + 1} > 0$, so that the minimum distance of the code ensemble is $d_{\min} = w_{\min} \simeq n\delta_{\min}$. Observe from this that the ensemble $d_{\min}$ grows linearly with n, which is certainly a desirable property.

For example, Gallager showed [22] that the typical minimum distance for regular LDPC codes with column weight 3 or greater increases linearly with code length n.

We now proceed to show how $\delta_{\min}$ may be numerically determined for protograph-based codes. Because the formulas in the previous section involve the number of copies Q instead of the number of code bits n, it is convenient to define the function

$$\tilde{r}(\tilde{\delta}) = \lim_{Q \to \infty} \sup \frac{\ln A_w}{Q}, \tag{8.50}$$

where $\tilde{\delta} = w/Q$. Note that $n = |S_t| \cdot Q$ and so

$$r(\delta) = \frac{1}{|S_t|} \tilde{r}(|S_t| \cdot \delta).$$

We also define $\max^*(x, y) \triangleq \ln(e^x + e^y)$ and we similarly define $\max^*$ when more than two variables are involved. When x and y are large and distinct (so that e^x and e^y are vastly different), then $\max^*(x, y) \simeq \max(x, y)$. Similar comments apply for more than two variables.

From (8.39), we have

$$\begin{aligned}
\ln A_w &= \max_{\{w_l:v_l\in S_t\}}{}^{*} \left\{ \max_{\{w_k:v_k\in S_p\}}{}^{*} \{\ln A(\mathbf{w})\} \right\}, \\
&\approx \max_{\{w_l:v_l\in S_t\}} \left\{ \max_{\{w_k:v_k\in S_p\}} \{\ln A(\mathbf{w})\} \right\}, \\
&= \max_{\{w_l:v_l\in S_t\}} \max_{\{w_k:v_k\in S_p\}} \left\{ \sum_{i=1}^{n_c} \ln A^{c_i}(\mathbf{w}_i) - \sum_{j=1}^{n_v} (q_{v_j}-1) \ln \binom{Q}{w_j} \right\},
\end{aligned}$$

under the constraint $\sum_{\{w_j:v_j\in S_t\}} w_j = w$. The second line holds when Q is large and the third line is obtained by invoking (8.38). Taking the limit as $Q \to \infty$ and applying the result (from Stirling's formula)

$$\lim_{Q\to\infty} \sup \ln \binom{Q}{w_j} \Big/ Q = H(\tilde{\delta}_j) = -(1-\tilde{\delta}_j)\ln(1-\tilde{\delta}_j) - \tilde{\delta}_j \ln \tilde{\delta}_j,$$

where $\tilde{\delta}_j = w_j/Q$, we obtain

$$\tilde{r}(\tilde{\delta}) = \max_{\{\tilde{\delta}_l:v_l\in S_t\}} \max_{\{\tilde{\delta}_k:v_k\in S_p\}} \left\{ \sum_{i=1}^{n_c} a^{c_i}(\tilde{\boldsymbol{\delta}}_i) - \sum_{j=1}^{n_v} (q_{v_j}-1) H(\tilde{\delta}_j) \right\} \tag{8.51}$$

under the constraint $\sum_{\{\tilde{\delta}_j:v_j\in S_t\}} \tilde{\delta}_j = \tilde{\delta}$. In (8.51), $a^{c_i}(\tilde{\boldsymbol{\delta}}_i)$ is the asymptotic weight-vector enumerator of the constraint node c_i, and $\tilde{\boldsymbol{\delta}}_i = \mathbf{w}_i/Q$. For a generic constituent CN code $\mathcal{C}$, such an enumerator is defined as

$$a^{\mathcal{C}}(\boldsymbol{\omega}) = \lim_{Q\to\infty} \sup \frac{\ln A^{\mathcal{C}}(\mathbf{w})}{Q}, \tag{8.52}$$

where $\boldsymbol{\omega} = \mathbf{w}/Q$.

We may obtain a simple expression for $a^{\mathcal{C}}(\boldsymbol{\omega})$ using the method of types; see Chapter 12 of [24]. We define the type $P_{\boldsymbol{\omega}}$ as the relative proportion of occurrences of each codeword of constituent CN code $\mathcal{C}$ in a sequence of Q codewords. In other words, $P_{\boldsymbol{\omega}} = [p_1, p_2, \ldots, p_K]$ is the empirical probability distribution of the codewords in $\mathcal{C}$ given a sequence of Q such codewords, where $p_k = n_k/Q$ and n_k is the number of occurrences of the kth codeword. Then the type class of $P_{\boldsymbol{\omega}}$, $T(P_{\boldsymbol{\omega}})$, is the set of all length-Q sequences of codewords in $\mathcal{C}$, each containing n_k occurrences of the kth codeword in $\mathcal{C}$, for $k = 1, 2, \ldots, K$. Observe that $|T(P_{\boldsymbol{\omega}})| = C(Q; n_1, n_2, \ldots, n_K)$. From [24, Theorem 12.1.3] $|T(P_{\boldsymbol{\omega}})| \to e^{Q\cdot H(P_{\boldsymbol{\omega}})}$ as $Q \to \infty$,

where $H(P_{\boldsymbol{\omega}}) = -\sum_{k=1}^{K} p_k \ln p_k$. Consequently, for $Q \to \infty$ we rewrite (8.46) as

$$\begin{aligned} A^{\mathcal{C}}(\mathbf{w}) &= \sum_{\{\mathbf{n}\}} C\left(Q; n_1, n_2, \ldots, n_K\right) \\ &= \sum_{\{P_{\boldsymbol{\omega}}\}} |T(P_{\boldsymbol{\omega}})| \\ &\to \sum_{\{P_{\boldsymbol{\omega}}\}} e^{Q \cdot H(P_{\boldsymbol{\omega}})}. \end{aligned} \tag{8.53}$$

It follows from (8.52) and (8.53) that

$$a^{\mathcal{C}}(\boldsymbol{\omega}) = \max_{\{P_{\boldsymbol{\omega}}\}} \{H(P_{\boldsymbol{\omega}})\}, \tag{8.54}$$

under the constraint that $\{P_{\boldsymbol{\omega}}\}$ is the set of solutions to $\boldsymbol{\omega} = P_{\boldsymbol{\omega}} \cdot \mathbf{M}^{\mathcal{C}}$, where $p_1, p_2, \ldots, p_K \geq 0$ and $\sum_{k=1}^{K} p_k = 1$. These are the asymptotic equivalents of the constraints mentioned below Equation (8.46).

Example 8.4. In this example we evaluate the asymptotic weight enumerators for several related protograph-based LDPC code ensembles. In particular, we first consider the (6,3) regular LDPC code ensemble, which has the protograph shown in Figure 8.10(a). (The simplest (6,3) protograph has two degree-3 VNs connected to a single degree-6 CN, but we use the form in Figure 8.10(a) so that we may build upon it in the rest of Figure 8.10.) Its asymptotic weight-enumerator curve appears in Figure 8.11, which shows that this ensemble has $\delta_{\min} = 0.023$. Consider also the asymptotic weight enumerator for a precoded version of the (6,3) ensemble, where precoding is performed by an accumulator (see the discussion of ARA codes in Chapter 6). The protograph for this ensemble is shown in Figure 8.10(b), and the asymptotic enumerator results are given in Figure 8.11, which shows that precoding increases $\delta_{\min}$ from 0.023 to 0.028.

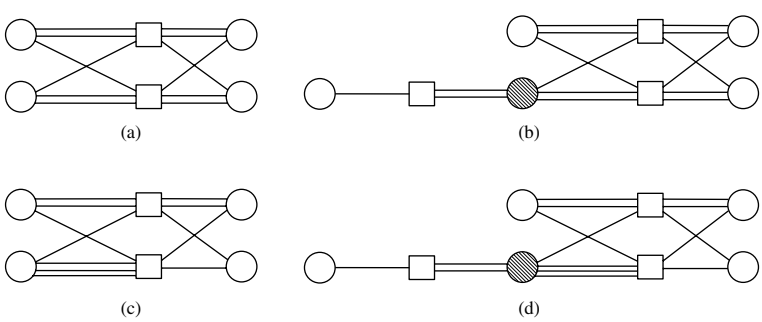


Figure 8.10 Protographs for several rate-1/2 LDPC codes: (a) (6,3) regular LDPC code, (b) precoded (6,3) regular LDPC code, (c) RJA LDPC code, and (d) ARJA LDPC code. (The shaded VNs represent punctured nodes.)

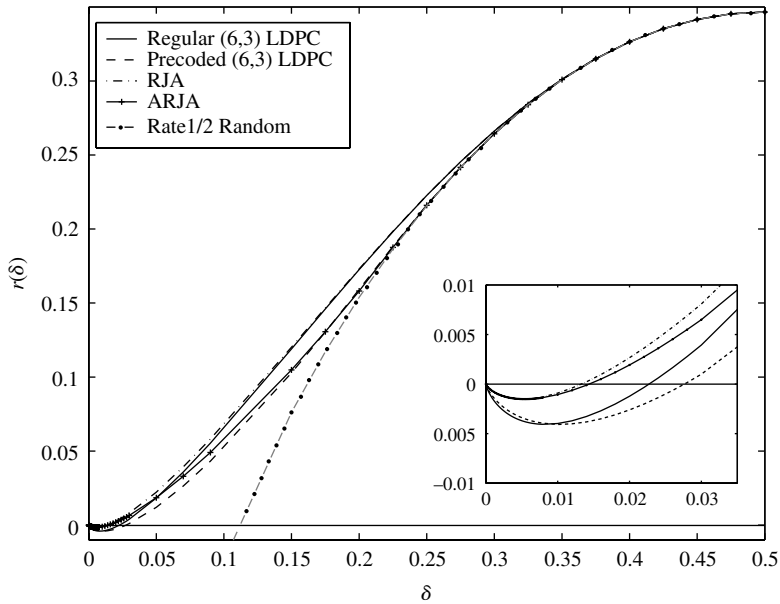


Figure 8.11 Asymptotic weight enumerators for several protograph-based LDPC code ensembles.

Next, we consider the RJA and the ARJA (precoded RJA) LDPC codes which have the protographs in Figures 8.10(c) and (d), respectively. The asymptotic weight enumerators for their ensembles appear in Figure 8.11. The latter figure shows that the RJA ensemble has $\delta_{\min} = 0.013$ and the ARJA has $\delta_{\min} = 0.015$.

Example 8.5. Consider the protograph in Figure 8.12, where the parity-check matrix $\mathbf{H}_1$ corresponds to the (7,4) Hamming code generated by $\mathbf{G}$ of Example 8.3. The parity-check matrix $\mathbf{H}_2$ corresponds to the following column-permutation of $\mathbf{H}_1$: $(6, 7, 1, 2, 3, 4, 5)$. That is, the first column of $\mathbf{H}_2$ is the sixth column of $\mathbf{H}_1, \ldots$, and the seventh column of $\mathbf{H}_2$ is the fifth column of $\mathbf{H}_1$. A code constructed per this protograph has rate 1/7 since each CN represents three redundant bits. The variable nodes $v_1, v_2, \ldots, v_7$ have the normalized weights $\tilde{\delta}_1, \tilde{\delta}_2, \ldots, \tilde{\delta}_7$. The asymptotic weight enumerator is $r(\delta) = \tilde{r}(7\delta)/7$, where

$$\tilde{r}(\tilde{\delta}) = \max_{\tilde{\delta}_1, \ldots, \tilde{\delta}_7} \left\{ a^{\mathbf{H}_1}(\tilde{\boldsymbol{\delta}}_1) + a^{\mathbf{H}_2}(\tilde{\boldsymbol{\delta}}_2) - \sum_{j=1}^{7} H(\tilde{\delta}_j) \right\}, \tag{8.55}$$

such that $\sum_{i=1}^{7} \tilde{\delta}_i = \tilde{\delta}$, and where $\tilde{\boldsymbol{\delta}}_1 = [\tilde{\delta}_1, \tilde{\delta}_2, \ldots, \tilde{\delta}_7]$ and $\tilde{\boldsymbol{\delta}}_2 = [\tilde{\delta}_6, \tilde{\delta}_7, \tilde{\delta}_1, \tilde{\delta}_2, \tilde{\delta}_3, \tilde{\delta}_4, \tilde{\delta}_5]$. We also evaluated the asymptotic weight enumerator for the rate-1/6 G-LDPC code which results from puncturing one of the VNs. The result is presented in Figure 8.13.

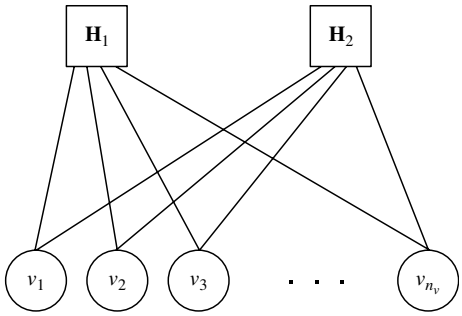


Figure 8.12 A rate-1/7, $n_v = 7$ (or rate-7/15, $n_v = 15$) G-LDPC protograph.

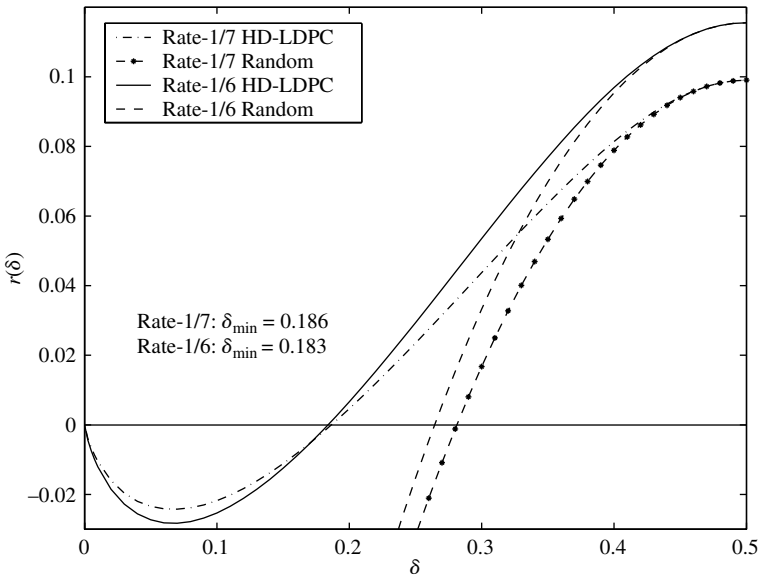


Figure 8.13 Asymptotic weight enumerators for the rate-1/7 (1/6) G-LDPC code.

Note that this ensemble has a relatively large $\delta_{\min}$. Consequently, a long code based on this protograph has, with probability near unity, a large minimum distance.

8.5.3 On the Complexity of Computing Asymptotic Ensemble Enumerators

The drawback of the above method for evaluating the asymptotic enumerators can be seen from (8.54): the number of the maximization arguments equals the number of the codewords K in the CN code $\mathcal{C}$. As an example, for the (15,11) Hamming code, the maximization is over $K = 2^{11} = 2048$ variables.

To alleviate this issue of having to maximize over a large number of variables, the following steps are considered. First, the protograph's VNs are partitioned

into subsets based on their neighborhoods. That is, two VNs belong to the same subset if and only if they have the same *type-neighborhood*, meaning that they are connected to an identical distribution of CN types. Given this partitioning of the set of VNs into subsets, the bits for a given CN code can themselves be partitioned into *bit subsets* in accordance with their membership in the VN subsets. Now, define the subset-weight vector (SWV) of a CN codeword as the vector whose components are the weights of bit subsets of the CN codeword. As an example, let $\bar{x} = [1010111]$ be a codeword of a length-7 CN code and let the CN code's bit subsets be $\{1, 2, 7\}, \{3, 4\}, \{5, 6\}$; then $\mathrm{SWV}(\bar{x}) = [2, 1, 2]$. Also, define the SWV enumerator as the number of CN codewords that have the same SWV.

Experimental results have led to the following conjecture which simplifies the computational complexity: *in the maximization in (8.54), the optimal point occurs when codewords of equal SWV have the same proportion of occurrence in the constituent CN code.* The implication is that most of the elements of $P_{\boldsymbol{\omega}} = [p_1, p_2, \ldots, p_K]$ are identical and so the maximization (8.54) is over a vastly reduced number of distinct variables.

This conjecture is used with some simple linear algebra to rewrite (8.54) as

$$a^{\mathcal{C}}(\boldsymbol{\omega}) = \max_{\{\hat{P}\}} \Big\{ H^*(\hat{P}) \Big\}, \tag{8.56}$$

under the constraint that $\{\hat{P}\}$ is the set of solutions to $\boldsymbol{\omega} = \hat{P} \cdot \hat{\mathbf{M}}^{\mathcal{C}}, p^{(1)}, p^{(2)}, \ldots \geq 0$ and $\Psi \cdot \hat{P}^{\mathrm{T}} = 1$, where $\hat{P} = [p^{(1)}, p^{(2)}, \ldots]$ is a vector of the distinct p_ks in $P_{\boldsymbol{\omega}}$, $\Psi = [\psi_1, \psi_2, \ldots]$ is a vector of the SWV enumerators of $\mathcal{C}$, and $\hat{\mathbf{M}}^{\mathcal{C}}$ is constructed from $\mathbf{M}^{\mathcal{C}}$ by adding the rows of $\mathbf{M}^{\mathcal{C}}$ having the same SWV. Note that it is possible to have identical columns in $\hat{\mathbf{M}}^{\mathcal{C}}$. This implies that the corresponding ω_ls in $\boldsymbol{\omega} = [\omega_1, \omega_2, \ldots, \omega_\mu]$ are equal. Finally, $H^*(\hat{P})$ is related to $H(P_{\boldsymbol{\omega}})$ as follows:

$$\begin{aligned} H(P_{\boldsymbol{\omega}}) &= -\sum_{k=1}^{K} p_k \ln p_k \\ &= -\sum \left(p^{(l)} \ln p^{(l)} \right) \cdot \psi_l \\ &\triangleq H^*(\hat{P}). \end{aligned}$$

Example 8.6. Consider again the protograph in Figure 8.12, but with (15,11) Hamming codes for the constraints $\mathbf{H}_1$ and $\mathbf{H}_2$, where

$$\mathbf{H}_1 = [\mathbf{M}_1 \quad \mathbf{M}_2] = \begin{bmatrix} 1\,0\,1\,0\,1\,0\,1\,0 & 1\,0\,1\,0\,1\,0\,1 \\ 0\,1\,1\,0\,0\,1\,1\,0 & 0\,1\,1\,0\,0\,1\,1 \\ 0\,0\,0\,1\,1\,1\,1\,0 & 0\,0\,0\,1\,1\,1\,1 \\ 0\,0\,0\,0\,0\,0\,0\,1 & 1\,1\,1\,1\,1\,1\,1 \end{bmatrix},$$

$$\mathbf{H}_2 = [\mathbf{M}_2 \quad \mathbf{M}_1].$$

Note that there are $K = 2048$ codewords in each constituent code, so it would be difficult to evaluate (8.54) for this code. However, we may apply the conjecture because all of the

VNs in this protograph have the same type-neighborhood. Also, after finding $\hat{\mathbf{M}}$, one finds that all of its columns are identical. Consequently, the VNs have the same normalized weight δ.

The asymptotic weight enumerator is

$$r(\delta) = \frac{1}{15} \max_{\delta} \Big\{ 2a^{\mathbf{H}_1}(\tilde{\boldsymbol{\delta}}) - 15H(\delta) \Big\},$$

where $\tilde{\boldsymbol{\delta}} = [\delta, \delta, \ldots, \delta]$ (15 of them). Now, to find $a^{\mathbf{H}_1}(\tilde{\boldsymbol{\delta}})$, define $p^{(\rho)}$ as the proportion of occurrence of a codeword of weight ρ in the constituent CN code, so $\hat{P} = [p^{(0)}, p^{(3)}, p^{(4)}, p^{(5)}, p^{(6)}, p^{(7)}, p^{(8)}, p^{(9)}, p^{(10)}, p^{(11)}, p^{(12)}, p^{(15)}]$ and $\Psi = [1, 35, 105, 168, 280, 435, 435, 280, 168, 105, 35, 1]$. Consequently,

$$a^{\mathbf{H}_1}(\tilde{\boldsymbol{\delta}}) = \max_{\{\hat{P}\}} \Big\{ H^*(\hat{P}) \Big\}, \tag{8.57}$$

under the constraint that $\{\hat{P}\}$ is the set of solutions to $\tilde{\boldsymbol{\delta}} = \hat{P} \cdot \hat{\mathbf{M}}^{\mathcal{C}}$, $p^{(\rho)} \geq 0$, for all $p^{(\rho)}$ in $\hat{P}$, and $\Psi \cdot \hat{P}^{\mathrm{T}} = 1$. Clearly, under these assumptions, the computation of $a^{\mathbf{H}_1}(\delta)$, and hence of $r(\delta)$, is vastly simplified. The $r(\delta)$ result appears in Figure 8.14. Also included in Figure 8.14 is $r(\delta)$ for a rate-1/2 code obtained by puncturing one bit in the protograph. Note that, for both cases, $d_{\min}$ for the ensemble increases linearly with n.

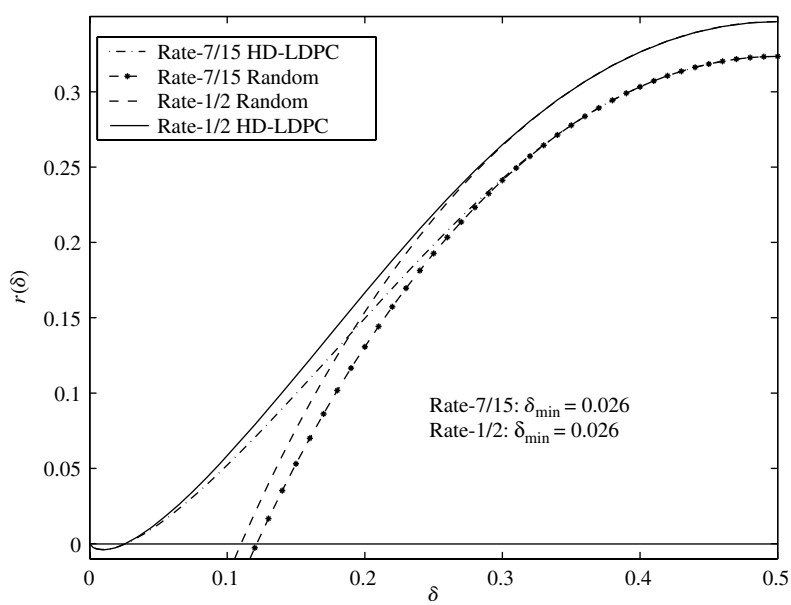


Figure 8.14 Asymptotic weight enumerators for the rate-7/15 (1/2) G-LDPC code.

8.5.4 Ensemble Trapping-Set Enumerators

Classically, code design has involved finding code ensembles with large minimum distances. For modern codes with iterative decoders, additional work is necessary because iterative decoding gives rise to decoder pitfalls as a result of their distributed processing nature. The decoder pitfall associated with iterative decoders is a trapping set (see Chapter 5). An (a, b) *general trapping set* [25], $\mathcal{T}_{a,b}$, is a set of VNs of size a, which induce a subgraph with exactly b odd-degree check nodes (and an arbitrary number of even-degree check nodes). We use the qualifier "general" to make the distinction from elementary trapping sets, to be defined later. On the AWGN channel, an LDPC code will suffer from an error floor due to a small minimum distance or to trapping sets unless one selects from a "good" ensemble a code that has a large minimum distance (grows linearly with code length) and trapping sets with a negligible probability of occurrence.

8.5.4.1 Finite-size Trapping-Set Enumerators

The method for determining trapping-set enumerators for an LDPC code ensemble characterized by a given protograph involves first creating a modified protograph from the original protograph. Then the codeword weight enumerator for the modified protograph is determined, from which the trapping-set enumerator may be obtained. We describe the technique as follows.

Assume that we are interested in trapping sets of weight a in the graph G in Figure 8.9. The value of the companion parameter b depends on which a VNs are of interest, so we set to "1" the values of the a VNs of interest and set to "0" the values of the remaining VNs. With the a VNs so fixed, we are now interested in which CNs "see" an odd weight among their neighboring VNs, for the number of such CNs is the corresponding parameter b. Such odd-weight CNs can be identified by the addition of auxiliary "flag" VNs to each CN (see protograph G' in Figure 8.15), where the value of an auxiliary VN equals "1" exactly when its corresponding original CN sees odd weight. The number of auxiliary VNs equal to "1" is precisely the parameter b for the set of a VNs that were set to "1". Thus, to obtain the trapping-set enumerator for the original protograph G, one may apply the codeword weight-enumerator technique to G', partitioning the set of VNs into the original set $(S_t \cup S_p)$ and an auxiliary flag set (S_f), much like we had earlier partitioned the set of VNs into subsets of transmitted and punctured VNs.

Note that the set $S_t \cup S_p$ (the VNs at the bottom of Figure 8.15) accounts for the weight a and the set S_f (the VNs at the top of Figure 8.15) accounts for the weight b. Also note that, in counting the number of trapping sets, we do not distinguish between transmitted and punctured VNs (refer to the definition of a trapping set). However, in evaluating the failure rate of a trapping set, one should make this distinction.

On the basis of the above discussion and (8.39), the trapping-set enumerator $A_{a,b}$ is given by

$$A_{a,b} = \sum_{\{w_j : v_j \in S_t \cup S_p\}} \sum_{\{w_k : v_k \in S_f\}} A(\mathbf{w}) \tag{8.58}$$

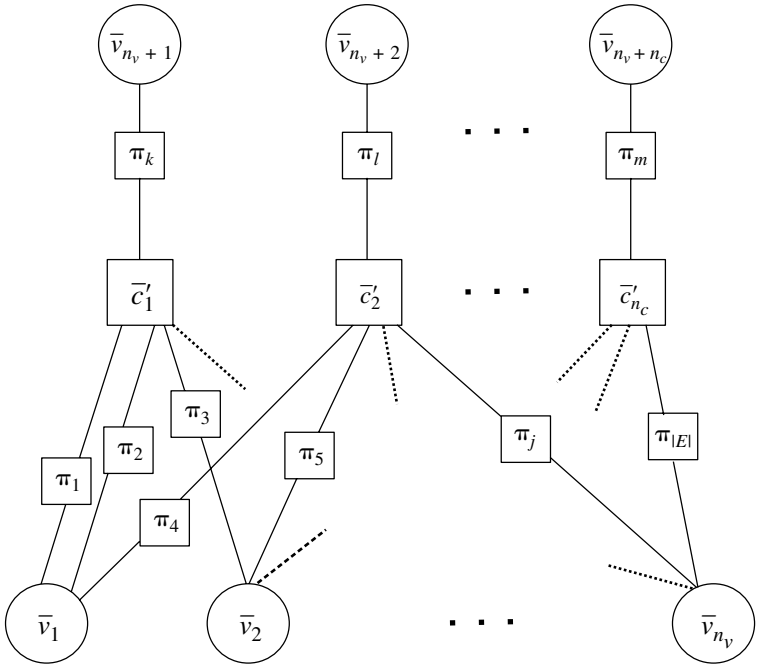


Figure 8.15 A modified vectorized protograph G' for trapping-set enumeration.

under the constraints $\sum_{\{w_j:v_j\in S_t\cup S_p\}} w_j = a$ and $\sum_{\{w_j:v_j\in S_f\}} w_j = b$, where

$$A(\mathbf{w}) = \frac{\prod_{i=1}^{n_c} A^{c'_i}(\mathbf{w}_i)}{\prod_{j=1}^{n_v} \binom{Q}{d_j}^{q_{v_j}-1}}. \tag{8.59}$$

Notice the use of c'_i instead of c_i in (8.59) to indicate that the weight-vector enumerators in (8.59) are the CNs in G'. Those weight-vector enumerators can be evaluated using (8.46).

8.5.4.2 *Elementary Trapping-Set Enumerators*

An *elementary* (a, b) *trapping set*, $\mathcal{T}^{(e)}_{(a,b)}$, is a set of VNs that induces a subgraph with only degree-1 and degree-2 check nodes, and exactly b of these CNs have degree 1. It has frequently been observed that the LDPC code error floors are due to elementary trapping sets. Some examples of elementary trapping sets are the (12,4) trapping set in the (2640,1320) Margulis code, and the (4,4) and (5,3) trapping sets in the (1008,504) and (816,408) MacKay codes. As a result, it is desired to compute the trapping-set enumerators just for the elementary trapping sets in the code ensembles.

To find elementary trapping-set enumerators, note that the only difference between the elementary and the general trapping set is the constraints on the

degrees of the CNs in the induced subgraph. This can be taken care of by choosing a proper matrix $\mathbf{M}^C$ in (8.46) and (8.54) when evaluating the weight-vector enumerators in (8.59) and the asymptotic weight-vector enumerators in (8.62), respectively. Consequently, the discussion regarding general trapping-set enumerators is valid for elementary trapping-set enumerators after redefining $\mathbf{M}^C$.

To find $\mathbf{M}^C$ for the case of elementary trapping sets, note that, when the bits of an elementary trapping set are set to "1" with all other bits in G set to "0", the CNs in G can see only certain bit patterns. Specifically, the set of patterns that can be seen by a degree-q_c CN c in G consists of the all-zeros word, the $\binom{q_c}{1}$ weight-1 words, and the $\binom{q_c}{2}$ weight-2 words. This implies that the possible patterns for the corresponding degree-(q_c+1) CN c' in G' are the all-zeros pattern and the $\binom{q_c+1}{2}$ patterns of weight-2. Therefore, $\mathbf{M}^C$ contains the all-zeros pattern and all of the weight-2 patterns.

8.5.4.3 *Asymptotic Trapping-Set Enumerators*

Analogous to the weight-enumerator case, define the *normalized logarithmic asymptotic trapping-set enumerator* (we will simply call it the asymptotic trapping-set enumerator), $r(\alpha, \beta)$, as

$$r(\alpha, \beta) = \lim_{n\to\infty} \sup \frac{\ln A_{a,b}}{n}, \tag{8.60}$$

where $\alpha = a/n$ and $\beta = b/n$ (recall that $n = |S_t| \cdot Q$ is the transmit code length). The derivation of an expression for (8.60) from (8.58) uses the same steps as were used in deriving (8.47) from (8.39). Thus, we omit the derivation and present the final result:

$$r(\alpha, \beta) = \frac{1}{|S_t|}\tilde{r}(\alpha|S_t|, \beta|S_t|), \tag{8.61}$$

where

$$\tilde{r}(\tilde{\alpha}, \tilde{\beta}) = \max_{\{\tilde{\delta}_l : v_l \in S_t \cup S_p\}} \left\{ \max_{\{\tilde{\delta}_k : v_k \in S_f\}} \left\{ \sum_{i=1}^{n_c} a^{c'_i}(\tilde{\boldsymbol{\delta}}_i) - \sum_{j=1}^{n_v} (q_{v_j} - 1) H(\tilde{\delta}_j) \right\} \right\}, \tag{8.62}$$

under the constraints $\sum_{\{\tilde{\delta}_j : v_j \in S_t \cup S_p\}} \tilde{\delta}_j = \tilde{\alpha}$ and $\sum_{\{\tilde{\delta}_j : v_j \in S_f\}} \tilde{\delta}_j = \tilde{\beta}$. The asymptotic weight-vector enumerator, $a^{c'_i}(\tilde{\boldsymbol{\delta}}_i)$, can be evaluated using (8.54). (To avoid ambiguity, note the use of $a^c(\cdot)$ to refer to the asymptotic weight-vector enumerator and the use of a to refer to the size of the (a,b) trapping set.) It is possible to establish an analogy to the typical minimum distance in the weight-enumerator problem, but this is beyond our scope.

8.5.5 Ensemble Stopping-Set Enumerators

In the context of iterative decoding of LDPC codes on the BEC, a *stopping set* S is a subset of the set of VNs whose neighboring SPC nodes of S are connected

to S at least twice. The implication of this definition is that, if all of the bits in a stopping set are erased, the iterative erasure decoder will never resolve the values of those bits. For *single-CN-type* G-LDPC codes, a *generalized stopping set*, S^d, is a subset of the set of VNs, such that all neighbors of S^d are connected to S^d at least d times, where $d-1$ is the erasure capability of the neighboring CNs.

In this section, in addition to single-CN-type G-LDPC codes, we consider multi-CN-type G-LDPC codes, for which a mixture of CN types is permissible. We assume the use of a standard iterative erasure decoding tailored to the Tanner graph of the code, with bounded-distance algebraic decoders employed at the CNs. That is, CN c_i can correct up to $d_{\min}^{(\mathrm{i})}-1$ erasures. A set of erased VNs will fail to decode exactly when all of the neighboring CNs see more erasures than they are capable of resolving. Consequently, we introduce a new definition for the generalized stopping set as follows. A generalized stopping set, S^D, of a G-LDPC code is a subset of the set of VNs, such that every neighboring CN c_i of S^D is connected to S^D at least $d_{\min}^{(\mathrm{i})}$ times, for all c_i in the neighborhood of S^D, where D is the set of $d_{\min}^{(\mathrm{i})}$s for the neighboring CNs. Hereafter, we will use the term stopping set to refer to a generalized stopping set.

Much like in the trapping-set case, the method for finding weight enumerators for protograph-based G-LDPC code ensembles can be leveraged to obtain stopping-set enumerators for these ensembles. To see how, let us consider the mapping ϕ from the set of stopping sets $\{S^D\}$ to $\mathbb{F}_2^n$ defined as $\phi(S^D)=[x_0,x_1,\ldots,x_{n-1}]$, where $x_j=1$, if and only if the corresponding VN v_j in the Tanner graph is in the stopping set S^D. The set of all binary words $\{\phi(S^D)\}$ corresponding to a Tanner graph G need not form a linear code, although the set is closed under bit-wise OR since the set of stopping sets $\{S^D\}$ is closed under unions. We call the (nonlinear) code $\{\phi(S^D)\}$ induced by the set of stopping sets of G under the map ϕ a *stopping-set code* and we denote it by $C_{ss}(G)=\left\{\phi(S^D)\right\}$. Note that the weight of $\phi(S^D)$ equals the size of S^D, so the weight enumerator for the stopping-set code $C_{ss}(G)$ is identical to the stopping-set enumerator for G. For example, in a graph G with a single CN $\mathcal{C}$, where $\mathcal{C}$ is a (μ,κ) linear block code of minimum distance $d_{\min}$, the stopping-set code is $C_{ss}(G)=\{\bar{x}\in\mathbb{F}_2^{\mu}\colon \mathrm{weight}(\bar{x})\geq d_{\min}\}$. Thus, the stopping-set enumerator for this simple graph is exactly the weight enumerator for $C_{ss}(G)$. This simple example also lets us speak of a stopping-set code for a single CN c_i, which is $C_{ss}(c_i)=\{\bar{x}\in\mathbb{F}_2^{\mu}\colon \mathrm{weight}(\bar{x})\geq d_{\min}^{(\mathrm{i})}\}$.

In the case of a non-trivial G-LDPC Tanner graph G, we would first find the stopping-set codes $C_{ss}(c_i)$ for each of the CNs c_i. We then form a new graph, G', from G by replacing CN c_i in G by $C_{ss}(c_i)$, for all i. The stopping-set enumerator for G is then given by the weight enumerator for $C_{ss}(G')$, which is given by $C_{ss}(G')=\{\bar{x}\in\mathbb{F}_2^n\colon \bar{x}$ satisfies the constraints of $G'\}$. In summary, the ensemble *stopping-set enumerator* for a protograph-based G-LDPC code with graph G is identically the ensemble *weight enumerator* for the graph G' formed from G by replacing the CNs in G by their corresponding stopping-set code constraint nodes.

Problems

8.1 Consider the (7,4) Hamming code generated in systematic from by the generator $g(x) = 1 + x + x^3$. Plot the BER P_b and FER P_{cw} using (8.3) and (8.4) for P_b and (8.1) and (8.2) for P_{cw}.

8.2 Consider the (8,4) extended Hamming code generated in systematic form by the generator $g(x) = 1 + x^2 + x^3 + x^4$. Find the following enumerators for this code: $A(W)$, $A(I, P)$, $A_i(P)$ for $i = 1, 2, 3, 4$; $A(I, W)$, $A_i(W)$ for $i = 1, 2, 3, 4$; and $B(W)$, $B(I, P)$, $B_i(P)$ for $i = 1, 2, 3, 4$.

8.3 Plot the P_b ensemble bound for PCCC_1 whose constituent codes both have the RSC generator matrix

$$G_1(D) = \begin{bmatrix} 1 & \dfrac{1}{1+D} \end{bmatrix}.$$

Compare this with the P_b ensemble bound for PCCC_2 whose constituent codes both have the $\bar{\text{R}}$SC generator matrix

$$G_2(D) = [1 + D \quad 1].$$

Assume an information block of 200 bits in both cases.

8.4 Plot the P_b bound for an SCCC ensemble whose outer code has the $\bar{\text{R}}$SC generator matrix $G(D) = [1 + D \quad 1]$ and whose inner code is the rate-1 accumulator with transfer function $1/(1 + D)$. Assume an information block of 200 bits.

8.5 Do the previous problem but now with outer code generator matrix $G(D) = \begin{bmatrix} 1 + D + D^2 & 1 + D \end{bmatrix}$.

8.6 Plot the P_b and P_{cw} bounds for the (non-systematic) rate-1/3 RA code ensemble with codeword length 900.

8.7 Find the input–output weight-enumerator coefficients $A_{i,w}$ for a rate-$1/(q + 1)$ *systematic* RA code.

8.8 Plot the P_b bounds for the rate-1/2 (2000,1000) IRA code ensembles with column-weight parameter $w_c = 3$, 4, and 5.

8.9 Consider the protograph-based LDPC codes considered in Section 8.5. For a given protograph LDPC code ensemble, show that the fraction of such codes that are quasi-cyclic goes to zero as the code length goes to infinity.

8.10 Consider a (3,2) SPC constraint node in a protograph. Find the vector weight enumerator $A(\mathbf{w})$ for $Q = 3$ copies of this code.

8.11 Consider a (4,3) SPC constraint node in a protograph. Find the vector weight enumerator $A(\mathbf{w})$ for $Q = 3$ copies of this code.

8.12 Show that the weight enumerator for a rate-R, length-n random linear code ensemble is

$$A_w = \binom{n}{w} 2^{-n(1-R)}.$$

From this and Stirling's approximation of $n!$ for large n, namely, $n! \simeq \sqrt{2\pi n} n^n / e^n$, show that, for random linear codes,

$$r(\delta) = H(\delta) - (1-R)\ln(2),$$

where $H(\delta) = -(1-\delta)\ln(1-\delta) - \delta \ln \delta$ and $\delta = w/n$.

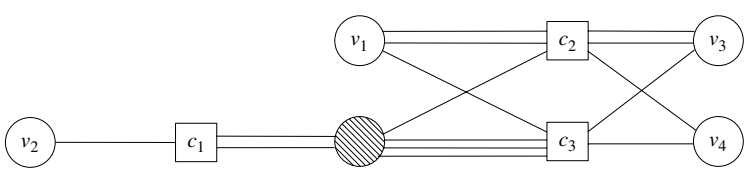
8.13 (Project) Reproduce the asymptotic weight enumerator curve $r(\delta)$ versus δ for the regular (6,3) code ensemble that appears in Figure 8.11. The (6,3) protograph has two degree-3 VNs connected to a single degree-6 CN. Your result should show that $\delta_{\min} = 0.023$.

8.14 (With acknowledgment of D. Divsalar and S. Abu-Surra) (a) Show that the weight-vector enumerator for Q copies of a (4,3) SPC constraint node can be written as

$$A_Q^{(4,3)}(\mathbf{w}) = \sum_{l=1}^{Q} \frac{A_Q^{(3,2)}([w_1, w_2, l]) A_Q^{(3,2)}([w_3, w_4, l])}{\binom{Q}{l}},$$

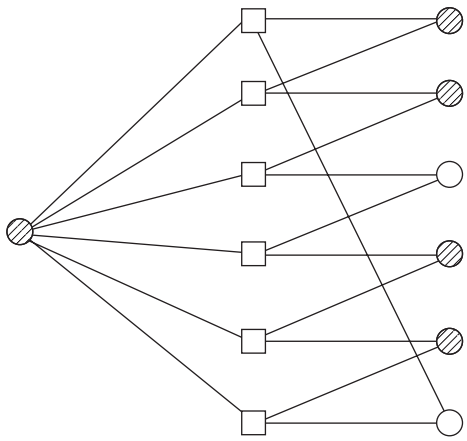
where $A_Q^{(k+1,k)}(\mathbf{w}) = A_Q^{(k+1,k)}([w_1, w_2, \ldots, w_{k+1}])$ is the vector weight enumerator for Q copies of a $(k+1,k)$ SPC constraint node. (Hint: Consider a concatenation of constraint nodes.) (b) Find an expression for the weight-vector enumerator for Q copies of a (6,5) SPC constraint node in terms of $A_Q^{(3,2)}(\mathbf{w})$.

8.15 (With acknowledgement of D. Divsalar and S. Abu-Surra) (a) For the protograph below, argue that the optimizing values for $\tilde{\boldsymbol{\delta}}_1$ and $\tilde{\boldsymbol{\delta}}_2$ in (8.51) for VNs v_1 and v_3, respectively, must be equal. (b) Find $\hat{\mathbf{M}}$ and Ψ, as defined in (8.56) for the constraint node c_2. Use $\hat{\mathbf{M}}$ to explain your answer part (a).

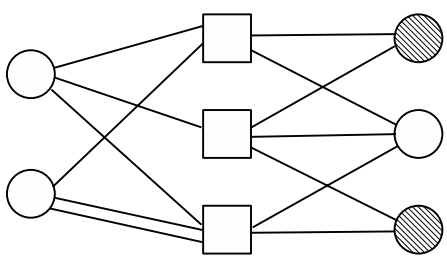


8.16 (With acknowledgement of D. Divsalar and C. Jones) The figure below presents the protograph of a rate-1/6 RA code punctured to rate-1/2. (The shaded VNs are punctured.) Show that the punctured protograph can be reduced to an

equivalent, simpler protograph with one (punctured) degree-6 VN, two degree-5 CNs, and two degree-2 VNs. Sketch this equivalent protograph.



8.17 (With acknowledgement of S. Abu-Surra) Argue that the minimum distance increases linearly with code length for the rate-2/5 ensemble corresponding to the protograph below, without the puncturing indicated. Hint: Using the ideas of the previous problem and Gallager's result for regular LDPC codes, show that the rate-2/3 ensemble corresponding to the punctured case is such that $d_{\min}$ grows linearly with code length. Then use the fact that the rate-2/3 ensemble's codewords are "sub-words" of the 2/5 ensemble's codewords.



8.18 Let $\lambda(X)$ and $\rho(X)$ be the edge-perspective degree distributions for an LDPC code ensemble as described in Chapter 5. Let $\lambda'(X)$ and $\rho'(X)$ be their (formal) derivatives. (a) What subset of variable nodes does $\lambda'(0)$ enumerate? (b) In [13] it is shown that, if $\lambda'(0)\rho'(1) < 1$, then $d_{\min}$ grows linearly with code length n with probability $\sqrt{\lambda'(0)\rho'(1)}$. Use this condition to check for linear $d_{\min}$ growth of the following rate-1/2 ensembles presented in Chapter 6: RA, IRA, and ARJA.

References

[1] S. Benedetto and G. Montorsi, "Unveiling turbo codes: some results on parallel concatenated coding schemes," *IEEE Trans. Information Theory*, vol. 42, no. 3, pp. 409–428, March 1996.

[2] S. Benedetto and G. Montorsi, "Design of parallel concatenated codes," *IEEE Trans. Communications*, vol. 44, no. 5, pp. 591–600, May 1996.

[3] S. Benedetto, D. Divsalar, G. Montorsi, and F. Pollara, "Serial concatenation of interleaved codes: performance analysis, design, and iterative decoding," *IEEE Trans. Information Theory*, vol. 44, no. 5, pp. 909–926, May 1998.

[4] S. Lin and D. J. Costello, Jr., *Error Control Coding*, 2nd edn., New Saddle River, NJ, Prentice-Hall, 2004.

[5] D. Divsalar, H. Jin, and R. McEliece, "Coding theorems for turbo-like codes," *Proc. 36th Annual Allerton Conf. on Communications Control, and Computing*, September 1998, pp. 201–210.

[6] H. Jin, A. Khandekar, and R. McEliece, "Irregular repeat–accumulate codes," *Proc. 2nd. Int. Symp. on Turbo Codes and Related Topics*, Brest, France, September 4, 2000, pp. 1–8.

[7] Y. Zhang, W. E. Ryan, and Y. Li, "Structured eIRA codes," *38th Asilomar Conf. on Signals, Systems and Computers*, November 2004, pp. 2005–2009.

[8] Y. Zhang and W. E. Ryan, "Structured IRA codes: performance analysis and construction," *IEEE Trans. Communications*, vol. 55, no. 5, pp. 837–844, May 2007.

[9] D. Divsalar, "Ensemble weight enumerators for protograph LDPC codes," *IEEE Int. Symp. on Information Theory*, July 2006, pp. 1554–1558.

[10] S. Abu-Surra, W. E. Ryan, and D. Divsalar, "Ensemble weight enumerators for protograph-based generalized LDPC codes," UCSD ITA Workshop, January 2007, pp. 342–348.

[11] S. Abu-Surra, W. E. Ryan, and D. Divsalar, "Ensemble trapping set enumerators for protograph-based LDPC codes," *Proc. 45th Annual Allerton Conf. on Communications Control, and Computing*, October 2007.

[12] S. Abu-Surra, W. E. Ryan, and D. Divsalar, "Ensemble enumerators for protograph-based generalized LDPC codes," *2007 IEEE Global Telecommunications Conf.*, November 2007, pp. 1492–1497.

[13] C. Di, T. Richardson, and R. Urbanke, "Weight distribution of low-density parity-check codes," *IEEE Trans. Information Theory*, vol. 52, no. 11, pp. 4839–4855, November 2006.

[14] O. Milenkovic, E. Soljanin, and P. Whiting, "Asymptotic spectra of trapping sets in regular and irregular LDPC code ensembles," *IEEE Trans. Information Theory*, vol. 53, no. 1, pp. 39–55, January 2007.

[15] T. Richardson and R. Urbanke, *Modern Coding Theory*, Cambridge, Cambridge University Press, 2008.

[16] D. Divsalar and C. Jones, "Protograph based low error floor LDPC coded modulation," *2005 IEEE MilCom Conf.*, October 2005, pp. 378–385.

[17] D. Divsalar, C. Jones, S. Dolinar, and J. Thorpe, "Protograph based LDPC codes with minimum distance linearly growing with block size," *2005 IEEE GlobeCom Conf.*, 5 pp. November–December 2006.

[18] D. Divsalar, S. Dolinar, and C. Jones, "Construction of protograph LDPC codes with linear minimum distance," *2006 Int. Symp. Information Theory*, July 2006, pp. 664–668.

[19] D. Divsalar, S. Dolinar, and C. Jones, "Protograph LDPC codes over burst erasure channels," *2006 IEEE MilCom Conf.*, October 2006, pp. 1–7.

[20] D. Divsalar and C. Jones, "Protograph LDPC codes with node degrees at least 3," *2006 IEEE GlobeCom Conf.*, November 2006, pp. 1–5.

[21] D. Divsalar, S. Dolinar, and C. Jones, "Short protograph-based LDPC codes," *2007 IEEE MilCom Conf.*, October 2007, pp. 1–6.
[22] R. G. Gallager, *Low-Density Parity-Check Codes*, Cambridge, MA, MIT Press, 1963.
[23] S. Abu-Surra, D. Divsalar, and W. E. Ryan, "Ensemble enumerators for protograph-based generalized LDPC codes," in preparation for submission to *IEEE Trans. Information Theory.*
[24] T. Cover and J. Thomas, *Elements of Information Theory*, New York, Wiley, 1991.
[25] T. Richardson, "Error-floors of LDPC codes," *41st Annual Allerton Conf. on Communications Control, and Computing*, September 2003.

9 Ensemble Decoding Thresholds for LDPC and Turbo Codes

The previous chapter examined code properties responsible for the floor region (high-SNR region) of LDPC and turbo codes. Specifically, the emphasis was on the computation of various weight enumerators for LDPC and turbo code ensembles because a poor weight spectrum leads to poor performance in a code's floor region both for iterative decoders and for maximum-likelihood decoders. That chapter also introduced ensemble enumerators for trapping sets and stopping sets, both of which can lead to poor floor performance in iterative decoders. In this chapter we examine the iterative decoding performance of LDPC and turbo code ensembles in the complementary low-SNR region, the "waterfall" region. We show that the iterative decoding of long LDPC and turbo codes displays a threshold effect in which communication is reliable beyond this threshold and unreliable below it. The threshold is a function of code ensemble properties and the tools introduced in this chapter allow the designer to predict the decoding threshold and its gap from Shannon's limit. The ensemble properties for LDPC codes are the degree distributions which are typically the design targets for the code-design techniques presented in subsequent chapters. The ensemble properties for turbo codes are the selected constituent codes. Our development borrows heavily from the references listed at the end of the chapter and our focus is on the binary-input AWGN channel. The references and the problems consider other channels.

9.1 Density Evolution for Regular LDPC Codes

We first summarize the main results of [1] with regard to iterative (message-passing) decoding of long, regular LDPC codes.

1. A *concentration theorem* is proven. This asserts that virtually all codes in an ensemble have the same behavior, so that the performance prediction of a specific (long) code is possible via the ensemble average performance.
2. For long codes, this average performance is equal to the performance of cycle-free codes, which is computable via an algorithm called density evolution. *Density evolution* refers to the evolution of the probability density functions (pdfs) of the messages being passed around in the iterative decoder, where the messages are modeled as random variables (r.v.s). Such knowledge of the pdfs allows

one to predict under which channel conditions (e.g., SNRs) the decoder bit error probability will converge to zero.

3. Long codes possess a *decoding threshold*, which partitions the channel parameter (e.g., SNR) space into one region for which reliable communication is possible and another region for which it is not. The density-evolution algorithm allows us to determine the decoding threshold of an LDPC code ensemble.

In this section we present the computation, via density evolution, of the decoding thresholds for long, regular LDPC codes with variable-node (VN) degree d_v and check-node (CN) degree d_c. The approach is quite general, and applies to a number of binary-input, symmetric-output channel models, although we focus primarily on the binary-input AWGN channel. The symmetric-output channel description for the binary-input AWGN channel means that the channel transition pdf satisfies $p(y|x=+1) = p(-y|x=-1)$. A similar relationship holds for the binary-symmetric channel (BSC) and other channels. Symmetry conditions are also required of the iterative decoder [1], all of which are satisfied by the sum–product algorithm, so we need not detail them here. It will be assumed throughout that the all-zeros codeword $\mathbf{c} = [0 \quad 0 \quad \ldots \quad 0]$ is sent. Under the mapping $x = (-1)^c$, this means that the all-ones word $\mathbf{x} = [+1 \quad +1 \quad \ldots \quad +1]$ is transmitted on the channel.

As mentioned above, the tool for determining the decoding threshold of a (d_v,d_c)-regular LDPC code is the density-evolution algorithm. Because $\mathbf{x} = [+1 \quad +1 \quad \ldots \quad +1]$ is transmitted, an error will be made at the decoder after the maximum number of iterations if any of the signs of the cumulative variable node LLRs, L_j^{total}, are negative. Let $p_\ell^{(v)}$ denote the pdf of a message $m^{(v)}$ to be passed from VN v to some check node during the ℓth iteration.[1] Note that, under the above symmetry conditions for channel and decoder, the pdfs are identical for all such outgoing VN messages. Then, no decision error will occur after an infinite number of iterations if

$$\lim_{\ell\to\infty} \int_{-\infty}^{0} p_\ell^{(v)}(\tau)d\tau = 0, \tag{9.1}$$

because the probability that any L_j^{total} is negative is zero when (9.1) holds. Note that $p_\ell^{(v)}(\tau)$ depends on the channel parameter, which we denote by α. For example, α is the crossover probability ε for the BSC and α is the standard deviation σ of the channel noise for the AWGN channel. Then the decoding threshold α^* is given by

$$\alpha^* = \sup\left\{\alpha\colon \lim_{\ell\to\infty} \int_{-\infty}^{0} p_\ell^{(v)}(\tau)d\tau = 0\right\}$$

[1] $m^{(v)}$ is one of the LLRs $L_{j\to i}$ from Chapter 5, but we avoid the latter notation in most of this chapter, since otherwise it would become overly cumbersome. Similarly, below we use $m^{(c)}$ for $L_{i\to j}$ to simplify the notation.

under the assumption that the codeword length $n \to \infty$. For the AWGN case, we will also say that the SNR, or E_b/N_0, value corresponding to $\alpha^* = \sigma^*$ is the decoding threshold. It is shown in [1] that if $\alpha > \alpha^*$ then Pr(error) will be bounded away from zero; otherwise, from the definition of α^*, when $\alpha < \alpha^*$, Pr(error) $\to 0$ as $\ell \to \infty$.

We now develop the density-evolution algorithm for computing $p_\ell^{(v)}(\tau)$. We start by recalling that an outgoing message from VN v may be written as

$$m^{(v)} = m_0 + \sum_{j=1}^{d_v-1} m_j^{(c)} = \sum_{j=0}^{d_v-1} m_j^{(c)}, \tag{9.2}$$

where $m_0 \equiv m_0^{(c)}$ is the message from the channel and $m_1^{(c)}, \ldots, m_{d_v-1}^{(c)}$ are the messages received from the $d_v - 1$ neighboring CNs. Now let $p_j^{(c)}$ denote the pdfs for the d_v incoming messages $m_{0,}^{(c)}, \ldots, m_{d_v-1}^{(c)}$. Then, ignoring the iteration-count parameter ℓ, we have from (9.2)

$$\begin{aligned} p^{(v)} &= p_0^{(c)} * p_1^{(c)} * \cdots * p_{d_v-1}^{(c)} \\ &= p_0^{(c)} * \left[p^{(c)}\right]^{*(d_v-1)}, \end{aligned} \tag{9.3}$$

where independence among the messages is assumed and $*$ denotes convolution. The $(d_v - 1)$-fold convolution in the second line follows from the fact that the pdfs $p_j^{(c)}$ are identical because the code ensemble is regular, and we write $p^{(c)}$ for this common pdf. The computation of $p^{(v)}$ in (9.3) may be done via the fast Fourier transform according to

$$p^{(v)} = \mathcal{F}^{-1}\left\{\mathcal{F}\left\{p_0^{(c)}\right\}\left[\mathcal{F}\left\{p^{(c)}\right\}\right]^{d_v-1}\right\}. \tag{9.4}$$

At this point, we have a technique for computing the pdfs $p^{(v)}$ from the pdfs of the messages $m^{(c)}$ emanating from the CNs.

We need now to develop a technique to compute the pdf $p^{(c)}$ for a generic message $m^{(c)}$ emanating from a check node. From Chapter 5, we may write for $m^{(c)}$

$$m^{(c)} = \left(\prod_{i=1}^{d_c-1} \tilde{s}_i^{(v)}\right) \cdot \phi\left(\sum_{i=1}^{d_c-1} \phi\left(\left|m_i^{(v)}\right|\right)\right), \tag{9.5}$$

where $\tilde{s}_i^{(v)} = \operatorname{sign}\left(m_i^{(v)}\right) \in \{+1, -1\}$ and $\phi(x) = -\ln\tanh(x/2)$. The messages $m_i^{(v)}$ are received by CN c from $d_c - 1$ of its d_c VN neighbors. We define $s_i^{(v)}$ informally as $s_i^{(v)} = \log_{-1}\left(\tilde{s}_i^{(v)}\right)$ so that $s_i^{(v)} = 1$ when $\tilde{s}_i^{(v)} = -1$ and $s_i^{(v)} = 0$ when

$\tilde{s}_i^{(v)} = +1$. Then the first factor in (9.5) is equivalently

$$s^{(c)} = \sum_{i=1}^{d_c-1} s_i^{(v)} \pmod 2.$$

Note also that the second factor in (9.5) is non-negative so that $s^{(c)}$ serves as the sign bit for the message $m^{(c)}$ and the second factor serves as its magnitude, $\left|m^{(c)}\right|$.

In view of this, to simplify the mathematics to follow, we represent the r.v. $m^{(c)}$ as the two-component random vector (r.v̄.)

$$\bar{m}^{(c)} = \left[s^{(c)}, \left|m^{(c)}\right|\right].$$

Likewise, for the r.v.s $m_i^{(v)}$ we let

$$\bar{m}_i^{(v)} = \left[s_i^{(v)}, \left|m_i^{(v)}\right|\right].$$

Note that $\bar{m}^{(c)}, \bar{m}_i^{(v)} \in \{0,1\} \times [0,\infty)$. With the r.v̄. representation, we rewrite (9.5) informally as

$$\bar{m}^{(c)} = \phi^{-1}\left(\sum_{i=1}^{d_c-1} \phi\left(\bar{m}_i^{(v)}\right)\right), \tag{9.6}$$

where we have replaced the outer ϕ in (9.5) by ϕ^{-1} for later use. (Recall that $\phi = \phi^{-1}$ for positive arguments.) We make use of (9.6) to derive the pdf of $\bar{m}^{(c)}$ (and hence of $m^{(c)}$) from $\{\bar{m}_i^{(v)}\}$ as follows.

1. Compute the pdf of the r.v̄. $\bar{z}_i = \phi\left(\bar{m}_i^{(v)}\right)$. When $s_i^{(v)} = 0$ (equivalently, $m_i^{(v)} > 0$), we have $z_i = \phi\left(m_i^{(v)}\right) = -\ln\tanh\left(m_i^{(v)}/2\right)$ so that

$$\begin{aligned} p\left(0, z_i\right) &= \left|\frac{dz_i}{dm_i^{(v)}}\right|^{-1} p^{(v)}\left(m_i^{(v)}\right)\Bigg|_{m_i^{(v)}=\phi^{-1}(z_i)} \\ &= \frac{1}{\sinh\left(z_i\right)} p^{(v)}(\phi(z_i)), \end{aligned}$$

where we have used the fact that $\phi^{-1}(z_i) = \phi(z_i)$ because $z_i > 0$. Similarly, when $s_i^{(v)} = 1$ (equivalently, $m_i^{(v)} < 0$), $z_i = \phi\left(-m_i^{(v)}\right) = -\ln\tanh\left(-m_i^{(v)}/2\right)$, and from this it is easily shown that

$$p\left(1, z_i\right) = \frac{1}{\sinh\left(z_i\right)} p^{(v)}(-\phi(z_i)).$$

2. Compute the pdf of the r.v̄. $\bar{w} = \sum_{i=1}^{d_c-1} \bar{z}_i = \sum_{i=1}^{d_c-1} \phi\left(\bar{m}_i^{(v)}\right)$. Under the assumption of independent messages $\bar{m}_i^{(v)}$, the r.v̄.s $\bar{z}_i$ are independent, so we may write

$$\begin{aligned} p(\bar{w}) &= p(\bar{z}_1) * p(\bar{z}_2) * \cdots * p(\bar{z}_{d_c-1}) \\ &= p(\bar{z})^{*(d_c-1)}, \end{aligned} \tag{9.7}$$

where we write the second line, a $(d_c - 1)$-fold convolution of $p(\bar{z})$ with itself, because the pdfs $p(\bar{z}_i)$ are identical. These convolutions may be performed using a two-dimensional Fourier transform, where the first component of $\bar{z}_i$ takes values in $\{0, 1\}$ and the second component takes values in $[0, \infty)$. Note that the discrete Fourier transform F_k of some function f_n: $\{0, 1\} \to R$, where R is some range, is given by $F_0 = f_0 + f_1$ and $F_1 = f_0 - f_1$. In an analogous fashion, we write for the two-dimensional transform of $p(\bar{z})$

$$\begin{aligned} \mathcal{F}\{p(\bar{z})\}_{(0,\omega)} &= \mathcal{F}\{p(0, z)\}_\omega + \mathcal{F}\{p(1, z)\}_\omega, \\ \mathcal{F}\{p(\bar{z})\}_{(1,\omega)} &= \mathcal{F}\{p(0, z)\}_\omega - \mathcal{F}\{p(1, z)\}_\omega. \end{aligned}$$

Then, from (9.7), we may write

$$\begin{aligned} \mathcal{F}\{p(\bar{w})\}_{(0,\omega)} &= \left[\mathcal{F}\{p(\bar{z})\}_{(0,\omega)}\right]^{d_c-1}, \\ \mathcal{F}\{p(\bar{w})\}_{(1,\omega)} &= \left[\mathcal{F}\{p(\bar{z})\}_{(1,\omega)}\right]^{d_c-1}. \end{aligned}$$

$p(\bar{w})$ is then obtained by inverse transforming the previous expressions.

3. Compute the pdf $p^{(c)}$ of $\bar{m}^{(c)} = \phi^{-1}(\bar{w}) = \phi^{-1}\left(\sum_{i=1}^{d_c-1} \phi\left(\bar{m}_i^{(v)}\right)\right)$. The derivation is similar to that of the first step. The solution is

$$p\left(0, m^{(c)}\right) = \frac{1}{\sinh\left(m^{(c)}\right)} p(0, z)\Big|_{z=\phi(m^{(c)})}, \quad \text{when } s^{(c)} = 0, \tag{9.8}$$

$$p\left(1, m^{(c)}\right) = \frac{1}{\sinh\left(-m^{(c)}\right)} p(1, z)\Big|_{z=\phi(-m^{(c)})}, \quad \text{when } s^{(c)} = 1. \tag{9.9}$$

We are now equipped with the tools for computing $p^{(v)}$ and $p^{(c)}$, namely Equation (9.4) and Steps 1–3 above. To perform density evolution to find the decoding threshold for a particular (d_v,d_c)-regular LDPC code ensemble, one mimics the iterative message-passing decoding algorithm, starting with the initial pdf $p_0^{(c)}$ for the channel LLRs. This is detailed in the algorithm description below.

Algorithm 9.1 Density Evolution for Regular LDPC Codes

1. Set the channel parameter α to some nominal value expected to be less than the threshold α^*. Set the iteration counter to $\ell = 0$.
2. Given $p_0^{(c)}$, obtain $p^{(v)}$ via (9.4) with $\mathcal{F}\{p^{(c)}\} = 1$ since initially $m^{(c)} = 0$.
3. Increment ℓ by 1. Given $p^{(v)}$, obtain $p^{(c)}$ using Steps 1–3 in the text above.
4. Given $p^{(c)}$ and $p_0^{(c)}$, obtain $p^{(v)}$ using (9.4).
5. If $\ell < \ell_{\max}$ and

$$\int_{-\infty}^{0} p^{(v)}(\tau)d\tau \leq p_e \tag{9.10}$$

for some prescribed error probability p_e (e.g., $p_e = 10^{-6}$), increment the channel parameter α by some small amount and go to 2. If (9.10) does not hold and $\ell < \ell_{\max}$, then go back to 3. If (9.10) does not hold and $\ell = \ell_{\max}$, then the previous α is the decoding threshold α^*.

Example 9.1. Consider the binary symmetric channel with channel inputs $x \in \{+1, -1\}$, channel outputs $y \in \{+1, -1\}$, and error probability ε. We are interested in the decoding threshold ε^* for regular LDPC codes (i.e., the channel parameter is $\alpha = \varepsilon$). A generic message $m_0^{(c)}$ from the channel is given by

$$m_0^{(c)} = \ln\left(\frac{P(y|x=+1)}{P(y|x=-1)}\right),$$

where x is the channel input and y is the channel output. It is shown in Problem 9.1 that, under the assumption that $x = +1$ is always transmitted, the pdf of $m_0^{(c)}$ is given by

$$p_0^{(c)}(\tau) = \varepsilon\delta\left(\tau - \ln\left(\frac{\varepsilon}{1-\varepsilon}\right)\right) + (1-\varepsilon)\delta\left(\tau - \ln\left(\frac{1-\varepsilon}{\varepsilon}\right)\right), \tag{9.11}$$

where $\delta(v) = 1$ when $v = 0$ and $\delta(v) = 1$ when $v \neq 0$. From the density-evolution algorithm above, we obtain the results in the table below [1], where $R = 1 - d_v/d_c$ is the code rate and ε_{cap} is the solution to $R = 1 - \mathcal{H}(\varepsilon) = 1 + \varepsilon\log_2(\varepsilon) + (1-\varepsilon)\log_2(1-\varepsilon)$:

d_v	d_c	R	ε^*	ε_{cap}
3	6	1/2	0.084	0.11
4	8	1/2	0.076	0.11
5	10	1/2	0.068	0.11
3	5	0.4	0.113	0.146
4	6	1/3	0.116	0.174
3	4	1/4	0.167	0.215

Example 9.2. Consider the binary-input AWGN channel with channel inputs $x \in \{+1, -1\}$ and channel outputs $y = x + n$, where $n \sim \mathcal{N}\left(0, \sigma^2\right)$ is a white-Gaussian-noise sample. A generic message $m_0^{(c)}$ from the channel is given by

$$m_0^{(c)} = \ln\left(\frac{P(y|x=+1)}{P(y|x=-1)}\right) = \frac{2y}{\sigma^2},$$

which is clearly conditionally Gaussian, so we need only determine its mean and variance. Under the assumption that only $x = +1$ is sent, $y \sim \mathcal{N}\left(+1, \sigma^2\right)$, so that the mean value and variance of $m_0^{(c)}$ are

$$E\left(m_0^{(c)}\right) = \frac{2E(y)}{\sigma^2} = \frac{2}{\sigma^2},$$
$$\operatorname{var}\left(m_0^{(c)}\right) = \frac{4}{\sigma^4}\operatorname{var}(y) = \frac{4}{\sigma^2}.$$

From the density-evolution algorithm, we obtain the results in the table below [1], where the channel parameter in this case is $\alpha = \sigma$. In the table, σ^* is the decoding threshold in terms of the noise standard deviation and σ_{cap} is the standard deviation for which the channel capacity $C = C(\sigma)$ is equal to R. Also included in the table are the E_b/N_0 values that correspond to σ^* and σ_{cap}.

d_v	d_c	R	σ^*	$(E_b/N_0)^*$ (dB)	σ_{cap}	$(E_b/N_0)_{\text{cap}}$ (dB)
3	6	1/2	0.881	1.11	0.979	0.187
4	8	1/2	0.838	1.54	0.979	0.187
5	10	1/2	0.794	2.01	0.979	0.187
3	5	0.4	1.009	−0.078	1.148	−1.20
4	6	1/3	1.011	−0.094	1.295	−2.25
3	4	1/4	1.267	−2.05	1.549	−3.80

9.2 Density Evolution for Irregular LDPC Codes

We extend the results of the previous section to irregular LDPC code ensembles characterized by the degree-distribution pair $\lambda(X)$ and $\rho(X)$ first introduced in Chapter 5. Recall that

$$\lambda(X) = \sum_{d=1}^{d_v} \lambda_d X^{d-1}, \tag{9.12}$$

where λ_d denotes the fraction of all edges connected to degree-d VNs and d_v denotes the maximum VN degree. Recall also that

$$\rho(X) = \sum_{d=1}^{d_c} \rho_d X^{d-1}, \tag{9.13}$$

where ρ_d denotes the fraction of all edges connected to degree-d CNs and d_c denotes the maximum CN degree. Our goal is to incorporate these degree distributions into the density-evolution algorithm so that the decoding threshold of irregular LDPC code ensembles may be computed. We can leverage the results for regular LDPC ensembles if we first re-state the density-evolution algorithm of the previous section in a more compact form. As before, we start with (9.3), but incorporate the iteration-count parameter ℓ, so that

$$p_\ell^{(v)} = p_0^{(c)} * \left(p_{\ell-1}^{(c)}\right)^{*(d_v-1)}. \tag{9.14}$$

Next, let Γ correspond to the "change of density" due to the transformation $\phi(\cdot)$ as occurs in the computation of $p^{(c)}$ in the previous section (Step 1). Similarly, let Γ^{-1} correspond to the change of density due to the transformation $\phi^{-1}(\cdot)$ (Step 3). Then, in place of Steps 1–3 of the previous section, we may write the shorthand

$$p_\ell^{(c)} = \Gamma^{-1}\left[\left(\Gamma\left[p_\ell^{(v)}\right]\right)^{*(d_c-1)}\right]. \tag{9.15}$$

Substitution of (9.15) into (9.14) then gives

$$p_\ell^{(v)} = p_0^{(c)} * \left(\Gamma^{-1}\left[\left(\Gamma\left[p_{\ell-1}^{(v)}\right]\right)^{*(d_c-1)}\right]\right)^{*(d_v-1)}. \tag{9.16}$$

Equation (9.16) is a compact representation of (9.3) together with Steps 1–3 in the previous section. That is, it represents the entire density-evolution algorithm for regular LDPC ensembles.

For irregular ensembles, (9.14) must be modified to average over all possible VN degrees. This results in

$$\begin{aligned} p_\ell^{(v)} &= p_0^{(c)} * \sum_{d=1}^{d_v} \lambda_d \cdot \left(p_{\ell-1}^{(c)}\right)^{*(d-1)} \\ &= p_0^{(c)} * \lambda_*\left(p_{\ell-1}^{(c)}\right), \end{aligned} \tag{9.17}$$

where the notation $\lambda_*(X)$ in the second line is defined in the first line. Similarly, for irregular ensembles (9.15) becomes

$$\begin{aligned} p_\ell^{(c)} &= \Gamma^{-1}\left[\sum_{d=1}^{d_c} \rho_d \cdot \left(\Gamma\left[p_\ell^{(v)}\right]\right)^{*(d-1)}\right] \\ &= \Gamma^{-1}\left[\rho_*\left(\Gamma\left[p_\ell^{(v)}\right]\right)\right], \end{aligned} \tag{9.18}$$

where the notation $\rho_*(X)$ in the second line is defined in the first line. Substitution of (9.18) into (9.17) yields the irregular counterpart to (9.16),

$$p_\ell^{(v)} = p_0^{(c)} * \lambda_*\Big(\Gamma^{-1}\Big[\rho_*\Big(\Gamma\Big[p_{\ell-1}^{(v)}\Big]\Big)\Big]\Big). \tag{9.19}$$

Observe that (9.19) reduces to (9.16) when $\lambda(X) = X^{d_v-1}$ and $\rho(X) = X^{d_c-1}$.

Analogously to the regular ensemble case, the density-evolution recursion (9.19) is used to obtain $p_\ell^{(v)}$ as a function of the channel parameter α, with $\lambda(X)$ and $\rho(X)$ fixed. That is, as specified in the algorithm presented in the previous section, α is incrementally increased until (9.10) fails; the value of α just prior to this failure is the decoding threshold α^*.

Density evolution for irregular LDPC codes determines the decoding threshold for a given degree-distribution pair, $\lambda(X)$ and $\rho(X)$, but it does not by itself find the optimal degree distributions in the sense of the minimum threshold. To do the latter, one needs an "outer" global optimization algorithm that searches the space of polynomials, $\lambda(X)$ and $\rho(X)$, for the optimum threshold, assuming a fixed code rate. The density-evolution algorithm is, of course, the inner algorithm that determines the threshold for each trial polynomial pair. The global optimization algorithm suggested in [2] is the *differential-evolution* algorithm. This algorithm is in essence a combination of a hill-climbing algorithm and a genetic algorithm. Many researchers have used it successfully to determine optimal degree distributions. One observation that has reduced the search space for optimal $\lambda(X)$ and $\rho(X)$ is that only two or three (consecutive) nonzero coefficients of $\rho(X)$ are necessary and the nonzero coefficients of $\lambda(X)$ need only be λ_2, λ_3, λ_{d_v}, and a few intermediate coefficients. Of course, any global optimization algorithm will require the following constraints on $\lambda(X)$ and $\rho(X)$:

$$\lambda(1) = \rho(1) = 1, \tag{9.20}$$

$$\int_0^1 \rho(X)dX = (1-R)\int_0^1 \lambda(X)dX, \tag{9.21}$$

where R is the design code rate.

Example 9.3. The authors of [2] have determined the following rate-1/2 optimal degree distributions for the binary-input AWGN channel for $d_v = 6$, 11, and 30. These are listed below together with their decoding thresholds. For comparison, the capacity limit for rate-1/2 coding on this channel is $(E_b/N_0)_{\text{cap}} = 0.187\,\text{dB}$ and the decoding threshold for a (3,6) regular LDPC code is 1.11 dB. Figure 9.1 presents the AWGN performance of a 0.5(200012,100283) LDPC code with degree distributions approximately equal to those given below. Observe that at $P_b = 10^{-5}$ the simulated performance is about 0.38 dB from

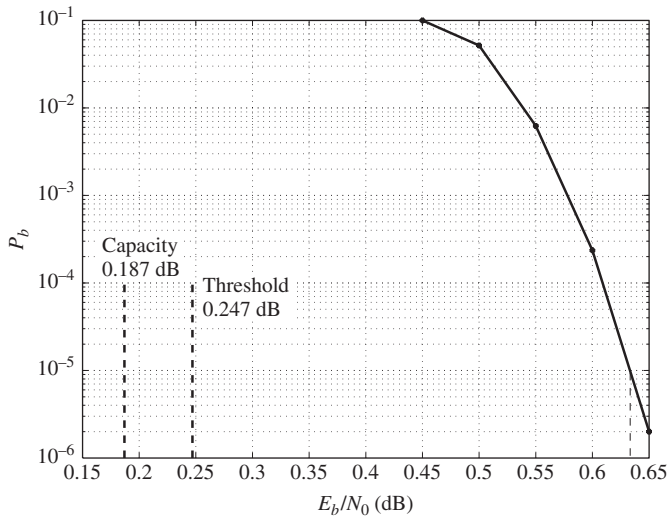


Figure 9.1 Performance of 0.5(200012,100283) LDPC code with approximately optimal degree distributions for the $d_v = 30$ case. (d_v is the maximum variable-node degree.)

the decoding threshold and about 0.45 dB from the capacity limit. For comparison, the original 0.5(131072,65536) turbo code performs about 0.51 dB from the capacity limit. As discussed in the example in the next section, it is possible to obtain performance extremely close to the capacity limit by allowing $d_v = 200$ with a codeword length of 10^7 bits.

$\underline{d_v = 6}$
$\lambda(X) = 0.332X + 0.247X^2 + 0.110X^3 + 0.311X^5,$
$\rho(X) = 0.766X^5 + 0.234X^6,$
$(E_b/N_0)^* = 0.627\,\text{dB}.$

$\underline{d_v = 11}$
$\lambda(X) = 0.239X + 0.295X^2 + 0.033X^3 + 0.433X^{10},$
$\rho(X) = 0.430X^6 + 0.570X^7,$
$(E_b/N_0)^* = 0.380\,\text{dB}.$

$\underline{d_v = 30}$
$$\lambda(X) = 0.196X + 0.240X^2 + 0.002X^5 + 0.055X^6 + 0.166X^7 + 0.041X^8 + 0.011X^9 + 0.002X^{27} + 0.287X^{29},$$
$\rho(X) = 0.007X^7 + 0.991X^8 + 0.002X^9,$
$(E_b/N_0)^* = 0.274\,\text{dB}.$

Example 9.4. It is important to remind the reader of the infinite-length, cycle-free assumptions underlying density evolution and the determination of optimal degree distributions. When optimal degree distributions are adopted in the design of short- or medium-length codes, the codes are generally susceptible to high error floors due to short cycles and trapping sets. In particular, the iterative decoder will suffer from the presence of cycles involving mostly, or only, degree-2 variable nodes.

Consider the design of an LDPC code with the parameters: 0.82(4161,3430) (to compete with other known codes with those parameters). Near-optimal degree distributions with $d_v = 8$ and $d_c = 20$ were found to be [4]

$$\begin{aligned} \lambda(X) &= 0.2343X + 0.3406X^2 + 0.2967X^6 + 0.1284X^7, \\ \rho(X) &= 0.3X^{18} + 0.7X^{19}. \end{aligned} \tag{9.22}$$

From $\lambda(X)$, the number of degree-2 variable nodes would be

$$4161 \cdot \tilde{\lambda}_2 = 4161 \cdot \left(\frac{\lambda_2/2}{\int_0^1 \lambda(X)dX} \right) = 1685.$$

In one of the Chapter 6 problems, it is stated that, in a given Tanner graph (equivalently, $\mathbf{H}$ matrix), the maximum number of degree-2 variable nodes possible before a cycle involving only these degree-2 nodes is created is $n - k - 1$. Thus, a (4161,3430) code with the above degree distributions will have many "degree-2 cycles" since 1685 is much greater than $n - k - 1 = 730$. Further, it can be shown (Problem 9.1) that the girth of this code can be no greater than 10, so these degree-2 cycles are likely somewhat short. Such graphical configurations give rise to an error floor when an iterative decoder is used.

Figure 9.2 presents simulations of the following four length-4161, rate-0.82 codes [4], where we observe that the irregular code with the near-optimal degree distribution in (9.22) does indeed appear to have the best decoding threshold, but it is achieved at the expense of a high error-rate floor due to its large number of degree-2 variable nodes and their associated cycles.

1. A (4161,3430) irregular LDPC code with the degree distributions given in (9.22).
2. A (4161,3431) (nearly) regular LDPC code due to MacKay having degree distributions

$$\begin{aligned} \lambda(X) &= X^3, \\ \rho(X) &= 0.2234X^{21} + 0.7766X^{22}. \end{aligned}$$

   Note that there are no degree-2 nodes.
3. A (4161,3430) regular finite-geometry-based LDPC due to Kou *et al.* [5] having degree distributions

$$\begin{aligned} \lambda(X) &= X^{64}, \\ \rho(X) &= X^{64}. \end{aligned}$$

   Note that there are no degree-2 nodes.
4. A (4161,3430) IRA code with $4161 - 3430 - 1 = 730$ degree-2 nodes, with $d_v = 8$ and $d_c = 20$. The optimal IRA-constrained degree distributions (cf. Section 9.5) were found

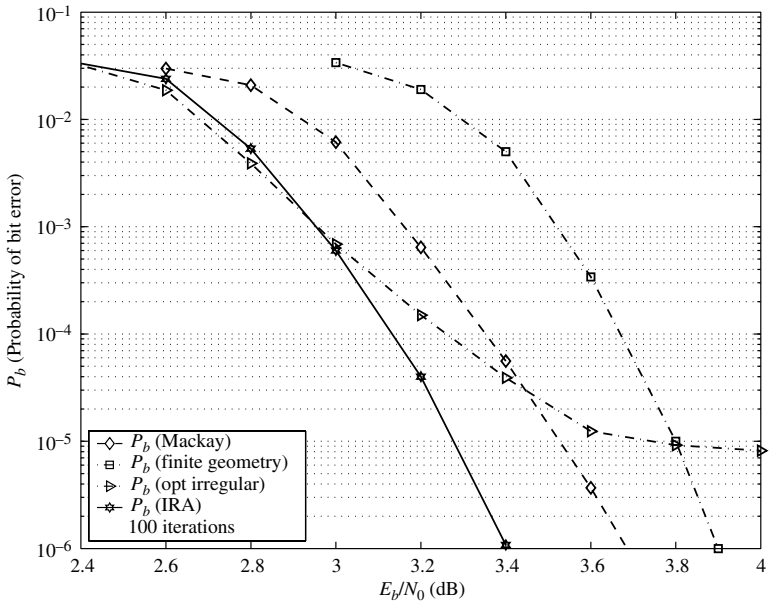


Figure 9.2 Comparison of four length-4161, rate-0.82 LDPC codes, including a near-optimal ("opt" in the figure) irregular one that displays a high error-rate floor.

to be

$$\lambda(X) = 0.00007X^0 + 0.1014X + 0.5895X^2 + 0.1829X^6 + 0.1262X^7,$$
$$\rho(X) = 0.3037X^{18} + 0.6963X^{19}.$$

9.3 Quantized Density Evolution

Clearly the algorithms of the previous two sections involve large amounts of numerical computation that could easily become unstable unless care is taken to avoid this. One way to ensure stability is to quantize all of the quantities involved and design the density-evolution algorithm on this basis. Quantized density evolution has the added advantage that it corresponds to a quantized iterative decoder that would be employed in practice. This section describes the approach, following [6].

Let Δ be the quantization resolution and let $Q(m)$ be the quantized representation of the message m, a real number. Then

$$Q(m) = \begin{cases} \lfloor m/\Delta + 0.5 \rfloor \cdot \Delta, & \text{if } m \geq \Delta/2, \\ \lceil m/\Delta - 0.5 \rceil \cdot \Delta, & \text{if } m \leq -\Delta/2, \\ 0, & \text{otherwise,} \end{cases}$$

where $\lfloor x \rfloor$ is the largest integer not greater than x and $\lceil x \rceil$ is the smallest integer not less than x. We will write $\breve{m}$ for $Q(m)$ so that the quantized version of (9.2) becomes

$$\breve{m}^{(v)} = \breve{m}_0 + \sum_{j=1}^{d_v-1} \breve{m}_j^{(c)} = \sum_{j=0}^{d_v-1} \breve{m}_j^{(c)}.$$

Because of the quantization, we speak of the probability mass function (pmf) of a random variable $\breve{m}$ rather than a pdf, and denote the pmf by $P_m[k] = \Pr(\breve{m} = k\Delta)$.

We first consider a (d_v,d_c)-regular LDPC code ensemble. Analogously to (9.3), we write for the quantized density evolution (actually, pmf evolution) of the variable nodes

$$\begin{aligned} P^{(v)} &= P_0^{(c)} * P_1^{(c)} * \cdots * P_{d_v-1}^{(c)} \\ &= P_0^{(c)} * \left[P^{(c)}\right]^{*(d_v-1)}, \end{aligned} \tag{9.23}$$

where $*$ now represents discrete convolution, $P^{(v)}$ is the pmf of $\breve{m}^{(v)}$, $P_j^{(c)}$ is the pmf of $\breve{m}_j^{(c)}$ for $j = 0, 1, \ldots, d_v - 1$, and $P_j^{(c)} = P^{(c)}$ for $j = 1, 2, \ldots, d_v - 1$. The computations in (9.23) can be efficiently performed using a fast Fourier transform.

As for the quantized density evolution for the check nodes, in lieu of the update equation in (9.5), we use the (quantized) box-plus ($\boxplus$) form of the computation,

$$\breve{m}^{(c)} = \boxplus_{i=1}^{d_c-1} \breve{m}_i^{(v)}, \tag{9.24}$$

where, for two quantized messages, $\breve{m}_1$ and $\breve{m}_2$,

$$\breve{m}_1 \boxplus \breve{m}_2 = \breve{\mathcal{B}}(\breve{m}_1, \breve{m}_2) \triangleq Q\left(2\tanh^{-1}\left(\tanh\left(\breve{m}_1/2\right)\tanh\left(\breve{m}_2/2\right)\right)\right) .$$

The box-plus operation $\breve{\mathcal{B}}(\cdot,\cdot)$ is implemented by a use of two-input look-up table. The pmf of $\breve{m} = \breve{m}_1 \boxplus \breve{m}_2$ is given by

$$P_m[k] = \sum_{(i,j):k\Delta=\breve{\mathcal{B}}(i\Delta,j\Delta)} P_{m_1}[i]P_{m_2}[j].$$

To simplify the notation, we write this as

$$P_m = P_{m_1} \boxplus P_{m_2}.$$

This can be applied to (9.24) $d_c - 2$ times to obtain

$$\begin{aligned} P^{(c)} &= P_1^{(v)} \boxplus P_2^{(v)} \boxplus \cdots \boxplus P_{d_c-1}^{(v)} \\ &= \left(P^{(v)}\right)^{\boxplus(d_c-1)}, \end{aligned} \tag{9.25}$$

where on the second line we used the fact that $P_i^{(v)} = P^{(v)}$ for $i = 1, 2, \ldots, d_c - 1$.

In Section 9.2 we substituted (9.15) into (9.14) (and also (9.18) into (9.17)), that is, the expression for $p^{(c)}$ into the expression for $p^{(v)}$. Here, following [6], we

go in the opposite direction and substitute (9.23) into (9.25) to obtain

$$P_\ell^{(c)} = \left(P_0^{(c)} * \left[P_{\ell-1}^{(c)} \right]^{*(d_v-1)} \right)^{\boxplus(d_c-1)}, \tag{9.26}$$

where we have added the iteration-count parameter ℓ. Note that we need not have obtained the analogous expression to (9.16) because $\Pr(\breve{m}^{(c)} < 0) = 0$ exactly when $\Pr(\breve{m}^{(v)} < 0) = 0$ under the assumption that +1s are transmitted on the channel.

In summary, the quantized density-evolution algorithm for regular LDPC code ensembles follows the algorithm of Section 9.1, but uses (9.26) in place of (9.16) and, in place of (9.10), uses the following stopping criterion:

$$\sum_{k<0} P_\ell^{(c)}[k] \le p_e. \tag{9.27}$$

For irregular LDPC code ensembles with degree distributions $\lambda(X)$ and $\rho(X)$, analogously to (9.17), (9.23) becomes

$$P_\ell^{(v)} = P_0^{(c)} * \lambda_* \left(P_{\ell-1}^{(c)} \right). \tag{9.28}$$

Also, analogous to (9.18), (9.25) becomes

$$\begin{aligned} P_\ell^{(c)} &= \sum_{d=1}^{d_c} \rho_d \cdot \left(P_\ell^{(v)} \right)^{\boxplus(d-1)} \\ &= \rho_\boxplus \left(P_\ell^{(v)} \right), \end{aligned} \tag{9.29}$$

where the notation $\rho_\boxplus(X)$ is obviously defined. Finally, substitution of (9.28) into (9.29) yields

$$P_\ell^{(c)} = \rho_\boxplus \left(P_0^{(c)} * \lambda_* \left(P_{\ell-1}^{(c)} \right) \right). \tag{9.30}$$

At this point, the quantized density-evolution algorithm follows the same procedure as the others described earlier, with the recursion (9.30) at the core of the algorithm and (9.27) as the stopping criterion.

Example 9.5. The authors of [6] have found optimal degree distributions for a rate-1/2 irregular LDPC code ensemble with $d_v = 200$. For 9 bits of quantization, they have found a decoding threshold of $\sigma^* = 0.975\,122$, corresponding to $(E_b/N_0)^* = 0.2188\,\text{dB}$. For 14 bits of quantization, the threshold is $\sigma^* = 0.977\,041$, corresponding to $(E_b/N_0)^* = 0.2017\,\text{dB}$, for an improvement of $0.0171\,\text{dB}$ relative to 9 bits. Again, for comparison, $(E_b/N_0)_{\text{cap}} = 0.187\,\text{dB}$. They have also designed and simulated a rate-1/2 LDPC code of length 10^7 having the $d_v = 200$ optimal degree distribution. Their code achieved a bit error rate of $P_b = 10^{-6}$ at $E_b/N_0 = 0.225\,\text{dB}$, only $0.038\,\text{dB}$ from the $0.187\,\text{dB}$ limit, and $0.0133\,\text{dB}$ from the $0.2017\,\text{dB}$ threshold.

9.4 The Gaussian Approximation

For the binary-input AWGN channel, an alternative to quantized density evolution that simplifies and stabilizes the numerical computations is approximate density evolution based on a *Gaussian approximation* (GA) [7]. The idea is to approximate the message pdfs by Gaussian densities (or Gaussian-mixture densities). Because Gaussian densities are fully specified by two parameters, the mean and variance, (approximate) density evolution entails only the evolution of these two parameters. A further simplification is possible under a consistency assumption, which allows the evolution of only the message means in order to determine approximate decoding thresholds.

A message m satisfies the *consistency condition* if its pdf p_m satisfies

$$p_m(\tau) = p_m(-\tau)e^{\tau}. \tag{9.31}$$

Under the assumption that all +1s are sent, from Example 9.2 the initial (channel) message for the AWGN channel is $m_0^{(c)} = 2y/\sigma^2$. This initial message $m_0^{(c)}$ has the normal pdf $\mathcal{N}\big(2/\sigma^2, 4/\sigma^2\big)$, from which

$$\begin{aligned} p_m(\tau) &= \frac{\sigma}{\sqrt{8\pi}} \exp\left[-\frac{\sigma^2}{8}\left(\tau - \frac{2}{\sigma^2}\right)^2\right] \\ &= \frac{\sigma}{\sqrt{8\pi}} \exp\left[-\frac{\sigma^2}{8}\left(-\tau - \frac{2}{\sigma^2}\right)^2 + \tau\right] \\ &= p_m(-\tau)e^{\tau}. \end{aligned}$$

Thus, the channel message satisfies the consistency condition, and the assumption is that other messages satisfy it at least approximately. Observe also that

$$\operatorname{var}\Big(m_0^{(c)}\Big) = 2E\Big(m_0^{(c)}\Big).$$

Such a relationship holds for all Gaussian pdfs that satisfy the consistency condition because

$$\frac{1}{\sqrt{2\pi}\sigma} \exp\left[-\frac{1}{2\sigma^2}(\tau - \mu)^2\right] = \frac{1}{\sqrt{2\pi}\sigma} \exp\left[-\frac{1}{2\sigma^2}(-\tau - \mu)^2\right] e^{\tau}$$

reduces to

$$\sigma^2 = 2\mu.$$

We call a normal density $\mathcal{N}(\mu, 2\mu)$ a *consistent normal density*. The implication is that one need only monitor the message means when performing density evolution using a Gaussian approximation with the consistency condition. It is interesting

to note that, if one is concerned with the evolution of the SNR of messages instead of pdfs, a convenient SNR definition is

$$\text{SNR} = \frac{\mu^2}{\sigma^2} = \frac{\mu}{2}.$$

So, also in the case of SNR evolution under the Gaussian/consistency assumptions, one need only propagate message means.

9.4.1 GA for Regular LDPC Codes

Examining now the propagation of means for regular code ensembles, we first take the expected value of (9.2), the update equation for the VN-to-CN messages $m_\ell^{(v)}$, to obtain (after adding the iteration-count parameter ℓ),

$$\begin{aligned}\mu_\ell^{(v)} &= \mu_0 + \sum_{j=1}^{d_v-1} \mu_{j,(\ell-1)}^{(c)} \\ &= \mu_0 + (d_v - 1)\,\mu_{\ell-1}^{(c)},\end{aligned} \tag{9.32}$$

where the second line follows from the fact that the code ensemble is regular so that all message means are identical during each iteration. For the CN-to-VN messages $m_\ell^{(c)}$, we rewrite (9.5) (or (9.24)) as

$$\tanh\left(\frac{m_\ell^{(c)}}{2}\right) = \prod_{i=1}^{d_c-1} \tanh\left(\frac{m_{i,\ell}^{(v)}}{2}\right). \tag{9.33}$$

Taking the expected value of this equation under an i.i.d. assumption for the messages $m_{i,\ell}^{(v)}$, we have

$$E\left\{\tanh\left(\frac{m_\ell^{(c)}}{2}\right)\right\} = \left[E\left\{\tanh\left(\frac{m_\ell^{(v)}}{2}\right)\right\}\right]^{d_c-1}, \tag{9.34}$$

where the expectations are taken with respect to a consistent normal density of the form $\mathcal{N}(\mu, 2\mu)$. We can therefore write (9.34) as

$$1 - \Phi\left(\mu_\ell^{(c)}\right) = \left[1 - \Phi\left(\mu_\ell^{(v)}\right)\right]^{d_c-1}, \tag{9.35}$$

where, for $\mu \geq 0$, we define

$$\Phi(\mu) \triangleq 1 - \frac{1}{\sqrt{4\pi\mu}} \int_{-\infty}^{\infty} \tanh(\tau/2) \exp\left[-(\tau-\mu)^2/(4\mu)\right] d\tau.$$

Note that $\Phi(\mu)$ need only be defined for $\mu \geq 0$ because we assume that only +1s are transmitted and hence all message means are positive. It can be shown that $\Phi(\mu)$ is

continuous and decreasing for $\mu \geq 0$, with $\lim_{\mu\to 0} \Phi(\mu) = 1$ and $\lim_{\mu\to\infty} \Phi(\mu) = 0$. Equation (9.35) can be rearranged as

$$\mu_\ell^{(c)} = \Phi^{-1}\left(1 - \left[1 - \Phi\left(\mu_0 + (d_v - 1)\mu_{\ell-1}^{(c)}\right)\right]^{d_c - 1}\right), \tag{9.36}$$

where we have also used (9.32) for $\mu_\ell^{(v)}$.

Example 9.6. Using (9.32), (9.36), and the following approximation [7] for $\Phi(\mu)$,

$$\Phi(\mu) \simeq \begin{cases} \exp\left(-0.4527\mu^{0.86} + 0.0218\right), & \text{for } 0 < \mu < 10, \\ \sqrt{\dfrac{\pi}{\mu}} \exp(-\mu/4)\left(1 - \dfrac{10}{7\mu}\right), & \text{for } \mu > 10, \end{cases}$$

the authors of [7] have obtained the decoding thresholds (σ_{GA} and $(E_b/N_0)_{\text{GA}}$) below for the binary-input AWGN channel using the Gaussian approximation. Also included in the table are the thresholds obtained using "exact" density evolution (σ^* and $(E_b/N_0)^*$). Notice that the gaps between the "exact" and GA results are at most 0.1 dB. Figure 9.3 displays the $\mu_\ell^{(c)}$ versus ℓ trajectories for the $(d_v,d_c) = (3,6)$ ensemble for E_b/N_0 (dB) equal to 1.162, 1.163, 1.165, 1.170 (equivalently, σ equal to 0.8748, 0.8747, 0.8745, 0.8740), where it is clearly seen that the threshold computed by the GA algorithm is $(E_b/N_0)_{\text{GA}} = 1.163$ dB (equivalently, $\sigma_{GA} = 0.8747$).

d_v	d_c	R	σ^*	$(E_b/N_0)^*$ (dB)	σ_{GA}	$(E_b/N_0)_{\text{GA}}$ (dB)
3	6	1/2	0.8809	1.102	0.8747	1.163
4	8	1/2	0.8376	1.539	0.8323	1.594
5	10	1/2	0.7936	2.008	0.7910	2.037
3	5	0.4	1.009	0.8913	1.0003	0.9665
4	6	1/3	1.0109	1.667	1.0036	1.730
3	4	1/4	1.2667	0.9568	1.2517	1.060

9.4.2 GA for Irregular LDPC Codes

For the irregular LDPC codes ensemble case, density evolution with the Gaussian approximation assumes that the outgoing messages both from the VNs and from the CNs are Gaussian (although they are not). Further, the assumption for both node types is that the incoming messages are Gaussian mixtures, that is, their pdfs are weighted sums of Gaussian pdfs.

Analogously to (9.32), the mean of an output message $m_\ell^{(v)}$ of a degree-d VN is given by

$$\mu_{\ell,d}^{(v)} = \mu_0 + (d-1)\mu_{\ell-1}^{(c)}, \tag{9.37}$$

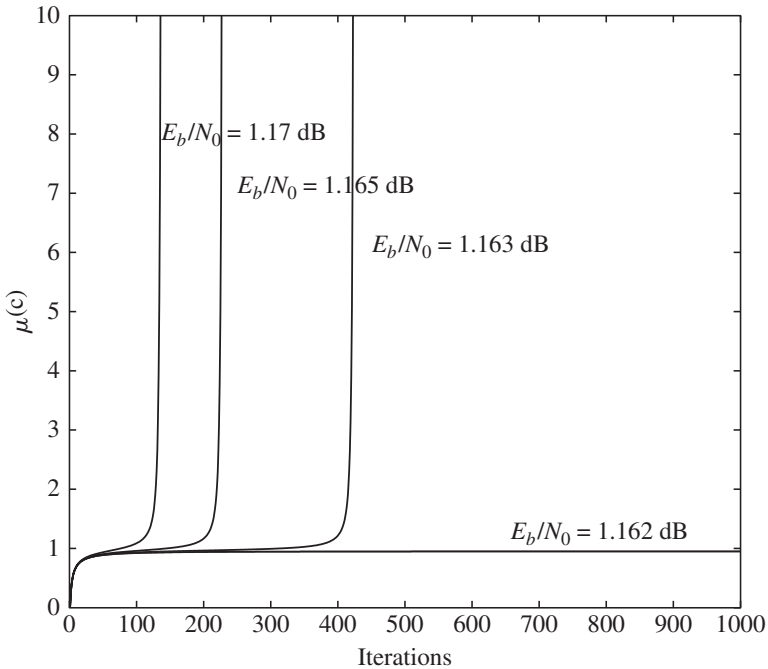


Figure 9.3 Plots of $\mu_\ell^{(c)}$ versus ℓ trajectories for the $(d_v,d_c) = (3,6)$ ensemble for E_b/N_0 (dB) = 1.162, 1.163, 1.165, 1.170 (σ = 0.8748, 0.8747, 0.8745, 0.8740).

where $\mu_{\ell-1}^{(c)}$ is the mean of the message $m_{\ell-1}^{(c)}$, a Gaussian mixture (in general), and the $d-1$ incoming messages are assumed to be i.i.d. Since the output message $m_\ell^{(v)}$ is (assumed) Gaussian with mean $\mu_\ell^{(v)}$, its variance would be $2\mu_\ell^{(v)}$ under the consistency condition. Because a randomly chosen edge is connected to a degree-d variable node with probability λ_d, averaging over all degrees d yields the following Gaussian mixture pdf for the variable node messages:

$$p_{\ell,d}^{(v)}(\tau) = \sum_{d=1}^{d_v} \lambda_d \cdot \mathcal{N}_\tau\left(\mu_{\ell,d}^{(v)}, 2\mu_{\ell,d}^{(v)}\right).$$

In this expression, $\mathcal{N}_\tau(\mu, \sigma^2)$ represents the pdf for a Gaussian r.v. with mean μ and variance σ^2.

This pdf is important to us because, analogously to (9.34) through (9.36), we will need to take the expected value of a function of the form $\tanh(m/2)$ with respect to the pdf of m. Specifically, the expected value within brackets $[\cdot]$ on the

right-hand side of (9.34) is

$$E\left\{\tanh\left(\frac{m_\ell^{(v)}}{2}\right)\right\} = \int_{-\infty}^{\infty} \tanh\left(m_\ell^{(v)}\right) \sum_{d=1}^{d_v} \lambda_d \cdot \mathcal{N}_\tau\left(\mu_{\ell,d}^{(v)}, 2\mu_{\ell,d}^{(v)}\right) d\tau$$
$$= 1 - \sum_{d=1}^{d_v} \lambda_d \Phi\left(\mu_{\ell,d}^{(v)}\right).$$

Referring to the development of (9.36), we have that the mean of a degree-δ check node output is

$$\mu_{\ell,\delta}^{(c)} = \Phi^{-1}\left(1 - \left[1 - \sum_{d=1}^{d_v} \lambda_d \Phi\left(\mu_{\ell,d}^{(v)}\right)\right]^{\delta-1}\right).$$

Finally, if we average over all check-node degrees δ, we have

$$\mu_\ell^{(c)} = \sum_{\delta=1}^{d_c} \rho_\delta \mu_{\ell,\delta}^{(c)}$$
$$= \sum_{\delta=1}^{d_c} \rho_\delta \Phi^{-1}\left(1 - \left[1 - \sum_{d=1}^{d_v} \lambda_d \Phi\left(\mu_0 + (d-1)\mu_{\ell-1}^{(c)}\right)\right]^{\delta-1}\right), \qquad (9.38)$$

where we have also used (9.37).

Equation (9.38) is the Gaussian-approximation density-evolution recursion sought. Its notation can be simplified as follows. Let $s \in [0, \infty)$, $t \in [0, \infty)$, and define

$$f_\delta(s,t) = \Phi^{-1}\left(1 - \left[1 - \sum_{d=1}^{d_v} \lambda_d \Phi(s + (d-1)t)\right]^{\delta-1}\right),$$
$$f(s,t) = \sum_{\delta=1}^{d_c} \rho_\delta f_\delta(s,t).$$

Hence, the simplified recursion is

$$t_\ell = f(s, t_{\ell-1}),$$

where $s = \mu_0$, $t_\ell = \mu_\ell^{(c)}$, and $t_0 = 0$. The threshold sought is

$$s^* = \inf\left\{s \colon s \in [0,\infty) \text{ and } \lim_{\ell\to\infty} t_\ell(s) = \infty\right\}.$$

Note that, since the mean of the initial message $m_0 = 2y/\sigma^2$ is $\mu_0 = 2/\sigma^2$, $s^* = \mu_0^*$ is of the form $s^* = 2/(\sigma^2)^*$, so that the noise threshold is

$$\sigma^* = \sqrt{2/s^*}.$$

9.5 On the Universality of LDPC Codes

A universal code is a code that may be used across a number of different channel types or conditions with little degradation relative to a good single-channel code. The explicit design of universal codes, which simultaneously seeks to solve a multitude of optimization problems, is a daunting task. This section shows that one channel may be used as a surrogate for an entire set of channels to produce good universal LDPC codes. This result suggests that sometimes a channel for which LDPC code design is simple (e.g., BEC) may be used as a surrogate for a channel for which LDPC code design is complex (e.g., Rayleigh). We examine in this section the universality of LDPC codes over the BEC, AWGN, and flat Rayleigh-fading channels in terms of decoding threshold performance. Using excess mutual information (defined below) as a performance metric, we present design results supporting the contention that an LDPC code designed for a single channel can be universally good across the three channels. The section follows [10]. See also [8, 9, 11].

We will examine the SPA decoding thresholds of (non-IRA) LDPC codes and IRA codes of rate 1/4, 1/2, and 3/4 designed under various criteria. Quantized density evolution is used for the AWGN and Rayleigh channels and the result in Section 9.8.2 is used for the BEC. While the Gaussian approximation could be used for the AWGN channel, that is not the case for the Rayleigh channel. To see this, observe that the model for the Rayleigh channel is

$$y_k = a_k x_k + n_k,$$

where a_k is a Rayleigh r.v., $x_k \in \{+1, -1\}$ represents a code bit, and $n_k \sim \mathcal{N}\left(0, \sigma^2\right)$ is an AWGN sample. The initial message is then

$$m_0 = 2\hat{a}_k y_k/\sigma^2,$$

where $\hat{a}_k$ is an estimate of a_k (i.e., $\hat{a}_k$ is channel state information, which we assume is perfect). Under the assumption that $x_k = +1$ (all-zeros codeword), the pdf of m_0 is the expectation, with respect to the Rayleigh pdf for a_k, of the $\mathcal{N}\left(2a/\sigma^2, 4a^2/\sigma^2\right)$ pdf conditioned on a_k. The resulting pdf is not nearly Gaussian.

Recall the constraints (9.20) and (9.21) for LDPC codes. The structure of IRA codes requires $n-k-1$ degree-2 nodes and a single degree-1 node, so that $\lambda_2 \cdot N_e = 2(n-k-1)$ and $\lambda_1 \cdot N_e = 1$, where N_e is the number of edges. On combining

these two equations with (9.20) and (9.21), we have

$$\frac{\lambda_2 R}{2(1-R)-2/n} = \sum_{i=3}^{d_v} \frac{\lambda_i}{i},$$
$$\frac{\lambda_2(1-R)}{2(1-R)-2/n} = \sum_{j} \frac{\rho_j}{j},$$
$$\sum_{i=1}^{d_v} \lambda_i = 1,$$
$$\sum_{i=2}^{d_c} \rho_i = 1.$$

For $n > 200$, this is approximately equivalent to

$$\sum_{j} \frac{\rho_j}{j} = \frac{1-R}{R} \sum_{i=3}^{d_v} \frac{\lambda_i}{i},$$
$$\sum_{i=1}^{d_v} \lambda_i = 1,$$
$$\sum_{i=2}^{d_c} \rho_i = 1.$$

Thus, for IRA codes of practical length, the infinite-length assumption can still be applied.

In all density-evolution computations, the maximum number of decoder iterations is $\ell_{\max} = 1000$ and the stopping threshold is $p_e = 10^{-6}$. The ten code-design criteria are listed in Table 9.1. We consider both a large and a small maximum variable-node degree ($d_v = 50$ and $d_v = 8$), the former to approach theoretical limits and the latter to accommodate low-complexity encoding and decoding. For each design criterion and for each target channel, we compare the threshold-to-capacity gaps for each degree-distribution pair obtained. For the AWGN and Rayleigh channels these gaps are called "excess SNR" and for the BEC these gaps are called "excess δ_0," where δ_0 is the BEC erasure probability. With AWGN and Rayleigh as the target channels, Figure 9.4 presents the excess-SNR results for the ten design criteria for all three code rates. This figure will be discussed shortly.

We can repeat this for the case where the BEC is the target channel and excess α is the performance metric, but we will find that such a plot would be redundant in view of a unifying performance metric we now consider. Specifically, it is convenient (in fact, proper) to present all of our results in a single plot using as the performance metric excess mutual information (MI) [10, 11], defined as

$$\text{excess MI} = I(\rho^*) - R.$$

Table 9.1. Design criteria

Entry	Type	Surrogate channel	d_v
1	LDPC	BEC	50
2	LDPC	AWGN-GA	50
3	LDPC	AWGN	50
4	LDPC	Rayleigh	50
5	LDPC	BEC	8
6	LDPC	AWGN-GA	8
7	LDPC	AWGN	8
8	LDPC	Rayleigh	8
9	IRA	BEC	8
10	IRA	Rayleigh	8

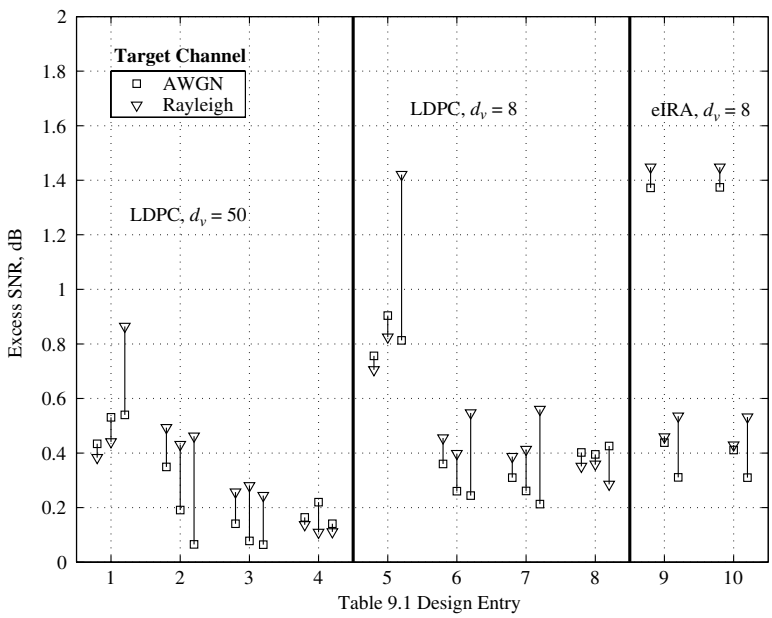


Figure 9.4 Excess SNR (E_b/N_0) for codes designed under the criteria of Table 9.1. The markers aligned with the entry numbers correspond to rate 1/2, those to the left correspond to rate 1/4, and those to the right correspond to rate 3/4.

In this expression, R is the design code rate and $I(\rho^*)$ is the MI for channel parameter ρ^* at the threshold. ρ is the erasure probability for the BEC and SNR E_b/N_0 for the AWGN and Rayleigh channels. Note that, when the BEC is the target channel, excess MI is equal to excess δ_0, obviating the need for an excess-δ_0 plot. We remark that, for the binary-input channels we consider, MI equals

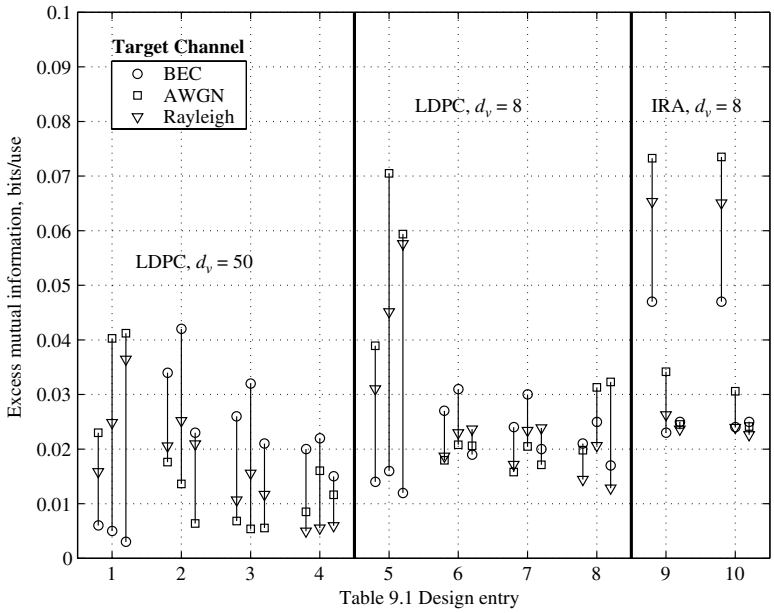


Figure 9.5 Excess MI for code ensembles on all three target channels designed under the criteria of Table 9.1. The markers aligned with the entry numbers correspond to rate 1/2, those to the left correspond to rate 1/4, and those to the right correspond to rate 3/4.

capacity, but we maintain the terminology "excess MI" for consistency with the literature [10, 11].

Figure 9.5 presents the results of the various density-evolution computations. Note in Figure 9.5 that the excess MI is minimized for all 30 cases when the target channel matches the design criterion. Below we partition the discussions of universality and surrogate-channel design corresponding to Figures 9.4 and 9.5 as follows: (a) $d_v = 50$ LDPC codes, (b) $d_v = 8$ LDPC codes, (c) $d_v = 8$ IRA codes, and (d) design via surrogate channels.

(a) $d_v = \mathbf{50}$ **LDPC codes.** Starting with the excess-SNR metric in Figure 9.4 ($d_v = 50$), we observe that, for each code rate, the Rayleigh design criterion (entry 4) leads to codes that are universally good on the AWGN and Rayleigh channels. Specifically, the worst-case excess SNR is only 0.21 dB for rate 1/2 codes on the AWGN channel. At the other extreme, for a code rate of 3/4, the BEC design criterion (entry 1) leads to a worst-case excess SNR of 0.9 dB on the Rayleigh channel. While using the Rayleigh channel as a surrogate leads to the best universal codes, it is at the expense of much greater algorithm complexity. Using the AWGN channel as a surrogate (entry 3) might be preferred since it yields results that are nearly as good. In fact, using the AWGN-GA design criterion (entry 2) also appears to lead to universal codes that are quite good.

Similar comments can be made ($d_v = 50$) when using excess MI as the performance metric as in Figure 9.5. We can add, however, that the BEC design criterion does not look quite as bad in this context. Consider that for the BEC surrogate channel (entry 1) the worst-case excess MI for rates 1/4, 1/2, and 3/4 are 0.023, 0.04, and 0.04, respectively. These correspond to worst-case throughput losses of $0.023/0.25 = 9.2\%$, $0.04/0.5 = 8\%$, and $0.04/0.75 = 5.3\%$, respectively. These are similar to the worst-case throughput losses of 8%, 4.4%, and 2% for the much more complex Rayleigh design criterion (entry 4).

(b) $d_v = 8$ **LDPC codes.** For $d_v = 8$, both in Figure 9.4 and in Figure 9.5, the BEC criterion (entry 5) leads to clearly inferior codes in terms of universality. For example, the worst-case excess SNR is 1.43 dB, which occurs for a rate-3/4 code on the Rayleigh channel. The corresponding excess MI value is 0.057, which corresponds to a throughput loss of 7.6%. On the other hand, the AWGN and Rayleigh criteria both lead to very good universal codes of nearly equal quality. The AWGN-GA criterion (entry 6) also results in codes that are good on all three channels.

(c) $d_v = 8$ **IRA codes.** Even though the structure of IRA codes forces additional constraints on the degree distributions of a code's Tanner graph, as shown in the next section on density evolution for IRA codes, the infinite-length assumption is still valid in the density-evolution process.

The empirical results in [12] indicate that the BEC design criterion may be used to design IRA codes for the Rayleigh-fading channel with negligible performance difference. Here, we reconsider this issue from the perspective of excess MI. As seen in Figure 9.5, there is negligible difference between the IRA codes designed using the BEC criterion (entry 9) and those designed using the Rayleigh criterion (entry 10). Thus the BEC design technique should be used in the case of IRA codes for all three target channels. We note that, for rates 1/2 and 3/4, there are small excess MI losses (and occasionally gains) on going from $d_v = 8$ LDPC codes (entry 8) to $d_v = 8$ IRA codes (entry 9). However, the excess MI loss is substantial for rate 1/4. This is because IRA codes are constrained to $n - k - 1$ degree-2 variable nodes, which is substantially more than the number required for the optimal threshold for rate 1/4. For example, with $d_v = 8$, for rate 1/4, $\lambda_2 \approx 0.66$ for IRA codes, but $\lambda_2 \approx 0.43$ for an optimum LDPC code.

(d) **Design via surrogate channels.** The previous paragraph argued that the BEC may be used as a surrogate with negligible penalty when the designer is interested only in rate-1/2 and -3/4 IRA codes. For BEC-designed LDPC codes ($d_v = 50$) on the Rayleigh channel, the throughput loss compared with that for Rayleigh-designed codes is quite small for all three rates, with the worst-case loss occurring for rate 1/4: $0.012/0.25 = 4.8\%$ (entries 1 and 4). Additionally, the GA is a good "surrogate" when the target is AWGN, as is well known. As an example, the throughput loss compared with the AWGN criterion at $d_v = 50$ and rate 1/2 is $0.015/0.5 = 3\%$.

In summary, the main results of this section are that (1) an LDPC code can be designed to be universally good across all three channels; (2) the Rayleigh

channel is a particularly good surrogate in the design of LDPC codes for the three channels, but the AWGN channel is typically an adequate surrogate; and (3) with the Rayleigh channel as the target, the BEC may be used as a faithful surrogate in the design of IRA codes with rates greater than or equal to 1/2, and there is a throughput loss of less than 6% if the BEC is used as a surrogate to design (non-IRA) LDPC codes.

9.6 EXIT Charts for LDPC Codes

As an alternative to density evolution, the extrinsic-information-transfer (EXIT) chart technique is a graphical tool for estimating the decoding thresholds of LDPC code and turbo code ensembles. The technique relies on the Gaussian approximation, but provides some intuition regarding the dynamics and convergence properties of an iteratively decoded code. The EXIT chart also possesses some additional properties, as covered in the literature [13–16] and briefly in Section 9.8. This section follows the development and the notation found in those publications.

The idea behind EXIT charts begins with the fact that the variable-node processors (VNPs) and check-node processors (CNPs) work cooperatively and iteratively to make bit decisions, with the metric of interest generally improving with each half-iteration. (Various metrics are mentioned below.) A transfer curve plotting the input metric versus the output metric can be obtained both for the VNPs and for the CNPs, where the transfer curve for the VNPs depends on the channel SNR. Further, since the output metric for one processor is the input metric for its companion processor, one can plot both transfer curves on the same axes, but with the abscissa and ordinate reversed for one processor. Such a chart aids in the prediction of the *decoding threshold* of the ensemble of codes characterized by given VN and CN degree distributions: the decoding threshold is the SNR at which the VNP transfer curve just touches the CNP curve, precluding convergence of the two processors. As with density evolution, decoding-threshold prediction via EXIT charts assumes a graph with no cycles, an infinite codeword length, and an infinite number of decoding iterations.

Example 9.7. An EXIT chart example is depicted in Figure 9.6 for the ensemble of regular LDPC codes on the binary-input AWGN channel with $d_v = 3$ and $d_c = 6$. In Figure 9.6, the metric used for the transfer curves is extrinsic mutual information, hence the name extrinsic-information-transfer (EXIT) chart. (The notation used in the figure for the various information measures is given later in this chapter.) The top (solid) $I_{E,V}$ versus $I_{A,V}$ curve is an extrinsic-information curve corresponding to the VNPs. It plots the mutual information, $I_{E,V}$, for the extrinsic information coming out of a VNP against the mutual information, $I_{A,V}$, for the extrinsic (*a priori*) information going into the VNP. The bottom (dashed) $I_{A,C}$ versus $I_{E,C}$ curve is an extrinsic-information curve

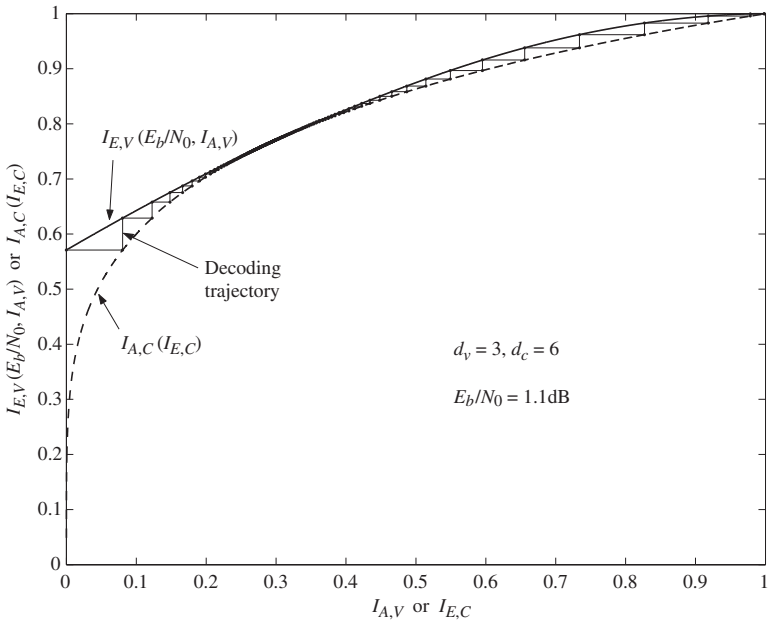


Figure 9.6 An EXIT chart example for the $(d_v,d_c) = (3,6)$ regular LDPC code ensemble.

corresponding to the CNPs. It plots the mutual information, $I_{A,C}$, for the extrinsic (*a priori*) information going into a CNP against the mutual information, $I_{E,C}$, for the extrinsic information coming out of the CNP. (This curve is determined by computing $I_{E,C}$ as a function of $I_{A,C}$ and then, for the purposes of the EXIT chart, plotted on reversed axes.) Between these two curves is the decoding trajectory for an iterative SPA decoder. Note that it "bounces" between the two curves because the extrinsic information coming out of the VNPs (CNPs) is the *a priori* information going into the CNPs (VNPs). The trajectory starts at the $(0, 0)$ point (zero information) and eventually converges to the $(1, 1)$ point (one bit of information, zero errors). The trajectory allows one to visualize the amount of information (in bits) that is being exchanged between the VNPs and CNPs. As the amount of information exchanged approaches unity, the error rate approaches zero.

As the channel SNR increases, the top (VNP) curve shifts upward, increasing the "tunnel" between the two curves and thus the decoder convergence rate. The SNR for this figure is just above the decoding threshold for the $(d_v,d_c) = (3,6)$ ensemble, that is, above $(E_b/N_0)_{\text{EXIT}} = 1.1$ dB. If the SNR is below this value, then the tunnel will be closed, precluding the decoder trajectory from making it through all the way to the $(I_{E,V}, I_{E,C}) = (1, 1)$ point for which the error rate is zero.

Other metrics, such as the SNR or mean [17, 18] and error probability [22] are possible, but mutual information generally gives the most accurate prediction of the decoding threshold [13, 19] and is a universally good metric across many channels [8–11, 16].

We now consider the computation of EXIT transfer curves both for VNPs and for CNPs, first for regular LDPC codes and then for irregular codes. Except for the inputs from the channel, we consider VNP and CNP inputs to be *a priori* information, designated by "A," and their outputs to be extrinsic information, designated by "E." We denote by $I_{E,V}$ the mutual information between a VNP (extrinsic) output and the code bit associated with that VNP. We denote by $I_{A,V}$ the mutual information between the VNP (*a priori*) inputs and the code bit associated with that VNP. The extrinsic-information transfer curve for the VNPs plots the mutual information $I_{E,V}$ as a function of the mutual information $I_{A,V}$. A similar notation holds for the CNPs, namely, $I_{E,C}$ and $I_{A,C}$.

9.6.1 EXIT Charts for Regular LDPC Codes

We focus first on the $I_{E,V}$ versus $I_{A,V}$ transfer curve for the VNPs. We adopt the consistent-Gaussian assumption described in Section 9.4 for the VNP extrinsic-information (*a priori*) inputs $L_{i'\to j}$ and its output $L_{j\to i}$. (The inputs from the channel are truly consistent, so no assumption is necessary.) Under this assumption, such an *a priori* VNP input has the form

$$L_{i'\to j} = \mu_a \cdot x_j + n_j,$$

where $n_j \sim \mathcal{N}\left(0, \sigma_a^2\right)$, $\mu_a = \sigma_a^2/2$, and $x_j \in \{\pm 1\}$. From the VNP update equation, $L_{j\to i} = L_{ch,j} + \sum_{i'\neq i} L_{i'\to j}$, and an independent-message assumption, $L_{j\to i}$ is Gaussian with variance $\sigma^2 = \sigma_{ch}^2 + (d_v - 1)\sigma_A^2$ (and hence mean $\sigma^2/2$), where σ_{ch}^2 is the variance of the consistent-Gaussian input from the channel, $L_{ch,j}$. For simplicity, below we will write L for the extrinsic LLR $L_{j\to i}$, x for the code bit x_j, X for the r.v. corresponding to the realization x, and $p_L(l|\pm)$ for $p_L(l|x = \pm 1)$. Then the mutual information between X and L is

$$\begin{aligned}
I_{E,V} &= H(X) - H(X|L) \\
&= 1 - E\left[\log_2\left(\frac{1}{p_{X|L}(x|l)}\right)\right] \\
&= 1 - \sum_{x=\pm 1} \frac{1}{2} \int_{-\infty}^{\infty} p_L(l|x) \cdot \log_2\left(\frac{p_L(l|+) + p_L(l|-)}{p_L(l|x)}\right) dl \\
&= 1 - \int_{-\infty}^{\infty} p_L(l|+) \log_2\left(1 + \frac{p_L(l|-)}{p_L(l|+)}\right) dl \\
&= 1 - \int_{-\infty}^{\infty} p_L(l|+) \log_2\left(1 + e^{-l}\right) dl,
\end{aligned}$$

where the last line follows from the consistency condition and because $p_L(l|x = -1) = p_L(-l|x = +1)$ for Gaussian densities.

Since $L_{j\to i} \sim \mathcal{N}\left(\sigma^2/2, \sigma^2\right)$ (when conditioned on $x_j = +1$), we have

$$I_{E,V} = 1 - \int_{-\infty}^{\infty} \frac{1}{\sqrt{2\pi}\sigma} e^{-(l-\sigma^2/2)^2/(2\sigma^2)} \log_2\left(1 + e^{-l}\right) dl. \tag{9.39}$$

For convenience we write this as

$$I_{E,V} = J(\sigma) = J\left(\sqrt{(d_v - 1)\sigma_A^2 + \sigma_{ch}^2}\right). \tag{9.40}$$

To express $I_{E,V}$ as a function of $I_{A,V}$, we first exploit the consistent-Gaussian assumption for the inputs $L_{j\to i}$ to write

$$I_{A,V} = J(\sigma_A), \tag{9.41}$$

so that from (9.40) we have

$$I_{E,V} = J(\sigma) = J\left(\sqrt{(d_v - 1)[J^{-1}(I_{A,V})]^2 + \sigma_{ch}^2}\right). \tag{9.42}$$

The inverse function $J^{-1}(\cdot)$ exists since $J(\sigma_A)$ is monotonic in σ_A. Lastly, $I_{E,V}$ can be parameterized by E_b/N_0 for a given code rate R since $\sigma_{ch}^2 = 4/\sigma_w^2 = 8R(E_b/N_0)$. Approximations of the functions $J(\cdot)$ and $J^{-1}(\cdot)$ are given in [14], although numerical computations of these are fairly simple with modern mathematics programs.

To obtain the CNP EXIT curve, $I_{E,C}$ versus $I_{A,C}$, we can proceed as we did in the VNP case, i.e., begin with the consistent-Gaussian assumption. However, this assumption is not sufficient because determining the mean and variance for a CNP output $L_{i\to j}$ is not straightforward, as is evident from the computation for CNPs in (9.5) or (9.24). Closed-form expressions have been derived for the check node EXIT curves [20, 21] and computer-based numerical techniques can also be used to obtain these curves. However, the simplest technique exploits the following duality relationship (which has been proven to be exact for the binary erasure channel [16]): the EXIT curve for a degree-d_c check node (i.e., rate-$(d_c - 1)/d_c$ SPC code) and that of a degree-d_c variable node (i.e., rate-$1/d_c$ repetition code (REP)) are related as

$$I_{E,\mathrm{SPC}}(d_c, I_A) = 1 - I_{E,\mathrm{REP}}(d_c, 1 - I_A).$$

This relationship was shown to be very accurate for the binary-input AWGN channel in [20, 21]. Thus,

$$\begin{aligned} I_{E,C} &= 1 - I_{E,V}(\sigma_{ch} = 0, d_v \leftarrow d_c, I_{A,V} \leftarrow 1 - I_{A,C}) \\ &= 1 - J\left(\sqrt{(d_c - 1)[J^{-1}(1 - I_{A,C})]^2}\right). \end{aligned} \tag{9.43}$$

Equations (9.42) and (9.43) were used to produce the plot in Figure 9.6.

9.6.2 EXIT Charts for Irregular LDPC Codes

For irregular LDPC codes, $I_{E,V}$ and $I_{E,C}$ are computed as weighted averages. The weighting is given by the coefficients of the "edge-perspective" degree-distribution polynomials $\lambda(X) = \sum_{d=1}^{d_v} \lambda_d X^{d-1}$ and $\rho(X) = \sum_{d=1}^{d_c} \rho_d X^{d-1}$, where λ_d is the fraction of edges in the Tanner graph connected to degree-d variable nodes, ρ_d is the fraction of edges connected to degree-d check nodes, and $\lambda(1) = \rho(1) = 1$. So, for irregular LDPC codes,

$$I_{E,V} = \sum_{d=1}^{d_v} \lambda_d I_{E,V}(d, I_{A,V}), \tag{9.44}$$

where $I_{E,V}(d)$ is given by (9.42) with d_v replaced by d, and

$$I_{E,C} = \sum_{d=1}^{d_c} \rho_d I_{E,C}(d, I_{A,C}), \tag{9.45}$$

where $I_{E,C}(d)$ is given by (9.43) with d_c replaced by d.

These equations allow one to compute the EXIT curves for the VNPs and the CNPs. For a given SNR and set of degree distributions, the latter curve is plotted on transposed axes together with the former curve which is plotted in the standard way. If the curves intersect, the SNR must be increased; if they do not intersect, the SNR must be decreased; the SNR at which they just touch is the decoding threshold. After each determination of a decoding threshold, an outer optimization algorithm chooses the next set of degree distributions until an optimum threshold is attained. However, the threshold can be determined quickly and automatically without actually plotting the two EXIT curves, thus allowing the programmer to act as the outer optimizer.

It has been shown [16] that, to optimize the decoding threshold on the binary erasure channel, the shapes of the VNP and CNP transfer curves must be well matched in the sense that the CNP curve fits inside the VNP curve (an example will follow). This situation has also been observed on the binary-input AWGN channel [14]. This is of course intuitively clear for, if the shape of the VNP curve (nearly) exactly matches that of the CNP curve, then the channel parameter can be adjusted (e.g., SNR lowered) to its (nearly) optimum value so that the VNP curve lies just above the CNP curve. Further, to achieve a good match, the number of different VN degrees need be only about 3 or 4 and the number of different CN degrees need be only 1 or 2.

Example 9.8. We consider the design of a rate-1/2 irregular LDPC code with four possible VN degrees and two possible CN degrees. Given that $\lambda(1) = \rho(1) = 1$ and $R = 1 - \int_0^1 \rho(X)dX / \int_0^1 \lambda(X)dX$, only two of the four coefficients for $\lambda(X)$ need be specified and only one of the two for $\rho(X)$ need be specified, after the six degrees have been specified. A non-exhaustive search yielded $\lambda(X) = 0.267X + 0.176X^2 + 0.127X^3 + 0.430X^9$ and

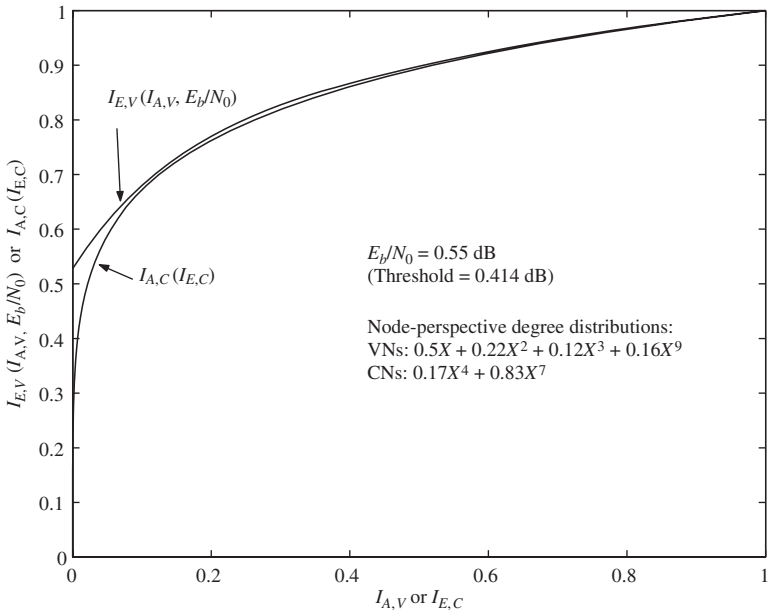


Figure 9.7 An EXIT chart for a rate-1/2 irregular LDPC code ensemble.

$\rho(X) = 0.113X^4 + 0.887X^7$ with a decoding threshold of $(E_b/N_0)_{\text{EXIT}} = 0.414\,\text{dB}$. The EXIT chart for $E_b/N_0 = 0.55\,\text{dB}$ is presented in Figure 9.7, where we see a narrow, but open, tunnel between the two transfer curves. Figure 9.7 also gives the "node-perspective" degree-distribution information. Thus, 17% of the CNs are degree-5 CNs and 85% are degree-8 CNs; 50% of the VNs are degree-2 VNs, 22% are degree-3 VNs, and so on.

The references contain additional information on EXIT charts, including EXIT charts for the Rayleigh channel, for higher-order modulation, and for multi-input/multi-output channels. The area property for EXIT charts and its significance is briefly reviewed in Section 9.8.

9.6.3 EXIT Technique for Protograph-Based Codes

We present in this section an extension of the EXIT approach to codes defined by protographs, following [23, 24]. This extension is a multidimensional numerical technique and hence does not have a two-dimensional EXIT-chart representation of the iterative decoding procedure. Still, the technique yields decoding thresholds for LDPC code ensembles specified by protographs. This multidimensional technique is facilitated by the relatively small size of protographs and permits the analysis of protograph code ensembles characterized by the presence of *exceptional node types*, i.e., node types that can lead to failed EXIT-based convergence.

Examples of exceptional node types are degree-1 variable nodes and punctured variable nodes.

A code ensemble specified by a protograph is a refinement (sub-ensemble) of a code ensemble specified simply by the protograph's (and hence LDPC code's) degree distributions. To demonstrate this, we recall the base matrix $\mathbf{B} = [b_{ij}]$ for a protograph, where b_{ij} is the number of edges between CN i and VN j in the protograph. As an example, consider the protographs with base matrices

$$\mathbf{B} = \begin{pmatrix} 2 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

and

$$\mathbf{B}' = \begin{pmatrix} 2 & 0 & 2 \\ 1 & 2 & 0 \end{pmatrix}.$$

The degree distributions for these protographs are identical and are easily seen to be

$$\lambda(X) = \frac{4}{7}X + \frac{3}{7}X^2,$$
$$\rho(X) = \frac{3}{7}X^2 + \frac{4}{7}X^3.$$

However, the ensemble corresponding to $\mathbf{B}$ has a threshold of $E_b/N_0 = 0.762$ dB and that corresponding to $\mathbf{B}'$ has a threshold at $E_b/N_0 = 0.814$ dB. For comparison, density evolution applied to the above degree distributions gives a threshold of $E_b/N_0 = 0.817$ dB.

As another example, let

$$\mathbf{B} = \begin{pmatrix} 1 & 2 & 1 & 1 & 0 \\ 2 & 1 & 1 & 1 & 0 \\ 1 & 2 & 0 & 0 & 1 \end{pmatrix}$$

and

$$\mathbf{B}' = \begin{pmatrix} 1 & 3 & 1 & 0 & 0 \\ 2 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 & 1 \end{pmatrix},$$

noting that they have identical degree distributions. We also puncture the bits corresponding to the second column in each base matrix. Using the multidimensional EXIT algorithm described below, the thresholds for $\mathbf{B}$ and $\mathbf{B}'$ in this case were computed to be 0.480 dB and (about) 4.7 dB, respectively.

Thus, standard EXIT analysis based on degree distributions is inadequate for protograph-based LDPC code design. In fact, the presence of degree-1 variable nodes as in our second example implies that there is a term in the summation in

(9.44) of the form

$$\lambda_1 I_{E,V}(1, I_{A,V}) = \lambda_1 J(\sigma_{ch}).$$

Since $J(\sigma_{ch})$ is always less than unity for $0 < \sigma_{ch} < \infty$ and since $\sum_{d=1}^{d_v} \lambda_d = 1$, the summation in (9.44), that is, $I_{E,V}$, will be strictly less than unity. Again, standard EXIT analysis implies failed convergence for codes with the same degree distributions as $\mathbf{B}$ and $\mathbf{B}'$. This is in contrast with the fact that codes in the $\mathbf{B}$ ensemble do converge when the SNR exceeds the threshold of 0.48 dB.

In the following, we present a multidimensional EXIT technique [23–25] that overcomes this issue and allows the determination of the decoding threshold for codes based on protographs (possibly with punctured nodes).

The algorithm presented in [23, 24] eliminates the average in (9.44) and considers the propagation of the messages on a decoding tree that is specified by the protograph of the ensemble. Let $\mathbf{B} = [b_{ij}]$ be the $M \times N$ base matrix for the protograph under analysis. Let $I_{E,V}^{j\to i}$ be the extrinsic mutual information between code bits associated with "type j" VNs and the LLRs $L_{j\to i}$ sent from these VNs to "type i" CNs. Similarly, let $I_{E,C}^{i\to j}$ be the extrinsic mutual information between code bits associated with "type j" VNs and the LLRs $L_{i\to j}$ sent from "type i" CNs to these VNs. Then, because $I_{E,C}^{i\to j}$ acts as *a priori* mutual information in the calculation of $I_{E,V}^{j\to i}$, following (9.42) we have (provided that there exists an edge between CN i and VN j, i.e., provided that $b_{ij} \neq 0$)

$$I_{E,V}^{j\to i} = J\left(\sqrt{\sum_{c=1}^{M}(b_{cj} - \delta_{ci})\left(J^{-1}(I_{E,C}^{c\to j})\right)^2 + \sigma_{ch,j}^2}\right), \tag{9.46}$$

where $\delta_{ci} = 1$ when $c = i$ and $\delta_{ci} = 0$ when $c \neq i$. $\sigma_{ch,j}^2$ is set to zero if code bit j is punctured. Similarly, because $I_{E,V}^{j\to i}$ acts as *a priori* mutual information in the calculation of $I_{E,C}^{i\to j}$, following (9.43) we have (when $b_{ij} \neq 0$)

$$I_{E,C}^{i\to j} = 1 - J\left(\sqrt{\sum_{v=1}^{N}(b_{iv} - \delta_{vj})\left(J^{-1}(1 - I_{E,V}^{v\to i})\right)^2}\right). \tag{9.47}$$

The multidimensional EXIT algorithm can now be presented (see below). This algorithm converges only when the selected E_b/N_0 is above the threshold. Thus, the threshold is the lowest value of E_b/N_0 for which all I_{CMI}^j converge to 1. As shown in [23, 24], the thresholds computed by this algorithm are typically within 0.05 dB of those computed by density evolution.

Algorithm 9.2 Multidimensional EXIT Algorithm

1. **Initialization.** Select E_b/N_0. Initialize a vector $\boldsymbol{\sigma}_{ch} = (\sigma_{ch,0}, \ldots, \sigma_{ch,N-1})$ such that
$$\sigma_{ch,j}^2 = 8R(E_b/N_0)_j,$$
where $(E_b/N_0)_j$ equals zero when x_j is punctured and equals the selected E_b/N_0 otherwise.
2. **VN to CN.** For $j = 0, \ldots, N-1$ and $i = 0, \ldots, M-1$, compute (9.46).
3. **CN to VN.** For $i = 0, \ldots, M-1$ and $j = 0, \ldots, N-1$, compute (9.47).
4. **Cumulative mutual information.** For $j = 0, \ldots, N-1$, compute
$$I_{\text{CMI}}^j = J\left(\sqrt{\sum_{c=1}^{M}\left(J^{-1}(I_{E,C}^{c\to j})\right)^2 + \sigma_{ch,j}^2}\right).$$
5. **Stopping criterion.** If $I_{\text{CMI}}^j = 1$ (up to the desired precision) for all j, then stop; otherwise, go to step 2.

9.7 EXIT Charts for Turbo Codes

The EXIT-chart technique for turbo codes is very similar to that for LDPC codes. For LDPC codes, there is one extrinsic-information transfer curve for the VNs and one for the CNs. The CNs are not connected to the channel, so only the VN transfer curve is affected by changes in E_b/N_0. The E_b/N_0 value at which the VN curve just touches the CN curve is the decoding threshold. For turbo codes, there is one extrinsic-information transfer curve for each constituent code (and we consider only turbo codes with two constituent codes). For parallel turbo codes, both codes are directly connected to the channel, so the transfer curves for both shift with shifts in E_b/N_0. For serial turbo codes, only the inner code is generally connected to the channel, in which case only the transfer curve for that code is affected by changes in E_b/N_0. Our discussion in this section begins with parallel turbo codes [13] and then shows how the technique is straightforwardly extended to serial turbo codes.

Recall from Chapter 7 (e.g., Figure 7.3) that each constituent decoder receives as inputs $L_{ch}^u = 2y^u/\sigma^2$ (channel LLRs for systematic bits), $L_{ch}^p = 2y^p/\sigma^2$ (channel LLRs for parity bits), and L_a (extrinsic information that was received from a counterpart decoder, used as *a priori* information). It produces as outputs L^{total} and $L_e = L^{\text{total}} - L_{ch}^u - L_a$, where the extrinsic information L_e is sent to the counterpart decoder which uses it as *a priori* information. Let I_E be the mutual information between the outputs L_e and the systematic bits u and let I_A be the mutual information between the inputs L_a and the systematic bits u. Then the transfer

curve for each constituent decoder is a plot of I_E versus I_A. Each such curve is parameterized by E_b/N_0.

Let $p_E(l|U)$ be the pdf of L_e conditioned on systematic input U. Then the mutual information between L_e and the systematic bits u is given by

$$I_E = 0.5 \sum_{u=\pm 1} \int_{-\infty}^{+\infty} p_E(l|U=u) \log_2 \left(\frac{2p_E(l|U=u)}{p_E(l|U=+1) + p_E(l|U=-1)} \right) dl. \tag{9.48}$$

The conditional pdfs that appear in (9.48) are approximately Gaussian, especially for higher SNR values, but more accurate results are obtained if the pdfs are estimated via simulation of the encoding and decoding of the constituent code. During the simulation, two conditional histograms of $L_e = L^{\text{total}} - L_{ch}^u - L_a$ are accumulated, one corresponding to $p_E(l|U=+1)$ and one corresponding to $p_E(l|U=-1)$, and the respective pdfs are estimated from these histograms. I_E is then computed numerically from (9.48) using these pdf estimates. Note that the inputs to this process are E_b/N_0 and L_a. In order to obtain the I_E versus I_A transfer curve mentioned in the previous paragraph, we need to obtain a relationship between the pdf of L_a and I_A.

Recall that the input L_a for one constituent decoder is the output L_e for the counterpart decoder. As mentioned in the previous paragraph, L_e (and hence L_a) is approximately conditionally Gaussian for both $+1$ and -1 inputs. For our purposes, it is (numerically) convenient to model L_a as if it were precisely conditionally Gaussian; doing so results in negligible loss in accuracy. Further, we assume consistency so that

$$L_a = \mu_a \cdot u + n_a, \tag{9.49}$$

where $n_a \sim \mathcal{N}\left(0, \sigma_a^2\right)$ and $\mu_a = \sigma_a^2/2$. In this case, following (9.39), we have

$$I_A = 1 - \int_{-\infty}^{\infty} \frac{1}{\sqrt{2\pi}\sigma_a} \exp\left(-\left(l - \sigma_a^2/2\right)^2/(2\sigma_a^2)\right) \log_2\left(1 + e^{-l}\right) dl. \tag{9.50}$$

Because I_A is monotonic in σ_a, there is a one-to-one correspondence between I_A and σ_a: given one, we may easily determine the other.

We are now prepared to enumerate the steps involved in obtaining the I_E versus I_A transfer curve for a constituent code. We assume a parallel turbo code with two constituent recursive systematic convolutional encoders.

1. Specify the turbo code rate R and the SNR E_b/N_0 of interest. From these, determine the AWGN variance $N_0/2$.
2. Specify the mutual information I_A of interest. From this determine σ_a, e.g., from a computer program that implements (9.50).
3. Run the RSC code simulator with the BCJR decoder under the model $y = x + n$, where $x \in \{\pm 1\}$ represent the transmitted code bits and $n \sim \mathcal{N}\left(0, \sigma^2\right)$, $\sigma^2 = N_0/2$. In addition to decoder inputs $L_{ch} = 2y/\sigma^2$ (L_{ch} includes L_{ch}^u and L_{ch}^p), the decoder has as inputs the *a priori* LLRs given by (9.49). The values

for the systematic bits u in (9.49) are exactly the systematic bits among the code bits x. The BCJR output is L^{total}, from which $L_e = L^{\text{total}} - L^u_{ch} - L_a$ may be computed for each systematic bit. For a long input block (100 000 bits provides very stable results), histograms corresponding to $p_E(l|U = +1)$ and $p_E(l|U = -1)$ can be produced from the values collected for L_e and used to estimate these conditional pdfs.

4. From (9.48) and the estimated pdfs, compute the mutual information I_E that corresponds to the I_A specified in Step 2.
5. If all of the I_A values of interest have been exhausted, then stop; otherwise go to Step 2.

Example 9.9. We consider the computation of EXIT charts for a rate-1/2 parallel turbo code possessing two identical RSC constituent codes with generator polynomials $(7,5)_8$. The two encoders are punctured to rate-2/3 to achieve the overall code rate of 1/2. In practice, only the first RSC code is terminated, but we can ignore termination issues because we choose 200 000 as the encoder input length. The procedure above is used to obtain the $I_{E,1}$ versus $I_{A,1}$ extrinsic-information transfer curve for the first code. Because the two encoders are identical, this is also the $I_{E,2}$ versus $I_{A,2}$ transfer curve for the second code. Also, because the extrinsic output for the first code is the *a priori* input for the second code, and the extrinsic output for the second code is the *a priori* input for the first code, we can plot the $I_{E,2}$ versus $I_{A,2}$ transfer curve on the same plot as the $I_{E,1}$ versus $I_{A,1}$ transfer curve, but with opposite abscissa–ordinate conventions. In this way we can show the decoding trajectories between the two decoders using mutual information as the figure of merit. This is demonstrated for $E_b/N_0 = 0.6$ and 1 dB in Figure 9.8, where E_b/N_0 is with respect to the rate-1/2 turbo code, not the rate-2/3 constituent codes.

Note for both SNR values that the transfer curve for the second code is just that of the first code mirrored across the 45-degree line because the codes are identical. (Otherwise, a separate transfer curve for the second constituent code would have to be obtained.) Observe also that the decoding trajectory for the 0.6-dB case fails to make it through to the $(I_A, I_E) = (1, 1)$ convergence point because 0.6 dB is below the decoding threshold so that the two transfer curves intersect. The threshold, the value of E_b/N_0 at which the two transfer curves touch, is left to the student in Problem 9.15. Note that, for SNR values just above the threshold, the tunnel between the two curves will be very narrow so that many decoding iterations would be required in order to reach the $(1, 1)$ point (see, for example, Figure 9.6). Finally, these curves correspond to a very long turbo code, for which the information exchanged between the constituent decoders can be assumed to be independent. For turbo codes that are not quite so long, the independence assumption becomes false after some number of iterations, at which point the actual decoder trajectory does not reach the transfer-curve boundaries [13]. As a consequence, more decoder iterations become necessary.

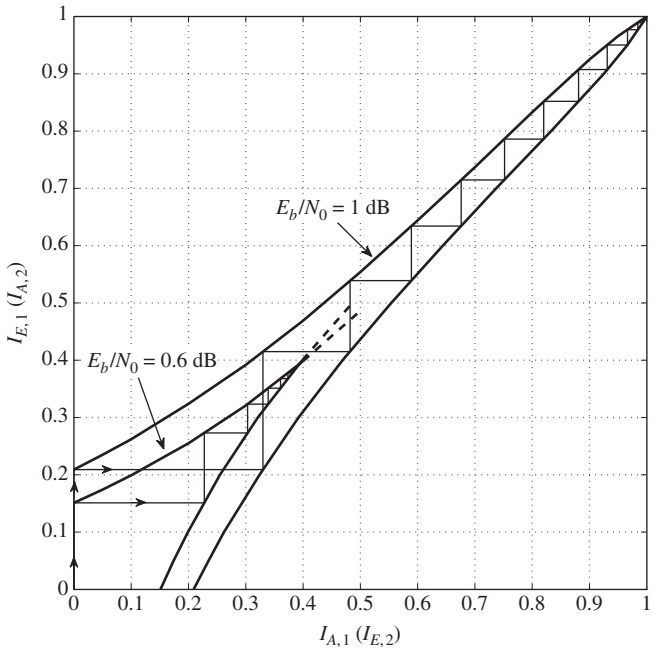


Figure 9.8 EXIT charts at $E_b/N_0 = 0.6$ and 1 dB for a rate-1/2 parallel turbo code with RSC constituent codes having generators $g^{(1)}(D) = 1 + D + D^2$ and $g^{(2)}(D) = 1 + D^2$.

The procedure for producing an EXIT chart for serial turbo codes is nearly identical to that for parallel turbo codes. We assume that only the inner code is connected to the channel. In this case, the transfer curve for that code is obtained in a fashion that is identical to that for a constituent code of a parallel turbo code. For the outer code, there are no inputs L_{ch}^u and L_{ch}^p from the channel, so its extrinsic output is computed as $L_e = L^{\text{total}} - L_a$ (i.e., not $L_e = L^{\text{total}} - L_{ch}^u - L_a$). Thus, to obtain the transfer curve for the outer code, one simulates the outer code decoder with inputs specified as in (9.49), from which estimates of the pdfs $p_E(l|U = +1)$ and $p_E(l|U = -1)$ are produced. These pdfs give I_E through (9.48) as before. Specifying the parameter μ_a (equivalently, σ_a^2) in (9.49) pins down I_A so each simulation of the model (9.49) with differing μ_a values gives different (I_A, I_E) pairs from which the outer-code transfer curve is drawn.

We remark that, unlike with LDPC codes, for which optimal degree distributions may be found via optimization techniques, for turbo codes, design involves a bit of trial and error. The reason is that there is a discrete search space for the turbo-code case, namely the set of constituent-code-generator polynomials.

9.8 The Area Property for EXIT Charts

When describing the design of iteratively decodable codes via EXIT charts, relying on common sense, it was remarked that the closer the shape of the CNP transfer curve matches the shape of the VNP transfer curve, the better the decoding threshold will be. The matched shapes allow one to lower the VNP curve (by lowering the SNR, for example) as much as possible without intersecting the CNP curve, thus improving (optimizing) the SNR decoding threshold. Information-theoretic support was given to this observation in [16, 26] for the BEC and later in [27, 28] for other binary-input memoryless channels. Following [16], we give a brief overview of this result in this section for serial-concatenated codes and LDPC codes, which have very similar descriptions. The results for parallel-concatenated codes are closely related, but require more subtlety [16]. The usual assumptions regarding independence, memorylessness, and long codes are in effect here.

9.8.1 Serial-Concatenated Codes

Consider a serial-concatenated code with EXIT characteristics $I_{E,1}(I_{A,1})$ for the outer code and $I_{E,2}(I_{A,2})$ for the inner code. For convenience, the dependence of $I_{E,2}$ on SNR is suppressed. For $j = 1, 2$, let

$$\mathcal{A}_j = \int_0^1 I_{E,j}(I_{A,j}) dI_{A,j}$$

be the area under the $I_{E,j}(I_{A,j})$ curve. Then it can be shown that, for the outer code,

$$\mathcal{A}_1 = 1 - R_1, \tag{9.51}$$

where R_1 is the rate of the outer code. Note that this is the area *above* the $I_{E,1}(I_{A,1})$ curve when plotted on swapped axes. For the inner code, the result is

$$\mathcal{A}_2 = \frac{I(\underline{X};\underline{Y})/n}{R_2}, \tag{9.52}$$

where R_2 is the rate of the inner code, n is the length of the inner code (and hence of the serial-concatenated code), and $I(\underline{X};\underline{Y})$ is the mutual information between the channel input vector (represented by $\underline{X}$) and the channel output vector (represented by $\underline{Y}$).

Equations (9.51) and (9.52) lead to profound results. First, we know that for successful iterative decoding the outer-code EXIT curve (plotted on swapped axes) must lie below the inner-code EXIT curve. This implies that $1 - \mathcal{A}_1 < \mathcal{A}_2$ or, from (9.51) and (9.52),

$$R_1 R_2 < I(\underline{X};\underline{Y})/n \leq C,$$

where C is the channel capacity. This is, of course, in agreement with Shannon's result that the overall rate must be less than capacity, but we may draw additional conclusions. Because $1 - \mathcal{A}_1 < \mathcal{A}_2$, let $1 - \mathcal{A}_1 = \gamma\mathcal{A}_2$ for some $\gamma \in [0, 1)$. Then, from (9.51) and (9.52),

$$R_1 R_2 = \gamma I(\underline{X}; \underline{Y})/n \leq \gamma C. \tag{9.53}$$

Because γ is a measure of the mismatch between the two areas, $1 - \mathcal{A}_1$ and $\mathcal{A}_2$, Equation 9.53 implies that the area-mismatch factor γ between the two EXIT curves leads directly to a rate loss relative to C by the same factor. Thus, the code-design problem in this sense leads to a curve-fitting problem: the shape of the outer-code curve must be identical to that of the inner code, otherwise a code rate loss will occur.

Note that, if $R_2 = 1$ (e.g., the inner code is an accumulator or an intersymbol-interference channel) and $I(\underline{X}; \underline{Y})/n = C$, then $\mathcal{A}_2 = C$ (from (9.52)) and $\mathcal{A}_2 - (1 - \mathcal{A}_1) = C - R_1$; that is, the area between the two EXIT curves is precisely the code rate loss. Further, if $R_2 < 1$, then $I(\underline{X}; \underline{Y})/n < C$ so that any inner code with $R_2 < 1$ creates an irrecoverable rate loss. One might consider a very strong inner code for which the loss is negligible, i.e., $I(\underline{X}; \underline{Y})/n \simeq C$, but the idea behind concatenated codes is to use weak, easily decodable component codes. Thus, rate-1 inner codes are highly recommended for serial-concatenated code design.

9.8.2 LDPC Codes

The LDPC code case is similar to that of the serial-concatenated code case in that the variable nodes act as the inner code and the check nodes act as the outer code (because the check nodes receive no channel information). Let $\mathcal{A}_C$ be the area under the $I_{E,C}(I_{A,C})$ curve and let $\mathcal{A}_V$ be the area under the $I_{E,V}(I_{A,V})$ curve. Then it can be shown that,

$$\mathcal{A}_C = 1/\bar{d}_c, \tag{9.54}$$

$$\mathcal{A}_V = 1 - (1 - C)/\bar{d}_v, \tag{9.55}$$

where C is the channel capacity as before and $\bar{d}_c$ ($\bar{d}_v$) is the average check (variable) node degree. $\bar{d}_c$ and $\bar{d}_v$ are easily derived from the degree distributions $\rho(X)$ and $\lambda(X)$ (or $\tilde{\rho}(X)$ and $\tilde{\lambda}(X)$). As we saw in our first description of EXIT charts in Section 9.6, the decoder will converge only if the CNP EXIT curve (plotted on swapped axes) lies below the VNP EXIT curve. This implies $1 - \mathcal{A}_C < \mathcal{A}_V$, or $1 - \mathcal{A}_V < \mathcal{A}_C$, which we may write as $1 - \mathcal{A}_V = \gamma\mathcal{A}_C$ for some $\gamma \in [0, 1)$. This equation, combined with (9.54) and (9.55) and some algebra, yields

$$R = \frac{C - (1 - \gamma)}{\gamma} < C, \tag{9.56}$$

where we have used the fact that $R = 1 - \bar{d}_v/\bar{d}_c$.

This equation is the LDPC-code counterpart to Equation (9.53). Thus, any area difference between the CNP and VNP EXIT curves (quantified by the factor γ) corresponds to a rate loss relative to C. Consider, for example, the case $\gamma = 1$ in (9.56). Consequently, capacity can be approached by closely matching the two transfer curves, for example, by curve fitting the VNP EXIT curve to the CNP EXIT curve.

Problems

9.1 Show that the pdf of the channel message $m_0^{(c)}$ for the BSC is given by (9.11) under the assumption that $x = +1$ is always transmitted.

9.2 (Density Evolution for the BEC [2]) Show that for the BEC with erasure probability δ_0 the recursion (9.19) simplifies to

$$\delta_\ell = \delta_0 \lambda(1 - \rho(1 - \delta_{\ell-1})),$$

where δ_ℓ is the probability of a bit erasure after the ℓth iteration. Note that convolution is replaced by multiplication. The decoding threshold δ_0^* in this case is the supremum of all erasure probabilities $\delta_0 \in (0, 1)$ for which $\delta_\ell \to 0$ as $\ell \to \infty$.

9.3 (BEC stability condition [2]) Expand the recursion in the previous problem to show that

$$\delta_\ell = \delta_0 \lambda'(0)\rho'(1)\delta_{\ell-1} + O(\delta_{\ell-1}^2),$$

from which we can conclude that $\delta_\ell \to 0$ provided $\delta_0 < \left[\lambda'(0)\rho'(1)\right]^{-1}$, that is, δ_0^* is upper bounded by $\left[\lambda'(0)\rho'(1)\right]^{-1}$.

9.4 Show that the girth of a (4161,3431) code with the degree distributions in (9.22) is at most 10. To do this, draw a tree starting from a degree-2 variable node, at level 0. At level 1 of the tree, there will be two check nodes; at level 3 there will be at least 38 variable nodes (since the check nodes have degree at least 19); and so on. Eventually, there will be a level L with more than $4161 - 3431 = 730$ check nodes in the tree, from which one can conclude that level L has repetitions of the 730 check nodes from which the girth bound may be deduced.

9.5 Reproduce Figure 9.3 of Example 9.6 using the Gaussian-approximation algorithm. While the expression given in the example for $\Phi(\mu)$ for $0 < \mu < 10$ is easily invertible, you will have to devise a way to implement $\Phi^{-1}(\cdot)$ in software for $\mu > 10$. One possibility is fzero in Matlab.

9.6 Repeat the previous problem for the rate-1/3 regular-(4,6) ensemble. You should find that $\sigma_{\text{GA}} = 1.0036$, corresponding to $(E_b/N_0)_{\text{GA}} = 1.730$ dB.

9.7 Using density evolution, it was shown in Example 9.3 that the decoding threshold for the irregular $d_v = 11$ ensemble is $(E_b/N_0)^* = 0.380$ dB. Use the Gaussian-approximation algorithm of Section 9.4.2 to find the estimated threshold $(E_b/N_0)_{\text{GA}}$ for the same degree-distribution pair.

9.8 For quantized density evolution of *regular* LDPC ensembles, find a recursion analogous to (9.16). For quantized density evolution of *irregular* LDPC ensembles, find a recursion analogous to (9.19).

9.9 Show that the consistency condition (9.31) is satisfied by the pdfs of the initial (channel) messages both from the BEC and from the BSC.

9.10 Write an EXIT-chart computer program to reproduce Figure 9.6.

9.11 Write an EXIT-chart computer program to reproduce Figure 9.7.

9.12 (EXIT-like chart using SNR as convergence measure [17]) Suppose that, instead of mutual information, we use SNR as the convergence measure in an LDPC-code ensemble EXIT chart. Consider a (d_v,d_c)-regular LDPC code ensemble. Show that there is a straight-line equation for the variable nodes given by $\mathrm{SNR}_{\mathrm{out}} = (d_v - 1)\mathrm{SNR}_{\mathrm{in}} + 2RE_b/N_0$. Show also that there is a large-SNR asymptote for the check nodes given by $\mathrm{SNR}_{\mathrm{out}} = \mathrm{SNR}_{\mathrm{in}} - 2\ln(d_c - 1)$.

9.13 Figure 9.7 allows four possible VN degrees and two possible CN degrees. Design degree distributions for a rate-1/2 ensemble with only one possible CN degree (d_c) and three possible VN degrees. Set $d_c = 8$ so that $\rho(X) = X^7$. Also, set the minimum and maximum VN degrees to be 2 and 18. You must choose (search for) a third VN degree. Since $\lambda(1) = 1$ and $R = 1/2 = 1 - \int_0^1 \rho(X)dX / \int_0^1 \lambda(X)dX$, only one coefficient of $\lambda(X)$ may be chosen freely, once all four degrees have been set. Try λ_2 in the vicinity of 0.5 and vary it by a small amount (in the range 0.49–0.51). You should be able to arrive at degree distributions with an EXIT-chart threshold close to $(E_b/N_0)_{\mathrm{EXIT}} = 0.5$ dB. Investigate the impact that the choice of d_c has on the threshold and convergence speed.

9.14 Use the multidimensional-EXIT-chart technique for protograph LDPC codes in Section 9.6.3 to confirm that the protographs with base matrices

$$\mathbf{B} = \begin{pmatrix} 2 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

and

$$\mathbf{B}' = \begin{pmatrix} 2 & 0 & 2 \\ 1 & 2 & 0 \end{pmatrix}$$

have decoding thresholds $E_b/N_0 = 0.762$ dB and $E_b/N_0 = 0.814$ dB, respectively. As mentioned in Section 9.6.3, they have identical degree distributions (what are they?), for which density evolution yields a threshold of $E_b/N_0 = 0.817$ dB.

9.15 Write a computer program to reproduce Figure 9.8. Also, use your program to find the decoding threshold for this turbo code, which is clearly between 0.6 dB and 1 dB from the figure.

9.16 Produce an EXIT chart for a rate-1/2 serial-concatenated convolutional code with a rate-1/2 outer (non-recursive) convolutional code with generators $g^{(1)}(D) = 1 + D + D^2$ and $g^{(2)}(D) = 1 + D^2$ and a rate-1 inner code that is simply the accumulator $1/(1 + D)$. Show that the decoding threshold is about 1 dB.

References

[1] T. Richardson and R. Urbanke, "The capacity of LDPC codes under message-passing decoding," *IEEE Trans. Information Theory*, vol. 47, no. 2, pp. 599–618, February 2001.

[2] T. Richardson and R. Urbanke, "Design of capacity-approaching irregular LDPC codes," *IEEE Trans. Information Theory*, vol. 47, no. 2, pp. 619–637, February 2001.

[3] T. Richardson and R. Urbanke, *Modern Coding Theory*, Cambridge, Cambridge University Press, 2008.

[4] M. Yang, W. E. Ryan, and Y. Li, "Design of efficiently encodable moderate-length high-rate irregular LDPC codes," *IEEE Trans. Communications*, vol. 49, no. 4, pp. 564–571, April 2004.

[5] Y. Kou, S. Lin, and M. Fossorier, "Low-density parity-check codes based on finite geometries: a rediscovery and new results," *IEEE Trans. Information Theory*, vol. 47, no. 11, pp. 2711–2736, November 2001.

[6] S.-Y. Chung, G. David Forney, Jr., T. Richardson, and R. Urbanke, "On the design of low-density parity-check codes within 0.0045 dB of the Shannon limit," *IEEE Communications Lett.*, vol. 4, no. 2, pp. 58–60, February 2001.

[7] S.-Y. Chung, T. Richardson, and R. Urbanke, "Analysis of sum–product decoding of LDPC codes using a Gaussian approximation," *IEEE Trans. Information Theory*, vol. 47, no. 2, pp. 657–670, February 2001.

[8] C. Jones, A. Matache, T. Tian, J. Villasenor, and R. Wesel, "The universality of LDPC codes on wireless channels," *Proc. Military Communications Conf. (MILCOM)*, October 2003.

[9] M. Franceschini, G. Ferrari, and R. Raheli, "Does the performance of LDPC codes depend on the channel?" *Proc. Int. Symp. Information Theory and its Applications*, 2004.

[10] F. Peng, W. E. Ryan, and R. Wesel, "Surrogate-channel design of universal LDPC codes," *IEEE Communications Lett.*, vol. 10, no. 6, pp. 480–482, June 2006.

[11] C. Jones, T. Tian, J. Villasenor, and R. Wesel, "The universal operation of LDPC codes over scalar fading channels," *IEEE Trans. Communications*, vol. 55, no. 1, pp. 122–132, January 2007.

[12] F. Peng, M. Yang and W. E. Ryan, "Simplified eIRA code design and performance analysis for correlated Rayleigh fading channels," *IEEE Trans. Wireless Communications*, pp. 720–725, March 2006.

[13] S. ten Brink, "Convergence behavior of iteratively decoded parallel concatenated codes," *IEEE Trans. Communications*, vol. 49, no. 10, pp. 1727–1737, October 2001.

[14] S. ten Brink, G. Kramer, and A. Ashikhmin, "Design of low-density parity-check codes for modulation and detection," *IEEE Trans. Communications*, vol. 52, no. 4, pp. 670–678, April 2004.

[15] S. ten Brink and G. Kramer, "Design of repeat–accumulate codes for iterative detection and decoding," *IEEE Trans. Signal Processing*, vol. 51, no. 11, pp. 2764–2772, November 2003.

[16] A. Ashikhmin, G. Kramer, and S. ten Brink, "Extrinsic information transfer functions: model and erasure channel properties," *IEEE Trans. Information Theory*, vol. 50, pp. 2657–2673, November 2004.

[17] D. Divsalar, S. Dolinar, and F. Pollara, "Iterative turbo decoder analysis based on density evolution," *IEEE J. Selected Areas in Communications*, vol. 19, no. 5, pp. 891–907, May, 2001.

[18] H. El Gamal and A. R. Hammons, "Analyzing the turbo decoder using the Gaussian approximation," *IEEE Trans. Information Theory*, vol. 47, no. 2, pp. 671–686, February 2001.

[19] M. Tüchler, S. ten Brink, and J. Hagenauer, "Measures for tracing convergence of iterative decoding algorithms," *Proc. 4th IEEE/ITG Conf. on Source and Channel Coding*, Berlin, January 2002.

[20] E. Sharon, A. Ashikhmin, and S. Litsyn, "EXIT functions for the Gaussian channel," *Proc. 40th Annu. Allerton Conf. on Communication, Control, Computers*, Allerton, IL, October 2003, pp. 972–981.

[21] E. Sharon, A. Ashikhmin, and S. Litsyn, "EXIT functions for binary input memoryless symmetric channels," *IEEE Trans. Communications*, pp. 1207–1214, July 2006.

[22] M. Ardakani and F. R. Kschischang, "A more accurate one-dimensional analysis and design of LDPC codes," *IEEE Trans. Communications*, vol. 52, no. 12, pp. 2106–2114, December 2004.

[23] G. Liva, *Block Codes Based on Sparse Graphs for Wireless Communication Systems*, Ph.D. thesis, Università degli Studi di Bologna, 2006.

[24] G. Liva and M. Chiani, "Protograph LDPC codes design based on EXIT analysis", *Proc. 2007 IEEE GlobeCom Conf.*, pp. 3250–3254, November 2007.

[25] G. Liva, S. Song, L. Lan, Y. Zhang, W. E. Ryan, and S. Lin, "Design of LDPC codes: a survey and new results," *J. Communications Software and Systems*, vol. 2, no. 9, pp. 191–211, September 2006.

[26] A. Ashikhman, G. Kramer, and S. ten Brink, "Extrinsic information transfer functions: a model and two properties," *36th Annual Conf. on Information Science Systems*, Princeton University, March 2002.

[27] C. Measson, A. Montanari, and R. Urbanke, "Why we cannot surpass capacity: the matching condition," *43rd Allerton Conf. on Communications, Control, and Computing*, September 2005.

[28] K. Bhattad and K. Narayanan, "An MSE based transfer chart to analyze iterative decoding schemes," *IEEE Trans. Information Theory*, vol. 53, no. 1, pp. 22–38, January 2007.

10 Finite-Geometry LDPC Codes

Finite geometries, such as Euclidean and projective geometries, are powerful mathematical tools for constructing error-control codes. In the 1960s and 1970s, finite geometries were successfully used to construct many classes of easily implementable majority-logic decodable codes. In 2000, Kou, Lin, and Fossorier [1–3] showed that finite geometries can also be used to construct LDPC codes that perform well and close to the Shannon theoretical limit with iterative decoding based on belief propagation. These codes are called *finite-geometry (FG)*-LDPC codes. FG-LDPC codes form the first class of LDPC codes that are constructed algebraically. Since 2000, there have been many major developments in construction of LDPC codes based on various structural properties of finite geometries [4–18]. In this chapter, we put together all the major constructions of LDPC codes based on finite geometries under a unified frame. We begin with code constructions based on Euclidean geometries and then go on to discuss those based on projective geometries.

10.1 Construction of LDPC Codes Based on Lines of Euclidean Geometries

This section presents a class of cyclic LDPC codes and a class of quasi-cyclic (QC) LDPC codes constructed using lines of Euclidean geometries. Before we present the constructions of these two classes of Euclidean-geometry (EG)-LDPC codes, we recall some fundamental structural properties of a Euclidean geometry that have been discussed in Chapter 2.

Consider the m-dimensional Euclidean geometry EG(m,q) over the Galois field GF(q). This geometry consists of q^m points and

$$J \triangleq J_{\mathrm{EG}}(m,1) = q^{m-1}(q^m-1)/(q-1) \tag{10.1}$$

lines. A point in EG(m,q) is simply an m-tuple over GF(q). A line in EG(m,q) is simply a one-dimensional subspace or its coset of the vector space $\mathbf{V}$ of all the m-tuples over GF(q). Each line consists of q points. Two lines either do not have any point in common or they have one and only one point in common. Let $\mathcal{L}$ be a line in EG(m,q) and $\mathbf{p}$ a point on $\mathcal{L}$. We say that $\mathcal{L}$ passes through the point $\mathbf{p}$. If two lines have a common point $\mathbf{p}$, we say that they intersect at $\mathbf{p}$. For any point

$\mathbf{p}$ in EG(m,q), there are

$$g \triangleq g_{\mathrm{EG}}(m,1,0) = (q^m - 1)/(q-1) \tag{10.2}$$

lines intersecting at $\mathbf{p}$ (or passing through $\mathbf{p}$). As described in Chapter 2, the Galois field GF(q^m) as an extension field of GF(q) is a realization of EG(m,q). Let α be a primitive element of GF(q^m). Then the powers of α,

$$\alpha^{-\infty} \triangleq 0, \alpha^0 = 1, \alpha, \alpha^2, \ldots, \alpha^{q-2},$$

represent the q^m points of EG(m,q), where $\alpha^{-\infty}$ represents the origin point of EG(m,q).

Let EG*(m,q) be a *subgeometry* of EG(m,q) obtained by removing the origin $\alpha^{-\infty}$ and all the lines passing through the origin from EG(m,q). This subgeometry consists of $q^m - 1$ non-origin points and

$$J_0 \triangleq J_{0,\mathrm{EG}}(m,1) = (q^{m-1} - 1)(q^m - 1)/(q-1) \tag{10.3}$$

lines not passing through the origin. Let $\mathcal{L}$ be a line in EG*(m,q) consisting of the points $\alpha^{j_1}, \alpha^{j_2}, \ldots, \alpha^{j_q}$, i.e.,

$$\mathcal{L} = \{\alpha^{j_1}, \alpha^{j_2}, \ldots, \alpha^{j_q}\}.$$

For $0 \le t < q^m - 1$,

$$\alpha^t \mathcal{L} = \{\alpha^{j_1+t}, \alpha^{j_2+t}, \ldots, \alpha^{j_q+t}\} \tag{10.4}$$

is also a line in EG*(m,q), where the powers of α in $\alpha^t\mathcal{L}$ are taken modulo $q^m - 1$. In Chapter 2, it has been shown that the $q^m - 1$ lines $\mathcal{L}, \alpha\mathcal{L}, \alpha^2\mathcal{L}, \ldots, \alpha^{q^m-2}\mathcal{L}$ are all different. Since $\alpha^{q^m-1} = 1$, $\alpha^{q^m-1}\mathcal{L} = \mathcal{L}$. These $q^m - 1$ lines form a *cyclic class*. The J_0 lines in EG*(m,q) can be partitioned into

$$K_c \triangleq K_{c,\mathrm{EG}}(m,1) = (q^{m-1} - 1)/(q-1) \tag{10.5}$$

cyclic classes, denoted $\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_{K_c}$. The above structure of the lines in EG*(m,q) is called the *cyclic structure*.

For any line $\mathcal{L}$ in EG*(m,q) not passing through the origin, we define the following ($q^m - 1$)-tuple over GF(2):

$$\mathbf{v}_{\mathcal{L}} = (v_0, v_1, \ldots, v_{q^m-2}), \tag{10.6}$$

whose components correspond to the $q^m - 1$ non-origin points, $\alpha^0, \alpha, \ldots, \alpha^{q^m-2}$, in EG*($m$,$q$), where $v_i = 1$ if α^i is a point on $\mathcal{L}$, otherwise $v_i = 0$. If $\mathcal{L} = \{\alpha^{j_1}, \alpha^{j_2}, \ldots, \alpha^{j_q}\}$, then $v_{j_1} = v_{j_2} = \cdots = v_{j_q} = 1$. The ($q^m - 1$)-tuple $\mathbf{v}_{\mathcal{L}}$ over GF(2) is called the *type-1 incidence vector* of $\mathcal{L}$. The weight of $\mathbf{v}_{\mathcal{L}}$ is q. From (10.4), we can readily see that, for $0 \le t < q^m - 1$, the incidence vector $\mathbf{v}_{\alpha^{t+1}\mathcal{L}}$ of the line $\alpha^{t+1}\mathcal{L}$ can be obtained by cyclically shifting all the components of the incidence vector $\mathbf{v}_{\alpha^t\mathcal{L}}$ of the line $\alpha^t\mathcal{L}$ one place to the right. We call $\mathbf{v}_{\alpha^{t+1}\mathcal{L}}$ the *right cyclic-shift* of $\mathbf{v}_{\alpha^t\mathcal{L}}$.

10.1.1 A Class of Cyclic EG-LDPC Codes

For each cyclic class $\mathcal{S}_i$ of lines in EG$^*(m,q)$ with $1 \leq i \leq K_c$, we form a $(q^m - 1) \times (q^m - 1)$ matrix $\mathbf{H}_{c,i}$ over GF(2) with the incidence vectors $\mathbf{v}_{\mathcal{L}}, \mathbf{v}_{\alpha\mathcal{L}}, \ldots, \mathbf{v}_{\alpha^{q^m-2}\mathcal{L}}$ of the lines $\mathcal{L}, \alpha\mathcal{L}, \ldots, \alpha^{q^m-2}\mathcal{L}$ in $\mathcal{S}_i$ as rows arranged in cyclic order. Then $\mathbf{H}_{c,i}$ is a $(q^m - 1) \times (q^m - 1)$ circulant over GF(2) with both column and row weights q. For $1 \leq k \leq K_c$, we form the following $k(q^m - 1) \times (q^m - 1)$ matrix over GF(2):

$$\mathbf{H}^{(1)}_{\mathrm{EG},c,k} = \begin{bmatrix} \mathbf{H}_{c,1} \\ \mathbf{H}_{c,2} \\ \vdots \\ \mathbf{H}_{c,k} \end{bmatrix}, \tag{10.7}$$

which has column and row weights kq and q, respectively. The subscript "c" of $\mathbf{H}^{(1)}_{\mathrm{EG},c,k}$ stands for "cyclic." Since the rows of $\mathbf{H}^{(1)}_{\mathrm{EG},c,k}$ correspond to lines in EG*(EG) (or EG(m,q)) and no two lines have more than one point in common, it follows that no two rows (or two columns) in $\mathbf{H}^{(1)}_{\mathrm{EG},c,k}$ have more than one 1-component in common. Hence $\mathbf{H}^{(1)}_{\mathrm{EG},c,k}$ satisfies the RC-constraint. The null space of $\mathbf{H}^{(1)}_{\mathrm{EG},c,k}$ gives a cyclic (kq,q)-regular LDPC code $\mathcal{C}_{\mathrm{EG},c,k}$ of length $q^m - 1$ with minimum distance at least $kq + 1$, whose Tanner graph has a girth of at least 6. The above construction gives a class of cyclic EG-LDPC codes. Cyclic EG-LDPC codes with $k = K_c$ were actually discovered in the late 1960s [19, 20, 25] and also shown to form a subclass of polynomial codes [21], but were not recognized to form a class of LDPC codes until 2000 [4].

Since $\mathcal{C}_{\mathrm{EG},c,k}$ is cyclic, it is uniquely specified by its generator polynomial $\mathbf{g}(X)$ (see Chapter 3). To find the generator polynomial of $\mathcal{C}_{\mathrm{EG},c,k}$, we first express each row of $\mathbf{H}^{(1)}_{\mathrm{EG},c,k}$ as a polynomial of degree $q^m - 2$ or less with the leftmost entry of the row as the constant term and the rightmost entry as the coefficient of X^{q^m-2}. Let $\mathbf{h}(X)$ be the *greatest common divisor* of the row polynomials of $\mathbf{H}^{(1)}_{\mathrm{EG},c,k}$. Let $\mathbf{h}^*(X)$ be the *reciprocal polynomial* of $\mathbf{h}(X)$. Then

$$\mathbf{g}(X) = (X^{q^m-1} - 1)/\mathbf{h}^*(X). \tag{10.8}$$

For $k = K_c$, the roots of $\mathbf{g}(X)$ can be completely determined [4, 21, 23–25]. Of particular interest is the cyclic code constructed from the subgeometry EG$^*(2,q)$ of the two-dimensional Euclidean geometry EG$(2,q)$. The subgeometry EG$^*(2,q)$ of EG$(2,q)$ consists of $q^2 - 1$ non-origin points and $q^2 - 1$ lines not passing through the origin. The $q^2 - 1$ lines of EG$^*(2,q)$ form a single cyclic class $\mathcal{S}$. Using the incidence vectors of the lines in this single cyclic class $\mathcal{S}$, we can form a $(q^2 - 1) \times (q^2 - 1)$ circulant matrix $\mathbf{H}^{(1)}_{\mathrm{EG},c,1}$ over GF(2) with both column and row weights q. The null space of $\mathbf{H}^{(1)}_{\mathrm{EG},c,1}$ gives a cyclic EG-LDPC code $\mathcal{C}_{\mathrm{EG},c,1}$ of length $q^2 - 1$

with minimum distance at least $q+1$. If q is a power of 2, say 2^s, $\mathcal{C}_{\mathrm{EG},c,1}$ has the following parameters [4,23]:

Length	$2^{2s}-1$,
Number of parity-check bits	3^s-1,
Minimum distance	2^s+1.

Let $\mathbf{g}(X)$ be the generator polynomial of $\mathcal{C}_{\mathrm{EG},c,1}$. The roots of $\mathbf{g}(X)$ in $\mathrm{GF}(2^s)$ can be completely determined [21, 23]. For any integer h such that $0 \le h \le 2^{2s}-1$, it can be expressed in *radix*-2^s form as follows:

$$h = c_0 + c_1 2^s, \tag{10.9}$$

where $0 \le c_0$ and $c_1 < 2^s$. The sum $W_{2^s}(h) = c_0 + c_1$ of the coefficients of h in the radix-2^s form is called the 2^s-*weight* of h. For any non-negative integer l, let $h^{(l)}$ be the remainder resulting from dividing $2^l h$ by $2^{2s}-1$. Then, $0 \le h^{(l)} < 2^{2s}-1$. The radix-2^s form and 2^s-weight of $h^{(l)}$ are $h^{(l)} = c_0^{(l)} + c_1^{(l)} 2^s$ and $W_{2^s}(h^{(l)}) = c_0^{(l)} + c_1^{(l)}$, respectively. Let α be a primitive element of $\mathrm{GF}(2^{2s})$. Then α^h is a root of $\mathbf{g}_{\mathrm{EG},2,c}(X)$ if and only if

$$0 < \max_{0 \le l < s} W_{2^s}(h^{(l)}) < 2^s. \tag{10.10}$$

Knowing its roots in $\mathrm{GF}(2^{2s})$, $\mathbf{g}(X)$ can be easily determined (see the discussion of BCH codes in Chapter 3). The smallest integer that does not satisfy the condition given by (10.10) is 2^s+1. Hence, $\mathbf{g}(X)$ has as roots the following consecutive powers of α:

$$\alpha, \alpha^2, \ldots, \alpha^{2^s}.$$

Then it follows from the BCH bound [23, 24, 26] (see Section 3.3) that the minimum distance of $\mathcal{C}_{\mathrm{EG},2,c}$ is at least 2^s+1, which is exactly equal to the column weight 2^s of the parity-check matrix $\mathbf{H}_{\mathrm{EG},c,1}$ of $\mathcal{C}_{\mathrm{EG},c,1}$ plus 1.

In fact, the minimum distance is exactly equal to 2^s+1. To show this, we consider the polynomial $X^{2^{2s}-1}-1$ over GF(2), which can be factored as follows:

$$X^{2^{2s}-1}-1 = (X^{2^s-1}-1)(1+X^{2^s-1}+X^{2(2^s-1)}+\cdots+X^{2^s(2^s-1)}). \tag{10.11}$$

First, we note that $X^{2^{2s}-1}-1$ has all the nonzero elements of $\mathrm{GF}(2^{2s})$ as roots and the factor $X^{2^s-1}-1$ has $\alpha^0, \alpha^{2^s+1}, \alpha^{2(2^s+1)}, \alpha^{3(2^s+1)}, \ldots, \alpha^{(2^s-2)(2^s+1)}$ as roots. It can be easily checked that powers of the roots of $X^{2^s-1}-1$ (i.e., $0, 2^s+1, \ldots, (2^s-2)(2^s+1)$) do not satisfy the condition given by (10.10). Hence, the polynomial $1+X^{2^s-1}+\cdots X^{2^s(2^s-1)}$ over GF(2) contains the roots α^h whose powers satisfy the condition given by (10.10). Consequently, the polynomial $1+X^{2^s-1}+\cdots+X^{2^s(2^s-1)}$ is divisible by $\mathbf{g}(X)$ and is a code polynomial (or codeword) in $\mathcal{C}_{\mathrm{EG},c,1}$. This code polynomial has weight 2^s+1. Since the minimum distance of $\mathcal{C}_{\mathrm{EG},c,1}$ is lower bounded by 2^s+1, the minimum distance of $\mathcal{C}_{\mathrm{EG},c,1}$ is exactly 2^s+1. It has been proved [14] that $\mathcal{C}_{\mathrm{EG},c,1}$ has no pseudo-codeword with weight less than its minimum weight 2^s+1, i.e., the Tanner graph of $\mathcal{C}_{\mathrm{EG},c,q}$ has no trapping set with size less than 2^s+1.

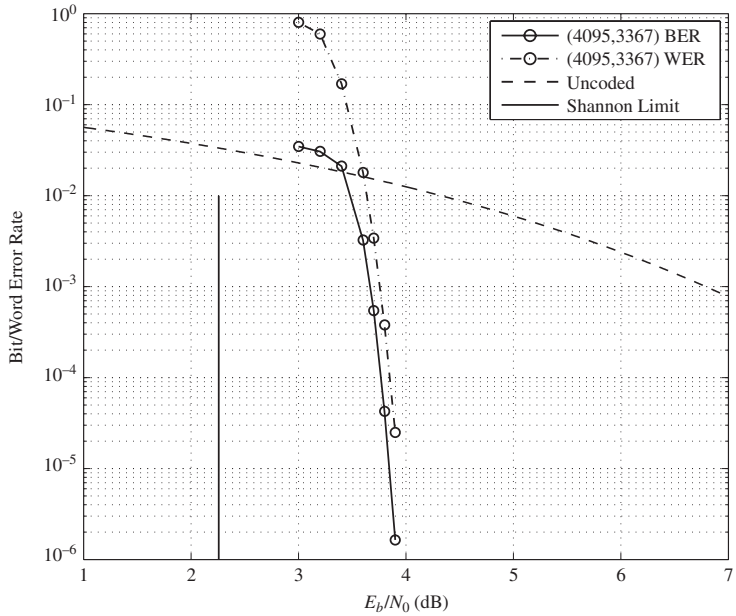


Figure 10.1 Error performance of the cyclic (4095,3367) EG-LDPC code given in Example 10.1.

In the following, we use an example to illustrate the above construction of cyclic EG-LDPC codes. In this and other examples in this chapter, we decode an LDPC code with iterative decoding using the SPA. The maximum number of decoding iterations is set to 100 (or 50). We also assume BPSK transmission over the binary-input AWGN channel.

Example 10.1. Consider the two-dimensional Euclidean geometry EG(2,2^6). The subgeometry EG*(2,2^6) of EG(2,2^6) consists of 4095 non-origin points and 4095 lines not passing through the origin of EG(2,2^6). Using the type-1 incidence vectors of the lines in EG*(2,2^6), we can construct a single 4095×4095 circulant matrix over GF(2) with both column and row weights 64. The null space of this circulant gives a (4095,3367) cyclic EG-LDPC code with rate 0.822 and minimum distance exactly equal to 65. The performance of this code decoded with iterative decoding using the SPA (100 iterations) is shown in Figure 10.1. At a BER of 10^{-6}, the code performs 1.65 dB from the Shannon limit. Since it has a very large minimum distance and a minimum trapping set of the same size, it has a very low error floor. Furthermore, decoding of this code converges very fast, as shown in Figure 10.2. We see that the performance curves of 10 and 100 iterations are basically on top of each other. This code has been considered by NASA for several possible space missions.

10.1.2 A Class of Quasi-Cyclic EG-LDPC Codes

Consider a cyclic class $\mathcal{S}_j$ of lines in the subgeometry EG$^*(m,q)$ of the m-dimensional Euclidean geometry EG(m,q) with $1 \leq j \leq K_c$. We can form a

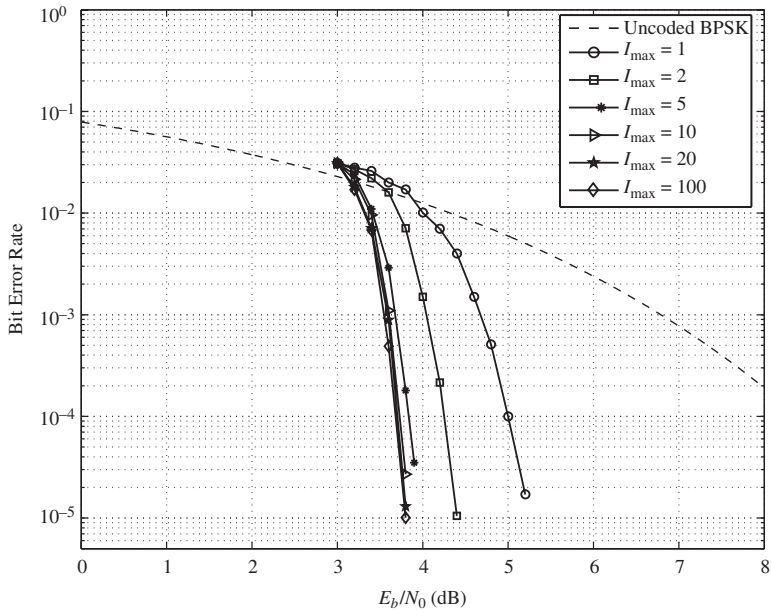


Figure 10.2 The convergence rate of decoding of the cyclic (4095,3367) EG-LDPC code given in Example 10.1.

$(q^m - 1) \times (q^m - 1)$ circulant $\mathbf{G}_{c,j}$ over GF(2) by taking the type-1 incidence vector of a line in $\mathcal{S}_j$ as the first column and its $q^m - 2$ *downward* cyclic-shifts as the other columns. In this case, the columns of $\mathbf{G}_{c,i}$ are the type-1 incidence vectors of the lines in $\mathcal{S}_i$. For $1 \leq k \leq K_c$, form the following $(q^m - 1) \times k(q^m - 1)$ matrix over GF(2):

$$\mathbf{H}^{(2)}_{\mathrm{EG},qc,k} = \begin{bmatrix} \mathbf{G}_{c,1} & \mathbf{G}_{c,2} & \cdots & \mathbf{G}_{c,k} \end{bmatrix}, \tag{10.12}$$

which consists of a row of k circulants over GF(2) of size $(q^m - 1) \times (q^m - 1)$. $\mathbf{H}^{(2)}_{\mathrm{EG},qc,k}$ has column and row weights q and kq, respectively. Actually, $\mathbf{H}^{(2)}_{\mathrm{EG},qc,k}$ is the transpose of the parity-check matrix $\mathbf{H}^{(1)}_{\mathrm{EG},c,k}$ given by (10.7). The null space of $\mathbf{H}^{(2)}_{\mathrm{EG},qc,k}$ gives a binary QC-LDPC code $\mathcal{C}^{(2)}_{\mathrm{EG},qc,k}$ of length $k(q^m - 1)$ with rate at least $(k-1)/k$ and minimum distance at least $q+1$, whose Tanner graph has a girth of at least 6. The subscript "qc" of $\mathcal{C}^{(2)}_{\mathrm{EG},qc,k}$ stands for "quasi-cyclic." For $k = 1, 2, \ldots, K_c$, we can construct a sequence of QC-LDPC codes of lengths $q^m - 1, 2(q^m - 1), \ldots, K_c(q^m - 1)$ based on the cyclic classes of lines in the subgeometry EG$^*(m,q)$ of the m-dimensional Euclidean geometry EG(m,q). The above construction gives a class of binary QC-EG-LDPC codes.

Example 10.2. Let the three-dimensional Euclidean geometry EG$(3,2^3)$ be the code-construction geometry. The subgeometry EG$^*(3,2^3)$ of EG$(3,2^3)$ consists of 511 non-origin points and 4599 lines not passing through the origin of EG$(3,2^3)$. The lines of EG$^*(3,2^3)$ can be partitioned into nine cyclic classes. Using the incidence vectors of the lines in

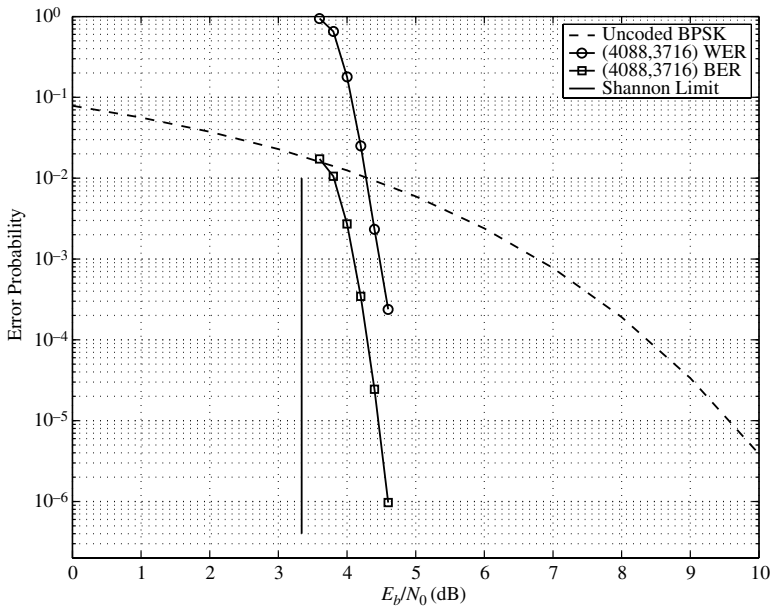


Figure 10.3 The error performance of the (4088,3716) QC-EG-LDPC code given in Example 10.2.

these nine cyclic classes, we can form nine 511×511 circulants over GF(2), each having both column weight 8 and row weight 8. Suppose we take eight of these nine circulants to form a 511×4088 matrix $\mathbf{H}^{(2)}_{\mathrm{EG},qc,8}$ over GF(2). $\mathbf{H}^{(2)}_{\mathrm{EG},qc,8}$ has column and row weights 8 and 64, respectively. The null space of $\mathbf{H}^{(2)}_{\mathrm{EG},qc,8}$ gives a (4088,3716) QC-EG-LDPC code with rate 0.909 and minimum distance at least 9. The error performance of this code decoded with iterative decoding using the SPA (50 iterations) is shown in Figure 10.3. At a BER of 10^{-6}, the code performs 1.2 dB from the Shannon limit.

Since a cyclic EG-LDPC code has a large minimum distance, it, in general, has a very low error floor.

10.2 Construction of LDPC Codes Based on the Parallel Bundles of Lines in Euclidean Geometries

In the previous section, we have presented two classes of structured LDPC codes constructed by exploiting the cyclic structure of lines in Euclidean geometries. Another important structure of lines in Euclidean geometries is the parallel structure as described in Chapter 2. On the basis of this parallel structure of lines, another class of EG-LDPC codes can be constructed.

Consider the m-dimensional Euclidean geometry EG(m,q). As described in Chapter 2, the $J \triangleq J_{\mathrm{EG}}(m,1) = q^{m-1}(q^m-1)/(q-1)$ lines in EG(m,q)

can be partitioned into $K_p = (q^m - 1)/(q - 1)$ parallel bundles, $\mathcal{P}_1(m, 1)$, $\mathcal{P}_2(m, 1)$, $\ldots, \mathcal{P}_{K_p}(m, 1)$, each consisting of q^{m-1} parallel lines. The lines in a parallel bundle contain all q^m points of EG(m,q), each point appearing on one and only one line. Each parallel bundle contains a line passing through the origin. For a line $\mathcal{L}$ in EG(m,q) passing or not passing through the origin of EG(m,q), we define a q^m-tuple over GF(2),

$$\mathbf{v}_{\mathcal{L}} = (v_{-\infty}, v_0, v_1, \ldots, v_{q^m-2}), \tag{10.13}$$

whose components correspond to all the points in EG(m,q) represented by the elements of GF(q^m), $\alpha^{-\infty} = 0, \alpha^0, \alpha, \ldots, \alpha^{q^m-2}$, where $v_j = 1$ if and only if the point α^j is on $\mathcal{L}$, otherwise $v_j = 0$. This q^m-tuple is called the *type-2 incidence vector* of line $\mathcal{L}$. Note that the type-2 incidence vector of a line in EG(m,q) contains a coordinate (or component) $v_{-\infty}$ corresponding to the origin $\alpha^{-\infty}$ of EG(m,q). The weight of $\mathbf{v}_{\mathcal{L}}$ is q. It is clear that the type-2 incidence vectors of two parallel lines do not have any 1-components in common.

For each parallel bundle $\mathcal{P}_i(m, 1)$ of lines in EG(m,q) with $1 \le i \le K_p$, we form a $q^{m-1} \times q^m$ matrix $\mathbf{H}_{p,i}$ over GF(2) with the type-2 incidence vectors of the q^{m-1} parallel lines in $\mathcal{P}_i(m, 1)$ as rows. The column and row weights of $\mathbf{H}_{p,i}$ are 1 and q, respectively. For $1 \le k \le K_p$, we form the following $kq^{m-1} \times q^m$ matrix over GF(2):

$$\mathbf{H}_{\mathrm{EG},p,k} = \begin{bmatrix} \mathbf{H}_{p,1} \\ \mathbf{H}_{p,2} \\ \vdots \\ \mathbf{H}_{p,k} \end{bmatrix}, \tag{10.14}$$

where the subscript "p" stands for "parallel bundle of lines." The rows of $\mathbf{H}_{\mathrm{EG},p,k}$ correspond to k parallel bundles of lines in EG(m,q). The column and row weights of $\mathbf{H}_{\mathrm{EG},p,k}$ are k and q, respectively. The null space of $\mathbf{H}_{\mathrm{EG},p,k}$ gives a (k,q)-regular LDPC code $\mathcal{C}_{\mathrm{EG},p,k}$ of length q^m with minimum distance at least $k + 1$. For $k = 1, 2, \ldots, K_p$, we can construct a family of regular LDPC codes of the same length q^m with different rates and minimum distances. The above construction gives a class of binary regular LDPC codes.

Example 10.3. Consider the three-dimensional Euclidean geometry EG(3,2^3) over GF(2^3). This geometry consists of 512 points and 4672 lines. Each line consists of eight points. The 4672 lines can be grouped into 73 parallel bundles, each consisting of 64 parallel lines. Suppose we take six parallel bundles of lines and form a 384×512 matrix $\mathbf{H}_{\mathrm{EG},p,6}$ with column and row weights 6 and 8, respectively. The null space of $\mathbf{H}_{\mathrm{EG},p,6}$ gives a (6,8)-regular (512,256) EG-LDPC code with rate 0.5 and minimum distance at least 7. The error performance of this code decoded with iterative decoding using the SPA (100 iterations) is shown in Figure 10.4.

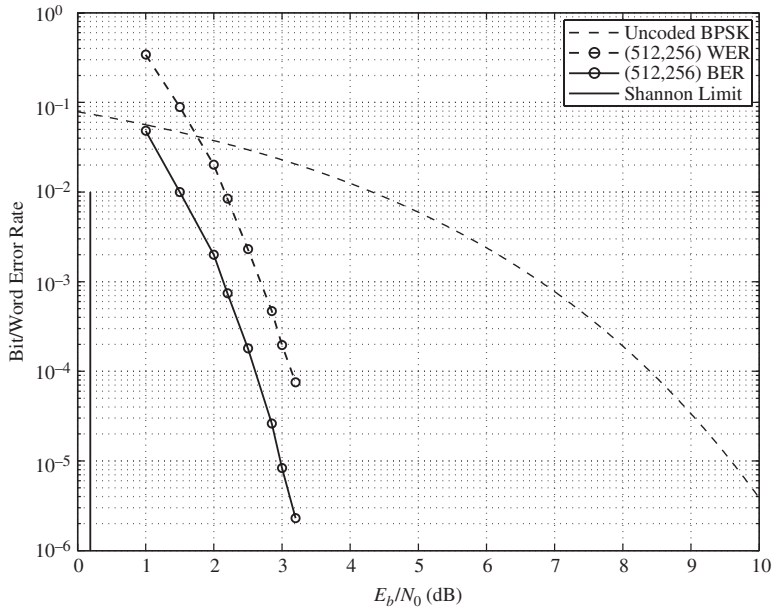


Figure 10.4 The error performance of the (512,256) EG-LDPC code given in Example 10.3.

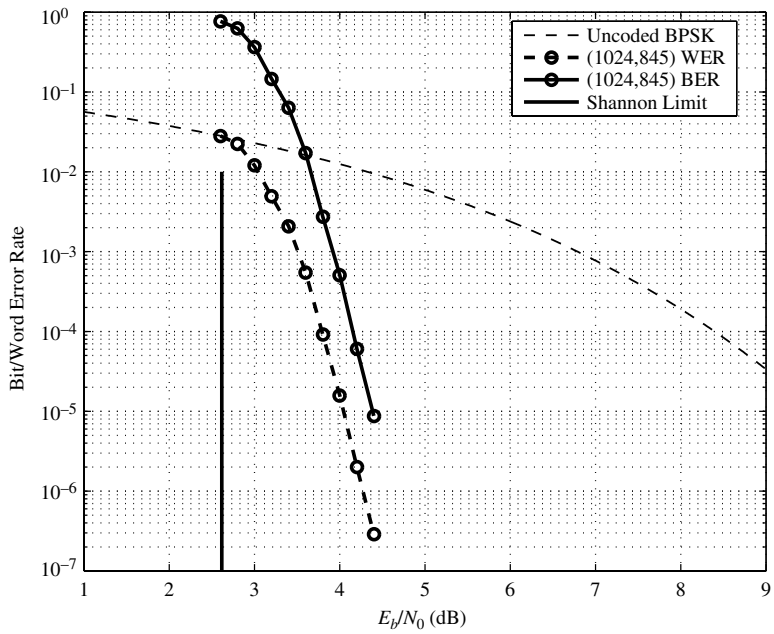


Figure 10.5 The error performance of the (1024,845) EG-LDPC code given in Example 10.4.

Example 10.4. Consider the two-dimensional Euclidean geometry EG(2,2^5). This geometry consists of 1024 points and 1056 lines. The 1056 lines can be partitioned into 33

parallel bundles, each consisting of 32 parallel lines. Suppose we take eight parallel bundles and form a 256×1024 matrix $\mathbf{H}_{\mathrm{EG},p,8}$ over GF(2) with column and row weights 8 and 32. The null space of $\mathbf{H}_{\mathrm{EG},p,8}$ gives an (8,32)-regular (1024,845) LDPC code with rate 0.825 and minimum distance at least 9. The error performance of this code over the binary-input AWGN channel decoded with iterative decoding using the SPA (100 iterations) is shown in Figure 10.5. At a BER of 10^{-6}, the code performs 1.9 dB from the Shannon limit.

10.3 Construction of LDPC Codes Based on Decomposition of Euclidean Geometries

An m-dimensional Euclidean geometry EG(m,q) can be decomposed into q parallel $(m-1)$-flats that are connected by lines. Highly structured EG-LDPC codes can be constructed in a way that utilizes this decomposition. Let $\mathcal{P}(m, m-1)$ be a parallel bundle of $(m-1)$-flats. This parallel bundle consists of q parallel $(m-1)$-flats, denoted by $\mathcal{F}_1, \mathcal{F}_1, \ldots, \mathcal{F}_q$. Each $(m-1)$-flat in $\mathcal{P}(m, m-1)$ consists of q^{m-1} points and

$$J_1 \triangleq J_{\mathrm{EG}}(m-1, 1) = q^{m-2}(q^{m-1}-1)/(q-1)$$

lines. The q parallel $(m-1)$-flats in $\mathcal{P}(m, m-1)$ contain all the q^m points of EG(m,q), each point appearing on one and only one flat in $\mathcal{P}(m, m-1)$. The lines on an $(m-1)$-flat $\mathcal{F}_i$ in $\mathcal{P}(m, m-1)$ contain only the points in $\mathcal{F}_i$. Since the $(m-1)$-flats in $\mathcal{P}(m, m-1)$ are mutually disjoint, the total number of lines contained in $\mathcal{P}(m, m-1)$ is

$$J_2 = qJ_1 = q^{m-1}(q^{m-1}-1)/(q-1).$$

Since there are in total $J = q^{m-1}(q^m-1)/(q-1)$ lines in EG(m,q), there are

$$J_3 = J - J_2 = q^{2(m-1)}$$

lines in EG(m,q) that are not contained in $\mathcal{P}(m, m-1)$. We denote this set of lines by E.

A line in E contains *one and only one point* from each of the q parallel $(m-1)$-flats in $\mathcal{P}(m, m-1)$. This follows from the facts that (1) $\mathcal{P}(m, m-1)$ contains all the points of EG(m,q); and (2) if a line has two points on an $(m-1)$-flat $\mathcal{F}_i$ in $\mathcal{P}(m, m-1)$, then the line is completely contained in $\mathcal{F}_i$. Since the $(m-1)$-flats in $\mathcal{P}(m, m-1)$ are disjoint, the lines in E basically perform the function of connecting them together. For this reason, the lines in E are called *connecting lines* of the parallel $(m-1)$-flats in $\mathcal{P}(m, m-1)$. The connecting lines in E can be partitioned into q^{m-1} groups, denoted $E_{m,1}^{(0)}, E_{m,1}^{(1)}, \ldots, E_{m,1}^{(q^{m-1}-1)}$. Each group, called a *connecting group* with respect to $\mathcal{P}(m, m-1)$, consists of q^{m-1} parallel lines and hence is a parallel bundle of lines in EG(m,q). We also call a connecting group $E_{m,1}^{(i)}$ a *connecting parallel bundle* with respect to $\mathcal{P}(m, m-1)$. The above decomposition

of an m-dimensional Euclidean geometry EG(m,q) into parallel $(m-1)$-flats connected by parallel bundle of lines is referred to as *geometry decomposition* [15].

Let $\mathbf{v} = (v_{-\infty}, v_0, v_1, \ldots, v_{q^m-2}) = (\mathbf{v}_0, \mathbf{v}_1, \ldots, \mathbf{v}_{q-1})$ be a q^m-tuple over GF(2) which consists of q *sections* of equal length q^{m-1} such that the q^{m-1} coordinates in the ith section $\mathbf{v}_i$ correspond to the q^{m-1} points of the ith $(m-1)$-flat $\mathcal{F}_i$ in $\mathcal{P}(m, m-1)$. Therefore, the coordinates of $\mathbf{v}$ correspond to the q^m points of the geometry EG(m,q). Let $\mathcal{L}$ be a line in a connecting parallel bundle $E_{m,1}^{(i)}$ with respect to $\mathcal{P}(m, m-1)$. Using the above sectionalized ordering of coordinates of a q^m-tuple, the type-2 incidence vector $\mathbf{v}_{\mathcal{L}} = (\mathbf{v}_0, \mathbf{v}_1, \ldots, \mathbf{v}_{q-1})$ of $\mathcal{L}$ consists of q sections of equal length q^{m-1} such that the ith section of $\mathbf{v}_i$ has *one and only one* 1-component corresponding to the point in the $(m-1)$-flat $\mathcal{F}_i$ where the line $\mathcal{L}$ passes through. Therefore, $\mathbf{v}_{\mathcal{L}}$ has one and only one 1-component in each section. This type-2 incidence vector is called a *sectionalized type-2 incidence vector*.

For each connecting parallel bundle $E_{m,1}^{(i)}$ of lines with $0 \le i < q^{m-1}$, we form a $q^{m-1} \times q^m$ matrix $\mathbf{H}_{d,i}$ over GF(2) using the sectionalized type-2 incidence vectors of the parallel lines in $E_{m,1}^{(i)}$ as rows. The subscript "d" of $\mathbf{H}_{d,i}$ stands for "decomposition." It follows from the sectionalized structure of the type-2 incidence vector of a connecting line in $E_{m,1}^{(i)}$ that $\mathbf{H}_{d,i}$ consists of a row of q permutation matrices of size $q^{m-1} \times q^{m-1}$,

$$\mathbf{H}_{d,i} = \begin{bmatrix} \mathbf{A}_{i,0} & \mathbf{A}_{i,1} & \cdots & \mathbf{A}_{i,q-1} \end{bmatrix}. \tag{10.15}$$

$\mathbf{H}_{d,i}$ is a matrix with column and row weights 1 and q, respectively. $\mathbf{H}_{d,i}$ is called the *incidence matrix* of the connecting parallel bundle $E_{m,1}^{(i)}$ of lines with respect to $\mathcal{P}(m, m-1)$. Form the following $q^{m-1} \times q$ array of $q^{m-1} \times q^{m-1}$ permutation matrices over GF(2):

$$\begin{aligned} \mathbf{H}_{\mathrm{EG},d}^{(1)} &= \begin{bmatrix} \mathbf{H}_{d,0} \\ \mathbf{H}_{d,1} \\ \vdots \\ \mathbf{H}_{d,q^{m-1}-1} \end{bmatrix} \\ &= \begin{bmatrix} \mathbf{A}_{0,0} & \mathbf{A}_{0,1} & \cdots & \mathbf{A}_{0,q-1} \\ \mathbf{A}_{1,0} & \mathbf{A}_{1,1} & \cdots & \mathbf{A}_{1,q-1} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{A}_{q^{m-1}-1,0} & \mathbf{A}_{q^{m-1}-1,1} & \cdots & \mathbf{A}_{q^{m-1}-1,q-1} \end{bmatrix}. \end{aligned} \tag{10.16}$$

$\mathbf{H}_{\mathrm{EG},d}^{(1)}$ is a $q^{2(m-1)} \times q^m$ matrix over GF(2) with column and row weights q^{m-1} and q, respectively. Since the rows of $\mathbf{H}_{\mathrm{EG},d}^{(1)}$ correspond to $q^{2(m-1)}$ lines in EG(m,q), no two rows (or two columns) of $\mathbf{H}_{\mathrm{EG},d}^{(1)}$ have more than one 1-component in common. Hence, $\mathbf{H}_{\mathrm{EG},d}^{(1)}$ satisfies the RC constraint.

For a pair (g, r) of integers with $1 \le g < q^{m-1}$ and $1 \le r \le q$, let $\mathbf{H}_{\mathrm{EG},d}^{(1)}(g, r)$ be a $g \times r$ subarray of $\mathbf{H}_{\mathrm{EG},d}^{(1)}$. Then $\mathbf{H}_{\mathrm{EG},d}^{(1)}(g, r)$ is a $gq^{m-1} \times rq^{m-1}$ matrix over GF(2)

with column and row weights g and r, respectively. The null space of $\mathbf{H}^{(1)}_{\mathrm{EG},d}(g,r)$ gives a (g,r)-regular EG-LDPC code $\mathcal{C}^{(1)}_{\mathrm{EG},d}$ of length rq^{m-1} with minimum distance at least $g+1$. $\mathbf{H}^{(1)}_{\mathrm{EG},d}(g,r)$ consists of r columns of permutation matrices, each column consisting of g permutation matrices. If we add l columns (or column-vectors) of $\mathbf{H}^{(1)}_{\mathrm{EG},d}(g,r)$, we obtain a column vector $\mathbf{h}$ with g sections of q^{m-1} bits each. For this column vector $\mathbf{h}$ to be zero, each section of $\mathbf{h}$ must be zero. For each section of $\mathbf{h}$ to be zero, l must be even. This implies that the minimum distance of $\mathcal{C}^{(1)}_{\mathrm{EG},d}$ must be even. Since the minimum distance of $\mathcal{C}^{(1)}_{\mathrm{EG},d}$ is at least $g+1$, the minimum distance of $\mathcal{C}^{(1)}_{\mathrm{EG},d}$ is at least $g+2$ for even g and at least $g+1$ for odd g. The above construction gives a class of regular EG-LDPC codes.

Example 10.5. Let the two-dimensional Euclidean geometry EG(2,2^6) be the code-construction geometry. This geometry consists of 4096 points and 4160 lines. Each line consists of 64 points. The lines in EG(2,2^6) can be partitioned into 65 parallel bundles, each consisting of 64 lines. Suppose we decompose this geometry in accordance with a chosen parallel bundle $\mathcal{P}(2,1)$ of lines. Then there are 64 connecting parallel bundles of lines, $E^{(0)}_{2,1}, E^{(1)}_{2,1}, \ldots, E^{(63)}_{2,1}$. For $0 \le i < 64$, the incidence matrix $\mathbf{H}_{d,i} = [\mathbf{A}_{i,0} \quad \mathbf{A}_{i,1} \quad \cdots \quad \mathbf{A}_{i,63}]$ of the connecting parallel bundle $E^{(i)}_{2,1}$ consists of a row of 64 permutation matrices of size 64×64. On the basis of (10.16), we can form a 64×64 array $\mathbf{H}^{(1)}_{\mathrm{EG},d}$ of 64×64 permutation matrices. From this array, we can construct a family of regular EG-LDPC codes with various lengths, rates, and minimum distances. Suppose we take the 6×32 subarray $\mathbf{H}^{(1)}_{\mathrm{EG},d}(6,32)$ at the upper-left corner of $\mathbf{H}^{(1)}_{\mathrm{EG},d}$ as the parity-check matrix for an EG-LDPC code. $\mathbf{H}^{(1)}_{\mathrm{EG},d}(6,32)$ is a 384×2048 matrix over GF(2) with column and row weights 6 and 32, respectively. The null space of this matrix gives a (6,32)-regular (2048,1723) EG-LDPC code with rate 0.841 and minimum distance at least 8. The error performance of this code over the binary-input AWGN channel decoded with iterative decoding using the SPA (100 iterations) is shown in Figure 10.6. At a BER of 10^{-6}, the code performs 1.55 dB from the Shannon limit and achieves a 6-dB coding gain over uncoded BPSK. This code is actually equivalent to the IEEE 802.3 standard (2048,1723) LDPC code for the 10-G Base-T Ethernet, which has no error floor down to a BER of 10^{-12}.

Example 10.6. We continue Example 10.5. Suppose we take a 8×64 subarray $\mathbf{H}^{(1)}_{\mathrm{EG},d}(8,64)$ from the 64×64 array $\mathbf{H}^{(1)}_{\mathrm{EG},d}$ given in Example 10.5, say the top eight rows of permutation matrices of $\mathbf{H}^{(1)}_{\mathrm{EG},d}$. $\mathbf{H}^{(1)}_{\mathrm{EG},d}(8,64)$ is a 512×4096 matrix over GF(2) with column and row weights 8 and 64, respectively. The null space of this matrix gives an (8,64)-regular (4096,3687) EG-LDPC code with rate 0.9 and minimum distance at least 10. The error performance, error floor, and convergence rate of decoding of this code are shown in Figures 10.7–10.9, respectively. At a BER of 10^{-6}, the code performs 1.3 dB from the Shannon limit as shown in Figure 10.7 and its error floor for BER is

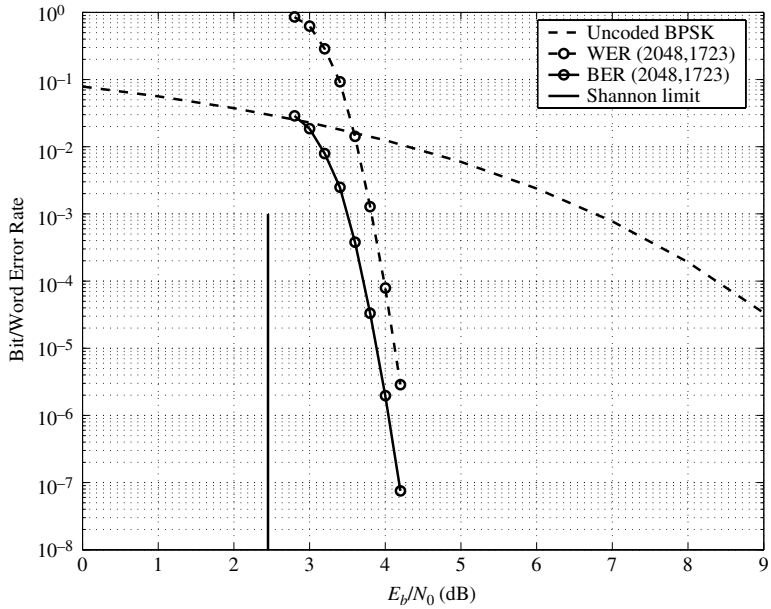


Figure 10.6 The error performance of the (2048,1723) LDPC code given in Example 10.5.

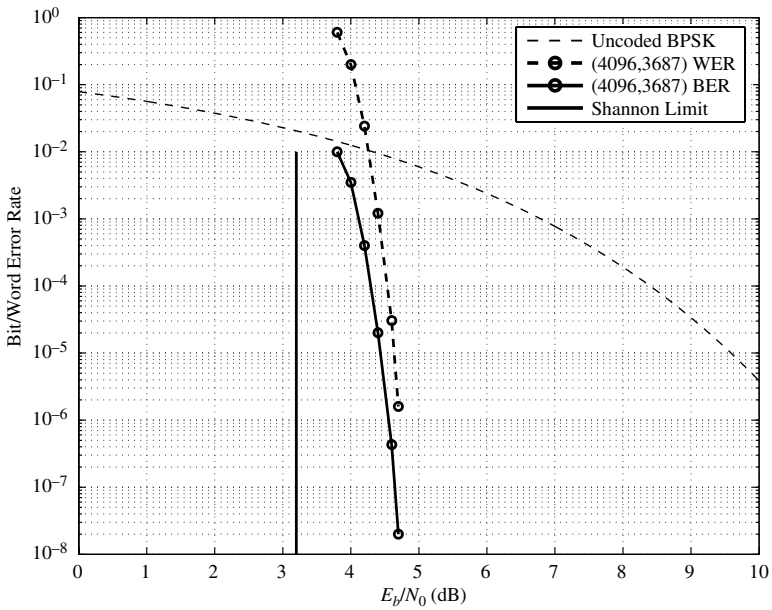


Figure 10.7 The error performance of the (4096,3687) LDPC code given in Example 10.6.

estimated to be below 10^{-21} as shown in Figure 10.8. Figure 10.9 shows that iterative decoding using the SPA converges very fast. At a BER of 10^{-6}, the gap between 10 iterations and 100 iterations is less than 0.2 dB.

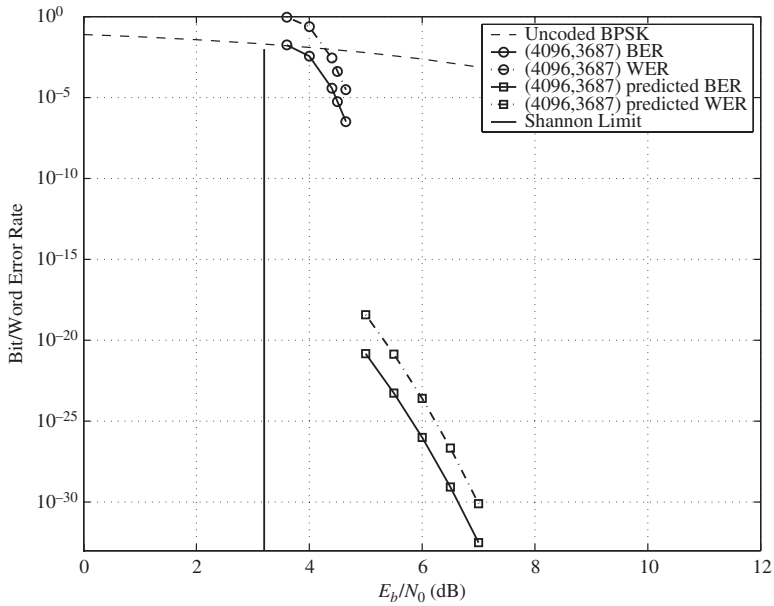


Figure 10.8 The estimated error floor of the (4096,3687) LDPC code given in Example 10.6.

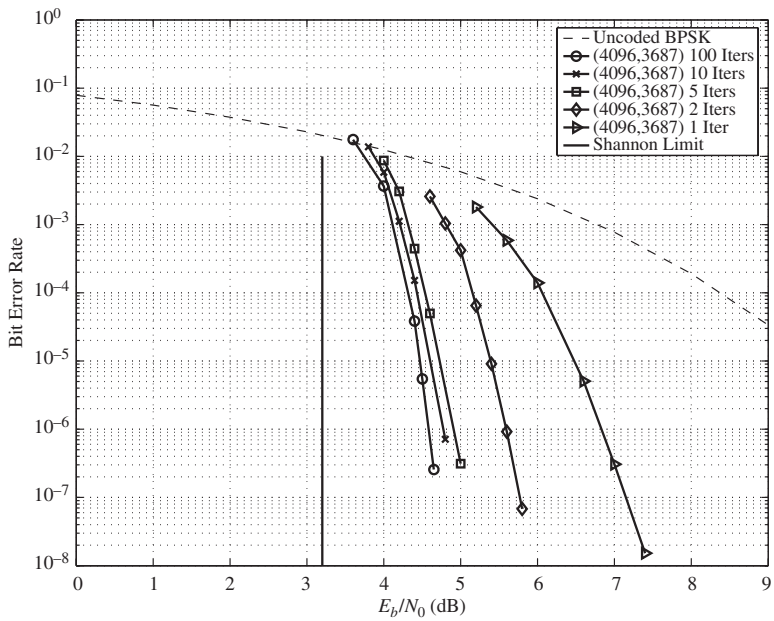


Figure 10.9 The convergence rate of decoding of the (4096,3687) LDPC code given in Example 10.6 (Iters, iterations).

For $m > 2$, the number of rows of permutation matrices in $\mathbf{H}^{(1)}_{\mathrm{EG},d}$ is much larger than the number of columns of permutation matrices in $\mathbf{H}^{(1)}_{\mathrm{EG},d}$. Using $\mathbf{H}^{(1)}_{\mathrm{EG},d}$, the longest length of a code that can be constructed is q^m. Let $\mathbf{H}^{(2)}_{\mathrm{EG},d}$ be the transpose of $\mathbf{H}^{(1)}_{\mathrm{EG},d}$, i.e.,

$$\mathbf{H}^{(2)}_{\mathrm{EG},d} = \left[\mathbf{H}^{(1)}_{\mathrm{EG},d}\right]^{\mathrm{T}}. \tag{10.17}$$

Then $\mathbf{H}^{(2)}_{\mathrm{EG},d}$ is a $q \times q^{m-1}$ array of permutation matrices of the following form:

$$\mathbf{H}^{(2)}_{\mathrm{EG},d} = \begin{bmatrix} \mathbf{B}_{0,0} & \mathbf{B}_{0,1} & \cdots & \mathbf{B}_{0,q^{m-1}-1} \\ \mathbf{B}_{1,0} & \mathbf{B}_{1,1} & \cdots & \mathbf{B}_{1,q^{m-1}-1} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{B}_{q-1,0} & \mathbf{B}_{q-1,1} & \cdots & \mathbf{B}_{q-1,q^{m-1}-1} \end{bmatrix}, \tag{10.18}$$

where $\mathbf{B}_{i,j} = \mathbf{A}^{\mathrm{T}}_{j,i}$ for $0 \le i < q$ and $0 \le j < q^{m-1}$. Using $\mathbf{H}^{(2)}_{\mathrm{EG},d}$, we can construct longer codes with higher rates. However, the largest minimum distance of an EG-LDPC code constructed from $\mathbf{H}^{(2)}_{\mathrm{EG},d}$ is lower bounded by $q+1$. The advantage of using the array $\mathbf{H}^{(1)}_{\mathrm{EG},d}$ for code construction is that the codes constructed have a wider range of minimum distances.

Example 10.7. Suppose we use the three-dimensional Euclidean geometry EG(3,13) over GF(13) for code construction. Suppose we decompose this geometry in terms of a parallel bundle $\mathcal{P}(3,2)$ of 2-flats. The decomposition results in a 169×13 array $\mathbf{H}^{(1)}_{\mathrm{EG},d}$ of 169×169 permutation matrices. Using this array, the longest length of a code that can be constructed is 2197. Suppose we take the transpose $\mathbf{H}^{(2)}_{\mathrm{EG},d}$ of $\mathbf{H}^{(1)}_{\mathrm{EG},d}$. Then $\mathbf{H}^{(2)}_{\mathrm{EG},d}$ is a 13×169 array of 169×169 permutation matrices. The longest length of a code that can be constructed from $\mathbf{H}^{(2)}_{\mathrm{EG},d}$ is 28561. Suppose we take a 6×48 subarray $\mathbf{H}^{(2)}_{\mathrm{EG},d}(6,48)$ from $\mathbf{H}^{(2)}_{\mathrm{EG},d}$. $\mathbf{H}^{(2)}_{\mathrm{EG},d}(6,48)$ is a 1014×8112 matrix over GF(2) with column and row weights 6 and 48, respectively. The null space of this matrix gives a (6,48)-regular (8112,7103) EG-LDPC code with rate 0.8756 and minimum distance at least 8. The error performance of this code with iterative decoding using the SPA (100 iterations) is shown in Figure 10.10. At a BER of 10^{-10}, the code performs 1.5 dB from the Shannon limit.

10.4 Construction of EG-LDPC Codes by Masking

In the previous section, we have presented two RC-constrained arrays of permutation matrices for constructing regular EG-LDPC codes. Although these arrays are highly structured, their constituent permutation matrices are densely packed. The density of such an array or its subarrays can be reduced by replacing a set of permutation matrices by zero matrices. This replacement of permutation matrices

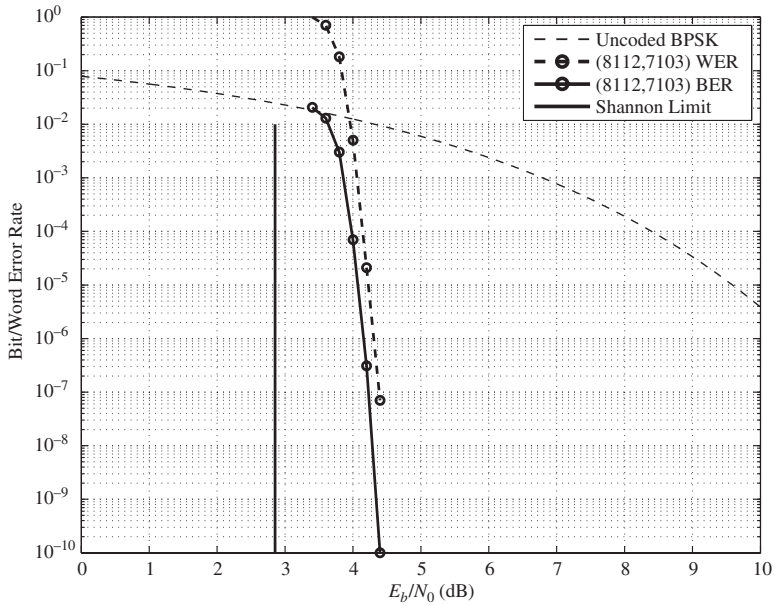


Figure 10.10 The error performance of the (8112,7103) LDPC code given in Example 10.7.

by zero matrices is referred to as *masking* [15, 23]. Masking an array of permutation matrices results in an array of permutation and zero matrices whose Tanner graph has fewer edges and hence has fewer short cycles and possibly larger girth. Masking subarrays of arrays $\mathbf{H}^{(1)}_{\mathrm{EG},d}$ and $\mathbf{H}^{(2)}_{\mathrm{EG},d}$ given by (10.16) and (10.18) results in new EG-LDPC codes, including irregular codes.

10.4.1 Masking

The masking operation can be modeled mathematically as a *special matrix product*. Let $\mathbf{H}(k,r) = [\mathbf{A}_{i,j}]$ be an RC-constrained $k \times r$ array of $n \times n$ permutation matrices over GF(2). $\mathbf{H}(k,r)$ may be a subarray of the array $\mathbf{H}^{(1)}_{\mathrm{EG},d}$ (or $\mathbf{H}^{(2)}_{\mathrm{EG},d}$) given by (10.16) (or (10.18)). Let $\mathbf{Z}(k,r) = [z_{i,j}]$ be a $k \times r$ matrix over GF(2). Define the following product of $\mathbf{Z}(k,r)$ and $\mathbf{H}(k,r)$:

$$\mathbf{M}(k,r) = \mathbf{Z}(k,r) \circledast \mathbf{H}(k,r) = [z_{i,j}\mathbf{A}_{i,j}]\,, \tag{10.19}$$

where $z_{i,j}\mathbf{A}_{i,j} = \mathbf{A}_{i,j}$ if $z_{i,j} = 1$ and $z_{i,j}\mathbf{A}_{i,j} = \mathbf{O}$, an $n \times n$ zero matrix, if $z_{i,j} = 0$. In this product operation, a set of permutation matrices in $\mathbf{H}(k,r)$ is masked by the 0-entries in $\mathbf{Z}(k,r)$. We call $\mathbf{Z}(k,r)$ the *masking matrix*, $\mathbf{H}(k,r)$ the *base array* (or matrix), and $\mathbf{M}(k,r)$ the *masked array* (or matrix). The distribution of permutation matrices in $\mathbf{M}(k,r)$ is identical to the distribution of the 1-entries in the masking matrix $\mathbf{Z}(k,r)$. The masked array $\mathbf{M}(k,r)$ is an array of permutation and zero matrices of size $n \times n$. It is a sparser matrix than the base matrix $\mathbf{H}(k,r)$. Since the base array $\mathbf{H}(k,r)$ satisfies the RC constraint, the masked array $\mathbf{M}(k,r)$ also satisfies the RC-constraint *regardless of* the masking matrix $\mathbf{Z}(k,r)$. Hence the Tanner

graph associated with $\mathbf{M}(k,r)$ has a girth of at least 6. It can be proved that, if the girth of the Tanner graph of $\mathbf{Z}(k,r)$ is $\lambda > 6$, the girth of $\mathbf{M}(k,r)$ is at least λ.

A masking matrix $\mathbf{Z}(k,r)$ can be either a *regular matrix* with constant column and constant row weights or an *irregular matrix* with *varying* column and row weights. If the masking matrix $\mathbf{Z}(k,r)$ is a regular matrix with column and row weights k_z and r_z with $1 \leq k_z \leq k$ and $1 \leq r_z \leq r$, respectively, then the masked matrix $\mathbf{M}(k,r)$ is a regular matrix with column and row weights k_z and r_z, respectively. Then the null space of $\mathbf{M}(k,r)$ gives a (k_z,r_z)-regular LDPC code $\mathcal{C}_{\text{EG,mas,reg}}$ where the subscripts "mas" and "reg" stand for "masking" and "regular," respectively. If the masking matrix $\mathbf{Z}(k,r)$ is irregular, then the column and row *weight distributions* of the masked matrix $\mathbf{M}(k,r)$ are identical to the column and row weight distributions of $\mathbf{Z}(k,r)$. The null space of $\mathbf{M}(k,r)$ gives an irregular EG-LDPC code $\mathcal{C}_{\text{EG,mas,irreg}}$, where the subscript "irreg" of $\mathcal{C}_{\text{EG,mas,irreg}}$ stands for "irregular."

10.4.2 Regular Masking

Regular masking matrices can be constructed algebraically. There are several construction methods [15]. One is presented in this section and the others will be presented in the next section and Chapter 11. Let $r = lk$, where l and k are two positive integers. Suppose we want to construct a $k \times r$ masking matrix $\mathbf{Z}(k,r)$ with column weight k_z and row weight $r_z = lk_z$. A k-tuple $\mathbf{g} = (g_0, g_1, \ldots, g_{k-1})$ over GF(2) is said to be *primitive* if executing k right cyclic-shifts of $\mathbf{g}$ gives k different k-tuples. For example, (1011000) is a primitive 7-tuple. Two primitive k-tuples are said to be *nonequivalent* if one cannot be obtained from the other by cyclic shifting. To construct $\mathbf{Z}(k,r)$, we choose l primitive nonequivalent k-tuples of weight k_z, denoted $\mathbf{g}_0, \mathbf{g}_1, \ldots, \mathbf{g}_{l-1}$. For each primitive k-tuple $\mathbf{g}_i$, we form a $k \times k$ circulant $\mathbf{G}_i$ with $\mathbf{g}_i$ as its top row and all the other $k-1$ right cyclic-shifts as the other $k-1$ rows. Then

$$\mathbf{Z}(k,r) = \begin{bmatrix} \mathbf{G}_0 & \mathbf{G}_1 & \cdots & \mathbf{G}_{l-1} \end{bmatrix}, \tag{10.20}$$

which consists of a row of l circulants of size $k \times k$. It is a $k \times lk$ matrix over GF(2) with column and row weights k_z and lk_z, respectively. The k-tuples, $\mathbf{g}_0, \mathbf{g}_1, \ldots, \mathbf{g}_{l-1}$, are called the *generators* of the circulants $\mathbf{G}_0, \mathbf{G}_1, \ldots, \mathbf{G}_{l-1}$, or the masking-matrix generators. These masking-matrix generators should be designed to achieve two objectives: (1) maximizing the girth of the Tanner graph of the masking matrix $\mathbf{Z}(k,r)$; and (2) preserving the minimum distance of the code given by the null space of base array $\mathbf{H}(k,r)$ after masking, i.e., keeping the minimum distance of the code given by the null space of the masked array $\mathbf{M}(k,r) = \mathbf{Z}(k,r) \circledast \mathbf{H}(k,r)$ the same as that of the code given by the null space of the base array $\mathbf{H}(k,r)$. *How to achieve these two objectives is unknown and is a very challenging research problem.*

A masking matrix that consists of a row of circulants can also be constructed by decomposing RC-constrained circulants constructed from finite geometries, e.g., the circulants constructed from the cyclic classes of lines of an Euclidean

geometry as described in Section 10.1.1. This construction will be presented in the next section.

Example 10.8. Consider the two-dimensional Euclidean geometry EG(2,2^7) over GF(2^7). On decomposing this geometry with respect to a chosen parallel bundle $\mathcal{P}(2,1)$ of lines, we obtain 128 connecting parallel bundles of lines with respect to $\mathcal{P}(2,1)$. From these 128 connecting parallel bundles of lines with respect to $\mathcal{P}(2,1)$, we can construct a 128×128 array $\mathbf{H}^{(1)}_{\mathrm{EG},d}$ of 128×128 permutation matrices. Take a 32×64 subarray $\mathbf{H}^{(1)}_{\mathrm{EG},d}(32,64)$ from $\mathbf{H}^{(1)}_{\mathrm{EG},d}$ as the base array for masking. Construct a 32×64 masking matrix $\mathbf{Z}(32,64) = [\mathbf{G}_0\mathbf{G}_1]$ over GF(2) that consists of a row of two 32×32 circulants, $\mathbf{G}_0$ and $\mathbf{G}_1$, whose generators are two primitive nonequivalent 32-tuples over GF(2) with weight 3:

$$\mathbf{g}_0 = (10100100000000000000000000000000)$$

and

$$\mathbf{g}_1 = (10000010000000100000000000000000).$$

These two masking-matrix generators are designed such that the Tanner graph of the masking matrix $\mathbf{Z}(k,r) = [\mathbf{G}_0\mathbf{G}_1]$ is free of cycles of length 4 and has a small number of cycles of length 6. By masking the base array $\mathbf{H}^{(1)}_{\mathrm{EG},d}(32,64)$ with $\mathbf{Z}(32,64)$, we obtain a masked array $\mathbf{M}(32,64)$ with 192 permutation and 1856 zero matrices of size 128×128. However, the base array $\mathbf{H}^{(1)}_{\mathrm{EG},d}(32,64)$ consists of 2048 permutation matrices. Therefore, $\mathbf{M}(32,64)$ is a much sparser matrix than $\mathbf{H}^{(1)}_{\mathrm{EG},d}(32,64)$. $\mathbf{M}(32,64)$ is a 4096×8192 matrix over GF(2) with column and row weights 3 and 6, respectively. The null space of $\mathbf{M}(32,64)$ gives a rate-1/2 (3,6)-regular (8192,4096) EG-LDPC code. The error performance of this code with iterative decoding using the SPA (100 iterations) is shown in Figure 10.11. At a BER of 10^{-9}, it performs 1.6 dB from the Shannon limit. It has no error floor down to a BER of 5×10^{-10}. Also included in Figure 10.11 is the error performance of a rate-1/2 (3,6)-regular (8192,4096) random code constructed by computer. We see that the EG-LDPC code outperforms the random code.

10.4.3 Irregular Masking

As defined in Chapter 5, an irregular LDPC code is given by the null space of a sparse matrix with varying column and/or row weights. Consequently, its Tanner graph $\mathcal{T}$ has varying variable-node degrees and/or varying check-node degrees. As shown in Chapter 5, the degree distributions of these two types of nodes are expressed in terms of two polynomials, $\tilde{\lambda}(X) = \sum_{i=1}^{d_{\tilde{\lambda}}} \tilde{\lambda}_i X^{i-1}$ and $\tilde{\rho}(X) = \sum_{i=1}^{d_{\tilde{\rho}}} \tilde{\rho}_i X^{i-1}$, where $\tilde{\lambda}_i$ and $\tilde{\rho}_i$ denote the fractions of variable and check nodes in $\mathcal{T}$ with degree i, respectively, and $d_{\tilde{\lambda}}$ and $d_{\tilde{\rho}}$ denote the maximum variable- and check-node degrees, respectively. Since the variable and check nodes of $\mathcal{T}$ correspond to the columns and rows of the adjacency matrix $\mathbf{H}$ of $\mathcal{T}$, $\tilde{\lambda}(X)$ and $\tilde{\rho}(X)$ also give the column and row weight distributions of $\mathbf{H}$. It has been shown

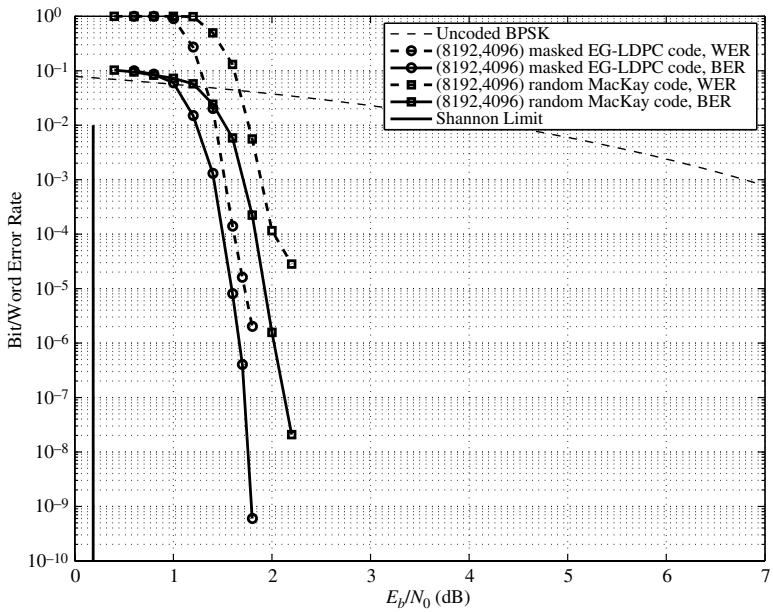


Figure 10.11 Error performances of the (8192,4096) masked EG-LDPC code and a (8192,4096) random MacKay code given in Example 10.8.

that the error performance of an irregular LDPC code depends on the variable- and check-node degree distributions of its Tanner graph $\mathcal{T}$ [27] and Shannon-limit-approaching LDPC codes can be designed by optimizing the two degree distributions by utilizing the evolution of the probability densities (called *density evolution*, Chapter 9) of the messages passed between the two types of nodes in a belief-propagation decoder. In code construction, once the degree distributions $\tilde{\lambda}(X)$ and $\tilde{\rho}(X)$ have been derived, a Tanner graph is constructed by connecting the variable nodes and check nodes with edges based on these two degree distributions. Since the selection of edges in the construction of a Tanner graph is not unique, edge selection is carried out in a *random manner* by computer search. During the edge-selection process, an effort must be made to ensure that the resultant Tanner graph does not contain short cycles, especially cycles of length 4. Once the code's Tanner graph $\mathcal{T}$ has been constructed, the incidence matrix $\mathbf{H}$ of $\mathcal{T}$ is derived from the edges that connect the variable and check nodes of $\mathcal{T}$ and is used as the parity-check matrix of an irregular LDPC code. The null space of $\mathbf{H}$ gives a *random-like* irregular LDPC code.

Geometry decomposition (presented in Section 10.3) and array masking (presented earlier) can be used for constructing irregular LDPC codes based on the degree distributions of variable and check nodes of their Tanner graphs derived from density evolution. First we choose an appropriate Euclidean geometry with which to construct an RC-constrained array $\mathbf{H}$ of permutation matrices (either $\mathbf{H}^{(1)}_{\mathrm{EG},d}$ or $\mathbf{H}^{(2)}_{\mathrm{EG},d}$) using geometry decomposition. Take a $k \times r$ subarray $\mathbf{H}(k,r)$ from $\mathbf{H}$ such that the null space of $\mathbf{H}(k,r)$ gives an LDPC code with length and

rate equal to or close to the desired length n and rate R. Let $\tilde{\lambda}(X)$ and $\tilde{\rho}(X)$ be the designed degree distributions of the variable and check nodes of the Tanner graph of a desired LDPC code of rate R. Construct a $k \times r$ masking matrix $\mathbf{Z}(k,r)$ over GF(2) with column and row weight distributions equal to or close to $\tilde{\lambda}(X)$ and $\tilde{\rho}(X)$, respectively. Then the masked matrix $\mathbf{M}(k,r) = \mathbf{Z}(k,r) \circledast \mathbf{H}(k,r)$ has column and row weight distributions *identical* to or *close* to $\tilde{\lambda}(X)$ and $\tilde{\rho}(X)$. The null space of $\mathbf{M}(k,r)$ gives an irregular LDPC code whose Tanner graph has variable- and check-node degree distributions identical to or close to $\tilde{\lambda}(X)$ and $\tilde{\rho}(X)$.

The above construction of irregular LDPC codes by masking and geometry decomposition is basically algebraic. It avoids the effort needed to construct a large random graph by computer search without cycles of length 4. Since the size of the masking matrix $\mathbf{Z}(k,r)$ is relatively small compared with the size of the code's Tanner graph, it is very easy to construct a matrix with column and row weight distributions identical to or close to the degree distributions $\tilde{\lambda}(X)$ and $\tilde{\rho}(X)$ derived from density evolution. Since the base array $\mathbf{H}(k,r)$ satisfies the RC constraint, the masked matrix $\mathbf{M}(k,r)$ also satisfies the RC-constraint regardless of whether the masking matrix $\mathbf{Z}(k,r)$ satisfies the RC-constraint. Hence the irregular LDPC code given by the null space of $\mathbf{M}(k,r)$ contains no cycles of length 4 in its Tanner graph.

In [27], the degree distributions of a code graph that optimize the code performance over the AWGN channel for a given code rate are derived under the assumptions of infinite code length, a cycle-free Tanner graph, and an infinite number of decoding iterations. These degree distributions are no longer optimal when used for constructing codes of finite length and, in general, result in high error floors, mostly due to the large number of low-degree variable nodes, especially degree-2 variable nodes. Hence they must be adjusted for constructing codes of finite length. Adjustment of degree distributions for finite-length LDPC codes will be discussed in Chapter 11. In the following, we simply give an example to illustrate how to construct an irregular code by masking an array of permutation matrices using degree distributions of the two types of node in a Tanner graph.

Example 10.9. The following degree distributions of variable and check nodes of a Tanner graph are designed for an irregular LDPC code of length around 4000 and rate 0.82: $\tilde{\lambda}(X) = 0.4052X^2 + 0.3927X^3 + 0.1466X^7 + 0.0555X^8$ and $\tilde{\rho}(X) = 0.3109X^{22} + 0.6891X^{23}$. From this pair of degree distributions, we construct a 12×63 masking matrix $\mathbf{Z}(12,63)$ with column and row weight distributions given in Table 10.1. Note that the column and row weight distributions of the constructed masking matrix $\mathbf{Z}(12,63)$ are not exactly identical to the designed degree distributions of the variable and check nodes given above. The average column and row weights of the masking matrix $\mathbf{Z}(12,63)$ are 4.19 and 23.66. Next we construct a 64×64 array $\mathbf{H}^{(1)}_{\mathrm{EG},d}$ of 64×64 permutation matrices over GF(2) based on the two-dimensional Euclidean geometry EG(2,2^6). Take a 12×63 subarray $\mathbf{H}^{(1)}_{\mathrm{EG},d}(12,63)$ from $\mathbf{H}^{(1)}_{\mathrm{EG},d}$. $\mathbf{H}^{(1)}_{\mathrm{EG},d}(12,63)$ is a 768×4032 matrix over GF(2) with column and row weights 12 and 63, respectively. By masking $\mathbf{H}^{(1)}_{\mathrm{EG},d}(12,63)$ with

Table 10.1. Column and row weight distributions of the masking matrix $\mathbf{Z}(12, 63)$ of Example 10.9

Column weight distribution		Row weight distribution	
Column weight	No. of columns	Row weight	No. of rows
3	25	23	4
4	25	24	8
8	9		
9	4		

Table 10.2. The weight distribution of the masked matrix $\mathbf{M}(12, 63)$ of Example 10.9

Column weight distribution		Row weight distribution	
Column weight	No. of columns	Row weight	No. of rows
3	1536	23	256
4	1664	24	512
8	576		
9	256		

$\mathbf{Z}(12, 63)$, we obtain a masked 768×4032 matrix $\mathbf{M}(12, 63) = \mathbf{Z}(12, 63) \circledast \mathbf{H}^{(1)}_{\mathrm{EG},d}(12, 63)$ with the column and row weight distributions given in Table 10.2. The null space of $\mathbf{M}(12, 63)$ gives a (4032,3264) irregular EG-LDPC code with rate 0.8113 whose Tanner graph has a girth of at least 6. The performance of this code over the binary-input AWGN channel decoded with the iterative decoding using the SPA with 100 iterations is shown in Figure 10.12. At a BER of 10^{-6}, it performs 1.2 dB from the Shannon limit.

10.5 Construction of QC-EG-LDPC Codes by Circulant Decomposition

In Section 10.1, we have shown that RC-constrained circulants can be constructed from type-1 incidence vectors of lines in a Euclidean geometry not passing through the origin. These circulants can be used to construct either cyclic EG-LDPC codes as shown in Section 10.1.1 or QC-EG-LDPC codes as shown in Section 10.1.2. In this section, we show that QC-EG-LDPC codes can be constructed by decomposing these circulants into arrays of circulants using *column and row decompositions* [28].

Consider an $n \times n$ circulant $\mathbf{G}$ over GF(2) with both column and row weight w. Since the column and row weights of circulant $\mathbf{G}$ are both w, we say that $\mathbf{G}$ has weight w. Label the rows and columns of $\mathbf{G}$ from 0 to $n - 1$. $\mathbf{G}$ can be decomposed into a row of $n \times n$ circulants by *column splitting*. For $1 \leq t \leq w$, let $w_0, w_1, \ldots, w_{t-1}$ be a set of positive integers such that $w = w_0 + w_1 + \cdots + w_{t-1}$. Let $\mathbf{g}$ be the first column of $\mathbf{G}$. Partition the locations of the w 1-components

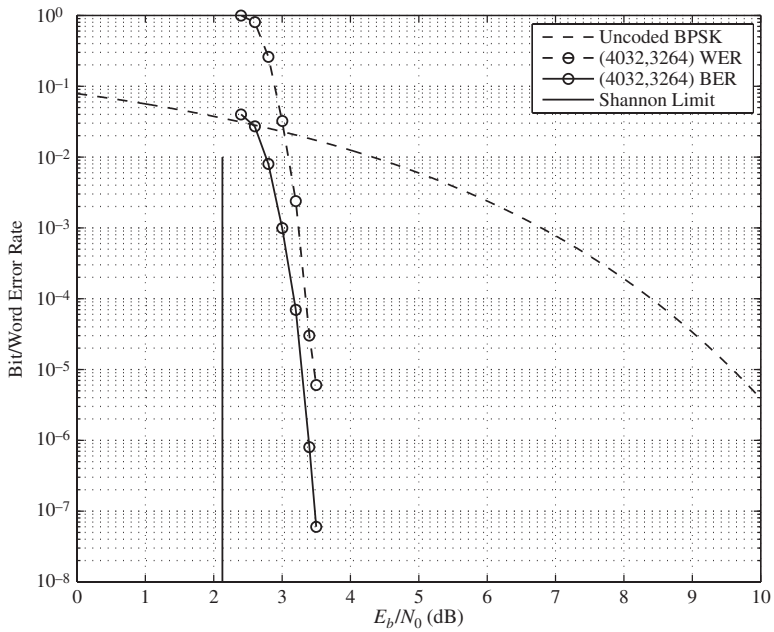


Figure 10.12 The error performance of the (4032,3264) irregular EG-LDPC code given in Example 10.9.

of $\mathbf{g}$ into t disjoint sets, $R_0, R_1, \ldots, R_{t-1}$. The jth location-set R_j consists of w_j locations of $\mathbf{g}$ where the components are 1-components. Split $\mathbf{g}$ into t columns of the same length n, $\mathbf{g}_0, \mathbf{g}_1, \ldots, \mathbf{g}_{t-1}$, with the w 1-components of $\mathbf{g}$ distributed among these new columns. For $0 \leq j < t$, we put the w_j 1-components of $\mathbf{g}$ at the locations in R_j into the jth new column $\mathbf{g}_j$ at the locations in R_j of $\mathbf{g}_j$ and set all the other components of $\mathbf{g}_j$ to zeros. For each new column $\mathbf{g}_j$, we form an $n \times n$ circulant $\mathbf{G}_j$ with $\mathbf{g}_j$ as the first column and its $n-1$ downward cyclic-shifts as the other $n-1$ columns. This results in t circulants of size $n \times n$, $\mathbf{G}_0, \mathbf{G}_1, \ldots, \mathbf{G}_{t-1}$, with weights $w_0, w_1, \ldots, w_{t-1}$, respectively. These circulants are called the *column descendants* of $\mathbf{G}$. The above process of column splitting decomposes the circulant $\mathbf{G}$ into a row of t $n \times n$ circulants,

$$\mathbf{G}_{\text{col,decom}} = \begin{bmatrix} \mathbf{G}_0 & \mathbf{G}_1 & \cdots & \mathbf{G}_{t-1} \end{bmatrix}. \tag{10.21}$$

$\mathbf{G}_{\text{col,decom}}$ is called a *column decomposition* of $\mathbf{G}$. The subscript "col, decom" stands for "column decomposition." Since $\mathbf{G}$ satisfies the RC constraint, it is clear that $\mathbf{G}_{\text{col,decom}}$ and each circulant $\mathbf{G}_j$ in $\mathbf{G}_{\text{col,decom}}$ also satisfy the RC-constraint, i.e., column decomposition preserves the RC-constraint structure.

Let c be a positive integer such that $1 \leq c \leq \max\{w_j\colon 0 \leq j < t\}$. For $0 \leq j < t$, let $w_{0,j}, w_{1,j}, \ldots, w_{c-1,j}$ be a set of non-negative integers such that $w_{0,j} + w_{1,j} + \cdots + w_{c-1,j} = w_j$. Each circulant $\mathbf{G}_j$ in $\mathbf{G}_{\text{col,decom}}$ can be decomposed into a column of c circulants of size $n \times n$ with weights $w_{0,j}, w_{1,j}, \ldots, w_{c-1,j}$, respectively. This is accomplished by splitting the first row $\mathbf{r}_j$ of $\mathbf{G}_j$ into c rows of the same length n, denoted $\mathbf{r}_{0,j}, \mathbf{r}_{1,j}, \ldots, \mathbf{r}_{c-1,j}$, with w_j 1-components of $\mathbf{r}_j$ distributed

among these c new rows, where the ith row $\mathbf{r}_{i,j}$ contains $w_{i,j}$ 1-components of $\mathbf{r}_j$. The row-splitting process is exactly the same as the column-splitting process described above. For $0 \leq j < t$, partition the locations of the w_j 1-components of $\mathbf{r}_j$ into c disjoint sets, $R_{0,j}, R_{1,j}, \ldots, R_{c-1,j}$. For $0 \leq i < c$, the ith location-set $R_{i,j}$ consists of $w_{i,j}$ locations of $\mathbf{r}_j$ where the components are 1-components. For $0 \leq i < c$, we put the $w_{i,j}$ 1-components of $\mathbf{r}_j$ at the locations in $R_{i,j}$ into the ith new row $\mathbf{r}_{i,j}$ at the locations in $R_{i,j}$ of $\mathbf{r}_{i,j}$ and set all the other components of $\mathbf{r}_{i,j}$ to zeros. For each new row $\mathbf{r}_{i,j}$ with $0 \leq i < c$, we form an $n \times n$ circulant $\mathbf{G}_{i,j}$ with $\mathbf{r}_{i,j}$ as the first row and its $n-1$ right cyclic-shifts as the other $n-1$ rows. The above row decomposition of $\mathbf{G}_i$ results in a column of c circulants of size $n \times n$,

$$\mathbf{G}_{\text{row,decom},j} = \begin{bmatrix} \mathbf{G}_{0,j} \\ \mathbf{G}_{1,j} \\ \vdots \\ \mathbf{G}_{c-1,j} \end{bmatrix}, \tag{10.22}$$

where the ith circulant $\mathbf{G}_{i,j}$ has weight $w_{i,j}$ with $0 \leq i < c$. The circulants $\mathbf{G}_{0,j}, \mathbf{G}_{1,j}, \ldots, \mathbf{G}_{c-1,j}$ are called the *row descendants* of $\mathbf{G}_j$ and $\mathbf{G}_{\text{row,decom},j}$ is called a *row decomposition* of $\mathbf{G}_j$. In row splitting, we allow $w_{i,j} = 0$. If $w_{i,j} = 0$, then $\mathbf{G}_{i,j}$ is an $n \times n$ zero matrix. Since each circulant $\mathbf{G}_j$ in $\mathbf{G}_{\text{col,decom}}$ satisfies the RC-constraint, the row decomposition $\mathbf{G}_{\text{row,decom},j}$ of $\mathbf{G}_j$ must also satisfy the RC-constraint.

On replacing each $n \times n$ circulant $\mathbf{G}_j$ in $\mathbf{G}_{\text{col,decom}}$ given by (10.21) by its row decomposition $\mathbf{G}_{\text{row,decom},j}$, we obtain the following RC-constrained $c \times t$ array of $n \times n$ circulants over GF(2):

$$\mathbf{H}_{\text{array,decom}} = \begin{bmatrix} \mathbf{G}_{0,0} & \mathbf{G}_{0,1} & \cdots & \mathbf{G}_{0,t-1} \\ \mathbf{G}_{1,0} & \mathbf{G}_{1,1} & \cdots & \mathbf{G}_{1,t-1} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{G}_{c-1,0} & \mathbf{G}_{c-1,1} & \cdots & \mathbf{G}_{c-1,t-1} \end{bmatrix}, \tag{10.23}$$

where the constituent circulant $\mathbf{G}_{i,j}$ has weight $w_{i,j}$ for $0 \leq i < c$ and $0 \leq j < t$. $\mathbf{H}_{\text{array,decom}}$ is called a $c \times t$ *array decomposition* of circulant $\mathbf{G}$. If all the constituent circulants in $\mathbf{H}_{\text{array,decom}}$ have the same weight σ, then $\mathbf{H}_{\text{array,decom}}$ is an RC-constrained $cn \times tn$ matrix with constant column and row weights $c\sigma$ and $t\sigma$, respectively. If all the constituent circulants of $\mathbf{H}_{\text{array,decom}}$ have weights equal to 1, then $\mathbf{H}_{\text{array,decom}}$ is a $c \times t$ array of $n \times n$ circulant permutation matrices.

For any pair (k, r) of integers with $1 \leq k \leq c$ and $1 \leq r \leq t$, let $\mathbf{H}_{\text{array,decom}}(k, r)$ be a $k \times r$ subarray of $\mathbf{H}_{\text{array,decom}}$. The null space of $\mathbf{H}_{\text{array,decom}}(k, r)$ gives a QC-EG-LDPC code $\mathcal{C}_{\text{EG},qc,\text{decom}}$ over GF(2) of length rn, whose Tanner graph has a girth of at least 6. The above construction gives a class of QC-EG-LDPC codes.

Example 10.10. Consider the three-dimensional Euclidean geometry EG(3,2^3). As shown in Example 10.2, using the type-1 incidence vectors of the lines in this geometry not passing through the origin, we can construct nine circulants of size 511×511, $\mathbf{G}_0, \mathbf{G}_1, \ldots, \mathbf{G}_8$, each with weight 8. Suppose we take eight of these circulants and arrange them in a row

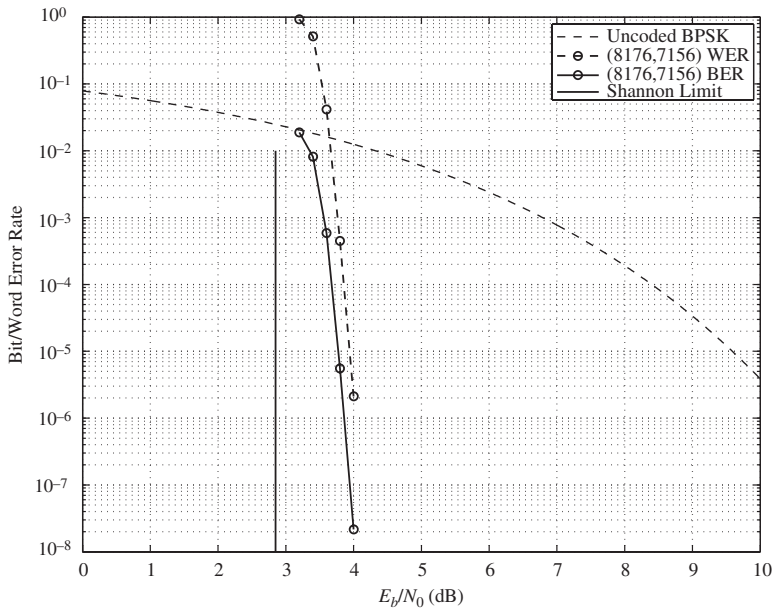


Figure 10.13 The error performance of the (8176,7156) QC-EG-LDPC code given in Example 10.10.

$\mathbf{G} = [\mathbf{G}_0\mathbf{G}_1 \cdots \mathbf{G}_7]$. For $0 \leq j < 8$, decompose each constituent circulant $\mathbf{G}_j$ in $\mathbf{G}$ into a 2×2 array of 511×511 circulants over GF(2),

$$\mathbf{M}_j = \begin{bmatrix} \mathbf{G}_{0,0}^{(j)} & \mathbf{G}_{0,1}^{(j)} \\ \mathbf{G}_{1,0}^{(j)} & \mathbf{G}_{1,1}^{(j)} \end{bmatrix},$$

where each constituent circulant in $\mathbf{M}_j$ has weight 2. $\mathbf{M}_j$ is a 1022×1022 matrix over GF(2) with both column weight and row weight 4. On replacing each constituent circulant $\mathbf{G}_j$ in $\mathbf{G}$ by its 2×2 array decomposition $\mathbf{M}_j$, we obtain a 2×16 array $\mathbf{H}_{\text{array,decom}}$ of 511×511 circulants, each with weight 2. $\mathbf{H}_{\text{array,decom}}$ is a 2044×8176 matrix with column and row weights 4 and 32, respectively. The null space of this matrix gives a (4,32)-regular (8176,7156) QC-EG-LDPC code with rate 0.8752, whose Tanner graph has a girth of at least 6. The error performance of this code decoded with iterative decoding using the SPA (50 iterations) is shown in Figures 10.13 and 10.14. At a BER of 10^{-6}, the code performs 1 dB from the Shannon limit. The error floor of this code for the bit-error-rate performance is estimated to be below 10^{-15}. This code has been selected for use in the NASA Landsat Data Continuation scheduled for launch in July 2011. Figure 10.14 shows the rate of decoding convergence of the code.

A VLSI decoder for this code has been built. Using this decoder, bit-error performance down to 10^{-14} can be simulated as shown in Figure 10.15. We see that there is no error floor even down to this low bit error rate. This code is also being considered for possible application to NASA's Cruise Exploration Shuttle mission.

Note that the matrix $\mathbf{G}_{\text{col,decom}}$ given by (10.21) consists of a row of t $n \times n$ circulants. For $1 \leq k \leq t$, any k circulants in $\mathbf{G}_{\text{col,decom}}$ can be used to form a

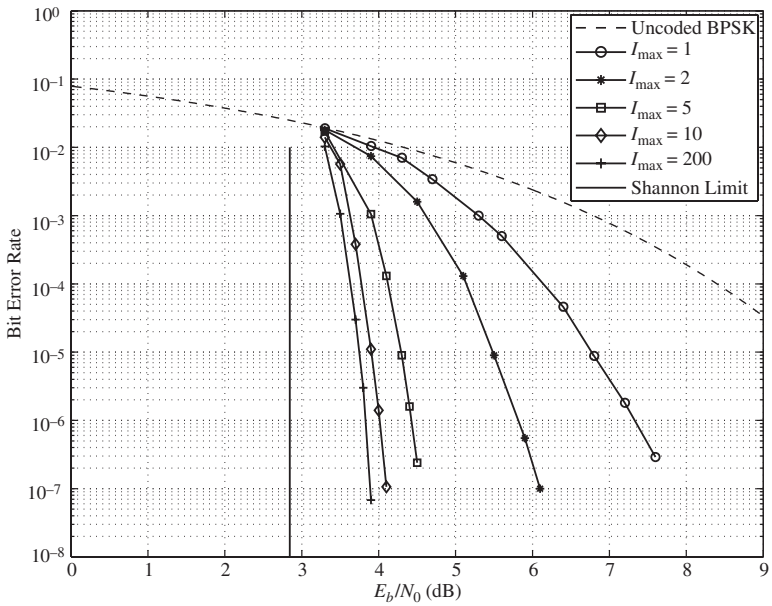


Figure 10.14 The convergence rate of decoding of the (8176,7156) QC-EG-LDPC code given in Example 10.10.

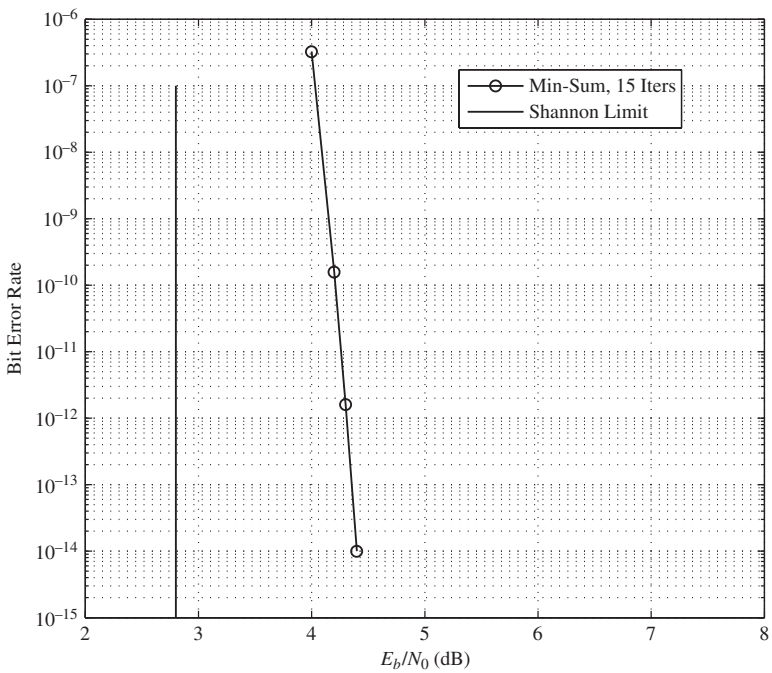


Figure 10.15 The VLSI-simulated error performance of the (8176,7156) QC-EG-LDPC code given in Example 10.10.

masking matrix for masking an $n \times kn$ array of permutation matrices constructed as in Section 10.3 for code construction. For example, consider the two-dimensional Euclidean geometry EG(2,2^3) over GF(2^3). Using the type-1 incidence vectors of lines in EG(2,2^3) not passing through the origin, we can construct a single 63×63 circulant $\mathbf{G}$ with both column weight and row weight 8. By column decomposition, we can decompose $\mathbf{G}$ into two 63×63 column-descendant circulants, $\mathbf{G}_0$ and $\mathbf{G}_1$, each having both column weight and row weight 4. Then $[\mathbf{G}_0\mathbf{G}_1]$ is a 63×126 matrix over GF(2) with column and row weights 4 and 8, respectively, and the matrix $[\mathbf{G}_0\mathbf{G}_1]$ can be used as a (4,8)-regular masking matrix. We can also decompose $\mathbf{G}$ into three 63×63 column-descendant circulants, $\mathbf{G}_0^*$, $\mathbf{G}_1^*$, and $\mathbf{G}_2^*$, with weights 3, 3, and 2, respectively, by column decomposition. We can use $\mathbf{G}_0^*$ and $\mathbf{G}_1^*$ to form a 63×126 (3,6)-regular masking matrix $[\mathbf{G}_0^*\mathbf{G}_1^*]$ or we can use $\mathbf{G}_0^*$, $\mathbf{G}_1^*$, and $\mathbf{G}_2^*$ to form an irregular masking matrix $[\mathbf{G}_0^*\mathbf{G}_1^*\mathbf{G}_2^*]$ with two different column weights.

If the constituent circulants of the array $\mathbf{H}_{\text{array,decom}}$ given by (10.23) are circulant permutation matrices, then $\mathbf{H}_{\text{array,decom}}$ or its subarray can be used as a base array for masking to construct QC-LDPC codes.

10.6 Construction of Cyclic and QC-LDPC Codes Based on Projective Geometries

In the last five sections, various methods have been presented for constructing cyclic, quasi-cyclic, and regular LDPC codes based on the cyclic and parallel structures of lines in Euclidean geometries. In this section, we will show that lines of projective geometries can also be used for constructing LDPC codes. However, the lines in a projective geometry have the cyclic structure but not the parallel structure. As a result, only the construction methods based on the cyclic structure of lines presented in Sections 10.1, 10.4, and 10.5 can be applied to construct cyclic and quasi-cyclic LDPC codes based on the lines of projective geometries. Projective geometries and their structural properties have been discussed in Chapter 2. In the following, we give a brief review of the structural properties of lines in these geometries for code construction.

10.6.1 Cyclic PG-LDPC Codes

Consider the m-dimensional projective geometry PG(m,q) over GF(q) with $m \geq 2$. This geometry consists of

$$n = (q^{m+1} - 1)/(q - 1) \tag{10.24}$$

points and

$$J_4 \triangleq J_{\text{PG}}(m, 1) = \frac{(q^m - 1)(q^{m+1} - 1)}{(q^2 - 1)(q - 1)} \tag{10.25}$$

lines. Each line consists of $q + 1$ points. Two lines in PG(m,q) either do not have any point in common or they intersect at one and only one point. For any given

point, there are

$$g \triangleq g_{\mathrm{PG}}(m,1) = (q^m - 1)/(q-1) \tag{10.26}$$

lines that intersect at this point. Let $\mathrm{GF}(q^{m+1})$ be the extension field of $\mathrm{GF}(q)$. Let α be a primitive element in $\mathrm{GF}(q^{m+1})$. The $n = (q^{m+1}-1)/(q-1)$ points of $\mathrm{PG}(m,q)$ can be represented by the n elements $\alpha^0, \alpha, \alpha^2, \ldots, \alpha^{n-1}$ of $\mathrm{GF}(q^{m+1})$ (see Chapter 2).

Let $\mathcal{L}$ be a line in $\mathrm{PG}(m,q)$. The incidence vector of $\mathcal{L}$ is defined as an n-tuple, $\mathbf{v}_{\mathcal{L}} = (v_0, v_1, \ldots, v_{n-1})$, whose components correspond to the n points of $\mathrm{PG}(m,q)$, where $v_j = 1$ if and only if α^j is a point on $\mathcal{L}$, otherwise $v_j = 0$. The weight of the incidence vector of a line in $\mathrm{PG}(m,q)$ is $q+1$. The (right or left) cyclic-shift of $\mathbf{v}_{\mathcal{L}}$ is the incidence vector of another line in $\mathrm{PG}(m,q)$. Form a $J_4 \times n$ matrix $\mathbf{H}_{\mathrm{PG},c}$ over $\mathrm{GF}(2)$ with the incidence vectors of all the J_4 lines in $\mathrm{PG}(m,q)$ as rows. $\mathbf{H}_{\mathrm{PG},c}$ has column and row weights $(q^m-1)/(q-1)$ and $q+1$, respectively. Since no two lines in $\mathrm{PG}(m,q)$ have more than one point in common, no two rows (or two columns) of $\mathbf{H}_{\mathrm{PG},c}$ have more than one 1-component in common. Hence, $\mathbf{H}_{\mathrm{PG},c}$ satisfies the RC-constraint. The null space of $\mathbf{H}_{\mathrm{PG},c}$ gives a cyclic LDPC code $\mathcal{C}_{\mathrm{PG},c}$ of length $n = (q^{m+1}-1)/(q-1)$ with minimum distance at least $(q^m-1)/(q-1)+1$.

The above construction gives a class of cyclic PG-LDPC codes. The generator polynomial of $\mathcal{C}_{\mathrm{PG},c}$ can be determined in exactly the same way as was given for a cyclic EG-LDPC code in Section 10.1.1. Cyclic PG-LDPC codes were also discovered in the late 1960s [19, 20, 29–31] and shown to form a subclass of polynomial codes [32, 38], again not being recognized as being LDPC codes until 2000 [1].

An interesting case is that in which the code-construction geometry is a two-dimensional projective geometry $\mathrm{PG}(2,q)$ over $\mathrm{GF}(q)$. For this case, the geometry consists of q^2+q+1 points and q^2+q+1 lines. The parity-check matrix $\mathbf{H}_{\mathrm{PG},c}$ of the cyclic code $\mathcal{C}_{\mathrm{PG},c}$ constructed from the incidence vectors of the lines in $\mathrm{PG}(2,q)$ consists of a single $(q^2+q+1) \times (q^2+q+1)$ circulant over $\mathrm{GF}(2)$ with both column weight and row weight $q+1$. If q is a power of 2, say 2^s, then $\mathcal{C}_{\mathrm{PG},c}$ has the following parameters [4, 23]:

Length	$2^{2s}+2^s+1$,
Number of parity-check bits	3^s+1,
Minimum distance (lower bound)	2^s+2.

Let $\mathbf{g}(X)$ be the generator polynomial of $\mathcal{C}_{\mathrm{PG},c}$ and let α be a primitive element of $\mathrm{GF}(2^{3s})$. Let h be a non-negative integer less than $2^{3s}-1$. Then α^h is a root of $\mathbf{g}(X)$ if and only if h is divisible by 2^s-1 and

$$0 \le \max_{0 \le l < s} W_{2^s}(h^{(l)}) = 2^s - 1. \tag{10.27}$$

This result was proved in [21].

Example 10.11. Consider the two-dimensional projective geometry $\mathrm{PG}(2,2^5)$ over $\mathrm{GF}(2^5)$. This geometry consists of 1057 points and 1057 lines. Each line consists of 33

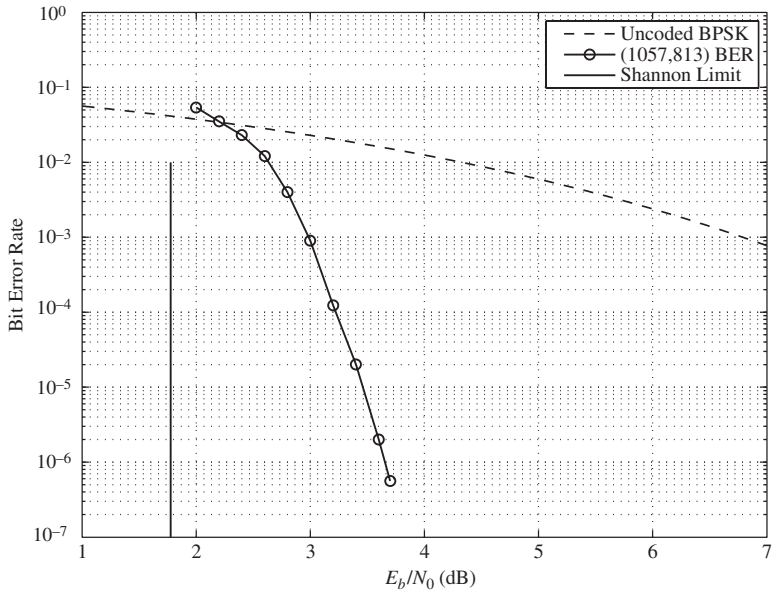


Figure 10.16 The error performance of the (1057,813) cyclic PG-LDPC code given in Example 10.11.

points. Using the incidence vectors of the 1057 lines in PG(2,2^5), we can form a single 1057×1057 circulant $\mathbf{H}_{\mathrm{PG},c}$ with both column weight and row weight 33. The null space of this circulant gives a (33,33)-regular (1057,813) cyclic PG-LDPC code with rate 0.7691 and minimum distance at least 34, whose Tanner graph has a girth of at least 6. The error performance of this code with iterative decoding using the SPA with 100 iterations is shown in Figure 10.16. At a BER of 10^{-6}, it performs 1.85 dB from the Shannon limit. Since the code has a very large minimum distance, its error performance should have a very low error floor.

In the above construction of a cyclic PG-LDPC code, we use the incidence vectors of all the lines in a projective geometry to form the parity-check matrix. A larger class of cyclic PG-LDPC codes can be constructed by grouping the incidence vectors of the lines in a projective geometry. For a Euclidean geometry EG(m,q), the type-1 incidence vector of any line is primitive, i.e., the incidence vector $\mathbf{v}_{\mathcal{L}}$ of a line $\mathcal{L}$ in EG(m,q) not passing through the origin and its $q^m - 2$ cyclic-shifts are all different $(q^m - 1)$-tuples over GF(2). However, this is not the case for a projective geometry PG(m,q). Whether the incidence vector of a line in PG(m,q) is primitive or not depends on whether the dimension m of the geometry is even or odd [4, 11, 12].

For even m, the incidence vector of each line in PG(m,q) is primitive. For this case, the incidence vectors of the lines in PG(m,q) can be partitioned into

$$K_{c,\mathrm{even}} = K_{c,\mathrm{PG}}^{(\mathrm{e})}(m,1) = (q^m - 1)/(q^2 - 1) \tag{10.28}$$

cyclic classes, $Q_0, Q_1, \ldots, Q_{K_{c,\text{even}}-1}$, each consisting of $n = (q^{m+1}-1)/(q-1)$ incidence vectors. For $0 \le i < K_{c,\text{even}}$, the incidence vectors of cyclic class Q_i can be obtained by cyclically shifting any incidence vector in Q_i n times. For each cyclic class Q_i, we can form an $n \times n$ circulant $\mathbf{H}_{c,i}$ using the incidence vectors in Q_i as rows arranged in cyclic order. Both the column weight and the row weight of $\mathbf{H}_{c,i}$ are $q+1$. For $1 \le k \le K_{c,\text{even}}$, we form the following $kn \times n$ matrix:

$$\mathbf{H}^{(1)}_{\text{PG},c,k} = \begin{bmatrix} \mathbf{H}_{c,0} \\ \mathbf{H}_{c,1} \\ \vdots \\ \mathbf{H}_{c,k-1} \end{bmatrix}, \tag{10.29}$$

which consists of a column of k circulants of size $n \times n$. $\mathbf{H}^{(1)}_{\text{PG},c,k}$ is a $kn \times n$ matrix over GF(2) with column and row weights $k(q+1)$ and $q+1$, respectively, and it satisfies the RC-constraint. The null space of $\mathbf{H}^{(1)}_{\text{PG},c,k}$ gives a cyclic PG-LDPC code of length n with minimum distance at least $k(q+1)+1$, whose Tanner graph has a girth of at least 6.

For odd $m \ge 3$, there are $l_0 = (q^{m+1}-1)/(q^2-1)$ lines in PG(m,q) whose incidence vectors are nonprimitive and there are

$$J_5 = \frac{q(q^{m+1}-1)(q^{m-1}-1)}{(q^2-1)(q-1)} \tag{10.30}$$

lines whose incidence vectors are primitive. The J_5 primitive incidence vectors of lines in PG(m,q) with odd m can be partitioned into

$$K_{c,\text{odd}} = K^{(\text{o})}_{c,\text{PG}}(m,1) = \frac{q(q^{m-1}-1)}{q^2-1} \tag{10.31}$$

cyclic classes, $Q_0, Q_1, \ldots, Q_{K_{c,\text{odd}}-1}$, each consisting of $n = (q^{m+1}-1)/(q-1)$ incidence vectors. Using these cyclic classes of incidence vectors, we can form $K_{c,\text{odd}}$ $n \times n$ circulants over GF(2), $\mathbf{H}_{c,0}, \mathbf{H}_{c,1}, \ldots, \mathbf{H}_{c,K_{c,\text{odd}}-1}$. These circulants can then be used to form low-density parity-check matrices of the form given by (10.29) to generate cyclic PG-LDPC codes.

10.6.2 Quasi-Cyclic PG-LDPC Codes

For $0 \le i < K_{c,\text{even}}$ (or $K_{c,\text{odd}}$), let $\mathbf{G}_{c,i}$ be an $n \times n$ circulant with the incidence vectors of the cyclic class Q_i as columns arranged in downward cyclic order. $\mathbf{G}_{c,i}$ has both column weight and row weight $q+1$. For $1 \le k < K_{c,\text{even}}$ (or $K_{c,\text{odd}}$), we form the following $n \times kn$ matrix over GF(2):

$$\mathbf{H}^{(2)}_{\text{PG},qc,k} = \begin{bmatrix} \mathbf{G}_{c,0} & \mathbf{G}_{c,1} & \cdots & \mathbf{G}_{c,k-1} \end{bmatrix}, \tag{10.32}$$

which consists of a row of k $n \times n$ circulants. The column and row weights of $\mathbf{H}^{(2)}_{\text{PG},qc,k}$ are $q+1$ and $k(q+1)$, respectively. The null space of $\mathbf{H}^{(2)}_{\text{PG},qc,k}$ gives a QC-PG-LDPC code of length kn with minimum distance at least $q+2$, whose Tanner

graph is at least 6. For a given projective geometry PG(m,q) with $1 \leq k \leq K_{c,\text{even}}$ (or $K_{c,\text{odd}}$), we can construct a sequence of QC-PG-LDPC codes with different lengths and rates. The above construction gives a class of QC-PG-LDPC codes.

Example 10.12. Consider the three-dimensional projective geometry PG(3,2^3) over GF(2^3). This geometry consists of 585 points and 4745 lines. Each line consists of nine points. The incidence vector of a line in PG(3,2^3) is a 585-tuple over GF(2) with weight 9. There are 65 lines in PG(3,2^3) whose incidence vectors are nonprimitive. The incidence vectors of the other 4680 lines in PG(3,2^3) are primitive. These primitive incidence vectors can be partitioned into eight cyclic classes, $Q_0, Q_1, \ldots, Q_7$, each consisting of 585 incidence vectors. For each cyclic class Q_j with $0 \leq j \leq 7$, we form a 585×585 circulant $\mathbf{G}_{c,j}$ with the 585 incidence vectors of Q_i as columns arranged in downward cyclic order. The column weight and row weight of $\mathbf{G}_{c,j}$ are both 9. Suppose we set $k = 6$ and form the following 585×3510 matrix over GF(2): $\mathbf{H}^{(2)}_{\text{PG},qc,6} = [\,\mathbf{H}_{c,0} \quad \mathbf{H}_{c,1} \quad \cdots \quad \mathbf{H}_{c,5}\,]$, which has column and row weights 9 and 54, respectively. The null space of this matrix gives a (9,54)-regular (3510,3109) QC-PG-LDPC code with rate 0.8858 and minimum distance at least 10. The error performance of this code with iterative decoding using the SPA with 100 iterations is shown in Figure 10.17. At a BER of 10^{-6}, it performs 1.3 dB from the Shannon limit.

If we decompose each $n \times n$ circulant $\mathbf{G}_{c,j}$ in $\mathbf{H}^{(2)}_{\text{PG},c,k}$ into a $c \times t$ array of $n \times n$ circulants as described in Section 10.5, we obtain a $c \times kt$ array $\mathbf{H}_{\text{array,decom}}$ of $n \times n$ circulants. The null space of $\mathbf{H}_{\text{array,decom}}$ gives a QC-PG-LDPC code of length nkt.

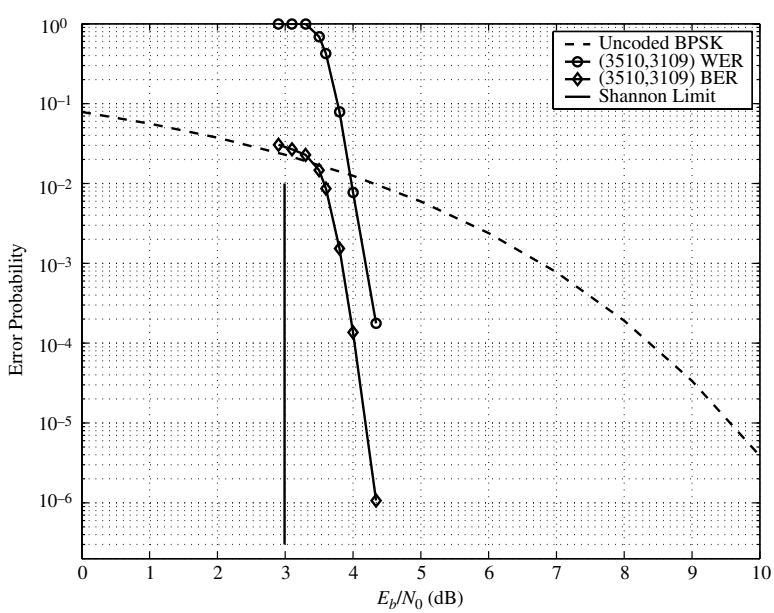


Figure 10.17 The error performance of the (3510,3109) QC-PG-LDPC code given in Example 10.12.

10.7 One-Step Majority-Logic and Bit-Flipping Decoding Algorithms for FG-LDPC Codes

As shown in earlier sections of this chapter, FG-LDPC codes perform very well over the binary-input AWGN channel with iterative decoding using the SPA. In this and the next two sections, we show that FG-LDPC codes can also be decoded with other decoding algorithms that require much less decoding (or computational) complexity to provide a wide range of effective trade-offs among error performance, decoding complexity, and decoding speed. These decoding algorithms include the *one-step majority-logic* (OSMLG) decoding algorithm, the *bit-flipping* (BF) decoding algorithm, and various *weighted BF (WBF) decoding algorithms*. The OSMLG and BF decoding algorithms are *hard-decision* decoding algorithms that can be easily implemented. The WBF decoding algorithms are *reliability-based* decoding algorithms, which require larger computational complexity than the hard-decision OSMLG and BF decoding algorithms, but provide better performance.

In this section, we first present the hard-decision OSMLG and BF decoding algorithms for LDPC codes. Before we introduce these decoding algorithms, we give a brief review of some basic concepts of linear block codes presented in Chapter 3 and introduce some new concepts that are essential for developing the OSMLG and BF decoding algorithms.

Consider an LDPC code C given by the null space of an RC-constrained $m \times n$ sparse matrix $\mathbf{H}$ over GF(2) with constant column and row weights g and r, respectively. Let $\mathbf{h}_0, \mathbf{h}_1, \ldots, \mathbf{h}_{m-1}$ denote the rows of $\mathbf{H}$, where the ith row $\mathbf{h}_i$ is expressed as the following n-tuple over GF(2):

$$\mathbf{h}_i = (h_{i,0}, h_{i,1}, \ldots, h_{i,n-1}),$$

for $0 \leq i < m$. As shown in Section 3.2, an n-tuple $\mathbf{v}$ over GF(2) is a codeword in C *if and only if* $\mathbf{v}\mathbf{H}^{\mathrm{T}} = \mathbf{0}$ (a zero m-tuple), i.e., the inner product $\mathbf{v} \cdot \mathbf{h}_i = 0$ for $0 \leq i < m$.

Let $\mathbf{v} = (v_0, v_1, \ldots, v_{n-1})$ be a codeword in C. Then, the condition $\mathbf{v}\mathbf{H}^{\mathrm{T}} = \mathbf{0}$ gives the following m constraints on the code bits of $\mathbf{v}$:

$$\mathbf{v} \cdot \mathbf{h}_i = v_0 h_{i,0} + v_1 h_{i,1} + \cdots + v_{n-1} h_{i,n-1} = 0, \tag{10.33}$$

for $0 \leq i < m$. The above m linear sums of code bits are called *parity-check-sums*. The constraint given by (10.33) is called the *zero-parity-check-sum constraint*. For $0 \leq j < n$, if $h_{i,j} = 1$, then the jth code bit v_j participates (or is contained) in the ith parity-check-sum given by (10.33). In this case, we say that the ith row $\mathbf{h}_i$ of $\mathbf{H}$ *checks* on the jth code bit v_j of $\mathbf{v}$ (or the jth code bit v_j of $\mathbf{v}$ is *checked* by the ith row $\mathbf{h}_i$ of $\mathbf{H}$). Since $\mathbf{H}$ has constant column weight g, there are g rows of $\mathbf{H}$ that check on v_j for $0 \leq j < n$. Hence, there are g zero-parity-check-sums that contain (or check on) the code bit v_j.

Suppose a codeword $\mathbf{v} = (v_0, v_1, \ldots, v_{n-1})$ in C is transmitted over the BSC (a binary-input AWGN channel with hard-decision output). Let $\mathbf{z} = (z_0, z_1, \ldots, z_{n-1})$ be the *hard-decision received* vector (an n-tuple over GF(2)). To decode $\mathbf{z}$ with any decoding method for a linear block code, the first step is to

compute its m-tuple *syndrome*,

$$\mathbf{s} = (s_0, s_1, \ldots, s_{m-1}) = \mathbf{z}\mathbf{H}^{\mathrm{T}}, \tag{10.34}$$

where

$$s_i = \mathbf{z} \cdot \mathbf{h}_i = z_0 h_{i,0} + z_1 h_{i,1} + \cdots + z_{n-1} h_{i,n-1}, \tag{10.35}$$

for $0 \leq i < m$, which is called a *syndrome sum* of $\mathbf{z}$. If $\mathbf{s} = \mathbf{0}$, then the received bits in $\mathbf{z}$ satisfy all the m zero-parity-check-sum constraints given by (10.33) and hence $\mathbf{z}$ is a codeword. In this case, the receiver assumes that $\mathbf{z}$ is the transmitted codeword and accepts it as the decoded codeword. If $\mathbf{s} \neq \mathbf{0}$, then the received bits in $\mathbf{z}$ do not satisfy all the m zero-parity-check-sum constraints given by (10.33) and $\mathbf{z}$ is not a codeword. In this case, we say that errors in $\mathbf{z}$ are *detected*. Then an error-correction process is initiated.

A syndrome sum s_i of the received sequence $\mathbf{z}$ that is not equal to zero is referred to as a *parity-check failure*, i.e., the received bits that participate in the syndrome sum s_i of $\mathbf{z}$ fail to satisfy the zero-parity-check-sum constraint given by (10.33). It is clear that the number of parity-check failures of $\mathbf{z}$ is equal to the number of 1-components in the syndrome $\mathbf{s}$. A syndrome sum s_i that contains a received bit z_j is said to *check on* z_j. Consider the syndrome sums of $\mathbf{z}$ that check on the received bit z_j but fail to satisfy the zero-parity-check-sum constraints given by (10.33). We call these syndrome sums the *parity-check failures* on z_j. The number of parity-check failures on a received bit z_j gives a *measure of the reliability* of the received bit z_j. The larger the number of parity-check failures on z_j, the less reliable z_j is. Conversely, the smaller the number of parity-check failures on z_j, the more reliable z_j is.

In the following, we will use the above-defined reliability measures of the bits of a hard-decision received sequence $\mathbf{z}$ to devise the OSMLG and BF decoding algorithms.

10.7.1 The OSMLG Decoding Algorithm for LDPC Codes over the BSC

Consider an LDPC code given by the null space of an RC-constrained $m \times n$ parity-check matrix $\mathbf{H}$ with constant column and row weights g and r, respectively. Express the m rows of $\mathbf{H}$ as follows:

$$\mathbf{h}_i = (h_{i,0}, h_{i,1}, \ldots, h_{i,n-1}),$$

for $1 \leq i < m$. For $0 \leq i < m$ and $0 \leq j < n$, we define the following two index sets:

$$N_i = \{j \colon 0 \leq j < n, h_{i,j} = 1\}, \tag{10.36}$$

$$M_j = \{i \colon 0 \leq i < m, h_{i,j} = 1\}. \tag{10.37}$$

The indices in N_i are simply the locations of the 1-components in the ith row $\mathbf{h}_i$ of $\mathbf{H}$. N_i is called the *support* of $\mathbf{h}_i$. The indices in M_j give the rows of $\mathbf{H}$ whose jth components are equal to 1. Since $\mathbf{H}$ satisfies the RC-constraint, it is clear that (1)

for $0 < i, i' < m$, and, $i \neq i'$, N_i and $N_{i'}$ have *at most one* index in common; and (2) for $0 \leq j, j' < n$, and $j \neq j'$, M_j and $M_{j'}$ have at most one index in common. Since $\mathbf{H}$ has constant column weight g and constant row weight r, $|M_j| = g$ for $0 \leq j < n$ and $|N_i| = r$ for $0 \leq i < m$.

For $0 \leq j < n$, define the following set of rows of $\mathbf{H}$:

$$\mathcal{A}_j = \{\mathbf{h}_i \colon 0 \leq i < m, i \in M_j\}. \tag{10.38}$$

Then it follows from the RC-constraint on the rows of $\mathbf{H}$ that $\mathcal{A}_j$ has the following structural properties: (1) every row in $\mathcal{A}_j$ checks on the code bit v_j; and (2) any code bit other than v_j is checked by at most one row in $\mathcal{A}_j$. The rows in $\mathcal{A}_j$ are said to be *orthogonal* on the code bit v_i (or the ith bit position of $\mathbf{v}$).

It follows from the definition of N_i that the ith syndrome sum s_i of the syndrome $\mathbf{s}$ of a hard-decision received sequence $\mathbf{z}$ given by (10.35) can be expressed in terms of the indices in N_i as follows:

$$s_i = \sum_{j \in N_i} z_j h_{i,j}, \tag{10.39}$$

for $0 \leq i < m$. The expression (10.39) simply says that only the received bits of $\mathbf{z}$ with indices in N_i participate in the ith syndrome sum s_i (or are checked by s_i). For $0 \leq j < n$, define the following subset of syndrome sums of $\mathbf{z}$:

$$S_j = \{s_i = \mathbf{z} \cdot \mathbf{h}_i \colon \mathbf{h}_i \in \mathcal{A}_j, i \in M_j\}. \tag{10.40}$$

Then the jth received bit z_j of $\mathbf{z}$ is contained in every syndrome sum in S_j and any received bit other than z_j is contained in (or is checked by) at most one syndrome sum in S_j. The g syndrome sums in S_j are said to be *orthogonal* on the jth received bit z_j or are called the *orthogonal syndrome sums* on z_j.

Let $\mathbf{e} = (e_0, e_1, \ldots, e_{n-1})$ be the error pattern caused by the channel noise during the transmission of the codeword $\mathbf{v}$ over the BSC. Then $\mathbf{z} = \mathbf{v} + \mathbf{e}$. Since $\mathbf{v} \cdot \mathbf{h}_i = \mathbf{0}$ for $1 \leq i < m$,

$$s_i = \mathbf{e} \cdot \mathbf{h}_i. \tag{10.41}$$

For $0 \leq j < n$, it follows from (10.41) and the orthogonality property of S_j that, for $i \in M_j$, we have the following g relationships between the g syndrome sums in S_j and the error digits, $e_0, e_1, \ldots, e_{n-1}$ in $\mathbf{z}$:

$$s_i = e_j + \sum_{l \in M_j, l \neq j} e_l h_{i,l}. \tag{10.42}$$

We see that all the above g syndrome sums contain (or check on) the error digit e_j. Furthermore, due to the orthogonal property of S_j (or $\mathcal{A}_j$), any error digit other than e_j is checked by (or appears in) at most one of the g syndrome sums given by (10.42). These syndrome sums are said to be *orthogonal on* the error digit e_j and are called *orthogonal syndrome sums* on e_j.

Next we show that the g orthogonal syndrome sums given by (10.42) can be used to decode the error digits of an error pattern $\mathbf{e}$. Suppose that there are $\lfloor g/2 \rfloor$ or fewer errors in the error pattern $\mathbf{e}$. The error digit e_j at the jth position of

$\mathbf{e}$ has two possible values, 1 or 0. If $e_j = 1$, the other nonzero error digits in $\mathbf{e}$ can distribute among at most $\lfloor g/2 \rfloor - 1$ syndrome sums orthogonal on e_j. Hence, at least $g - \lfloor g/2 \rfloor + 1$ (*more than half*) of the orthogonal syndrome sums on e_j are equal to $e_j = 1$. If $e_j = 0$, the nonzero error digits in $\mathbf{e}$ can distribute among at most $\lfloor g/2 \rfloor$ syndrome sums orthogonal on e_j. Hence, at least $g - \lfloor g/2 \rfloor$ (*half or more than half*) of the orthogonal syndrome sums on e_j are equal to $e_j = 0$. Let $w(\mathbf{e})$ be the weight of the error pattern $\mathbf{e}$ (or the number of errors in $\mathbf{e}$). For $w(\mathbf{e}) \leq \lfloor g/2 \rfloor$, the above analysis simply says that (1) the value of the jth error digit e_j of $\mathbf{e}$ is equal to the value assumed by a *majority* of the orthogonal syndrome sums on e_j; and (2) if no value, 0 or 1, is assumed by a clear majority of the orthogonal syndrome sums on e_j (i.e., *half of them are equal to 0 and half of them are equal to 1*), then the value of the jth error digit e_j is equal to 0. Given the above facts, a simple algorithm for decoding the error digits in an error pattern $\mathbf{e}$ can be devised as follows.

For $1 \leq j < n - 1$, the error digit e_j at the jth location of $\mathbf{e}$ is decoded as "1" if a majority of the orthogonal syndrome sums on e_j assumes the value "1"; otherwise, e_j is decoded as "0."

The above method for decoding the error pattern $\mathbf{e}$ contained in a hard-decision received sequence $\mathbf{z}$ is referred to as *OSMLG decoding*. Correct decoding of an error pattern $\mathbf{e}$ is *guaranteed* if the number of errors in $\mathbf{e}$ is $\lfloor g/2 \rfloor$ or less. However, correct decoding is *not guaranteed* if an error pattern $\mathbf{e}$ contains more than $\lfloor g/2 \rfloor$ errors (see Problem 10.13). Therefore, an LDPC code C given by the null space of an RC-constrained parity-check matrix $\mathbf{H}$ with constant column weight g is capable of correcting *any combination* of $\lfloor g/2 \rfloor$ or fewer errors with OSMLG decoding. The *parameter* $\lfloor g/2 \rfloor$ is called the *OSMLG error-correction capability* of the code C. Even though OSMLG decoding guarantees to correct all the error patterns with just $\lfloor g/2 \rfloor$ or fewer errors, it actually can correct a *very large fraction* of error patterns with more than $\lfloor g/2 \rfloor$ errors. Codes that can be decoded with OSMLG decoding are said to be *OSMLG-decodable* and are called *OSMLG-decodable codes*.

The above OSMLG decoding is based on the number of parity-check failures in the set of syndrome sums orthogonal on each received bit. The larger the number of the parity-check failures in the set of syndrome sums orthogonal on a received bit, the less reliable the received bit is. Conversely, the smaller the number of parity-check failures in the set of syndrome sums orthogonal on a received bit, the more reliable the received bit is. Therefore, the number of parity-check failures orthogonal on a received bit can be used as a *measure of the reliability* of the received bit.

For an LDPC code C given by the null space of an RC-constrained parity-check matrix $\mathbf{H}$ with constant column weight g, a set of g syndrome sums orthogonal on each received bit can be formed. Consider the jth received bit z_j in the received sequence $\mathbf{z}$. As shown earlier, the set S_j of g syndrome sums orthogonal on z_j is given by (10.40). For $0 \leq j < n$, define the following *integer sum*:

$$\sum_{i \in M_j} s_i. \tag{10.43}$$

The range of this sum consists of $g+1$ non-negative integers from 0 to g, denoted by $[0, g]$. For OSMLG decoding, z_j is decoded as "1" if

$$\sum_{i \in M_j} s_i > \lfloor g/2 \rfloor, \tag{10.44}$$

otherwise, z_j is decoded as "0." The above inequality can be put into the following form:

$$\sum_{i \in M_j} (2s_i - 1) > 0. \tag{10.45}$$

Let

$$\Lambda_j = \sum_{i \in M_j} (2s_i - 1). \tag{10.46}$$

The range of Λ_j consists of $2g+1$ integers from $-g$ to g, denoted by $[-g, g]$. From (10.46), we see that the greater the number of syndrome sums orthogonal on the received bit z_j that satisfy the zero-parity-check-sum constraint given by (10.33), the more negative Λ_j is and hence the more reliable the received bit z_j is. Conversely, the greater the number of syndrome sums orthogonal on z_j that fail the zero-parity-check-sum constraint given by (10.33), the more positive Λ_j is and hence the less reliable z_j is. Therefore, Λ_j also gives a measure of the reliability of the received bit z_j.

With the above measure of reliability of a received bit, the OSMLG decoding algorithm can be formulated as follows.

Algorithm 10.1 The OSMLG Decoding Algorithm

1. Compute the syndrome $\mathbf{s} = (s_0, s_1, \ldots, s_{m-1})$ of the received sequence $\mathbf{z} = (z_0, z_1, \ldots, z_{n-1})$. If $\mathbf{s} = \mathbf{0}$, stop decoding; otherwise, go to Step 2.
2. For each received bit z_j of $\mathbf{z}$ with $0 \le j < n$, compute its reliability measure Λ_j. Go to Step 3.
3. For $0 \le j < n$, if $\Lambda_j > 0$, decode the error digit e_j as "1"; otherwise, decode e_j as "0." Form the estimated error pattern $\mathbf{e}^* = (e_0, e_1, \ldots, e_{n-1})$. Go to Step 4.
4. Decode $\mathbf{z}$ into $\mathbf{v}^* = \mathbf{z} + \mathbf{e}^*$. Compute $\mathbf{s}^* = \mathbf{v}^* \mathbf{H}^{\mathrm{T}}$. If $\mathbf{s}^* = \mathbf{0}$, the decoding is successful; otherwise, declare a decoding failure.

Since OSMLG decoding involves only logical operations, an OSMLG decoder can be implemented with a combinational logic circuit. For high-speed decoding, all the error digits can be decoded in parallel. For a cyclic OSMLG-decodable code, the OSMLG decoding can be carried out in *serial manner*, decoding one error digit at a time using the same decoding circuit. Serial decoding further reduces decoding complexity. The OSMLG decoding is effective only when the column weight g of the RC-constrained parity-check matrix $\mathbf{H}$ of an LDPC code is relatively large. For more on majority-logic decoding, readers are referred to [23].

The classes of FG-LDPC codes presented in Sections 10.1–10.3 and 10.6 are *particularly effective* with OSMLG decoding. For a code in each of these classes,

a large number of syndrome sums orthogonal on every error digit can be formed for decoding. Consider the classes of cyclic EG-LDPC codes constructed on the basis of Euclidean geometries over finite fields given in Section 10.1.1. Let $C^{(1)}_{\mathrm{EG},c,k}$ be the cyclic EG-LDPC code of length $n = q^m - 1$ constructed on the basis of the m-dimensional Euclidean geometry EG(m,q) over GF(q), where $1 \le k \le K_c$, with $K_c = (q^{m-1} - 1)/(q - 1)$. The parity-check matrix $\mathbf{H}^{(1)}_{\mathrm{EG},c,k}$ of this cyclic EG-LDPC code is given by (10.7) and consists of a column of k circulants of size $(q^m - 1) \times (q^m - 1)$, each having both column weight and row weight q. The parity-check matrix $\mathbf{H}^{(1)}_{\mathrm{EG},c,k}$ has constant column and row weights kq and q, respectively. Since $\mathbf{H}^{(1)}_{\mathrm{EG},c,k}$ satisfies the RC constraint, kq syndrome sums orthogonal on each received bit can be formed. Hence, this code $C^{(1)}_{\mathrm{EG},c,k}$ given by the null space of $\mathbf{H}^{(1)}_{\mathrm{EG},c,k}$ is capable of correcting any combination of $\lfloor kq/2 \rfloor$ or fewer errors. Therefore, for $k = 1, 2, \ldots, K_c$, a sequence of OSMLG-decodable cyclic EG-LDPC codes can be constructed on the basis of the m-dimensional Euclidean geometry EG(m, q) over GF(q).

Example 10.13. Consider the (4095,3367) cyclic EG-LDPC code $C^{(1)}_{\mathrm{EG},c,1}$ given in Example 10.1. This code is constructed using the two-dimensional Euclidean geometry EG(2,2^6) over GF(2^6). The parity-check matrix $\mathbf{H}^{(1)}_{\mathrm{EG},c,1}$ of this code consists of a single 4095×4095 circulant with both column weight and row weight 64, which is formed by the type-1 incidence vectors of the lines of EG(2,2^6) not passing through the origin. Since the parity-check matrix of this code satisfies the RC-constraint, 64 syndrome sums orthogonal on any error-digit position can be formed. Hence, this cyclic EG-LDPC code is capable of correcting any combination of 32 or fewer errors with the OSMLG decoding. The minimum distance of this code is exactly 65 (1 greater than the column weight of the parity-check matrix of the code). The error performance of this code with OSMLG decoding over the hard-decision AWGN channel is shown in Figure 10.18. At a BER of 10^{-6}, it achieves a coding gain of 4.7 dB over the uncoded BPSK system. Figure 10.18 also includes the error performance of the (4095,3367) cyclic EG-LDPC code over the binary-input AWGN channel decoded using the SPA with 100 iterations. We see that the OSMLG decoding of this code loses about 2 dB in SNR compared with the SPA decoding of the code at the BER of 10^{-6} and below. However, the OSMLG decoding requires *much less* computational complexity than the SPA decoding. Also included in Figure 10.18 is the error performance of a (4095,3367) BCH code with *designed minimum distance* 123, which is about twice that of the (4095,3367) cyclic EG-LDPC code. This BCH code is capable of correcting 61 (*designed error-correction capability*) or fewer errors with the (*hard-decision*) Berlekamp–Massey (BM) algorithm [23, 26, 33]. In terms of error-correction capability, the BCH code is twice as powerful as the cyclic EG-LDPC code. However, from Figure 10.18, we see that at a BER of 10^{-6}, the (4095,3367) BCH code decoded with the BM algorithm has a coding gain of less than 0.5 dB over the (4095,3367) cyclic EG-LDPC code decoded with OSMLG decoding. Decoding the (4095,

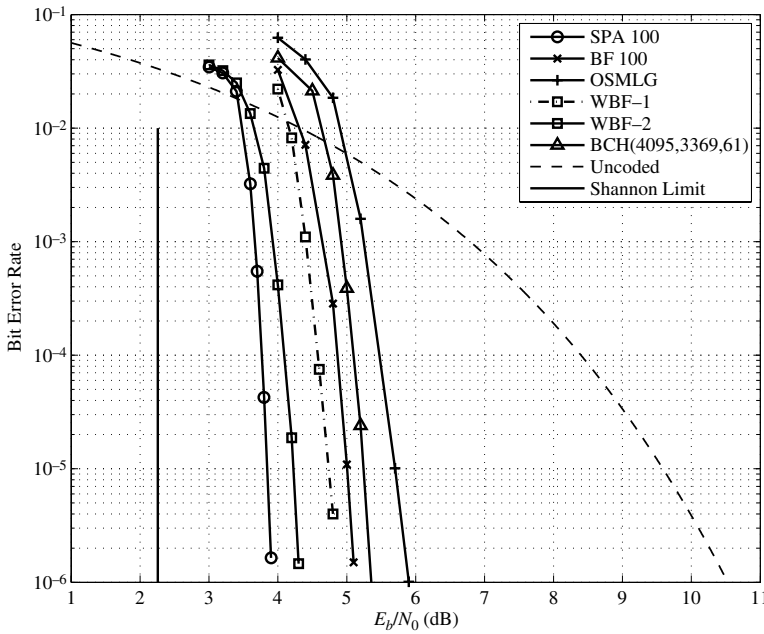


Figure 10.18 Error performances of the (4095,3367) cyclic EG-LDPC code given in Example 10.13 with various decoding methods.

3367) BCH code with BM algorithm requires computations over the Galois field GF(2^{12}). However, OSMLG decoding of the (4095,3367) cyclic EG-LDPC code requires only simple logical operations. Furthermore, the BM decoding algorithm is an iterative decoding algorithm. To correct 61 or fewer errors, it requires 61 iterations [23, 26]. The large number of decoding iterations causes a long decoding delay. Therefore, the BM decoding algorithm is much more complex than the OSMLG decoding algorithm and requires a longer decoding time. One of the reasons why the (4095,3367) BCH code does not provide an impressive coding gain over the (4095,3367) cyclic EG-LDPC code even though it has a much larger error-correction capability than the (4095,3367) cyclic EG-LDPC code is that the BM algorithm can correct only error patterns with numbers of errors up to its designed error-correction capability of 61. Any error pattern with more than 61 errors will cause a decoding failure. However, OSMLG decoding of the (4095,3367) cyclic EG-LDPC code can correct a very large fraction of error patterns with numbers of errors much larger than its OSMLG error-correction capability of 32.

Analysis of the OSMLG error-correction capabilities of FG-LDPC codes in the other classes given in Sections 10.1–10.3 and 10.6 can be carried out in the same manner as for the cyclic EG-LDPC codes. Consider the cyclic PG-LDPC code $C^{(1)}_{\mathrm{PG},c,k}$ of length $n = (q^{m+1} - 1)/(q - 1)$ given by the null space of the parity-check matrix $\mathbf{H}^{(1)}_{\mathrm{PG},c,k}$ of (10.29) which consists of a column of k circulants of size $n \times n$ (see Section 10.6.1). The k circulants of $\mathbf{H}^{(1)}_{\mathrm{PG},c,k}$ are constructed from k cyclic

classes of lines in the m-dimensional projective geometry PG(m,q) with $2 \leq m$ and $1 \leq k \leq K_{c,\text{even}}$ (or $K_{c,\text{odd}}$), where $K_{c,\text{even}}$ and $K_{c,\text{odd}}$ are given by (10.28) and (10.31), respectively. Each circulant of $H^{(1)}_{\text{PG},c,k}$ has both column weight and row weight $q+1$. The column and row weights of $H^{(1)}_{\text{PG},c,k}$ are $k(q+1)$ and $q+1$, respectively. Since $\mathbf{H}^{(1)}_{\text{PG},c,k}$ satisfies the RC-constraint, $k(q+1)$ syndrome sums orthogonal on every code bit position can be formed for OSMLG decoding. Hence, the OSMLG error-correction capability of the cyclic PG-LDPC code $C^{(1)}_{\text{PG},c,k}$ is $\lfloor k(q+1)/2) \rfloor$.

Example 10.14. For $m = 2$, let the two-dimensional projective geometry PG(2,2^6) over GF(2^6) be the code-construction geometry. This geometry consists of 4161 lines. Since $m = 2$ is even, $K_{c,\text{even}} = 1$. Using (10.29), we can form a parity-check matrix $H^{(1)}_{\text{PG},c,1}$ with a single circulant constructed from the incidence vectors of the 4161 lines of PG(2,2^6). The column weight and row weight of $\mathbf{H}_{\text{PG},c,1}$ are 65. The null space of $\mathbf{H}^{(1)}_{\text{PG},c,1}$ gives a (4161,3431) cyclic PG-LDPC code with rate 0.823 and minimum distance at least 66. For this code, 65 syndrome sums orthogonal on each code bit position can be formed. Hence, the OSMLG error-correction capability of this cyclic PG-LDPC code is 32, the same as the OSMLG error-correction capability of the (4095,3367) cyclic EG-LDPC code given in Example 10.13. The error performances of this code with various decoding algorithms are shown in Figure 10.19.

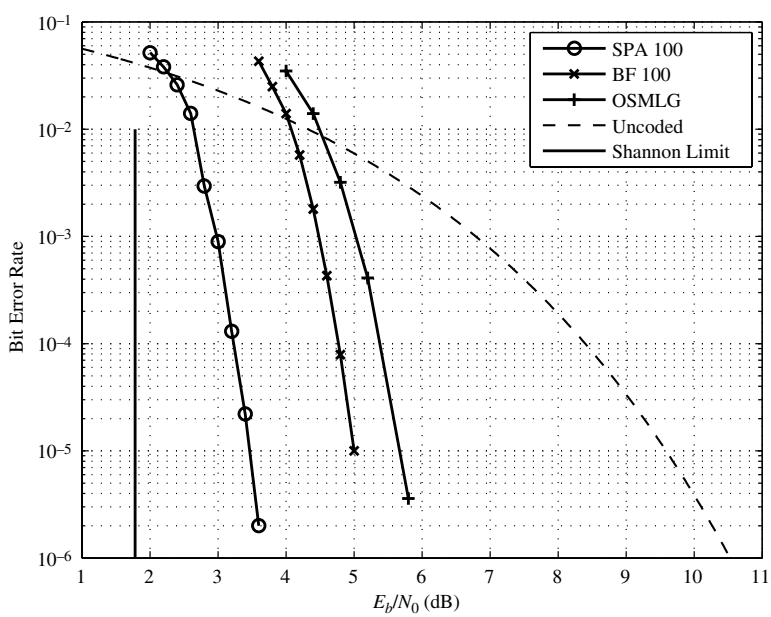


Figure 10.19 Error performances of the (4161,3431) cyclic PG-LDPC code given in Example 10.14 with various decoding methods.

Consider the regular LDPC codes constructed on the basis of decomposition of Euclidean geometries given in Section 10.3. From the m-dimensional Euclidean geometry $\mathrm{EG}(m, q)$ over $\mathrm{GF}(q)$, an RC-constrained $q^{m-1} \times q$ array $\mathbf{H}^{(1)}_{\mathrm{EG},d}$ of $q^{m-1} \times q^{m-1}$ permutation matrices can be formed. Let $1 \leq g \leq q^{m-1}$ and $1 \leq r \leq q$. The null space of any $g \times r$ subarray $\mathbf{H}^{(1)}_{\mathrm{EG},d}(g, r)$ of $\mathbf{H}^{(1)}_{\mathrm{EG},d}$ gives a (g,r)-regular LDPC code $C^{(1)}_{\mathrm{EG},d}$ of length $n = rq^{m-1}$ that is OSMLG decodable. For this code, g syndrome sums orthogonal on any code bit position can be formed for OSMLG decoding. Hence, the OSMLG error-correction capability is $\lfloor g/2 \rfloor$. Furthermore, any subarray of the $q \times q^{m-1}$ array $\mathbf{H}^{(2)}_{\mathrm{EG},d} = [\mathbf{H}^{(1)}_{\mathrm{EG},d}]^{\mathrm{T}}$ also gives an OSMLG-decodable EG-LDPC code. Decomposition of Euclidean geometries gives a large class of OSMLG-decodable EG-LDPC codes.

10.7.2 The BF Algorithm for Decoding LDPC Codes over the BSC

The BF algorithm for decoding LDPC codes over the BSC was discussed in Section 5.7.4. In this section, we show that bit-flipping decoding is very effective for decoding FG-LDPC codes in terms of the trade-off between error performance and decoding complexity. Before we do so, we give a brief review of BF decoding that is based on the concept of a reliability measure of received bits developed in the previous section.

Again we consider an LDPC code C given by the null space of an RC-constrained $m \times n$ parity-check matrix $\mathbf{H}$ with column and row weights g and r, respectively. Let $\mathbf{z} = (z_0, z_1, \ldots, z_{n-1})$ be the hard-decision received sequence. The first step of decoding $\mathbf{z}$ is to compute its syndrome $\mathbf{s} = (s_0, s_1, \ldots, s_{m-1}) = \mathbf{z}\mathbf{H}^{\mathrm{T}}$. For each received bit z_j with $0 \leq j < n$, we determine the number f_j of syndrome sums orthogonal on z_i that fail to satisfy the zero-parity-check-sum constraint given by (10.33). As we described in the previous section, f_j is a measure of how reliable the received bit z_j is. For $0 \leq j < n$, the range of f_j is $[0, g]$. Form the integer n-tuple $\mathbf{f} = (f_0, f_1, \ldots, f_{n-1})$. Then $\mathbf{f} = \mathbf{s}\mathbf{H}$, where the operations in taking the product $\mathbf{s}\mathbf{H}$ are carried out over the integer system (with integer additions). This integer n-tuple $\mathbf{f}$ is referred to as the *reliability profile* of the received sequence $\mathbf{z}$. It is clear that $\mathbf{f}$ is the *all-zero n-tuple* if and only if $\mathbf{s} = \mathbf{0}$.

The next step of decoding $\mathbf{z}$ is to identify the components of $\mathbf{f}$ that are greater than a *preset threshold* δ. The bits of the received sequence $\mathbf{z}$ corresponding to these components are regarded as not reliable. We flip all these unreliable received bits, which results in a new received sequence $\mathbf{z}^{(1)} = (z^{(1)}_0, z^{(1)}_1, \ldots, z^{(1)}_{n-1})$. Then we compute the syndrome $\mathbf{s}^{(1)} = (s^{(1)}_0, s^{(1)}_1, \ldots, s^{(1)}_{n-1}) = \mathbf{z}^{(1)}\mathbf{H}^{\mathrm{T}}$ of the modified received sequence $\mathbf{z}^{(1)}$. If $\mathbf{s}^{(1)} = \mathbf{0}$, we stop decoding and accept $\mathbf{z}^{(1)}$ as the decoded codeword. If $\mathbf{s}^{(1)} \neq \mathbf{0}$, we compute the reliability profile $\mathbf{f}^{(1)} = (f^{(1)}_0, f^{(1)}_1, \ldots, f^{(1)}_{n-1})$ of $\mathbf{z}^{(1)}$. Given $\mathbf{f}^{(1)}$, we repeat the above bit-flipping process to construct another modified received sequence $\mathbf{z}^{(2)}$. Then we test the syndrome $\mathbf{s}^{(2)} = \mathbf{z}^{(2)}\mathbf{H}^{\mathrm{T}}$. If $\mathbf{s}^{(2)} = \mathbf{0}$, we stop decoding; otherwise, we continue the bit-flipping process. The bit-flipping process continues until a zero syndrome is obtained or a preset

maximum number of iterations is reached. If the syndrome is equal to zero, decoding is successful; otherwise a decoding failure is declared.

The above BF decoding process is an iterative decoding algorithm. The threshold δ is a design parameter that should be chosen in such a way as to optimize the error performance while minimizing the computations of parity-check sums. The computations of parity-check sums are binary operations; however, the computations of the reliability profiles and comparisons with the threshold are integer operations. Hence, the computational complexity of the above BF decoding is larger than that of OSMLG decoding. The value of the threshold depends on the code parameters, g and r, and the SNR. The optimum threshold for bit-flipping has been derived by Gallager [32], but we will not discuss it here. Instead, we present a simple bit-flipping mechanism that leads to a very simple BF decoding algorithm as follows.

Algorithm 10.2 A Simple BF Decoding Algorithm

Initialization: Set $k = 0$, $\mathbf{z}^{(0)} = \mathbf{z}$, and the maximum number iterations to $k_{\max}$.

1. Compute the syndrome $\mathbf{s}^{(k)} = \mathbf{z}^{(k)}\mathbf{H}^{\mathrm{T}}$ of $\mathbf{z}^{(k)}$. If $\mathbf{s}^{(k)} = \mathbf{0}$, stop decoding and output $\mathbf{z}^{(k)}$ as the decoded codeword; otherwise go to Step 2.
2. Compute the reliability profile $\mathbf{f}^{(k)}$ of $\mathbf{z}^{(k)}$.
3. Identify the set F_k of bits in $\mathbf{z}^{(k)}$ that have the largest parity-check failures.
4. Flip the bits in F_k to obtain an n-tuple $\mathbf{z}^{(k+1)}$ over GF(2).
5. $k \leftarrow k + 1$. If $k > k_{\max}$, declare a decoding failure and stop the decoding process; otherwise, go to Step 1.

Since at each iteration, we flip the bits with the largest parity-check failures (LPCFs), we call the above simple BF decoding the LPCF-BF decoding algorithm.

The LPCF-BF decoding algorithm is very effective for decoding FG-LDPC codes since it allows the reliability measure of each received bit to vary over a large range due to the large column weights of the parity-check matrices of these codes. To demonstrate this, we again use the (4095,3367) cyclic EG-LDPC code given in Example 10.1 (or Example 10.13). The parity-check matrix of this code has column weight 64. Hence, the reliability measure f_j of each received bit z_j can vary between 0 and 64. The error performance of this code over the BSC decoded using the LPCF-BF decoding presented above with 100 iterations is shown in Figure 10.18. We see that, at a BER of 10^{-6}, LPCF-BF decoding outperforms OSMLG decoding by about 0.7 dB. Of course, this coding gain is achieved at the expense of a larger computational complexity. We also see that the (4095,3367) cyclic EG-LDPC code decoded with LPCF-BF decoding outperforms the (4095,3369) BCH code decoded by BM decoding, with less computational complexity.

10.8 Weighted BF Decoding: Algorithm 1

The performance of the simple hard-decision LPCF-BF decoding algorithm presented in Section 10.7.2 can be improved by *weighting* its bit-flipping decision

function with some type of *soft* reliability information about the received symbols. In this and following sections, we present three weighted BF decoding algorithms with increasing performance but also increasing decoding complexity. We will show that these weighted BF decoding algorithms are very effective for decoding FG-LDPC codes.

Let C be an LDPC code of length n given by the null space of an $m \times n$ RC-constrained parity-check matrix $\mathbf{H}$ with constant column and row weights g and r, respectively. Suppose a codeword $\mathbf{v} = (v_0, v_1, \ldots, v_{n-1})$ in C is transmitted over the binary-input AWGN channel with two-sided power-spectral density $N_0/2$. Assuming transmission using BPSK signaling with unit energy per signal, the transmitted codeword $\mathbf{v}$ is mapped into a sequence of BPSK signals that is represented by a bipolar code sequence,

$$(2v_0 - 1, 2v_1 - 1, \ldots, 2v_{n-1} - 1),$$

where the jth component $2v_j - 1 = +1$ for $v_j = 1$ and $2v_j - 1 = -1$ for $v_j = 0$. Let $\mathbf{y} = (y_0, y_1, \ldots, y_{n-1})$ be the sequence of *samples* (or *symbols*) at the output of the channel-receiver sampler. This sequence is commonly called a *soft-decision* received sequence. The samples of $\mathbf{y}$ are real numbers with $y_j = (2v_j - 1) + x_j$ for $0 \leq j < n$, where x_j is a Gaussian random variable with zero mean and variance $N_0/2$. For $0 \leq j < n$, suppose each sample y_j of $\mathbf{y}$ is decoded *independently* in accord with the following hard-decision rule:

$$z_j = \begin{cases} 0, & \text{for } y_j \leq 0, \\ 1, & \text{for } y_j > 0. \end{cases} \tag{10.47}$$

Then we obtain a binary sequence $\mathbf{z} = (z_0, z_1, \ldots, z_{n-1})$, the hard-decision received sequence. The jth bit z_j of $\mathbf{z}$ is simply an estimate of the jth code bit v_j of the transmitted codeword $\mathbf{v}$. If $z_j = v_j$ for $0 \leq j < n$, then $\mathbf{z} = \mathbf{v}$; otherwise, $\mathbf{z}$ contains transmission errors. Therefore, $\mathbf{z}$ is an estimate of the transmitted codeword $\mathbf{v}$ *prior* to decoding. The above hard-decision rule is optimal in the sense of minimizing the estimation error probability of a code bit as described in Chapter 1.

With the hard-decision rule given by (10.47), the magnitude $|y_j|$ of the jth sample y_j of $\mathbf{y}$ can be used as a reliability measure of the hard-decision decoded bit z_j, since the magnitude of the log-likelihood ratio,

$$\log\left[\frac{\Pr(y_j|v_j = 1)}{\Pr(y_j|v_j = 0)}\right],$$

associated with the hard-decision given by (10.47) is proportional to $|y_j|$. The larger the magnitude $|y_j|$ of y_j, the more reliable the hard-decision estimation z_j of v_j is.

For $0 \leq i < m$, we define

$$\phi_i = \min_{j \in N_i} |y_j|, \tag{10.48}$$

which is the *minimum magnitude* of the samples of $\mathbf{y}$ whose hard-decision bits participate in the ith syndrome sum s_i given by (10.39). The value of ϕ_i gives an indication of how much *confidence* we have on the ith syndrome sum s_i satisfying the zero-parity-check-sum constraint given by (10.33). The larger ϕ_i, the more reliable the hard-decision bits that participate in the ith syndrome sum s_i and the higher the probability that s_i satisfies the zero-parity-check-sum constraint given by (10.33). Therefore, ϕ_i may be taken as a reliability measure of the syndrome sum s_i.

Suppose we weight each term $2s_i - 1$ in the summation of (10.46) by ϕ_i for $i \in M_j$. We obtain the following weighted sum:

$$E_j = \sum_{i \in M_j} (2s_i - 1)\phi_i, \tag{10.49}$$

which is simply the sum of weighted syndrome sums orthogonal on the jth hard-decision bit z_j of $\mathbf{z}$. E_j is a real number with a range from $-|M_j|\phi_i$ to $+|M_j|\phi_i$. Since $|M_j| = g$, the range of E_j is the real-number interval $[-g\phi_i, +g\phi_i]$. For hard-decision OSMLG decoding, $\Lambda_j = \sum_{i \in M_j} (2s_i - 1)$ gives a measure of the reliability of z_j that is used as the decoding function to determine whether the jth received bit z_j is error-free or erroneous. Then E_j gives a *weighted reliability measure* of the hard-decision received bit z_j. From (10.49), we see that the more negative the reliability measure E_j, the more syndrome sums orthogonal on z_j satisfy the zero-parity-check-sum constraint given by (10.33) and hence the more reliable z_j is. In this case, z_j is less likely to be different from the transmitted code bit v_j and hence is most likely to be error-free. Conversely, the more positive the reliability measure E_j, the more syndrome sums orthogonal on z_j fail to satisfy the zero-parity-check-sum constraint given by (10.33) and hence the less reliable z_j is. In this case, z_j is most likely erroneous and should be flipped. Form over the real-number system the n-tuple

$$\mathbf{E} = (E_0, E_1, \ldots, E_{n-1}), \tag{10.50}$$

whose components give the reliability measures of the bits in the hard-decision received sequence $\mathbf{z} = (z_0, z_1, \ldots, z_{n-1})$. This real n-tuple $\mathbf{E}$ is called the *weighted reliability profile* of the hard-decision received sequence $\mathbf{z}$. The received bit in $\mathbf{z}$, say z_j, that has the *largest* E_j in the weighted reliability profile $\mathbf{E}$ of $\mathbf{z}$ is the *most unreliable* bit in $\mathbf{z}$ and should be flipped in a BF decoding algorithm.

With the concepts developed above and the weighted reliability measure of a hard-decision received bit defined by (10.49), an iterative weighted BF decoding algorithm can be formulated. For $0 \le k \le k_{\max}$, let $\mathbf{z}^{(k)} = (z_0^{(k)}, z_1^{(k)}, \ldots, z_{n-1}^{(k)})$ be the modified received sequence available at the beginning of the kth iteration of the BF-decoding process and let $\mathbf{E}^{(k)} = (E_0^{(k)}, E_1^{(k)}, \ldots, E_{n-1}^{(k)})$ be the weighted reliability profile of $\mathbf{z}^{(k)}$ computed via (10.49) and (10.50).

Algorithm 10.3 Weighted BF Decoding Algorithm 1

Initialization: Set $k = 0$, $\mathbf{z}^{(0)} = \mathbf{z}$, and the maximum number of iterations to $k_{\max}$. Compute and store ϕ_i for $0 \leq i < m$.

1. Compute the syndrome $\mathbf{s}^{(k)} = \mathbf{z}^{(k)}\mathbf{H}^{\mathrm{T}}$ of $\mathbf{z}^{(k)}$. If $\mathbf{s}^{(k)} = \mathbf{0}$, stop decoding and output $\mathbf{z}^{(k)}$ as the decoded codeword; otherwise go to Step 2.
2. Compute the reliability profile $\mathbf{E}^{(k)}$ of $\mathbf{z}^{(k)}$ on the basis of (10.49). Go to Step 3.
3. Identify the bit position j for which E_j is largest. Go to Step 4. (Remark: If j is not unique, then choose a j at random.)
4. Flip the jth received bit $z_j^{(k)}$ of $\mathbf{z}^{(k)}$ to obtain a modified received sequence $\mathbf{z}^{(k+1)}$. Go to Step 5.
5. $k \leftarrow k + 1$. If $k > k_{\max}$, declare a decoding failure and stop the decoding process; otherwise, go to Step 1.

The above weighted BF decoding algorithm 1 was first proposed in [4]. The five decoding steps of the algorithm are exactly the same as those of the hard-decision LPCF-BF decoding algorithm. However, the weighted BF decoding algorithm is more complex in computation than the LPCF-BF decoding algorithm, since it requires real-number additions and comparisons in order to carry out Steps 2 and 3 rather than the integer comparisons and additions used in the same two steps in the LPCF-BF decoding algorithm. Furthermore, additional memory is needed, to store ϕ_i for $0 \leq i < m$.

To show the performance improvement of weighted BF decoding algorithm 1 over the hard-decision LPCF-BF decoding algorithm, we again consider the (4095,3367) cyclic EG-LDPC code $C^{(1)}_{\mathrm{EG},c,1}$ constructed from the two-dimensional Euclidean geometry EG(2,2^6) over GF(2^6) given in Example 10.13. The error performance of this code over the binary-input AWGN channel decoded using weighted BF decoding algorithm 1 with a maximum of 100 iterations is shown in Figure 10.18. We see that, at a BER of 10^{-6}, weighted BF decoding algorithm 1 achieves a 0.6 dB coding gain over the hard-decision LPCF-BF-decoding algorithm. Of course, this coding gain is achieved at the expense of a larger computational complexity.

Steps 3 and 4 of weighted BF decoding algorithm 1 can be modified to allow flipping of multiple bits at time at Step 4 (see Problem 10.18).

10.9 Weighted BF Decoding: Algorithms 2 and 3

The performance of weighted BF decoding algorithm 1 presented in Section 10.8 can be further enhanced by improving the reliability measure of a received bit given by (10.49) and preventing the possibility of the decoding being trapped in a *loop*.

Of course, this performance enhancement comes with a cost in terms of additional computational complexity and memory requirement. In this section, we present two enhancements of weighted BF decoding algorithm 1. These enhancements are based on the work on weighted BF decoding given in [17,34–37].

For $0 \leq i < m$ and $j \in N_i$, we define a new reliability measure of the syndrome sum s_i that checks on the jth received bit z_j as follows:

$$\phi_{i,j} = \min_{j' \in N_i \setminus j} |y_{j'}|. \tag{10.51}$$

We notice that the above reliability measure $\phi_{i,j}$ of the syndrome sum s_i checking on the jth received bit z_j is different from the reliability measure ϕ_i of the same syndrome sum that is defined by (10.48). The reliability measure ϕ_i of the syndrome sum s_i is defined by considering the reliabilities of *all the received bits* participating in s_i, while the reliability measure $\phi_{i,j}$ of s_i is defined by *excluding* the reliability measure $|y_j|$ of the jth received bit z_j that is checked by s_i. Therefore, $\phi_{i,j}$ is a function both of the row index i and of the column index j of the parity-check matrix $\mathbf{H}$ of the code to be decoded, while ϕ_i is a function of the row index i alone.

Next define a new weight reliability measure of a hard-decision received bit z_j as follows:

$$E_{j,\epsilon} = \sum_{i \in M_j} (2s_i - 1)\phi_{i,j} - \epsilon |y_j|, \tag{10.52}$$

which consists of two parts. The first part of $E_{j,\epsilon}$ is the sum part of (10.52), which contains the reliability information coming from all the syndrome sums orthogonal on the jth received bit z_j but not including the reliability information of z_j. This part indicates to *what extent* the received bit z_j should be flipped. The second part $\epsilon|y_j|$ of $E_{j,\epsilon}$ gives reliability information on the received bit z_j. It basically indicates to what extent the received bit z_j should maintain its value unchanged. The parameter ϵ of the second part of $E_{j,\epsilon}$ is a positive real number, a design parameter, which is called the *confidence coefficient* of received bit z_j. For a given LDPC code, the confidence coefficient ϵ should be chosen so as to optimize the reliability measure E_j and hence to minimize the error rate of the code with BF decoding. The optimum value of ϵ is hard to derive analytically and it is usually found by computer simulation [17,35,37]. Experimental results show that the optimum choice of ϵ varies slightly with the SNR [35]. For simplicity, it is kept constant during the decoding process.

Weighted BF decoding algorithm 1 presented in Section 10.8 can be modified by using the new reliability measures of a syndrome sum and a received bit defined by (10.51) and (10.52), respectively. For $0 \leq k < k_{\max}$, let $\mathbf{z}^{(k)} = (z_0^{(k)}, z_1^{(k)}, \ldots, z_{n-1}^{(k)})$ be the hard-decision received sequence generated in the kth decoding iteration and let $\mathbf{E}_\epsilon^{(k)} = (E_{0,\epsilon}^{(k)}, E_{1,\epsilon}^{(k)}, \ldots, E_{n-1,\epsilon}^{(k)})$ be the weighted reliability profile of $\mathbf{z}^{(k)}$, where $E_{j,\epsilon}^{(k)}$ is the weighted reliability measure of the jth bit of $\mathbf{z}^{(k)}$, computed using (10.52) and the syndrome $\mathbf{s}^{(k)}$ of $\mathbf{z}^{(k)}$.

Algorithm 10.4 Weighted BF Decoding Algorithm 2

Initialization: Set $k = 0$, $\mathbf{z}^{(0)} = \mathbf{z}$, and the maximum number of iterations to $k_{\max}$. Store $\epsilon|y_j|$ for $0 \le j < n$. Compute and store $\phi_{i,j}$ for $0 \le i < m$ and $j \in N_i$.

1. Compute the syndrome $\mathbf{s}^{(k)} = \mathbf{z}^{(k)}\mathbf{H}^{\mathrm{T}}$ of $\mathbf{z}^{(k)}$. If $\mathbf{s}^{(k)} = \mathbf{0}$, stop decoding and output $\mathbf{z}^{(k)}$ as the decoded codeword; otherwise, go to Step 2.
2. Compute the weighted reliability profile $\mathbf{E}_{\epsilon}^{(k)}$ on the basis of (10.51) and (10.52). Go to Step 3.
3. Identify the bit position j for which $E_{j,\epsilon}^{(k)}$ is largest. Go to Step 4. (Remark: If j is not unique, then choose a j at random.)
4. Flip the jth received bit of $\mathbf{z}^{(k)}$ to obtain a modified received sequence $\mathbf{z}^{(k+1)}$. Go to Step 5.
5. $k \leftarrow k + 1$. If $k > k_{\max}$, stop decoding; otherwise, go to Step 1.

On comparing (10.51) with (10.49), we see that weighted BF decoding algorithm 2 requires more real-number additions than does weighted BF decoding algorithm 1. Furthermore, it requires more storage than does algorithm 1 because it has to store $\epsilon|y_j|$ for $0 \le j < n$ in addition to $\phi_{i,j}$ for $0 \le i < m$ and $j \in N_i$.

Again, we consider the (4095,3367) cyclic EG-LDPC code constructed in Example 10.13. The bit-error performance of this code over the binary-input AWGN channel decoded using the weighted BF decoding algorithm 2 with a maximum of 100 iterations is shown in Figure 10.18. The confidence coefficient ϵ is chosen to be 2.64 by computer search. At a BER of 10^{-6}, weighted BF decoding algorithm 2 outperforms weighted BF decoding algorithm 1 by 0.5 dB and it performs only 0.5 dB from the SPA with a maximum of 50 iterations.

During the BF decoding process, it may happen that the generated received sequence at some iteration, say the kth iteration, is a *repetition* of the received sequence generated at an earlier iteration, say the k_0th iteration, with $1 \le k_0 < k$, i.e., $\mathbf{z}^{(k)} = \mathbf{z}^{(k_0)}$. In this case, $\mathbf{s}^{(k)} = \mathbf{s}^{(k_0)}$ and $\mathbf{E}_{\epsilon}^{(k)} = \mathbf{E}_{\epsilon}^{(k_0)}$. Consequently, the decoder may enter into a *loop* and generate the same set of received sequences over and over again until a preset maximum number of iterations is reached. This loop is referred to as a *trapping loop*. When a BF decoder enters into a trapping loop, decoding will never converge.

A trapping loop can be detected and avoided by carefully choosing the bit to be flipped such that all the generated received sequences $\mathbf{z}^{(k)}, 0 \le k < k_{\max}$, are different. The basic idea is very straightforward. We list all the received sequences, say $\mathbf{z}^{(0)}$ to $\mathbf{z}^{(k-1)}$, that have been generated and compare the newly generated received sequence $\mathbf{z}^{(k)}$ with them. Suppose $\mathbf{z}^{(k)}$ is obtained by flipping the jth bit $z_j^{(k-1)}$ of $\mathbf{z}^{(k-1)}$, which has largest value $E_{j,\epsilon}^{(k-1)}$ in the reliability profile $\mathbf{E}_{\epsilon}^{(k-1)}$ of $\mathbf{z}^{(k-1)}$. If the newly generated sequence $\mathbf{z}^{(k)}$ is different from the sequences on the list, we add $\mathbf{z}^{(k)}$ to the list and continue the decoding process. If the newly generated sequence $\mathbf{z}^{(k)}$ has already been generated before, we discard it and flip

the bit of $\mathbf{z}^{(k-1)}$, say the lth bit, $z_l^{(k-1)}$, with the *next-largest component* in $\mathbf{E}_\varepsilon^{(k-1)}$. We keep flipping the next unreliable bits of $\mathbf{z}^{(k-1)}$, one at a time, until we generate a sequence that has never been generated before. Then we add this new sequence to the list and continue the decoding process. This approach to detecting and avoiding a trapping loop in weighted BF decoding was first proposed in [36].

It seems that the above straightforward approach to detecting and avoiding a trapping loop requires a large memory to store all the generated received sequences and a large number of vector comparisons. Fortunately, it can be accomplished with a simple mechanism that does not require either a large memory to store all the generated received sequences or a large number of vector comparisons. For $0 \le l < n$, define the following n-tuple over GF(2):

$$\mathbf{u}_l = (0, 0, \ldots, 0, 1, 0, \ldots, 0),$$

whose lth component is equal to 1, with all the other components being equal to 0. This n-tuple over GF(2) is called a *unit-vector*. Let j_k be the bit position of $\mathbf{z}^{(k-1)}$ chosen to be flipped at in the kth decoding iteration. Then

$$\mathbf{z}^{(k)} = \mathbf{z}^{(k-1)} + \mathbf{u}_{j_k}, \tag{10.53}$$

$$\mathbf{z}^{(k)} = \mathbf{z}^{(k_0)} + \mathbf{u}_{j_{k_0+1}} + \cdots + \mathbf{u}_{j_k}, \tag{10.54}$$

for $0 \le k_0 < k$. For $0 \le l \le k$, define the following sum of unit-vectors:

$$\mathbf{U}_l^{(k)} = \sum_{t=l}^{k} \mathbf{u}_{j_t}. \tag{10.55}$$

It follows from (10.54) and (10.55) that we have

$$\mathbf{z}^{(k)} = \mathbf{z}^{(k_0)} + \mathbf{U}_{k_0+1}^{(k)}. \tag{10.56}$$

It is clear that $\mathbf{z}^{(k)} = \mathbf{z}^{(k_0)}$ if and only if $\mathbf{U}_{k_0+1}^{(k)} = \mathbf{0}$. This implies that, to avoid a possible trapping loop, the vector-sums $\mathbf{U}_1^{(k)}, \mathbf{U}_2^{(k)}, \ldots, \mathbf{U}_{k-1}^{(k)}$ must all be *nonzero* vectors. Note that all these vector-sums depend only on the bit positions, $j_1, j_2, \ldots, j_k$, at which bits were flipped from the first iteration to the kth iteration. Using our knowledge of these bit positions, we don't have to store all the generated received sequences.

Note that $\mathbf{U}_l^{(k)}$ can be constructed recursively:

$$\mathbf{U}_l^{(k)} = \begin{cases} \mathbf{u}_{j_k}, & \text{for } l = k, \\ \mathbf{U}_{l+1}^{(k)} + \mathbf{u}_{j_l}, & \text{for } 1 \le l < k. \end{cases} \tag{10.57}$$

To check whether $\mathbf{U}_l^{(k)}$ is zero, we define w_l as the Hamming weight of $\mathbf{U}_l^{(k)}$. Then w_l can be computed recursively as follows, without counting the number of 1s in $\mathbf{U}_l^{(k)}$:

$$w_l = \begin{cases} 1, & \text{if } l = k \\ w_{l+1} + 1, & \text{if the } l\text{th bit of } \mathbf{U}_{l+1}^{(k)} = 0, \\ w_{l+1} - 1, & \text{if the } l\text{th bit of } \mathbf{U}_{l+1}^{(k)} = 1. \end{cases} \tag{10.58}$$

Obviously, $\mathbf{U}_l^{(k)} = \mathbf{0}$ if and only if $w_l = 0$.

Let B denote a list containing the bit positions that would cause trapping loops during decoding. This list of the bit positions is called a *loop-exclusion list*, which will be built up as the decoding iterations go on. With the above developments, weighted BF decoding algorithm 2 can be modified with a simple loop-detection mechanism as follows.

Algorithm 10.5 Weighted BF Decoding Algorithm 3 with Loop Detection

Initialization: Set $k = 0$, $\mathbf{z}^{(0)} = \mathbf{z}$, the maximum number of iterations to $k_{\max}$ and the loop-exclusion list $B = \varnothing$ (empty set). Compute and store $\phi_{i,j}$ for $0 \le i < m$ and $j \in N_i$. Store $\epsilon|y_j|$ for $0 \le j < n$.

1. Compute the syndrome $\mathbf{s}^{(k)} = \mathbf{z}^{(k)}\mathbf{H}^{\mathrm{T}}$ of $\mathbf{z}^{(k)}$. If $\mathbf{s}^{(k)} = \mathbf{0}$, stop decoding and output $\mathbf{z}^{(k)}$ as the decoded codeword; otherwise, go to Step 2.
2. $k \leftarrow k + 1$. If $k > k_{\max}$, declare a decoding failure and stop the decoding process; otherwise, go to Step 3.
3. Compute the weighted reliability profile $\mathbf{E}_\epsilon^{(k)}$ of $\mathbf{z}^{(k)}$. Go to Step 4.
4. Choose the bit position
$$j_k = \arg \max_{0 \le j < n, j \notin B} E_{j,\epsilon}^{(k)}.$$
Go to Step 5.
5. Compute $\mathbf{U}_l^{(k)}$ and w_l on the basis of (10.57) and (10.58) for $1 \le l < k$. If $w_l = 0$ for any l, then $B \leftarrow B \cup \{j_k\}$ and go to Step 5. If all $w_1, w_2, \ldots, w_{k-1}$ are nonzero, go to Step 6.
6. Compute $\mathbf{z}^{(k)} = \mathbf{z}^{(k-1)} + \mathbf{u}_{j_k}$. Go to Step 1.

Compared with weighted BF decoding algorithm 2, weighted BF decoding algorithm 3 with loop detection requires additional memory to store a loop-exclusion list and more computations to carry out Steps 4 and 5. Avoiding trapping loops in flipping during decoding would often make the decoding of many LDPC codes, be they algebraically or randomly constructed, converge faster and would lower their error floors. However, it might not make much difference for the finite-geometry codes constructed in Sections 10.1–10.3 and 10.6 because they already perform well with the weighted BF decoding algorithms without loop detection in terms of decoding convergence and the error floor. Clearly, loop detection can be included in the LPCF-BF and weighted BF decoding algorithms (see Problem 10.19).

10.10 Concluding Remarks

Before we conclude this chapter, we must emphatically point out that all the hard-decision and reliability-based decoding algorithms presented in Sections 10.7–10.9 can be applied to decode any LDPC code whose parity-check matrix satisfies the RC-constraint, not just the FG-LDPC codes constructed in Sections 10.1–10.3 and 10.6. In fact, they are very effective at decoding the codes constructed on the basis of finite fields to be presented in the next chapter. These decoding algorithms, together with the iterative decoding algorithms based on belief propagation presented in Chapter 5, provide a wide spectrum of trade-offs among error performance, decoding complexity, and decoding speed.

See also [38–43].

Problems

10.1 Construct a cyclic-EG-LDPC code based on the lines of the two-dimensional Euclidean geometry EG(2,13) over GF(13) not passing through the origin of the geometry. Determine the length, the dimension, the minimum distance, and the generator polynomials of the code. Compute the bit- and word-error performances of the code over the binary-input AWGN channel decoded using the SPA with a maximum of 50 iterations.

10.2 Construct a cyclic-EG-LDPC code based on the lines of the two-dimensional Euclidean geometry EG(2,2^4) over GF(2^4) not passing through the origin of the geometry. Determine its generator polynomial and compute its bit- and word-error performances over the binary-input AWGN channel using the SPA with a maximum of 50 iterations.

10.3 Consider the three-dimensional Euclidean geometry EG(3,7) over GF(7). How many cyclic classes of lines not passing through the origin of this geometry can be formed? Construct a QC-EG-LDPC code of the longest possible length based on these cyclic classes of lines. Compute the bit- and word-error performances of this code over the binary-input AWGN channel using the SPA with a maximum of 50 iterations.

10.4 Consider the two-dimensional Euclidean geometry EG(2,2^6) over GF(2^6). How many parallel bundles of lines can be formed? Suppose we choose six parallel bundles of lines in this geometry to construct a regular LDPC code.

1. Determine the length and dimension of the code.
2. Compute the bit- and word-error performances of the code over the binary-input AWGN channel decoded using the SPA with a maximum of 50 iterations.
3. Display the rate of decoding convergence of the constructed code with 1, 3, 5, 10, 20, and 50 iterations.

10.5 The incidence vectors of the lines in the two-dimensional Euclidean geometry EG(2,2^5) over GF(2^5) not passing through the origin of the geometry form a single 1023×1023 circulant with both column weight and row weight 32.

1. Using column splitting, decompose this circulant into a row $\mathbf{G}_{\text{col,decom}} = [\mathbf{G}_0 \quad \mathbf{G}_1 \quad \cdots \quad \mathbf{G}_{15}]$ of 16 1023×1023 circulants, each having column weight 4 and row weight 4. Take the first eight circulants of $\mathbf{G}_{\text{col,decom}}$ to form a parity-check matrix $\mathbf{H}_{\text{col,decom}}(1,8) = [\mathbf{G}_0 \quad \mathbf{G}_1 \quad \cdots \quad \mathbf{G}_7]$. Determine the QC-LDPC code given by the null space of $\mathbf{H}_{\text{col,decom}}(1,8)$. Compute its bit- and word-error performances over the binary-input AWGN channel using the SPA with 50 iterations.
2. Using row splitting, decompose each circulant of $\mathbf{H}_{\text{col,decom}}(1,8)$ into a column of two 1023×1023 circulants of weight 2. This decomposition results in a 2×8 array $\mathbf{H}_{\text{array,decom}}(2,8)$ of 1023×1023 circulants, each with weight 2. Determine the QC-LDPC code given by the array $\mathbf{H}_{\text{array,decom}}(2,8)$ and compute its bit- and word-error performances over the binary-input AWGN channel using the SPA with a maximum 50 iterations.

10.6 This problem is a continuation of 10.5.

1. Using row splitting, decompose each circulant of $\mathbf{H}_{\text{col,decom}}(1,8)$ into a column of four 1023×1023 circulant permutation matrices. This decomposition results in a 4×8 array $\mathbf{H}_{\text{array,decom}}(4,8)$ of 1023×1023 circulant permutation matrices. Determine the QC-LDPC code given by the null space of $\mathbf{H}_{\text{array,decom}}(4,8)$ and compute its bit- and word-error performances over the binary-input AWGN channel using the SPA with 50 iterations.
2. Replace a circulant permutation matrix in each column of the 4×8 array $\mathbf{H}_{\text{array,decom}}(4,8)$ that you have constructed by a zero matrix such that the resultant array $\mathbf{H}^*_{\text{array,decom}}(4,8)$ has three circulant permutation matrices and one zero matrix in each column, and six circulant permutation matrices and two zero matrices in each row. Determine the QC-LDPC code given by the null space of $\mathbf{H}^*_{\text{array,decom}}(4,8)$ and compute its bit- and word-error performances over the binary-input AWGN channel using the SPA with a maximum of 50 iterations.

10.7 Decompose the two-dimensional Euclidean geometry EG(2,2^6) based on a chosen parallel bundle $\mathcal{P}(2,1)$ of lines (see Example 10.5). From this decomposition, a 64×64 array $\mathbf{H}^{(1)}_{\text{EG},d}$ of 64×64 permutation matrices over GF(2) can be constructed. Take the first four rows of $\mathbf{H}^{(1)}_{\text{EG},d}$ to form a 4×64 subarray $\mathbf{H}^{(1)}_{\text{EG},d}(4,64)$, which is a 256×4096 matrix over GF(2) with column and row weights 4 and 64, respectively. Determine the (4,64)-regular LDPC code given by the null space of $\mathbf{H}^{(1)}_{\text{EG},d}(4,64)$ and compute its bit- and word-error performances of the code over the binary-input AWGN channel using the SPA with a maximum of 50 iterations.

10.8 This problem is a continuation of Problem 10.7. Suppose we take a 32×64 subarray $\mathbf{H}^{(1)}_{\mathrm{EG},d}(32,64)$ from the 64×64 array $\mathbf{H}^{(1)}_{\mathrm{EG},d}$ of 64×64 permutation matrices over GF(2) constructed in Problem 10.7, say the first 32 rows of $\mathbf{H}^{(1)}_{\mathrm{EG},d}$. Use the subarray $\mathbf{H}^{(1)}_{\mathrm{EG},d}(32,64)$ as a base array for masking. Construct a masking matrix $\mathbf{Z}(32,64)$ over GF(2) that consists of two 32×32 circulants in a row, each with column weight 3 and row weight 3. Then $\mathbf{Z}(32,64)$ is a 32×64 matrix over GF(2) with column and row weights 3 and 6, respectively. Masking $\mathbf{H}^{(1)}_{\mathrm{EG},d}(32,64)$ with $\mathbf{Z}(32,64)$ results in a 32×64 masked array $\mathbf{M}^{(1)}_{\mathrm{EG},d}(32,64)$ that is a 4096×2048 matrix over GF(2) with column and row weights 3 and 6, respectively. Determine the code given by the null space of $\mathbf{M}^{(1)}_{\mathrm{EG},d}(32,64)$ and compute its bit- and word-error performance over the binary-input AWGN channel using the SPA with a maximum of 50 iterations.

10.9 Prove that, for even m, the incidence vector of any line in the m-dimensional projective geometry PG(m,q) over GF(q) is primitive.

10.10 Prove that, for odd m, there are $(q^{m+1}-1)/(q^2-1)$ lines in the m-dimensional projective geometry PG(m,q) over GF(q) whose incidence vectors are not primitive.

10.11 Consider the cyclic-PG-LDPC code whose parity-check matrix is formed by using the incidence vectors of the lines of the two-dimensional projective geometry PG(2,2^4) over GF(2^4) as rows. Determine the code and compute its bit- and word-error performances over the binary-input AWGN channel using the SPA with a maximum of 50 iterations. Also display the rate of decoding convergence of this code for iterations 1, 3, 5, 10, and 50.

10.12 Consider the three-dimensional Euclidean geometry EG(3,2^3) over GF(2^3). The Galois field GF(2^9) as an extension field of GF(2^3) is a realization of EG(3,2^3). Let α be a primitive element of GF(2^9). Let $\mathcal{F}$ be a 2-flat in EG(3,2^3) not passing through the origin of the geometry. Define the incidence vector of $\mathcal{F}$ as a 511-tuple over GF(2),

$$\mathbf{v}_{\mathcal{F}} = (v_0, v_1, \ldots, v_{510}),$$

whose components correspond to the non-origin points $\alpha^0, \alpha, \ldots, \alpha^{511}$ of EG(3,2^3), where $v_i = 1$ if and only if α^i is a point on $\mathcal{F}$, otherwise $v_i = 0$. Form a matrix $\mathbf{H}$ with the incidence vectors of the 2-flats of EG(3,2^3) not passing through the origin of the geometry as rows. Prove that the null space of $\mathbf{H}$ gives a cyclic code. Does the Tanner graph of this code contain cycles of length 4? If it does, how many cycles of length 4 are there? Decode this code using the SPA with a maximum of 50 iterations and compute its bit- and word-error performances over the binary-input AWGN channel.

10.13 Consider an OSMLG-decodable code C with OSMLG error-correction capability $\lfloor g/2 \rfloor$. Show that there is at least one error pattern with $\lfloor g/2 \rfloor + 1$ errors

that is not correctable with OSMLG decoding. Also show that there are some error patterns with more than $\lfloor g/2 \rfloor$ errors that are correctable with OSMLG decoding.

10.14 Let the two-dimensional Euclidean geometry EG(2,2^5) over GF(2^5) be the code-construction geometry. A cyclic EG-LDPC code of length 1023 can be constructed by utilizing the type-1 incidence vectors of the lines in EG(2,2^5) not passing through the origin.

1. Determine the dimension of this code and its OSMLG error-correction capability.
2. Compute its error performance over the BSC with OSMLG decoding.
3. Compute its error performance over the BSC with LPCF-BF decoding using 10 and 50 iterations, respectively.
4. Compute its error performances over the binary-input AWGN channel using weighted BF decoding algorithms 1 and 2 with 50 iterations.
5. Compute its error performance over the binary-input AWGN channel decoded with the SPA using 50 iterations.

10.15 A 64×64 array $H^{(1)}_{\mathrm{EG},d}$ of 64×64 permutation matrices over GF(2) can be constructed by utilizing decomposition of the two-dimensional Euclidean geometry EG(2,2^6) over GF(2^6) (see Example 10.5). Take a 16×64 subarray $H^{(1)}_{\mathrm{EG},d}(16, 64)$ from $H^{(1)}_{\mathrm{EG},d}$.

1. Determine the EG-LDPC code given by the null space of $H^{(1)}_{\mathrm{EG},d}(16, 64)$.
2. Compute the error performance of the code over the BSC decoded with the LPCF-BF decoding algorithm using 100 iterations.
3. Compute the error performances of the code over the binary-input AWGN channel decoded with weighted BF decoding algorithms 1 and 2 with 100 iterations.
4. Compute the error performance of the code over the binary-input AWGN channel decoded with the SPA with 100 iterations.

10.16 Decode the (6,32)-regular (2048,1723) LDPC code constructed in Example 10.5 by utilizing decomposition of the Euclidean geometry EG(2,2^6) over GF(2^6) with the weighted BF decoding algorithm 2 using 50 and 100 iterations, and compute its error performances over the binary-input AWGN channel.

10.17 Decode the (3510,3109) QC-PG-LDPC code given in Example 10.12 with weighted BF decoding algorithm 2 using 50 and 100 iterations, and compute its error performances over the binary-input AWGN channel.

10.18 Modify weighted BF decoding algorithm 1 to allow flipping of multiple bits at Step 4.

10.19 Devise an LPCF-BF-decoding algorithm with loop detection. With your algorithm, decode the (4095,3367) cyclic EG-LDPC code constructed in Example 10.1 (or Example 10.13).

References

[1] Y. Kou, S. Lin, and M. Fossorier, "Low density parity check codes based on finite geometries: a rediscovery," *Proc. IEEE Int. Symp. Information Theory*, Sorrento, June 25–30, 2000, p. 200.

[2] Y. Kou, S. Lin, and M. Fossorier, "Construction of low density parity check codes: a geometric approach," *Proc. 2nd Int. Symp. Turbo Codes and Related Topics*, Brest, September 2000, pp. 137–140.

[3] Y. Kou, S. Lin, and M. Fossorier, "Low density parity-check codes: construction based on finite geometries," *Proc. IEEE Globecom*, San Francisco, CA, November–December, 2000, pp. 825–829.

[4] Y. Kou, S. Lin, and M. Fossorier, "Low-density parity-check codes based on finite geometries: a rediscovery and new results," *IEEE Trans. Information Theory*, vol. 47, no. 7, pp. 2711–2736, November 2001.

[5] Y. Kou, J. Xu, H. Tang, S. Lin, and K. Abdel-Ghaffar, "On circulant low density parity check codes," *Proc. IEEE Int. Symp. Information Theory*, Lausanne, June–July 2002, p. 200.

[6] S. Lin, H. Tang, and Y. Kou, "On a class of finite geometry low density parity check codes," *Proc. IEEE Int. Symp. Information Theory*, Washington, DC, June 2001, p. 24.

[7] S. Lin, H. Tang, Y. Kou, and K. Abdel-Ghaffar, "Codes on finite geometries," *Proc. IEEE Information Theory Workshop*, Cairns, Australia, September 2–7, 2001, pp. 14–16.

[8] H. Tang, "Codes on finite geometries," Ph.D. dissertation, University of California, Davis, CA, 2002.

[9] H. Tang, Y. Kou, J. Xu, S. Lin, and K. Abdel-Ghaffar, "Codes on finite geometries: old, new, majority-logic and iterative decoding," *Proc. 6th Int. Symp. Communication Theory and Applications*, Ambleside, UK, July 2001, pp. 381–386.

[10] H. Tang, J. Xu, Y. Kou, S. Lin, and K. Abdel-Ghaffar, "On algebraic construction of Gallager low density parity-check codes," *Proc. IEEE Int. Symp. Information Theory*, Lausanne, June–July, 2002, p. 482.

[11] H. Tang, J. Xu, Y. Kou, S. Lin, and K. Abdel-Ghaffar, "On algebraic construction of Gallager and circulant low density parity check codes," *IEEE Trans. Information Theory*, vol. 50, no. 6, pp. 1269–1279, June 2004.

[12] H. Tang, J. Xu, S. Lin, and K. Abdel-Ghaffar, "Codes on finite geometries," *IEEE Trans. Information Theory*, vol. 51, no. 2, pp. 572–596, 2005.

[13] Y. Y. Tai, L. Lan, L.-Q. Zeng, S. Lin, and K. Abdel Ghaffar, "Algebraic construction of quasi-cyclic LDPC codes for AWGN and erasure channels," *IEEE Trans. Communications*, vol. 54, no. 10, pp. 1765–1774, October 2006.

[14] P. O. Vontobel and R. Koetter, *IEEE Trans. Information Theory*, vol. 54, 2009, to be published.

[15] J. Xu, L. Chen, I. Djurdjevic, S. Lin, and K. Abdel-Ghaffar, "Construction of regular and irregular LDPC codes: geometry decomposition and masking," *IEEE Trans. Information Theory*, vol. 53, no. 1, pp. 121–134, January 2007.

[16] L.-Q. Zeng, L. Lan, Y. Y. Tai, B. Zhou, S. Lin, and K. Abdel-Ghaffar, "Construction of nonbinary quasi-cyclic LDPC codes: a finite geometry approach," *IEEE Trans. Communications*, vol. 56, no. 3, pp. 378–387, March 2008.

[17] J. Zhang and M. P. C. Fossorier, "A modified weighted bit-flipping decoding for low-density parity-check codes," *IEEE Communication Lett.*, vol. 8, pp. 165–167, March 2004.

[18] B. Zhou, J.-Y. Kang, Y. Y. Tai, S. Lin, and Z. Ding, "High performance non-binary quasi-cyclic LDPC codes on Euclidean geometries," *IEEE Trans. Communications*, vol. 57. no. 4, 2009, to

be published.

[19] L. D. Rudolph, "Geometric configuration and majority logic decodable codes," M.E.E. thesis, University of Oklahoma, Norman, OK, 1964.

[20] L. D. Rudolph, "A class of majority logic decodable codes," *IEEE Trans. Information Theory*, vol. 13. pp. 305–307, April 1967.

[21] T. Kasami, S. Lin, and W. W. Peterson, "Polynomial codes," *IEEE Trans. Information Theory*, vol. 14, no. 6, pp. 807–814, November 1968.

[22] S. J. Johnson and S. R. Weller, "Codes for iterative decoding form partial geometries," *IEEE Trans. Communications*, vol. 52, no. 2, pp. 236–243, February 2004.

[23] S. Lin and D. J. Costello, Jr., *Error Control Coding: Fundamentals and Applications*, 2nd edn., Upper Saddle River, NJ, Prentice-Hall, 2004.

[24] W. W. Peterson and E. J. Weldon, *Error-Correcting Codes*, 2nd edn., Cambridge, MA, MIT Press, 1972.

[25] E. J. Weldon, Jr., "Euclidean geometry cyclic codes," *Proc. Symp. Combinatorial Mathematics*, University of North Carolina, Chapel Hill, NC, April 1967.

[26] E. R. Berlekamp, *Algebraic Coding Theory*, New York, McGraw-Hill, 1968. Revised edn., Laguna Hills, NY, Aegean Park Press, 1984.

[27] T. Richardson, A. Shokrollahi, and R. Urbanke, "Design of capacity approaching irregular codes," *IEEE Trans. Information Theory*, vol. 47, no. 2, pp. 619–637, February 2001.

[28] L. Chen, J. Xu, I. Djurdjevic, and S. Lin, "Near Shannon limit quasi-cyclic low-density parity-check codes," *IEEE Trans. Communications*, vol. 52, no. 7, pp. 1038–1042, July 2004.

[29] J. M. Goethals and P. Delsarte, "On a class of majority-logic decodable codes," *IEEE Trans. Information Theory*, vol. 14, no. 2, pp. 182–189, March 1968.

[30] K. J. C. Smith, "Majority decodable codes derived from finite geometries," *Institute of Statistics Memo Series*, no. 561, University of North Carolina, Chapel Hill, NC, 1967.

[31] E. J. Weldon, Jr., "New generalization of the Reed–Muller codes, part II: nonprimitive codes," *IEEE Trans. Information Theory*, vol. 14, no. 2, pp. 199–205, March 1968.

[32] R. G. Gallager, "Low-density parity-check codes," *IRE Trans. Information Theory*, vol. IT-8, no. 1, pp. 21–28, January 1962.

[33] J. L. Massey, "Shift-register synthesis and BCH decoding," *IEEE Trans. Information Theory*, vol. IT-15, no. 1, pp. 122–127, January 1969.

[34] J. Chen and M. P. C. Fossorier, "Near optimum universal belief propagation based decoding of low-density parity check codes," *IEEE Trans. Communications*, vol. 50, no. 3, pp. 406–614, March 2002.

[35] M. Jiang, C. Zhao, Z. Shi, and Y. Chen, "An improvement on the modified weighted bit-flipping decoding algorithm for LDPC codes," *IEEE Communications Lett.*, vol. 9, no. 7, pp. 814–816, September 2005.

[36] Z. Liu and D. A. Pados, "A decoding algorithm for finite-geometry LDPC codes," *IEEE Trans. Communications*, vol. 53, no. 3, pp. 415–421, March 2005.

[37] M. Shan, C. Zhao, and M. Jian, "An improved weighted bit-flipping algorithm for decoding LDPC codes," *IEE Proc. Communications*, vol. 152, pp. 1350–2425, 2005.

[38] T. Kasami and S. Lin, "On majority-logic decoding for duals of primitive polynomial codes," *IEEE Trans. Information Theory*, vol. 17, no. 3, pp. 322–331, May 1971.

[39] S. Lin, "On the number of information symbols in polynomial codes," *IEEE Trans. Information Theory*, vol. 18, no. 6, pp. 785–794, November 1972.

[40] N. Miladinovic and M. P. Fossorier, "Improved bit-flipping decoding of low-density parity-check codes," *IEEE Trans. Information Theory*, vol. 51, no. 4, pp. 1594–1606, April 2005.

[41] N. Mobini, A. H. Banihashemi, and S. Hemati, "A differential binary-passing LDPC decoder," *Proc. IEEE Globecom*, Washington, DC, November 2007, pp. 1561–1565.

[42] T. J. Richardson, "Error floors of LDPC codes," *Proc. 41st Allerton Conf. on Communications, Control and Computing*, Monticello, IL, October 2003, pp. 1426–1435.

[43] E. J. Weldon, Jr., "Some results on majority-logic decoding," Chapter 8, *Error-Correcting Codes*, H. Mann, ed., New York, John Wiley, 1968.

11 Constructions of LDPC Codes Based on Finite Fields

In the late 1950s and early 1960s, finite fields were successfully used to construct linear block codes, especially cyclic codes, with large minimum distances for hard-decision algebraic decoding, such as BCH codes [1, 2] and RS codes [3]. There have recently been major developments in using finite fields to construct LDPC codes [4–11]. LDPC code constructions based on finite fields perform well over the binary-input AWGN channel with iterative decoding based on belief propagation. Most importantly, these finite-field LDPC codes were shown to have low error floors, which is important for communication and data-storage systems, for which very low bit and/or word error rates are required. Furthermore, most of the LDPC code construction/based on the basis of finite fields are quasi-cyclic and hence they can be efficiently encoded using simple shift-registers with linear complexity [12] (see Chapter 3). This chapter is devoted to constructions of LDPC codes based on finite fields.

11.1 Matrix Dispersions of Elements of a Finite Field

Consider the Galois field GF(q), where q is a power of a prime. Let α be a primitive element of GF(q). Then the q powers of α,

$$\alpha^{-\infty} \triangleq 0, \alpha^0 = 1, \alpha, \ldots, \alpha^{q-2},$$

form all the q elements of GF(q) and $\alpha^{q-1} = 1$. GF(q) consists of two groups, the *additive* and *multiplicative* groups. The q elements of GF(q) under addition form the additive group and the $q-1$ nonzero elements of GF(q) under multiplication form the multiplicative group of GF(q).

For each nonzero element α^i with $0 \leq i < q-1$, we define a $(q-1)$-tuple over GF(2),

$$\mathbf{z}(\alpha^i) = (z_0, z_1, \ldots, z_{q-2}), \tag{11.1}$$

whose components correspond to the nonzero elements $\alpha^0, \alpha, \ldots, \alpha^{q-2}$ of GF(q), where the ith component $z_i = 1$ and all the other $q-2$ components are set to zero. This $(q-1)$-tuple over GF(2) with a single 1-component is referred to as the

location-vector of element α^i with respect to the multiplicative group of GF(q). We call $\mathbf{z}(\alpha^i)$ the $\mathcal{M}$-*location-vector* of α^i, where "$\mathcal{M}$" stands for "multiplicative group." It is clear that the 1-components of the $\mathcal{M}$-location-vectors of two different nonzero elements of GF(q) reside at two different positions. The $\mathcal{M}$-location-vector $\mathbf{z}(0)$ of the 0 element of GF(q) is defined as the all-zero $(q-1)$-tuple, $(0, 0, \ldots, 0)$.

Let δ be a nonzero element of GF(q). For $1 \leq i < q-1$, the $\mathcal{M}$-location-vector $\mathbf{z}(\alpha^i\delta)$ of element $\alpha^i\delta$ is the cyclic-shift (one place to the right) of the $\mathcal{M}$-location-vector $\mathbf{z}(\alpha^{i-1}\delta)$ of the element $\alpha^{i-1}\delta$. Since $\alpha^{q-1} = 1$, $\mathbf{z}(\alpha^{q-1}\delta) = \mathbf{z}(\delta)$. This is to say that $\mathbf{z}(\delta)$ is the cyclic-shift of $\mathbf{z}(\alpha^{q-2}\delta)$. Form a $(q-1) \times (q-1)$ matrix $\mathbf{A}$ over GF(2) with the $\mathcal{M}$-location-vectors of

$$\delta,\ \alpha\delta,\ \ldots,\ \alpha^{q-2}\delta$$

as rows. Then $\mathbf{A}$ is a $(q-1) \times (q-1)$ *circulant permutation matrix (CPM)*, i.e., $\mathbf{A}$ is a permutation matrix for which each row is the right cyclic-shift of the row above it and the first row is the right cyclic-shift of the last row. $\mathbf{A}$ is referred to as the $(q-1)$-*fold matrix dispersion* (or *expansion*) of the field element δ over GF(2). It is clear that the $(q-1)$-fold matrix dispersions of two different nonzero elements of GF(q) are two different CPMs over GF(2).

11.2 A General Construction of QC-LDPC Codes Based on Finite Fields

This section presents a general method for constructing QC-LDPC codes based on finite fields [8, 9]. Let α be a primitive element of GF(q). Construction begins with an $m \times n$ matrix over GF(q),

$$\mathbf{W} = \begin{bmatrix} \mathbf{w}_0 \\ \mathbf{w}_1 \\ \vdots \\ \mathbf{w}_{m-1} \end{bmatrix} = \begin{bmatrix} w_{0,0} & w_{0,1} & \cdots & w_{0,n-1} \\ w_{1,0} & w_{1,1} & \cdots & w_{1,n-1} \\ \vdots & \vdots & \ddots & \vdots \\ w_{m-1,0} & w_{m-1,1} & \cdots & w_{m-1,n-1} \end{bmatrix}, \tag{11.2}$$

whose rows satisfy the following two constraints: (1) for $0 \leq i < m$ and $0 \leq k, l < q-1$ and $k \neq l$, $\alpha^k\mathbf{w}_i$ and $\alpha^l\mathbf{w}_i$ have at most one position where both of them have the same symbol from GF(q) (i.e., they differ in at least $n-1$ positions); and (2) for $0 \leq i, j < m$, $i \neq j$, and $0 \leq k, l < q-1$, $\alpha^k\mathbf{w}_i$ and $\alpha^l\mathbf{w}_j$ differ in at least $n-1$ positions. The above two constraints on the rows of matrix $\mathbf{W}$ are referred to as α-*multiplied row constraints* 1 and 2, respectively. The first α-multiplied row constraint implies that each row has at most one 0-component and the second α-multiplied row constraint implies that any two rows of $\mathbf{W}$ differ in at least $n-1$ positions. Matrices over GF(q) that satisfy α-multiplied row constraints 1 and 2 are referred to as α-*multiplied row-constrained* matrices.

On replacing each entry of $\mathbf{W}$ by its $(q-1)$-fold matrix dispersion over GF(2), we obtain the following $m \times n$ array of $(q-1) \times (q-1)$ square submatrices:

$$\mathbf{H}_{\text{disp}} = \begin{bmatrix} \mathbf{A}_{0,0} & \mathbf{A}_{0,1} & \cdots & \mathbf{A}_{0,n-1} \\ \mathbf{A}_{1,0} & \mathbf{A}_{1,1} & \cdots & \mathbf{A}_{1,n-1} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{A}_{m-1,0} & \mathbf{A}_{m-1,1} & \cdots & \mathbf{A}_{m-1,n-1} \end{bmatrix}, \tag{11.3}$$

where $\mathbf{A}_{i,j}$ is a $(q-1) \times (q-1)$ CPM if the entry $w_{i,j}$ is a nonzero element of GF(q) and a $(q-1) \times (q-1)$ zero matrix if $w_{i,j} = 0$. $\mathbf{H}_{\text{disp}}$ is an $m(q-1) \times n(q-1)$ matrix over GF(2). It follows from α-multiplied row constraint 1 that each row of the array $\mathbf{H}_{\text{disp}}$ has at most one $(q-1) \times (q-1)$ zero matrix. It follows from α-multiplied row constraint 2 that any two rows of $\mathbf{H}_{\text{disp}}$ can have at most one place where both of them have the same CPM, i.e., except at this place, the two CPMs at any other place in these two rows are different. This implies that $\mathbf{H}_{\text{disp}}$, as an $m(q-1) \times n(q-1)$ matrix over GF(2), satisfies the RC-constraint. $\mathbf{H}_{\text{disp}}$ is called the *multiplicative* $(q-1)$*-fold array dispersion* (or simply array dispersion) of $\mathbf{W}$. The subscript "disp" of $\mathbf{H}_{\text{disp}}$ stands for "dispersion."

For any pair (g, r) of integers with $1 \leq g \leq m$ and $1 \leq r \leq n$, let $\mathbf{H}_{\text{disp}}(g, r)$ be a $g \times r$ subarray of $\mathbf{H}_{\text{disp}}$. $\mathbf{H}_{\text{disp}}(g, r)$ is a $g(q-1) \times r(q-1)$ matrix over GF(2). Since it is a submatrix of $\mathbf{H}_{\text{disp}}$, it also satisfies the RC-constraint. Since $\mathbf{H}_{\text{disp}}(g, r)$ is an array of circulant permutation and/or zero matrices of the same size, the null space of $\mathbf{H}_{\text{disp}}(g, r)$ gives a QC-LDPC code $\mathcal{C}_{qc,\text{disp}}$ of length $r(q-1)$ with rate at least $(r-g)/r$, whose Tanner graph has a girth of at least 6. If $\mathbf{H}_{\text{disp}}(g, r)$ does not contain zero matrices, it has constant column and row weights g and r, respectively. Then $\mathcal{C}_{qc,\text{disp}}$ is a (g,r)-regular QC-LDPC code with minimum distance at least $g+2$ for even g and $g+1$ for odd g. The proof is exactly the same as that given in Section 10.3 for regular LDPC codes constructed from arrays of permutation matrices. If $\mathbf{H}_{\text{disp}}(g, r)$ contains zero matrices, it might not have constant column or row weights. In this case, the null space of $\mathbf{H}_{\text{disp}}(g, r)$ gives an irregular QC-LDPC code.

The above $(q-1)$-fold array dispersion of a matrix $\mathbf{W}$ over a finite field GF(q) that satisfies two α-multiplied row-constraints gives a method for constructing QC-LDPC codes. For any given finite field GF(q), a family of QC-LDPC codes with various lengths, rates, and minimum distances can be constructed. Now the question is that of how to construct α-multiplied row-constrained matrices over finite fields for array dispersion. In the next six sections, various methods for constructing α-multiplied row-constrained matrices will be presented.

11.3 Construction of QC-LDPC Codes Based on the Minimum-Weight Codewords of an RS Code with Two Information Symbols

Consider a cyclic $(q-1,2,q-2)$ RS code $\mathcal{C}_b$ over GF(q) of length $q-1$ with two information symbols whose generator polynomial $\mathbf{g}(X)$ has $\alpha, \alpha^2, \ldots, \alpha^{q-3}$ as roots (see Section 3.4). The minimum distance (or minimum weight) of this RS code is $q-2$. It consists of one codeword of weight 0, $(q-1)^2$ codewords of weight $q-2$, and $2(q-1)$ codewords of weight $q-1$ [13]. Since the two polynomials over GF(q), namely

$$\mathbf{v}_1(X) = 1 + X + X^2 + \cdots + X^{q-2}$$

and

$$\mathbf{v}_2(X) = 1 + \alpha X + \alpha^2 X + \cdots + \alpha^{q-2} X^{q-2},$$

have $\alpha, \alpha^2, \ldots, \alpha^{q-2}$ as roots, they are divisible by $\mathbf{g}(X)$ and hence are code polynomials of $\mathcal{C}_b$. Their corresponding codewords are $\mathbf{v}_1 = (1, 1, \ldots, 1)$ and $\mathbf{v}_2 = (1, \alpha, \ldots, \alpha^{q-2})$, respectively. They both have weight $q-1$. Let

$$\mathbf{w}_0 = \mathbf{v}_2 - \mathbf{v}_1 = (0, \alpha - 1, \alpha^2 - 1, \ldots, \alpha^{q-2} - 1). \tag{11.4}$$

Then $\mathbf{w}_0$ is a codeword in $\mathcal{C}_b$ with weight $q-2$ (minimum weight). The $q-1$ components of $\mathbf{w}_0$ are different elements of GF(q). There is one and only one 0 element in $\mathbf{w}_0$.

Form the following $(q-1) \times (q-1)$ matrix over GF(q) with $\mathbf{w}_0$ and its $q-2$ right cyclic-shifts, $\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_{q-2}$, as rows:

$$\begin{aligned}
\mathbf{W}^{(1)} &= \begin{bmatrix} \mathbf{w}_0 \\ \mathbf{w}_1 \\ \vdots \\ \mathbf{w}_{q-2} \end{bmatrix} \\
&= \begin{bmatrix} 0 & \alpha - 1 & \alpha^2 - 1 & \cdots & \alpha^{q-2} - 1 \\ \alpha^{q-2} - 1 & 0 & \alpha - 1 & \cdots & \alpha^{q-3} - 1 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \alpha - 1 & \alpha^2 - 1 & \alpha^3 - 1 & \cdots & 0 \end{bmatrix}.
\end{aligned} \tag{11.5}$$

Each row (or each column) of $\mathbf{W}^{(1)}$ contains one and only one 0 entry and the $q-1$ 0 entries of $\mathbf{W}^{(1)}$ lie on the main diagonal of $\mathbf{W}^{(1)}$. Since $\mathcal{C}_b$ is cyclic and $\mathbf{w}_0$ is a minimum-weight codeword in $\mathcal{C}_b$, all the rows of $\mathbf{W}^{(1)}$ are minimum-weight codewords of $\mathcal{C}_b$. Any two rows of $\mathbf{W}^{(1)}$ differ in all $q-1$ positions. For $0 \le i < q-1$ and $0 \le k, l < q-1$ and $k \ne l$, $\alpha^k \mathbf{w}_i$ and $\alpha^l \mathbf{w}_i$ are two different minimum-weight codewords in $\mathcal{C}_b$ and they both have a 0 element at position i. Hence, they differ in all the other $q-2$ positions. Therefore, the rows of $\mathbf{W}^{(1)}$ satisfy α-multiplied row constraint 1. For $0 \le i, j < q-1$, $i \ne j$, and $0 \le k, l < q-1$, $\alpha^k \mathbf{w}_i$ and $\alpha^l \mathbf{w}_j$ are two different minimum-weight codewords in $\mathcal{C}_b$, and they differ in

at least $q-2$ places. Consequently, the rows of $\mathbf{W}^{(1)}$ satisfy α-multiplied row constraint 2.

On replacing each entry in $\mathbf{W}^{(1)}$ by its multiplicative $(q-1)$-fold matrix dispersion, we obtain the following $(q-1)\times(q-1)$ array of $(q-1)\times(q-1)$ circulant permutation and zero matrices:

$$\mathbf{H}^{(1)}_{qc,\text{disp}} = \begin{bmatrix} \mathbf{O} & \mathbf{A}_{0,1} & \cdots & \mathbf{A}_{0,q-2} \\ \mathbf{A}_{0,q-2} & \mathbf{O} & \cdots & \mathbf{A}_{0,q-3} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{A}_{0,1} & \mathbf{A}_{0,2} & \cdots & \mathbf{O} \end{bmatrix}. \tag{11.6}$$

The array $\mathbf{H}^{(1)}_{qc,\text{disp}}$ has the following structural properties: (1) each row (or column) has one and only one zero matrix; (2) all the zero matrices lie on the main diagonal of the array; (3) each row (or column) is a right (or downward) cyclic-shift of the row above it (or the column on its left) and the first row (or first column) is the right (or downward) cyclic-shift of the last row (or last column); and (4) all the $q-2$ CPMs in each row (or each column) are different from each other. $\mathbf{H}^{(1)}_{qc,\text{disp}}$ is a $(q-1)^2\times(q-1)^2$ matrix over GF(2) that satisfies the RC constraint and has both column weight and row weight $q-2$. Actually, the $(q-1)^2$ rows of $\mathbf{H}^{(1)}_{qc,\text{disp}}$ correspond to the $(q-1)^2$ minimum-weight codewords of $\mathcal{C}_b$. A row in $\mathbf{H}^{(1)}_{qc,\text{disp}}$ is obtained by representing the $q-1$ components of a minimum-weight codeword in $\mathcal{C}_b$ by their respective $\mathcal{M}$-location-vectors.

For any pair $(g,\ r)$ of integers with $1\le g,r<q$, let $\mathbf{H}^{(1)}_{qc,\text{disp}}(g,r)$ be a $g\times r$ subarray of $\mathbf{H}^{(1)}_{qc,\text{disp}}$. Then $\mathbf{H}^{(1)}_{qc,\text{disp}}(g,r)$ is a $g(q-1)\times r(q-1)$ matrix over GF(2) that satisfies the RC-constraint. The null space of $\mathbf{H}^{(1)}_{qc,\text{disp}}(g,r)$ gives a QC-LDPC code $\mathcal{C}^{(1)}_{qc,\text{disp}}$ of length $r(q-1)$ with rate at least $(r-g)/r$, whose Tanner graph has a girth of at least 6. If $\mathbf{H}^{(1)}_{qc,\text{disp}}(g,r)$ lies either above or below the main diagonal of $\mathbf{H}^{(1)}_{qc,\text{disp}}$, it contains no zero submatrices of $\mathbf{H}^{(1)}_{qc,\text{disp}}$ and hence has constant column and row weights g and r, respectively. Then $\mathcal{C}^{(1)}_{qc,\text{disp}}$ is a (g,r)-regular QC-LDPC code with minimum distance at least $g+2$ for even g and $g+1$ for odd g. If $\mathbf{H}^{(1)}_{qc,\text{disp}}(g,r)$ contains some zero submatrices of $\mathbf{H}^{(1)}_{qc,\text{disp}}$, it may have two different column weights $g-1$ and g, and/or two different row weights $r-1$ and r. In this case, $\mathcal{C}^{(1)}_{qc,\text{disp}}$ is a near-regular QC-LDPC code with minimum distance at least g for even g and $g+1$ for odd g. For a given field GF(q), we can construct a family of QC-LDPC codes of various lengths, rates, and minimum distances.

The above construction gives a class of RS-based QC-LDPC codes. If we mask subarrays of the array $\mathbf{H}^{(1)}_{qc,\text{disp}}$ given by (11.6), we can construct many more QC-LDPC codes, regular or irregular.

In the following, we give several examples to illustrate the above code construction with or without masking. Again, we assume BPSK transmission over the binary-input AWGN channel.

Example 11.1. Consider the (256,2,255) RS code over the prime field GF(257). On the basis of this RS code and (11.4)–(11.6), we can construct a 256×256 array $\mathbf{H}^{(1)}_{qc,\text{disp}}$ of circulant permutation and zero matrices over GF(2) of size 256×256 with the 256 zero matrices on the main diagonal of $\mathbf{H}^{(1)}_{qc,\text{disp}}$. For any pair $(g,\ r)$ of integers with $1 \leq g, r < 257$, the null space of a $k \times r$ subarray $\mathbf{H}^{(1)}_{qc,\text{disp}}(g,r)$ of $\mathbf{H}^{(1)}_{qc,\text{disp}}$ gives a QC-LDPC code whose Tanner graph has a girth of at least 6. If we choose $r = 2^i$ with $2 \leq i \leq 8$ and various gs, we can construct a family of QC-LDPC codes of lengths that are powers of 2. For example, we set $g = 4$ and $r = 32$, taking a 4×32 subarray $\mathbf{H}^{(1)}_{qc,\text{disp}}(4,32)$ from $\mathbf{H}^{(1)}_{qc,\text{disp}}$ above or below the main diagonal of $\mathbf{H}^{(1)}_{qc,\text{disp}}$. $\mathbf{H}^{(1)}_{qc,\text{disp}}(4,32)$ is a 1024×8192 matrix over GF(2) with column and row weights 4 and 32, respectively. The null space of this matrix gives a (4,32)-regular (8192,7171) RS-based QC-LDPC code with rate 0.8753. The error performance of this code over the binary-input AWGN channel with iterative decoding using the SPA (100 iterations) is shown in Figure 11.1. At a BER of 10^{-6}, it performs 1 dB from the Shannon limit.

Suppose we set $g = 4$ and $r = 64$. Take a 4×64 subarray $\mathbf{H}^{(1)}_{qc,\text{disp}}(4,64)$ from $\mathbf{H}^{(1)}_{qc,\text{disp}}$, avoiding the zero submatrices on the main diagonal of $\mathbf{H}^{(1)}_{qc,\text{disp}}$. $\mathbf{H}^{(1)}_{qc,\text{disp}}(4,64)$ is a 1024×16384 matrix over GF(2) with column and row weights 4 and 64, respectively. The null space of this matrix gives a (4,64)-regular (16384,15363) RS-based QC-LDPC code with rate 0.9376. Its error performance with iterative decoding using the SPA (100 iterations) is also shown in Figure 11.1. At a BER of 10^{-6}, it performs 0.75 dB from the Shannon limit. If we set $g = 4$ and $r = 128$, we can construct a (4,128)-regular RS-based (32786,31747)

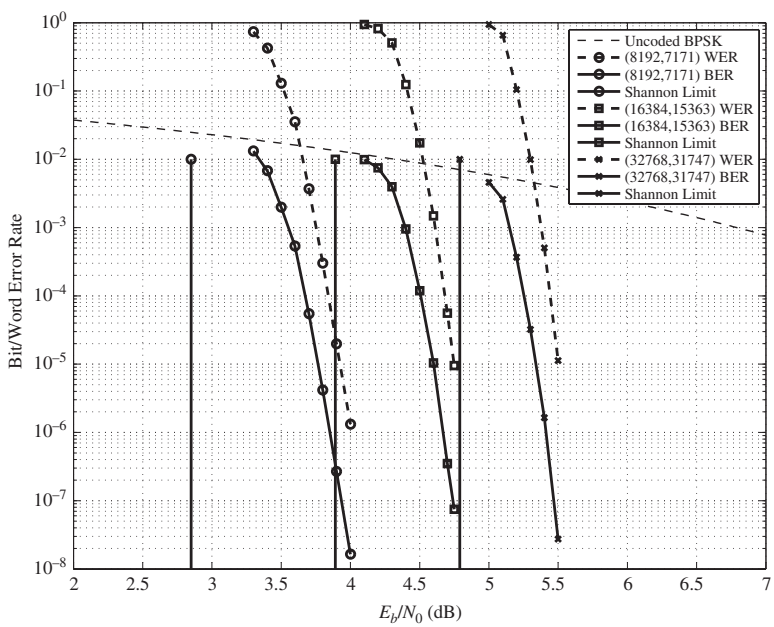


Figure 11.1 Error performances of the (8192,7171), (16384,15363), and (32786,31747) QC-LDPC codes given in Example 11.1.

QC-LDPC code with rate 0.9688. Its error performance decoded with iterative decoding using the SPA (100 iterations) is also shown in Figure 11.1. At a BER of 10^{-6}, it performs 0.6 dB from the Shannon limit.

Example 11.2. A subarray of the array given by (11.6) can be masked to form the parity-check matrix of a QC-LDPC code. Consider the (72,2,71) RS code over the prime field GF(73). From this RS code and (11.4)–(11.6), we can construct a 72×72 array $\mathbf{H}^{(1)}_{qc,\text{disp}}$ of circulant permutation and zero matrices of size 72×72. Set $g = 32$ and $r = 64$. Take a 32×64 subarray $\mathbf{H}^{(1)}_{qc,\text{disp}}(32, 64)$ from $\mathbf{H}^{(1)}_{qc,\text{disp}}$ as the base array for masking, avoiding zero submatrices on the main diagonal of $\mathbf{H}^{(1)}_{qc,\text{disp}}$. Construct a 32×64 masking matrix $\mathbf{Z}(32, 64)$ that consists of a row of two 32×32 circulants. The generators of the two circulants in $\mathbf{Z}(32, 64)$ are two primitive and nonequivalent 32-tuples with weight 3:

$$\mathbf{g}_1 = (10100100000000000000000000000000)$$

and

$$\mathbf{g}_2 = (10000010000000100000000000000000).$$

On masking the base array $\mathbf{H}^{(1)}_{qc,\text{disp}}(32, 64)$ with $\mathbf{Z}(32, 64)$, we obtain a 32×64 masked array $\mathbf{M}(32, 64) = \mathbf{Z}(32, 64) \circledast \mathbf{H}^{(1)}_{qc,\text{disp}}(32, 64)$. The masked array $\mathbf{M}(32, 64)$ consists of 192 CPMs and 1856 zero matrices of size 72×72. It is a 2304×4608 matrix with column and row weights 3 and 6, respectively. The null space of $\mathbf{M}(32, 64)$ gives a (3,6)-regular (4608,2304) QC-LDPC code with rate 0.5 whose Tanner graph has a girth of at least 6. The error performance of this code decoded with iterative decoding using the SPA with 50 iterations is shown in Figure 11.2. At a BER of 10^{-6}, it performs 1.7 dB from the Shannon limit. At a BER of 10^{-9}, it performs 1.9 dB from the Shannon limit and has no error floor down to a BER of 3×10^{-10}. It has a beautiful waterfall performance. It is a very good LDPC code.

Suppose we construct a 32×64 irregular masking matrix $\mathbf{Z}^*(32, 64)$ by computer in accordance with the following degree distributions of variable and check nodes of a Tanner graph, designed for a code of length 4608 with rate 0.5:

$$\tilde{\lambda}(X) = 0.25X + 0.625X^2 + 0.125X^8$$

and

$$\tilde{\rho}(X) = X^6.$$

The column and row weight distributions of $\mathbf{Z}^*(32, 64)$ are given in Table 11.1.

On masking the base array $\mathbf{H}^{(1)}_{qc,\text{disp}}(32, 64)$ given above with the irregular masking matrix $\mathbf{Z}^*(32, 64)$, we obtain a 32×64 irregular masked matrix $\mathbf{M}^*(32, 64)$. The average column and row weights of $\mathbf{M}^*(32, 64)$ are 3.5 and 7. The null space of $\mathbf{M}^*(32, 64)$ gives a (4608,2304) irregular QC-LDPC code. The error performance of this code with iterative decoding using the SPA with 50 iterations is also shown in Figure 11.2. At a BER of 10^{-6},

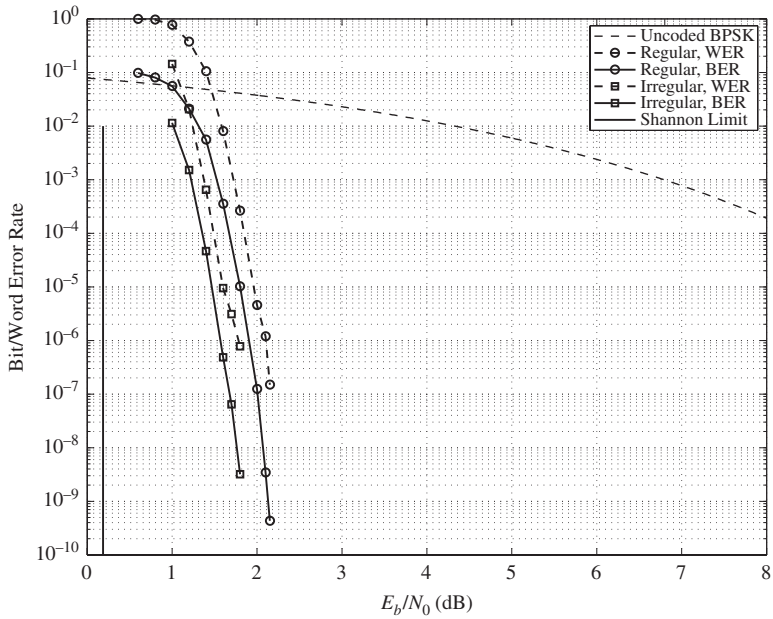


Figure 11.2 Error performances of the (4608,2304) regular and irregular QC-LDPC codes given in Example 11.2.

Table 11.1. Column and row distributions of the masking matrix $\mathbf{Z}^*(32, 64)$ of Example 11.2

Column weight distribution		Row weight distribution	
Column weight	No. of columns	Row weight	No. of rows
2	16	7	32
3	40		
9	8		

it performs 1.45 dB from the Shannon limit. We see that, in the waterfall region, the irregular (4608,2304) QC-LDPC code outperforms the (3,6)-regular (4608,2304) QC-LDPC code by 0.25 dB. However, the irregular code requires larger computation complexity per iteration.

Example 11.3. In this example, we construct another rate-1/2 irregular QC-LDPC code by masking a subarray of the 256×256 array $\mathbf{H}^{(1)}_{qc,\text{disp}}$ constructed from the (256,2,255) RS code over the prime field GF(257) given in Example 11.1. Take a 16×32 subarray $\mathbf{H}^{(1)}_{qc,\text{disp}}(16, 32)$ from $\mathbf{H}^{(1)}_{qc,\text{disp}}$ as the base array for masking, avoiding zero submatrices on the main diagonal of $\mathbf{H}^{(1)}_{qc,\text{disp}}$. Construct a 16×32 masking matrix $\mathbf{Z}(16, 32)$ over GF(2) based on the following variable- and check-node degree distributions of a Tanner graph

Table 11.2. Column and row weight distributions of the masking matrix $\mathbf{Z}(16, 32)$ of Example 11.3

Column weight distribution		Row weight distribution	
Column weight	No. of columns	Row weight	No. of rows
2	8	7	16
3	20		
9	4		

Table 11.3. Column and row weight distributions of the masked matrix $\mathbf{M}(16, 32)$ of Example 11.3

Column weight distribution		Row weight distribution	
Column weight	No. of columns	Row weight	No. of rows
2	2048	7	4096
3	5120		
9	1024		

designed for a rate-1/2 irregular code of length 8196:

$$\tilde{\lambda}(X) = 0.25X + 0.625X^2 + 0.125X^8$$

and

$$\tilde{\rho}(X) = X^6.$$

The column and weight distributions of $\mathbf{Z}(16, 32)$ are given in Table 11.2. The row weight and average column weight are 7 and 3.5, respectively.

Masking the base array $\mathbf{H}^{(1)}_{qc,\text{disp}}(16, 32)$ with $\mathbf{Z}(16, 32)$ results in a 16×32 masked array $\mathbf{M}(16, 32) = \mathbf{Z}(16, 32) \circledast \mathbf{H}^{(1)}_{qc,\text{disp}}(16, 32)$ with 112 circulant permutation and 400 zero matrices of size 256×256. $\mathbf{M}(16, 32)$ is an irregular 4096×8192 matrix over GF(2) with column and row weight distributions given in Table 11.3.

The null space of $\mathbf{M}(16, 32)$ gives an (8192,4096) irregular QC-LDPC code with rate 0.5. The error performances of this code over the binary-input AWGN channel with iterative decoding using the SPA with 20, 50, and 100 iterations are shown in Figure 11.3. With 100 iterations, the code performs 1.2 dB from the Shannon limit at a BER of 10^{-6} and a little less than 1.4 dB from the Shannon limit at a BER of 10^{-9}. Decoding of this code converges relatively fast. At a BER of 10^{-6}, the gap between 20 and 100 iterations is less than 0.25 dB, while the gap between 50 and 100 iterations is less than 0.05 dB.

Example 11.4. The objective of this example is to demonstrate that a long algebraic LDPC code can perform very close to the Shannon limit. Consider the (511,2,510) RS code over GF(2^9). From this RS code and (11.4)–(11.6), we can construct a

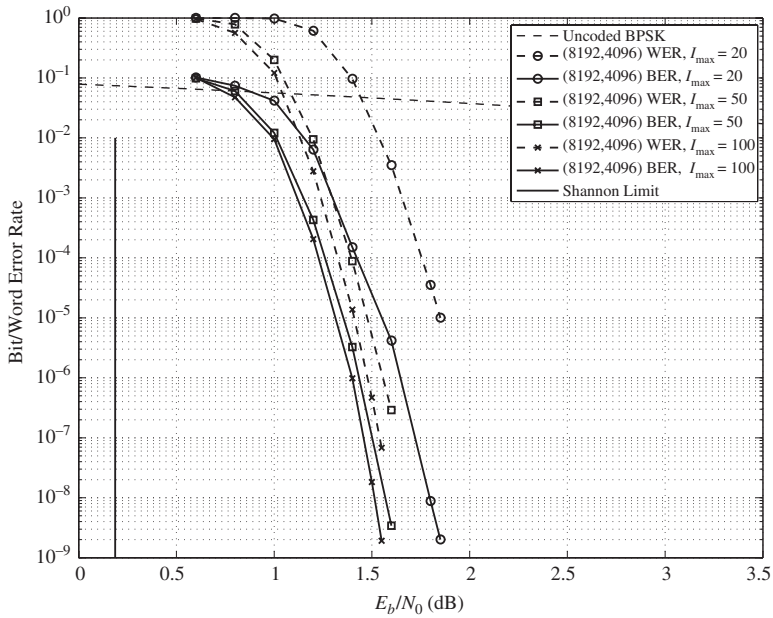


Figure 11.3 The error performance of the (8192,4096) irregular QC-LDPC of Example 11.3.

511×511 array $\mathbf{H}^{(1)}_{qc,\text{disp}}$ of 511×511 circulant permutation and zero matrices over GF(2). Take a 63×126 subarray $\mathbf{H}^{(1)}_{qc,\text{disp}}(63, 126)$ from $\mathbf{H}^{(1)}_{qc,\text{disp}}$ above its main diagonal. So $\mathbf{H}^{(1)}_{qc,\text{disp}}(63, 126)$ is a solid array of CPMs. We use this subarray as the base array for masking to construct an irregular QC-LDPC code. Construct a 63×126 masking matrix by computer based on the following variable- and check-node degree distributions of a Tanner graph of a code of rate 1/2:

$$\begin{aligned}\tilde{\lambda}(X) = {} & 0.4410X + 0.3603X^2 + 0.001\,71X^5 + 0.035\,43 + 0.093\,31X^7 \\ & + 0.0204X^8 + 0.0048X^9 + 0.000\,353X^{27} + 0.042\,92X^{29}\end{aligned}$$

and

$$\tilde{\rho}(X) = 0.008\,42X^7 + 0.990\,23X^8 + 0.001\,35X^9.$$

These two degree distributions are derived using density evolution with the assumption of infinite code length, a cycle-free Tanner graph, and an infinite number of decoding iterations. In accordance with these two degree distributions, we construct a 63×126 masking matrix by computer with column and row weight distributions given in Table 11.4. Note that the column weight distribution in terms of fractions is close (but not identical) to the variable-node degree distribution $\tilde{\lambda}(X)$ given above and the row weight distribution in terms of fractions is quite different from the check-node degree distribution $\tilde{\rho}(X)$ given above.

On masking the base array $\mathbf{H}^{(1)}_{qc,\text{disp}}$ with $\mathbf{Z}(63, 126)$, we obtain a 63×126 masked array $\mathbf{M}(63, 126) = \mathbf{Z}(63, 126) \circledast \mathbf{H}^{(1)}_{qc,\text{disp}}(63, 126)$. It is a 32193×64386 matrix over GF(2). The null space of $\mathbf{M}(63, 126)$ gives a (64386,32193) irregular QC-LDPC code whose error

Table 11.4. Column and row weight distributions of the masking matrix $\mathbf{Z}(63, 126)$ of Example 11.4

Column weight distribution		Row weight distribution	
Column weight	No. of columns	Row weight	No. of rows
2	55	5	1
3	45	6	1
7	4	7	2
8	12	8	5
9	3	9	19
10	1	10	35
30	6		

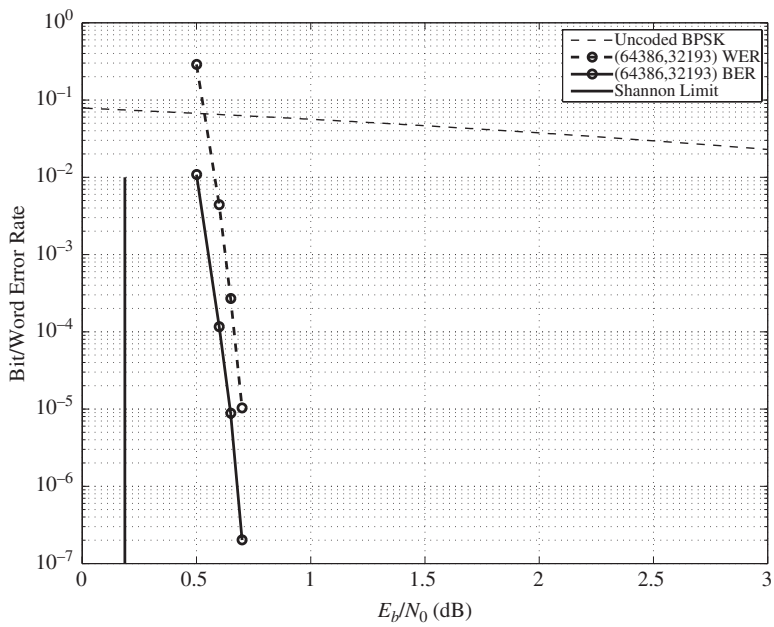


Figure 11.4 The error performance of the (64386,32193) irregular QC-LDPC code of Example 11.4.

performance with iterative decoding using the SPA (100 iterations) is shown in Figure 11.4. At a BER of 10^{-6}, it performs only 0.48 dB from the Shannon limit.

The RS-based QC-LDPC codes presented in this section can be effectively decoded with the OSMLG, BF, and WBF decoding algorithms presented in Sections 10.7–10.9.

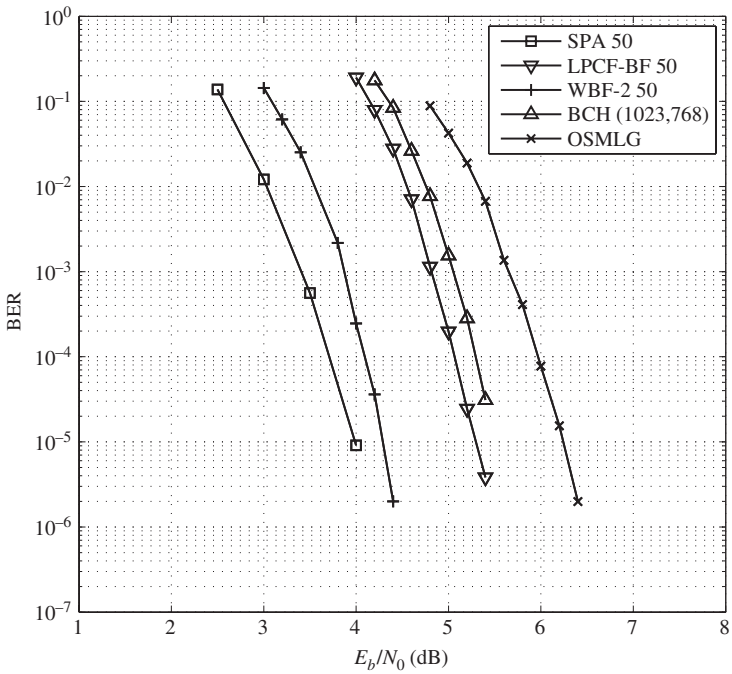


Figure 11.5 Error performances of the (961,721) QC-LDPC code given in Example 11.5 decoded with various decoding algorithms.

Example 11.5. Consider the (31,2,30) RS code C_b over GF(2^5). From this RS code, we can construct a 31×31 array $H^{(1)}_{qc,\text{disp}}$ of 31×31 circulant permutation and zero matrices over GF(2). This array is a 961×961 matrix over GF(2) with both column weight and row weight 30. The null space of this array gives a (961,721) QC-LDPC code over GF(2) with rate 0.750 and minimum distance at least 32. The OSMLG error-correction capability of this code is 15. The error performances of this code decoded with the OSMLG, LPCF-BF, WBF-2, and SPA decoding algorithms are shown in Figure 11.5. Also included in this figure is the performance of a (1023,768) BCH code decoded with the BM algorithm. This BCH code is constructed using the Galois field GF(2^5) and has rate 0.751 and minimum distance 53. We see that the (961,721) QC-LDPC code decoded with the simple LPCF-BF decoding algorithm outperforms the (1023,768) BCH code decoded with the BM algorithm which requires computations over GF(2^5).

11.4 Construction of QC-LDPC Codes Based on the Universal Parity-Check Matrices of a Special Subclass of RS Codes

In this section, we show that an RC-constrained array of CPMs can be constructed from the universal parity-check matrix for a family of RS codes over a given field

GF(q). This universal parity-check matrix is a matrix for which the null space of any number of consecutive rows gives an RS code over GF(q).

Let α be a primitive element of Galois field GF(q). Let m be the *largest prime factor* of $q-1$ and let $q-1=cm$. Let $\beta=\alpha^c$. Then β is an element in GF(q) of order m, i.e., m is the *smallest integer* such that $\beta^m=1$. The set

$$\mathcal{G}_m=\{\beta^0=1,\beta,\ldots,\beta^{m-1}\}$$

forms a cyclic subgroup of the multiplicative group $\mathcal{G}_{q-1}=\{\alpha^0=1,\alpha,\ \ldots,\alpha^{q-2}\}$. Form the following $m\times m$ matrix over GF(q):

$$\begin{aligned}\mathbf{W}^{(2)}&=\begin{bmatrix}\mathbf{w}_0\\ \mathbf{w}_1\\ \mathbf{w}_2\\ \vdots\\ \mathbf{w}_{m-1}\end{bmatrix}\\ &=\begin{bmatrix}1 & 1 & 1 & \cdots & 1\\ 1 & \beta & \beta^2 & \cdots & \beta^{m-1}\\ 1 & \beta^2 & (\beta^2)^2 & \cdots & (\beta^2)^{m-1}\\ \vdots & \vdots & \vdots & \ddots & \vdots\\ 1 & \beta^{m-1} & (\beta^{m-1})^2 & \cdots & (\beta^{m-1})^{m-1}\end{bmatrix},\end{aligned}\tag{11.7}$$

where the powers of β are taken modulo m. For $1\le t\le m$, any t consecutive rows of $\mathbf{W}^{(2)}$ form a parity-check matrix of a cyclic (m,$m-t$,$t+1$) RS code over GF(q), including the *end-around* case (e.g., the last l rows plus the first $t-l$ rows with $0\le l\le t$). Its generator polynomial has t consecutive powers of β as roots. For this reason, we call $\mathbf{W}^{(2)}$ a *universal parity-check matrix* for a family of RS codes.

Since m is a prime, we can easily prove that $\mathbf{W}^{(2)}$ has the following structural properties: (1) except for the first row, all the entries in a row are different and they form all the m elements of the cyclic subgroup $\mathcal{G}_m$ of the multiplicative group of GF(q); (2) except for the first column, all the entries in a column are different and they form all the m elements of $\mathcal{G}_m$; (3) any two rows have only the first entries identical (equal to 1) and are different in all the other $m-1$ positions; and (4) any two columns have only the first entries identical (equal to 1) and are different in all the other $m-1$ positions.

$\mathbf{W}^{(2)}$ satisfy α-multiplied row constraints 1 and 2. The proof of this is given in the following two lemmas.

Lemma 11.1. Let $\mathbf{w}_i$ be the ith row of $\mathbf{W}^{(2)}$ with $0\le i<m$, For $0\le k,l<m$, and $k\ne l$, $\alpha^k\mathbf{w}_i$ and $\alpha^l\mathbf{w}_i$ differ in all $m-1$ positions.

Proof. This lemma is a direct consequence of the fact that all the components of $\mathbf{w}_i$ are nonzero. □

Lemma 11.2. Let $\mathbf{w}_i$ and $\mathbf{w}_j$ be two rows of $\mathbf{W}^{(2)}$ with $0\le i,j<m$, and $i\ne j$. For $0\le k$, $l<q-1$, the two m-tuples, $\alpha^k\mathbf{w}_i$ and $\alpha^l\mathbf{w}_j$, over GF(q) can have at

most one position where they have identical components, i.e., they differ in at least $q-1$ positions.

Proof. Suppose $\alpha^k \mathbf{w}_i$ and $\alpha^l \mathbf{w}_j$ have two positions, say s and t with $t > s$, where they have identical components. Then $\alpha^k(\beta^i)^s = \alpha^l(\beta^j)^s$ and $\alpha^k(\beta^i)^t = \alpha^l(\beta^j)^t$. We assume that $i < j$. Then the above two equalities imply that $\beta^{(j-i)(t-s)} = 1$. Since the order of β is m, $(j-i)(t-s)$ must be a multiple of m. However, since m is a prime and both $j-i$ and $t-s$ are smaller than m, we have that $j-i$ and $t-s$ must be relatively prime to m. Consequently, $(j-i)(t-s)$ cannot be a multiple of m and hence the above hypothesis that $\alpha^k \mathbf{w}_i$ and $\alpha^l \mathbf{w}_j$ have two positions where they have identical components is invalid. This proves the lemma. □

It follows from Lemmas 11.1 and 11.2 that the matrix $\mathbf{W}^{(2)}$ given by (11.7) satisfies α-multiplied row constraints 1 and 2. Hence, $\mathbf{W}^{(2)}$ can be used as the base matrix for array dispersion. On replacing each entry of $\mathbf{W}^{(2)}$ by its multiplicative $(q-1)$-fold matrix dispersion, we obtain the following RC-constrained $m \times m$ array of $(q-1) \times (q-1)$ CPMs:

$$\mathbf{H}^{(2)}_{qc,\text{disp}} = \begin{bmatrix} \mathbf{A}_{0,0} & \mathbf{A}_{0,1} & \cdots & \mathbf{A}_{0,m-1} \\ \mathbf{A}_{1,0} & \mathbf{A}_{1,1} & \cdots & \mathbf{A}_{1,m-1} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{A}_{m-1,0} & \mathbf{A}_{m-1,1} & \cdots & \mathbf{A}_{m-1,m-1} \end{bmatrix}. \tag{11.8}$$

Since all the entries of $\mathbf{W}^{(2)}$ are nonzero, $\mathbf{H}^{(2)}_{qc,\text{disp}}$ contains no zero matrix. $\mathbf{H}^{(2)}_{qc,\text{disp}}$ is an $m(q-1) \times m(q-1)$ matrix over GF(2) with both column weight and row weight equal to m.

For any pair (g, r) of integers with $1 \le g, r \le m$, let $\mathbf{H}^{(2)}_{qc,\text{disp}}(g,r)$ be a $g \times r$ subarray of $\mathbf{H}^{(2)}_{qc,\text{disp}}$. $\mathbf{H}^{(2)}_{qc,\text{disp}}(g,r)$ is a $g(q-1) \times r(q-1)$ matrix over GF(2) with column and row weights g and r, respectively. The null space of $\mathbf{H}^{(2)}_{qc,\text{disp}}(g,r)$ gives a (g,r)-regular QC-LDPC code $\mathcal{C}^{(2)}_{qc,\text{disp}}$ of length $r(q-1)$ with rate at least $(r-g)/r$ and minimum distance $g+2$ for even g and $g+1$ for odd g. The above construction gives a class of QC-LDPC codes.

Example 11.6. Let GF(2^7) be the field for code construction. Since $2^7 - 1 = 127$ is a prime, the largest prime factor of 127 is itself. Using (11.7) and (11.8), we can construct a 127×127 array $\mathbf{H}^{(2)}_{qc,\text{disp}}$ of 127×127 CPMs over GF(2). Take a 4×40 subarray $\mathbf{H}^{(2)}_{qc,\text{disp}}(4,40)$ from $\mathbf{H}^{(2)}_{qc,\text{disp}}$, say the 4×40 subarray at the upper-left corner of $\mathbf{H}^{(2)}_{qc,\text{disp}}$. $\mathbf{H}^{(2)}_{qc,\text{disp}}(4,40)$ is a 508×5080 matrix over GF(2) with column and row weights 4 and 40, respectively. The null space of this matrix gives a (4,40)-regular (5080,4575) QC-LDPC code with rate 0.9006. The error performance of this code over the binary-input AWGN channel decoded with iterative decoding using the SPA (100 iterations) is shown in Figure 11.6. At a BER of 10^{-6}, it performs 1.1 dB from the Shannon limit. Also included in Figure 11.6 is a (5080,4575) random MacKay code constructed by computer search, whose parity-check matrix has column weight 4 and average row weight 40. We see that

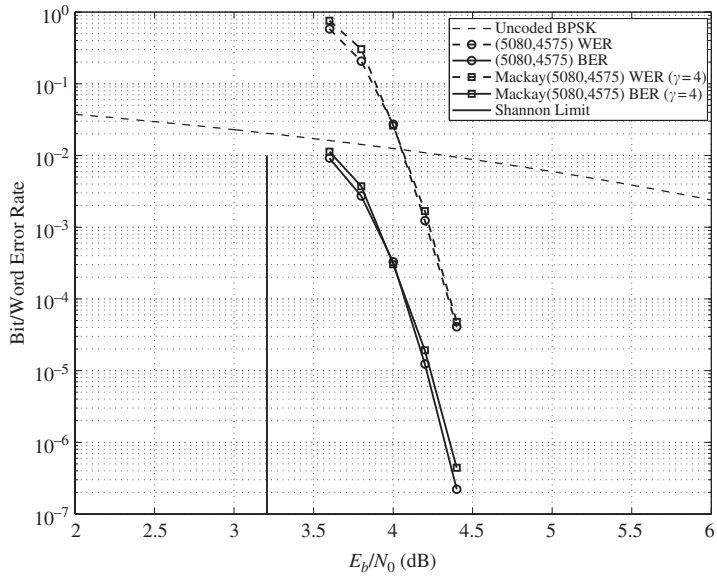


Figure 11.6 Error performances of the (5080,4575) QC-LDPC code of Example 11.6 and a (5080,4565) random MacKay code over the binary-input AWGN channel.

the error performance curves of the two codes are almost on top of each other, but the quasi-cyclic code performs slightly better than the corresponding random MacKay code.

The above array dispersion of $\mathbf{W}^{(2)}$ is based on the multiplicative group $\mathcal{G}_{q-1} = \{\alpha^0 = 1, \alpha, \ldots, \alpha^{q-2}\}$ of GF(q). Every nonzero entry in $\mathbf{W}^{(2)}$ is replaced by a $(q-1) \times (q-1)$ CPM. Since the entries in $\mathbf{W}^{(2)}$ are elements in the cyclic subgroup $\mathcal{G}_m$ of $\mathcal{G}_{q-1}$, $\mathbf{W}^{(2)}$ can be dispersed using a cyclic subgroup of $\mathcal{G}_{q-1}$ that contains $\mathcal{G}_m$ as a subgroup.

Suppose that $q-1$ can be factored as $q-1 = ltm$, where m again is the largest prime factor of $q-1$. Let $\beta = \alpha^{lt}$ and $\delta = \alpha^l$. The orders of β and δ are m and tm, respectively. Then the two sets of elements, $\mathcal{G}_m = \{\beta^0 = 1, \beta, \ldots, \beta^{m-1}\}$ and $\mathcal{G}_{tm} = \{\delta^0 = 1, \delta, \ldots, \delta^{tm-1}\}$, form two cyclic subgroups of $\mathcal{G}_{q-1}$, and $\mathcal{G}_{tm}$ contains $\mathcal{G}_m$ as a subgroup. $\mathcal{G}_{tm}$ is a super group of $\mathcal{G}_m$. Next we define the location-vector of an element δ^i in $\mathcal{G}_{tm}$ as a tm-tuple over GF(2),

$$\mathbf{z}(\delta^i) = (z_0, z_1, \ldots, z_{tm-1}),$$

whose components correspond to the tm elements of $\mathcal{G}_{km}$, where $z_i = 1$ and all the other components are set to zero. We refer to this location-vector as the location-vector of δ^i *with respect to the group* $\mathcal{G}_{tm}$ (or call it the $\mathcal{G}_{tm}$-location-vector of δ^i). Let σ be an element of $\mathcal{G}_{tm}$. Then $\sigma, \delta\sigma, \ldots, \delta^{tm-1}\sigma$ are distinct and they form all the tm elements of $\mathcal{G}_{tm}$. Form a $tm \times tm$ matrix $\mathbf{A}^*$ over $\mathcal{G}_{tm}$ with the $\mathcal{G}_{tm}$-location-vectors of $\sigma, \delta\sigma, \ldots, \delta^{tm-1}\sigma$ as rows. Then $\mathbf{A}^*$ is a $tm \times tm$ CPM over GF(2). $\mathbf{A}^*$ is called the tm-fold matrix dispersion of σ with respect to the cyclic group $\mathcal{G}_{tm}$.

Since $\mathcal{G}_m$ is a cyclic subgroup of $\mathcal{G}_{tm}$, $\mathbf{W}^{(2)}$ satisfies δ-multiplied row constraints 1 and 2. Now, on replacing each entry in $\mathbf{W}^{(2)}$ by its tm-fold matrix dispersion, we obtain the following RC-constrained $m \times m$ array of $tm \times tm$ CPMs over GF(2):

$$\mathbf{H}^{(3)}_{qc,\text{disp}} = \begin{bmatrix} \mathbf{A}^*_{0,0} & \mathbf{A}^*_{0,1} & \cdots & \mathbf{A}^*_{0,m-1} \\ \mathbf{A}^*_{1,0} & \mathbf{A}^*_{1,1} & \cdots & \mathbf{A}^*_{1,m-1} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{A}^*_{m-1,0} & \mathbf{A}^*_{m-1,1} & \cdots & \mathbf{A}^*_{m-1,m-1} \end{bmatrix}. \tag{11.9}$$

$\mathbf{H}^{(3)}_{qc,\text{disp}}$ is a $tm^2 \times tm^2$ matrix over GF(2) with both column weight and row weight m. For any pair (g, r) of integers with $1 \le g, r \le m$, let $\mathbf{H}^{(3)}_{qc,\text{disp}}(g, r)$ be a $g \times r$ subarray of $\mathbf{H}^{(3)}_{qc,\text{disp}}$. $\mathbf{H}^{(3)}_{qc,\text{disp}}(g, r)$ is a $gtm \times rtm$ matrix over GF(2) with column and row weights g and r, respectively. The null space of $\mathbf{H}^{(3)}_{qc,\text{disp}}(g, r)$ gives a (g,r)-regular QC-LDPC code of length rtm, whose Tanner graph has a girth of at least 6.

If the array dispersion of $\mathbf{W}^{(2)}$ is carried out with respect to the cyclic subgroup $\mathcal{G}_m$ of the multiplicative group $\mathcal{G}_{q-1}$ of GF(q), we obtain an $m \times m$ array $\mathbf{H}^{(3)}_{qc,\text{disp}}$ of $m \times m$ CPMs. From the subarrays of this array, we can construct QC-LDPC codes. The above construction based on various supergroups of $\mathcal{G}_m$ gives various sizes of CPMs in $\mathbf{H}^{(3)}_{qc,\text{disp}}$. As a result, we obtain various families of QC-LDPC codes.

Example 11.7. Let GF(2^{10}) be the code-construction field. We can factor $2^{10} - 1 = 1023$ as the product $3 \times 11 \times 31$. The largest prime factor of 1023 is 31. Let $m = 31$, $l = 11$, and $t = 3$. Let α be a primitive element of GF(2^{10}), $\beta = \alpha^{3\times 11}$ and $\delta = \alpha^{11}$. Then $\mathcal{G}_{31} = \{\beta^0 = 1, \beta, \ldots, \beta^{30}\}$ and $\mathcal{G}_{93} = \{\delta^0 = 1, \delta, \ldots, \delta^{92}\}$ form two cyclic subgroups of the multiplicative group $\mathcal{G}_{1023} = \{\alpha^0 = 1, \alpha, \ldots, \alpha^{1022}\}$ of GF(2^{10}). $\mathcal{G}_{93}$ contains $\mathcal{G}_{31}$ as a subgroup. Utilizing (11.7), we form a 31×31 matrix $\mathbf{W}^{(2)}$ with entries from $\mathcal{G}_{31}$, which satisfies δ-multiplied row constraints 1 and 2. On replacing each entry in $\mathbf{W}^{(2)}$ by its 93-fold matrix dispersion with respect to $\mathcal{G}_{93}$, we obtain a 31×31 array $\mathbf{H}^{(3)}_{qc,\text{disp}}$ of 93×93 CPMs. Suppose we take a 4×16 subarray $\mathbf{H}^{(3)}_{qc,\text{disp}}(4, 16)$ from $\mathbf{H}^{(3)}_{qc,\text{disp}}$. $\mathbf{H}^{(3)}_{qc,\text{disp}}(4, 16)$ is a 372×1488 matrix over GF(2) with column and row weights 4 and 16, respectively. The null space of $\mathbf{H}^{(3)}_{qc,\text{disp}}(4, 16)$ gives a (4,16)-regular (1488,1125) QC-LDPC code with rate 0.756. The error performance of this code decoded with iterative decoding using the SPA (50 iterations) is shown in Figure 11.7.

Example 11.8. Suppose we take an 8×16 subarray $\mathbf{H}^{(3)}_{qc,\text{disp}}(8, 16)$ from the array $\mathbf{H}^{(3)}_{qc,\text{disp}}$ of 93×93 CPMs constructed in Example 11.7. We use this subarray as the base array for masking. Construct an 8×16 masking matrix $\mathbf{Z}(8, 16)$ that consists of a row of two 8×8 circulants with generators $\mathbf{g}_1 = (01011001)$ and $\mathbf{g}_2 = (10010110)$. The masking matrix $\mathbf{Z}(8, 16)$ has column and row weights 4 and 8, respectively. Masking $\mathbf{H}^{(3)}_{qc,\text{disp}}(8, 16)$ with $\mathbf{Z}(8, 16)$, we obtain an 8×16 masked array $\mathbf{M}(8, 16)$ which is a 744×1488 matrix with

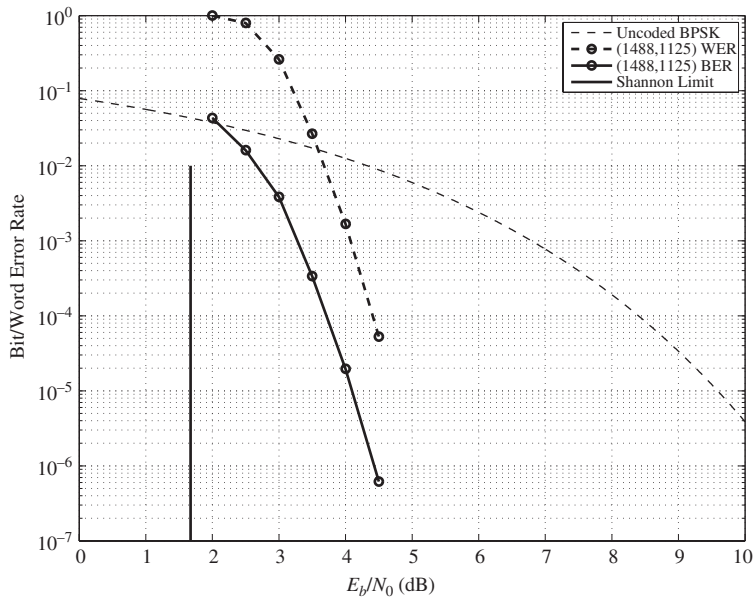


Figure 11.7 The error performance of the (1488,1125) QC-LDPC code of Example 11.7.

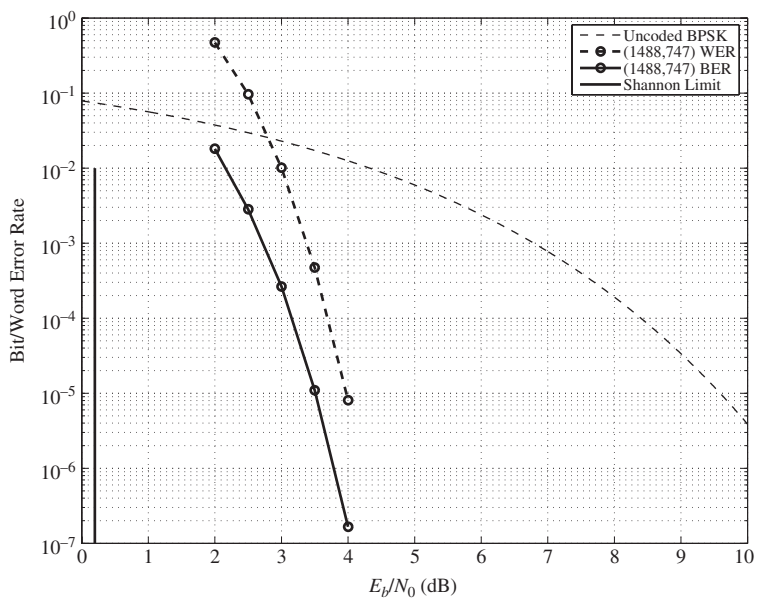


Figure 11.8 The error performance of the (1488,747) QC-LDPC code of Example 11.8.

column and row weights 4 and 8, respectively. The null space of $\mathbf{M}(8, 16)$ gives a (4,8)-regular (1488,747) code with rate 0.502. The error performance of this code with iterative decoding using the SPA is shown in Figure 11.8.

11.5 Construction of QC-LDPC Codes Based on Subgroups of a Finite Field

Subgroups of the additive or multiplicative groups of a finite field can be used to constructed RC-constrained arrays of CPMs. QC-LDPC codes can be constructed from these arrays [10].

11.5.1 Construction of QC-LDPC Codes Based on Subgroups of the Additive Group of a Finite Field

Let $q = p^m$, where p is a prime and m is a positive integer. Let GF(q) be an extension field of the prime field GF(p). Let α be a primitive element of GF(q). Then $\alpha^0, \alpha, \ldots, \alpha^{m-1}$ are linearly independent over GF(q). They form a basis of GF(q), called the polynomial basis. Any element α^i of GF(q) can be expressed as a linear combination of $\alpha^0, \alpha, \ldots, \alpha^{m-1}$ as follows:

$$\alpha^i = c_{i,0}\alpha^0 + c_{i,1}\alpha + \cdots + c_{i,m-1}\alpha^{m-1}$$

with $c_{i,j} \in \mathrm{GF}(p)$. We say that the m independent elements, $\alpha^0, \alpha, \ldots, \alpha^{m-1}$, over GF($p$) span the field GF($q$). For $1 \le t < m$, let $\mathcal{G}_1 = \{\beta_0 = 0, \beta_1, \ldots, \beta_{p^t-1}\}$ be the additive subgroup of GF(q) generated by the linear combinations of $\alpha^0, \alpha, \ldots, \alpha^{t-1}$, i.e.,

$$\beta_i = c_{i,0}\alpha^0 + c_{i,1}\alpha + \cdots + c_{i,t-1}\alpha^{t-1}.$$

Let $\mathcal{G}_2 = \{\delta_0 = 0, \delta_1, \ldots, \delta_{p^{m-t}-1}\}$ be the additive subgroup of GF(q) generated by the linear combinations of $\alpha^t, \alpha^{t+1}, \ldots, \alpha^{m-1}$, i.e.,

$$\delta_i = c_{i,t}\alpha^t + c_{i,t+1}\alpha^{t+1} + \cdots + c_{i,m-1}\alpha^{m-1}.$$

It is clear that $\mathcal{G}_1 \cap \mathcal{G}_2 = \{0\}$. For $0 \le i < p^{m-t}$, define the following set of elements:

$$\delta_i + \mathcal{G}_1 = \{\delta_i, \delta_i + \beta_1, \ldots, \delta_i + \beta_{p^t-1}\}.$$

This set is simply a coset of $\mathcal{G}_1$ with coset leader δ_i. There are p^{m-t} cosets of $\mathcal{G}_1$, including $\mathcal{G}_1$ itself. These p^{m-t} cosets of $\mathcal{G}_1$ form a *partition* of the q elements of the field GF(q). Two cosets of $\mathcal{G}_1$ are mutually disjoint.

Form the following $p^{m-t} \times p^t$ matrix over GF(q):

$$\mathbf{W}^{(4)} = \begin{bmatrix} \mathbf{w}_0 \\ \mathbf{w}_1 \\ \vdots \\ \mathbf{w}_{p^{m-t}-1} \end{bmatrix} = \begin{bmatrix} 0 & \beta_1 & \cdots & \beta_{p^t-1} \\ \delta_1 & \delta_1 + \beta_1 & \cdots & \delta_1 + \beta_{p^t-1} \\ \vdots & \vdots & \ddots & \vdots \\ \delta_{p^{m-t}-1} & \delta_{p^{m-t}-1} + \beta_1 & \cdots & \delta_{p^{m-t}-1} + \beta_{p^t-1} \end{bmatrix}, \quad (11.10)$$

where the ith row consists of the elements of the ith coset of $\mathcal{G}_i$ with $0 \le i < p^{m-t}$. Every element of GF(q) appears once and only once in $\mathbf{W}^{(4)}$. Except for the first row, every row $\mathbf{w}_i$ of $\mathbf{W}^{(4)}$ consists of only nonzero elements of GF(q). The first row of $\mathbf{W}^{(4)}$ contains the 0 element of GF(q) at the first position. Except for the

first column, every column of $\mathbf{W}^{(4)}$ consists of only nonzero elements of GF(q). The first column of $\mathbf{W}^{(4)}$ contains the 0 element of GF(q) at the first position. Since two cosets of $\mathcal{G}_1$ are disjoint, two different rows $\mathbf{w}_i$ and $\mathbf{w}_j$ of $\mathbf{W}^{(4)}$ differ in all positions. We also note that the elements of each column form a coset of $\mathcal{G}_2$. As a result, two different columns of $\mathbf{W}^{(4)}$ differ in all positions. From the above structural properties of $\mathbf{W}^{(4)}$ we can easily prove that $\mathbf{W}^{(4)}$ satisfies α-multiplied row constraints 1 and 2.

On replacing each entry of $\mathbf{W}^{(4)}$ by its multiplicative $(q-1)$-fold matrix dispersion, we obtain the following RC-constrained $p^{m-t} \times p^t$ array $\mathbf{H}^{(4)}_{qc,\text{disp}}$ of $(q-1) \times (q-1)$ CPMs:

$$\mathbf{H}^{(4)}_{qc,\text{disp}} = \begin{bmatrix} \mathbf{A}_{0,0} & \mathbf{A}_{0,1} & \cdots & \mathbf{A}_{0,p^t-1} \\ \mathbf{A}_{1,0} & \mathbf{A}_{1,1} & \cdots & \mathbf{A}_{1,p^t-1} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{A}_{p^{m-t}-1,0} & \mathbf{A}_{p^{m-t}-1,1} & \cdots & \mathbf{A}_{p^{m-t}-1,p^t-1} \end{bmatrix}, \tag{11.11}$$

where $\mathbf{A}_{0,0}$ is a $(q-1) \times (q-1)$ zero matrix, the only zero submatrix in the array. Since all the entries in $\mathbf{W}^{(4)}$ are different, all the permutation matrices in $\mathbf{H}^{(4)}_{qc,\text{disp}}$ are different.

For any pair (g, r) of integers, with $1 \le g \le p^{m-t}$ and $1 \le r \le p^t$, let $\mathbf{H}^{(4)}_{qc,\text{disp}}(g,r)$ be a $g \times r$ subarray of $\mathbf{H}^{(4)}_{qc,\text{disp}}$, which does not contain the single zero matrix of $\mathbf{H}^{(4)}_{qc,\text{disp}}$. $\mathbf{H}^{(4)}_{qc,\text{disp}}(g,r)$ is a $g(q-1) \times r(q-1)$ matrix over GF(2) with column and row weights g and r, respectively. The null space of $\mathbf{H}^{(4)}_{qc,\text{disp}}(g,r)$ gives a (g,r)-regular QC-LDPC code $\mathcal{C}^{(4)}_{qc,\text{disp}}$ of length $r(q-1)$, whose Tanner graph has a girth of at least 6. The above construction gives a class of QC-LDPC codes.

Example 11.9. In this example, we choose $m = 8$ and use GF(2^8) as the code-construction field. Let α be a primitive element of GF(2^8). Set $t = 5$. Then $m - t = 3$. Let $\mathcal{G}_1$ and $\mathcal{G}_2$ be two subgroups of the additive group of GF(2^8) with orders 32 and 8 spanned by the elements in $\{\alpha^0, \alpha, \alpha^2, \alpha^3, \alpha^4\}$ and the elements in $\{\alpha^5, \alpha^6, \alpha^7\}$, respectively. Via these two groups, we can form an 8×32 array $\mathbf{H}^{(4)}_{qc,\text{disp}}$ of 255 CPMs of size 255×255 and a single 255×255 zero matrix. Choose $g = 4$ and $r = 32$. Take a 4×32 subarray $\mathbf{H}^{(4)}_{qc,\text{disp}}(4,32)$ from $\mathbf{H}^{(4)}_{qc,\text{disp}}$, avoiding the single zero matrix. $\mathbf{H}^{(4)}_{qc,\text{disp}}(4,32)$ is a 1020×8160 matrix over GF(2) with column and row weights 4 and 32. The null space of this matrix gives a (4,32)-regular (8160,7159) QC-LDPC code with rate 0.8773. The error performance of this code decoded with iterative decoding using the SPA with 100 iterations is shown in Figure 11.9. At a BER of 10^{-6}, the code performs 0.98 dB from the Shannon limit.

Suppose we take the 8×16 subarray $\mathbf{H}^{(4)}_{qc,\text{disp}}(8,16)$ from $\mathbf{H}^{(4)}_{qc,\text{disp}}$ and use it as a base array for masking. Construct an 8×16 masking matrix $\mathbf{Z}(8,16) = [\mathbf{G}_0 \mathbf{G}_1]$ over

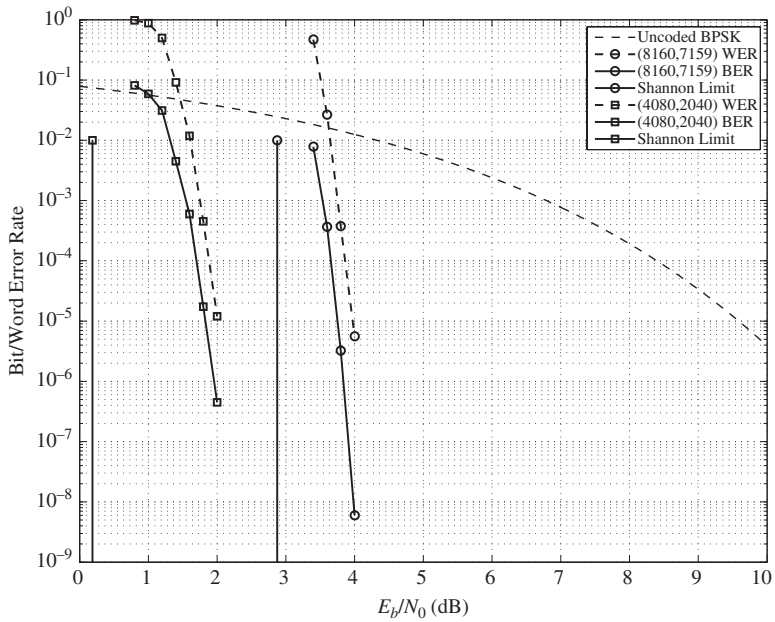


Figure 11.9 Error performances of the (8160,7159) QC-LDPC code and the (4080,2040) QC-LDPC code given in Example 11.9.

GF(2) that consists of two 8×8 circulants, $\mathbf{G}_0$ and $\mathbf{G}_1$. The two circulants in $\mathbf{Z}(8,16)$ are generated by $\mathbf{g}_0 = (01011000)$ and $\mathbf{g}_1 = (00101010)$, respectively. Then $\mathbf{Z}(8,16)$ is a (3,6)-regular masking matrix with column and row weights 3 and 6, respectively. On masking $\mathbf{H}^{(4)}_{qc,\text{disp}}$ with $\mathbf{Z}(8,16)$, we obtain an 8×16 masked array $\mathbf{M}^{(4)}(8,16) = \mathbf{Z}(8,16) \circledast \mathbf{H}^{(4)}_{qc,\text{disp}}$ that is a 2040×4080 matrix over GF(2) with column and row weights 3 and 6, respectively. The null space of $\mathbf{M}^{(4)}(8,16)$ gives a (3,6)-regular (4080,2040) QC-LDPC code with rate 1/2, whose bit and word error performances with iterative decoding using SPA (50 iterations) are also shown in Figure 11.9. We see that, at a BER of 10^{-6}, it performs 1.77 dB from the Shannon limit.

11.5.2 Construction of QC-LDPC Codes Based on Subgroups of the Multiplicative Group of a Finite Field

Again we use GF(q) for code construction. Let α be a primitive element of GF(q). Suppose $q-1$ is not prime. We can factor $q-1$ as a product of two relatively prime factors c and m such that $q-1 = cm$. Let $\beta = \alpha^c$ and $\delta = \alpha^m$. Then $\mathcal{G}_{c,1} = \{\beta^0 = 1, \beta, \ldots, \beta^{m-1}\}$ and $\mathcal{G}_{c,2} = \{\delta^0 = 1, \delta, \ldots, \delta^{c-1}\}$ form two cyclic subgroups of the multiplicative group of GF(q) and $\mathcal{G}_{c,1} \cap \mathcal{G}_{c,2} = \{1\}$. For $0 \le i < c$, the set

$$\delta^i \mathcal{G}_{c,1} = \{\delta^i, \delta^i\beta, \ldots, \delta^i\beta^{m-1}\}$$

forms a multiplicative coset of $\mathcal{G}_{c,1}$. The cyclic subgroup $\mathcal{G}_{c,1}$ has c multiplicative cosets, including itself. For $0 \leq j < m$, the set

$$\beta^j \mathcal{G}_{c,2} = \{\beta^j, \beta^j\delta, \ldots, \beta^j\delta^{c-1}\}$$

forms a multiplicative coset of $\mathcal{G}_{c,2}$. The cyclic subgroup $\mathcal{G}_{c,2}$ has m multiplicative cosets, including itself.

Form the following $c \times m$ matrix over GF(q):

$$\mathbf{W}^{(5)} = \begin{bmatrix} \mathbf{w}_0 \\ \mathbf{w}_1 \\ \vdots \\ \mathbf{w}_{c-1} \end{bmatrix} = \begin{bmatrix} \beta^0 - 1 & \beta - 1 & \cdots & \beta^{m-1} - 1 \\ \delta\beta^0 - 1 & \delta\beta - 1 & \cdots & \delta\beta^{m-1} - 1 \\ \vdots & \vdots & \ddots & \vdots \\ \delta^{c-1}\beta^0 - 1 & \delta^{c-1}\beta - 1 & \cdots & \delta^{c-1}\beta^{m-1} - 1 \end{bmatrix}, \tag{11.12}$$

where (1) the components of the ith row are obtained by subtracting the 1 element of GF(q) from each element in the ith multiplicative coset of $\mathcal{G}_{c,1}$ and (2) the components of the jth column are obtained by subtracting the 1 element from each element in the jth multiplicative coset of $\mathcal{G}_{c,2}$. $\mathbf{W}^{(5)}$ has the following structural properties: (1) there is one and only one 0 entry, which is located the upper-left corner, and all the other $cm - 1$ entries are nonzero elements in GF(q); (2) all the entries are different elements in GF(q); (3) any two rows differ in all the m positions; and (4) any two columns differ in all c positions. Given all the above structural properties, we can easily prove that $\mathbf{W}^{(5)}$ satisfies α-multiplied row constraints 1 and 2.

By dispersing each nonzero entry of $\mathbf{W}^{(5)}$ into a $(q-1) \times (q-1)$ CPM and the single 0 entry into a $(q-1) \times (q-1)$ zero matrix, we obtain the following RC-constrained $c \times m$ array of $cm - 1$ CPMs of size $(q-1) \times (q-1)$ and a single $(q-1) \times (q-1)$ zero matrix:

$$\mathbf{H}^{(5)}_{qc,\text{disp}} = \begin{bmatrix} \mathbf{O} & \mathbf{A}_{0,1} & \cdots & \mathbf{A}_{0,m-1} \\ \mathbf{A}_{1,0} & \mathbf{A}_{1,1} & \cdots & \mathbf{A}_{1,m-1} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{A}_{c-1,0} & \mathbf{A}_{c-1,1} & \cdots & \mathbf{A}_{c-1,m-1} \end{bmatrix}. \tag{11.13}$$

Since all the nonzero entries of $\mathbf{W}^{(5)}$ are different, all the CPMs in $\mathbf{H}^{(5)}_{qc,\text{disp}}$ are different. The null space of any subarray of $\mathbf{H}^{(5)}_{qc,\text{disp}}$ gives a QC-LDPC code whose Tanner graph has a girth of at least 6. The above construction gives a class of QC-LDPC codes.

Of course, masking subarrays of the arrays given by (11.11) and (11.13) will result in many more QC-LDPC codes.

Example 11.10. Let GF(2^8) be the code-construction field. We find that $2^8 - 1 = 255$ can be factored as the product of 5 and 51, which are relatively prime. Let $\beta = \alpha^5$ and $\delta = \alpha^{51}$. The sets $\mathcal{G}_{c,1} = \{\beta^0, \beta, \ldots, \beta^{50}\}$ and $\mathcal{G}_{c,2} = \{\delta^0, \delta, \delta^2, \delta^3, \delta^4\}$ form two cyclic groups of the multiplicative group of GF(2^8) and $\mathcal{G}_{c,1} \cap \mathcal{G}_{c,2} = \{1\}$. Given these two cyclic subgroups of GF(2^8), (11.12) and (11.13), we can construct a 5×51 array $\mathbf{H}^{(5)}_{qc,\text{disp}}$ of 254 CPMs of size 255×255 and a single 255×255 zero matrix. Take a 4×16 subarray $\mathbf{H}^{(5)}_{qc,\text{disp}}(4, 16)$ from $\mathbf{H}^{(5)}_{qc,\text{disp}}$, avoiding the single zero matrix. $\mathbf{H}^{(5)}_{qc,\text{disp}}(4, 16)$ is a 1020×4080 matrix with column and row weights 4 and 16, respectively. The null space of this matrix gives a (4,16)-regular (4080,3093) QC-LDPC code with rate 0.758. The error performance of this code with iterative decoding using the SPA with 50 iterations is shown in Figure 11.10. At a BER of 10^{-6}, it performs 1.3 dB from the Shannon limit.

Suppose we remove the first column of $\mathbf{H}^{(5)}_{qc,\text{disp}}$. This removal results in a 5×50 subarray $\mathbf{H}^{(5)}_{qc,\text{disp}}(5, 50)$ which is a 1275×12750 matrix with column and row weights 5 and 50, respectively. The null space of $\mathbf{H}^{(5)}_{qc,\text{disp}}(5, 50)$ gives a (5,50)-regular (12750,11553) QC-LDPC code with rate 0.9061. The error performance of this code is also shown in Figure 11.10. At a BER of 10^{-6}, it performs 0.9 dB from the Shannon limit.

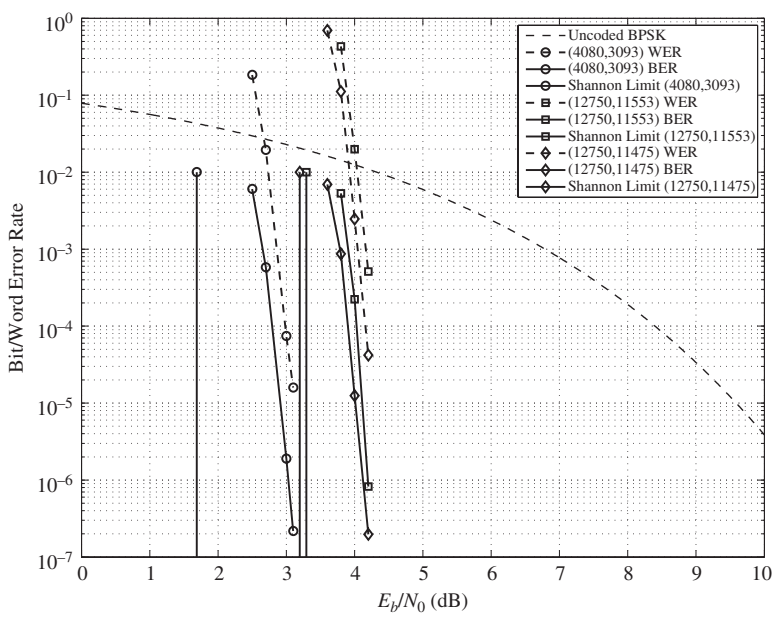


Figure 11.10 Error performances of the (4080,3093) and (12750,11553) QC-LDPC codes given in Example 11.10.

Suppose we use $\mathbf{H}^{(5)}_{qc,\text{disp}}(5,50)$ as the base array for masking. Construct a 5×50 masking matrix $\mathbf{Z}(5,50) = [\mathbf{G}_0\mathbf{G}_1 \cdots \mathbf{G}_9]$ over GF(2), where, for $0 \le i < 9$,

$$\mathbf{G}_i = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 \end{bmatrix}.$$

$\mathbf{Z}(5,50)$ has column and row weights 3 and 30, respectively. On masking $\mathbf{H}^{(5)}_{qc,\text{disp}}(5,50)$ with $\mathbf{Z}(5,50)$, we obtain a 5×50 masked array $\mathbf{M}^{(5)}(5,50), = \mathbf{Z}(5,10) \circledast \mathbf{H}^{(5)}(5,50)$, which is a 1275×12750 matrix over GF(2) with column and row weights 3 and 30, respectively. The null space of this matrix gives a (3,30)-regular (12750,11475) QC-LDPC code with rate 0.9, whose bit and word error performances with iterative decoding using the SPA (50 iterations) are also shown in Figure 11.10.

11.6 Construction of QC-LDPC Code Based on the Additive Group of a Prime Field

In the last four sections, RC-constrained arrays of CPMs were constructed by exploiting location-vectors and matrix dispersions of elements of a finite field with respect to the multiplicative group (or its cyclic subgroups) of the field. However, an RC-constrained array of CPMs can also be constructed from location-vectors and matrix dispersions of elements of a prime field with respect to its additive group.

Let p be a prime. Then the set of integers $\{0, 1, \ldots, p-1\}$ forms a field GF(p) under modulo-p addition and multiplication. Such a field is called a prime field (see Chapter 2). For each element i in GF(p), its location-vector *with respect to the additive group* of GF(p) is defined as a unit p-tuple over GF(2),

$$\mathbf{z}(i) = (z_0, z_1, \ldots, z_{p-1}), \tag{11.14}$$

whose components correspond to all the elements in GF(p), including the 0 element, where the ith component $z_i = 1$ and all the other components are set to zeros. This location-vector of element i is called the $\mathcal{A}$-*location-vector* of i, where "$\mathcal{A}$" stands for "additive." The $\mathcal{A}$-location-vector of the 0 element of GF(p) is $\mathbf{z}(0) = (1, 0, \ldots, 0)$. It is clear that the 1-components of the $\mathcal{A}$-location-vectors of two different elements in GF(p) are at two different positions. For any element k in GF(p), the $\mathcal{A}$-location-vector $\mathbf{z}(k+1)$ of the element $k+1$ is the right cyclic-shift of the $\mathcal{A}$-location-vector $\mathbf{z}(k)$ of the element k. The $\mathcal{A}$-location-vector $\mathbf{z}(0)$ of the 0 element of GF(p) is the right cyclic-shift of the $\mathcal{A}$-location-vector $\mathbf{z}(p-1)$ of the element $p-1$. For a given element k in GF(p), $k+0, k+1, k+2, \ldots, k+(p-1)$ (modulo p) are different integers and they form all p elements of GF(p). Form

a $p \times p$ matrix $\mathbf{A}$ with the the $\mathcal{A}$-location-vectors of $k+0, k+1, \ldots, k+(p-1)$ (modulo p) as rows. Then $\mathbf{A}$ is a $p \times p$ CPM and is referred to as the *additive p-fold matrix dispersion* of the element k.

Form the following $p \times p$ matrix over GF(p):

$$\mathbf{W}^{(6)} = \begin{bmatrix} \mathbf{w}_0 \\ \mathbf{w}_1 \\ \vdots \\ \mathbf{w}_i \\ \vdots \\ \mathbf{w}_{p-1} \end{bmatrix}$$

$$= \begin{bmatrix} 0\cdot 0 & 0\cdot 1 & 0\cdot 2 & \cdots & 0\cdot(p-1) \\ 1\cdot 0 & 1\cdot 1 & 1\cdot 2 & \cdots & 1\cdot(p-1) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ i\cdot 0 & i\cdot 1 & i\cdot 2 & \cdots & i\cdot(p-1) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ (p-1)\cdot 0 & (p-1)\cdot 1 & (p-1)\cdot 2 & \cdots & (p-1)\cdot(p-1) \end{bmatrix}, \quad (11.15)$$

where the multiplication of two elements in GF(p) is carried out with modulo-p multiplication. Label the rows and columns of $\mathbf{W}^{(6)}$ from 0 to $p-1$. $\mathbf{W}^{(6)}$ has the following structural properties: (1) all the components of the 0th row $\mathbf{w}_0$ are zeros: (2) all the components of any row $\mathbf{w}_i$ other than the 0th row $\mathbf{w}_0$ are different and they form all the p elements of GF(p); (3) all the elements in the 0th column are zeros; (4) all the components of any column other than the 0th column are different and they form all the p elements of GF(p); and (5) any two different rows (or two different columns) have the 0 element in common at the first position and differ in all the other $p-1$ positions. From the structural properties of $\mathbf{W}^{(6)}$, we can easily prove that the rows of $\mathbf{W}^{(6)}$ satisfy two constraints given by the following two lemmas.

Lemma 11.3. Let $\mathbf{w}_i = (i\cdot 0, i\cdot 1, \ldots, i\cdot(p-1))$ with $0 \le i < p$ be the ith row of $\mathbf{W}^{(6)}$. For two different elements, k and l, in GF(p), the two p-tuples over GF(p), $(i\cdot 0+k, i\cdot 1+k, \ldots, i\cdot(p-1)+k)$ and $(i\cdot 0+l, i\cdot 1+l, \ldots, i\cdot(p-1)+l)$ differ in all p positions.

Lemma 11.4. For $0 \le i, j < p$ and $i \ne j$, let $\mathbf{w}_i = (i\cdot 0, i\cdot 1, \ldots, i\cdot(p-1))$ and $\mathbf{w}_j = (j\cdot 0, j\cdot 1, \ldots, j\cdot(p-1))$ be two different rows of $\mathbf{W}^{(6)}$. For two elements, k and l, in GF(p) with $0 \le k, l < p$, the two p-tuples over GF(p), $(i\cdot 0+k, i\cdot 1+k, \ldots, i\cdot(p-1)+k)$ and $(j\cdot 0+l, j\cdot 1+l, \ldots, j\cdot(p-1)+l)$ differ in at least $p-1$ positions.

The two constraints on the rows of $\mathbf{W}^{(6)}$ given by Lemmas 11.3 and 11.4 are referred to as *additive row constraints* 1 and 2.

On replacing each entry in $\mathbf{W}^{(6)}$ by its additive p-fold matrix dispersion, we obtain the following $p \times p$ array of $p \times p$ CPMs over GF(2):

$$\mathbf{H}^{(6)}_{qc,\text{disp}} = \begin{bmatrix} \mathbf{A}_{0,0} & \mathbf{A}_{0,1} & \cdots & \mathbf{A}_{0,p-1} \\ \mathbf{A}_{1,0} & \mathbf{A}_{1,1} & \cdots & \mathbf{A}_{1,p-1} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{A}_{p-1,0} & \mathbf{A}_{p-1,1} & \cdots & \mathbf{A}_{p-1,p-1} \end{bmatrix}. \tag{11.16}$$

All the CPMs in the top row and the leftmost column of $\mathbf{H}^{(6)}_{qc,\text{disp}}$ are identity matrices and $\mathbf{H}^{(6)}_{qc,\text{disp}}$ contains no zero matrix. It follows from additive row constraints 1 and 2 that $\mathbf{H}^{(6)}_{qc,\text{disp}}$ is an RC-constrained array, i.e., since it is a $p^2 \times p^2$ matrix over GF(2), no two rows (or two columns) have more than one 1-component in common.

For a pair (g, r) of integers with $1 \leq g, r \leq p$, let $\mathbf{H}^{(6)}_{qc,\text{disp}}(g, r)$ be a $g \times r$ subarray of $\mathbf{H}^{(6)}_{qc,\text{disp}}$. $\mathbf{H}^{(6)}_{qc,\text{disp}}(g, r)$ is a $g(p-1) \times r(p-1)$ matrix over GF(2) with column and row weights g and r, respectively. The null space of $\mathbf{H}^{(6)}_{qc,\text{disp}}(g, r)$ gives a (g,r)-regular QC-LDPC code, whose Tanner graph has a girth of at least 6. The above construction gives a class of QC-LDPC codes.

Example 11.11. Suppose we use the additive group of the prime field GF(73) for code construction. From the additive 73-fold matrix dispersions of the elements in GF(73), (11.15) and (11.16), we can construct a 73×73 array $\mathbf{H}^{(6)}_{qc,\text{disp}}$ of 73×73 CPMs over GF(2). Choose $g = 6$ and $r = 72$. Take a 6×72 subarray $\mathbf{H}^{(6)}_{qc,\text{disp}}(6, 72)$ from $\mathbf{H}^{(6)}_{qc,\text{disp}}$. This subarray is a 438×5256 matrix with column and row weights 6 and 72, respectively. The null space of $\mathbf{H}^{(6)}_{qc,\text{disp}}(6, 72)$ gives a (6,72)-regular (5256,4823) QC-LDPC code with rate 0.9176. Since the column weight of $\mathbf{H}^{(6)}_{qc,\text{disp}}(6, 72)$ is 6, the minimum distance of the code is lower bounded by 8; however, the estimated minimum distance of the code is 20. The error performances of this code decoded with iterative decoding using the SPA with various numbers of decoding iterations are shown in Figure 11.11. At a BER of 10^{-6} with 50 decoding iterations, the code performs 1.3 dB from the Shannon limit. We also see that the decoding of this code converges very fast. At a BER of 10^{-6}, the performance gap between 5 and 50 iterations is about 0.2 dB, while the gap between 10 and 50 iterations is about 0.1 dB. The code also has a very low error floor as shown in Figure 11.12. The estimated error floor of the code is below 10^{-25} for bit-error performance and below 10^{-22} for word-error performance.

An FPGA decoder for this code has been built. Using this decoder and 15 iterations of the SPA, the performance of this code can be simulated down to a BER of 10^{-12} and a WER of almost 10^{-10} as shown in Figure 11.13. From Figure 11.11 and 11.13, we see that, to achieve a BER of 10^{-7}, both 15 and 50 iterations of the SPA require 5 dB.

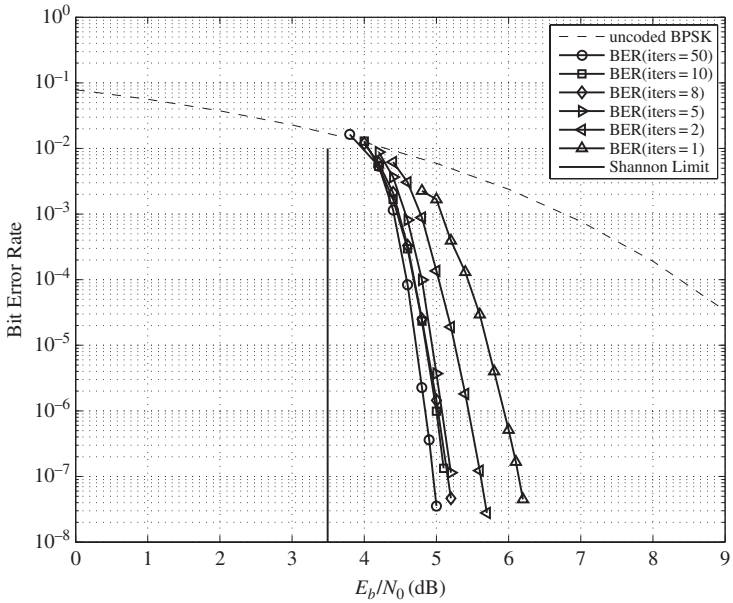


Figure 11.11 The error performance of the (5256,4823) QC-LDPC code given in Example 11.11.

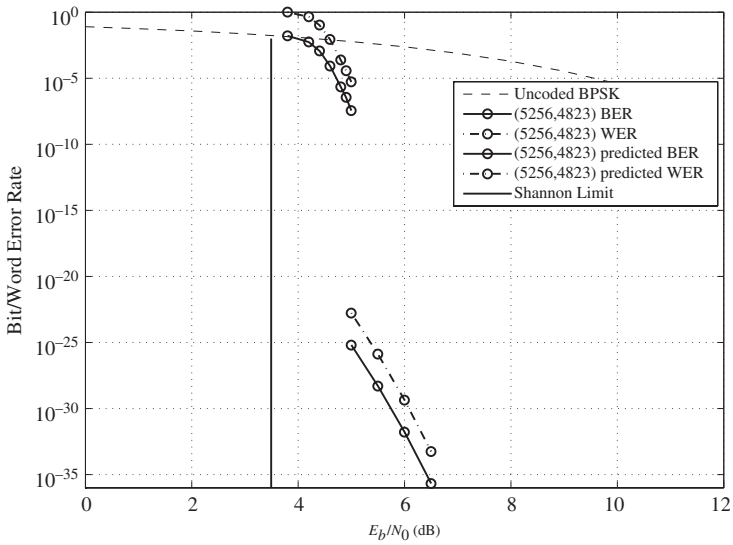


Figure 11.12 The estimated error floor of the (5256,4823) QC-LDPC code given in Example 11.11.

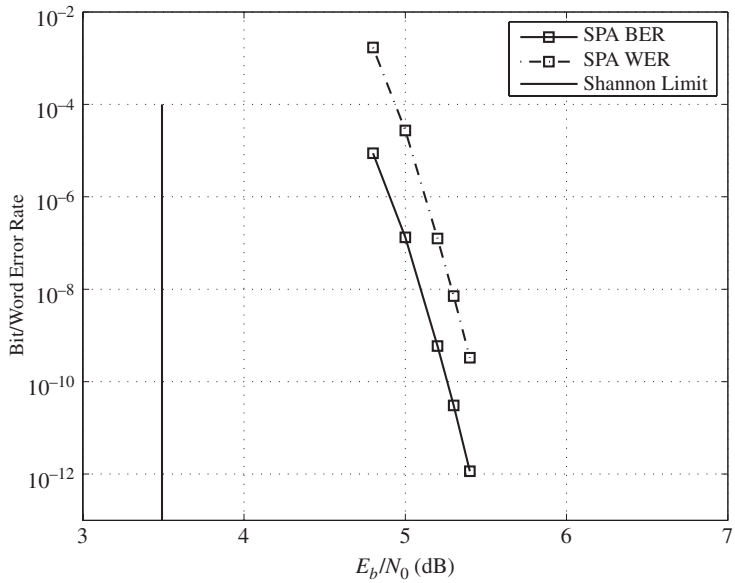


Figure 11.13 Bit-error and word-error performances of the (5256,4823) QC-LDPC code given in Example 11.11 simulated by an FPGA decoder using 15 iterations of the SPA.

Example 11.12. Suppose we take 32×64 subarray $\mathbf{H}^{(6)}(32, 64)$ from the 73×73 array $\mathbf{H}^{(6)}_{qc,\text{disp}}$ constructed on the basis of GF(73) given in Example 11.11. We use this subarray as a base array for masking. Construct a 32×64 masking matrix $\mathbf{Z}(32, 64) = [\mathbf{G}_0 \mathbf{G}_1]$ that consists of two 32×32 circulants, $\mathbf{G}_0$ and $\mathbf{G}_1$, whose generators (top rows) are

$$\mathbf{g}_0 = (00000000010000000101000000010000)$$

and

$$\mathbf{g}_1 = (00000000010100100000000000001000).$$

On masking $\mathbf{H}^{(6)}(32, 64)$ by $\mathbf{Z}(32, 64)$, we obtain a masked array $\mathbf{M}^{(6)}(32, 64) = \mathbf{Z}(32, 64) \circledast \mathbf{H}^{(6)}(32, 64)$, which is a 2336×4672 matrix over GF(2) with column and row weights 4 and 8, respectively. The null space of this matrix gives a (4,8)-regular (4672,2339) QC-LDPC code with rate 0.501. The bit- and word-error performances of this code are shown in Figure 11.14. At the BER of 10^{-6}, it performs 2.05 dB from the Shannon limit.

11.7 Construction of QC-LDPC Codes Based on Primitive Elements of a Field

This section gives a method of constructing QC-LDPC codes based on primitive elements of a finite field. Consider the Galois field GF(q), where q is a power of a prime. The number of primitive elements in GF(q) can be enumerated with Euler's

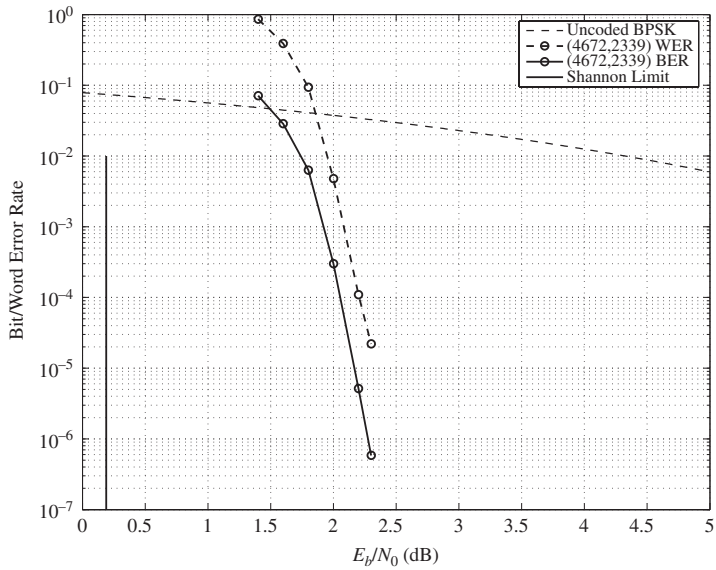


Figure 11.14 Error performances of the (4672,2339) QC-LDPC codes given in Example 11.12.

formula as given by (2.42). First we factor $q-1$ as a product of powers of primes,

$$q-1 = p_1^{k_1} p_2^{k_2} \cdots p_t^{k_t},$$

where p_i is a prime with $1 \le i \le t$. Then the number of primitive elements in GF(q) is given by

$$K = (q-1)\prod_{i=1}^{t}(1-1/p_i).$$

Let $\{\alpha^{j_1}, \alpha^{j_2}, \ldots, \alpha^{j_K}\}$ be the set of K primitive elements of GF(q). Let $j_0 = 0$. Form the following $(K+1)\times(K+1)$ matrix over GF(q):

$$\mathbf{W}^{(7)} = \begin{bmatrix} \mathbf{w}_0 \\ \mathbf{w}_1 \\ \vdots \\ \mathbf{w}_K \end{bmatrix} = \begin{bmatrix} \alpha^{j_0-j_0}-1 & \alpha^{j_1-j_0}-1 & \cdots & \alpha^{j_K-j_0}-1 \\ \alpha^{j_0-j_1}-1 & \alpha^{j_1-j_1}-1 & \cdots & \alpha^{j_K-j_1}-1 \\ \vdots & \vdots & \ddots & \vdots \\ \alpha^{j_0-j_K}-1 & \alpha^{j_1-j_K}-1 & \cdots & \alpha^{j_K-j_K}-1 \end{bmatrix}. \tag{11.17}$$

From (11.17), we can readily see that matrix $\mathbf{W}^{(7)}$ has the following structural properties: (1) all the entries of a row (or a column) are distinct elements in GF(q); (2) each row (each column) contains one and only one zero element; (3) any two rows (or two columns) differ in every position; and (4) the $K+1$ zero elements lie on the main diagonal of the matrix.

Lemma 11.5. The matrix $\mathbf{W}^{(7)}$ satisfies α-multiplied row constraints 1 and 2.

Proof. The proof of this lemma is left as an exercise. □

Since $\mathbf{W}^{(7)}$ satisfies α-multiplied row constraints 1 and 2, it can be used as a base matrix for array dispersion for constructing QC-LDPC codes. If we replace each entry by its multiplicative $(q-1)$-fold matrix dispersion, we obtain the following RC-constrained $(K+1)\times(K+1)$ array $\mathbf{H}^{(7)}_{qc,\text{disp}}$ of circulant permutation and zero matrices, with the zero matrices on the main diagonal of the array:

$$\mathbf{H}^{(7)}_{qc,\text{disp}} = \begin{bmatrix} \mathbf{A}_{0,0} & \mathbf{A}_{0,1} & \cdots & \mathbf{A}_{0,K} \\ \mathbf{A}_{1,0} & \mathbf{A}_{1,1} & \cdots & \mathbf{A}_{1,K} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{A}_{K,0} & \mathbf{A}_{K,1} & \cdots & \mathbf{A}_{K,K} \end{bmatrix}. \tag{11.18}$$

For any pair $(g,\, r)$ of integers, with $1 \le g, r \le K$, let $\mathbf{H}^{(7)}_{qc,\text{disp}}(g,r)$ be a $g \times r$ subarray of $\mathbf{H}^{(7)}_{qc,\text{disp}}$. The null space of $\mathbf{H}^{(7)}_{qc,\text{disp}}(g,r)$ gives a QC-LDPC code of length $r(q-1)$ whose Tanner graph is free of cycles of length 4. The construction based on the subarrays of $\mathbf{H}^{(7)}_{qc,\text{disp}}$ gives a family of QC-LDPC codes. Of course, masking subarrays of $\mathbf{H}^{(7)}_{qc,\text{disp}}$ gives more QC-LDPC codes.

Example 11.13. Let $\text{GF}(2^6)$ be the code-construction field. Note that $2^6 - 1 = 63$ can be factored as $3^2 \times 7$. Using Euler's formula, we find that $\text{GF}(2^6)$ has $K = 63 \times (1 - 1/3)(1 - 1/7) = 36$ primitive elements. Using (11.17) and (11.18), we can construct a 37×37 array $\mathbf{H}^{(7)}_{qc,\text{disp}}$ of 63×63 circulant permutation and zero matrices. The zero matrices lie on the main diagonal of $\mathbf{H}^{(7)}_{qc,\text{disp}}$. Choose $g = 6$ and $r = 37$. Take the first six rows of $\mathbf{H}^{(7)}_{qc,\text{disp}}$ to form a 6×37 subarray $\mathbf{H}^{(7)}_{qc,\text{disp}}(6, 37)$ which is a 378×2331 matrix over GF(2) with row weight 36 and two column weights, 5 and 6. Its null space gives a (2331,2007) QC-LDPC code with rate 0.861. The performance of this code decoded using the SPA with 50 iterations is shown in Figure 11.15. At a BER of 10^{-6}, it performs 1.5 dB from the Shannon limit. The error floor of this code is estimated to lie below the bit-error performance of 10^{-16} and below the block-error performance of 10^{-14} as shown in Figure 11.16. The decoding of this code also converges fast, as shown in Figure 11.17. At the BER of 10^{-6}, the performance gap between 5 and 50 iterations is less than 0.4 dB.

11.8 Construction of QC-LDPC Codes Based on the Intersecting Bundles of Lines of Euclidean Geometries

Consider the m-dimensional Euclidean geometry EG(m,q) over GF(q). Let α be a primitive element of GF(q^m). Represent the q^m points of EG(m,q) by the q^m elements of GF(q^m), $\alpha^{-\infty} = 0$, $\alpha^0 = 1$, α, …, α^{q^m-2}. Consider the subgeometry EG*(m,q) obtained by removing the origin $\alpha^{-\infty}$ and all the lines passing through the origin in EG(m,q). Then all the nonzero elements of GF(q^m) represent the non-origin points of EG*(m,q). Consider the bundle of $K = q(q^{m-1} - 1)/(q-1)$ lines that intersect at the point α^0. This bundle of lines is called the *intersecting bundle* of lines at the point a^0, denoted $\mathbf{I}(\alpha^0)$. Denote the lines in $\mathbf{I}(\alpha^0)$ by $\mathcal{L}_0$,

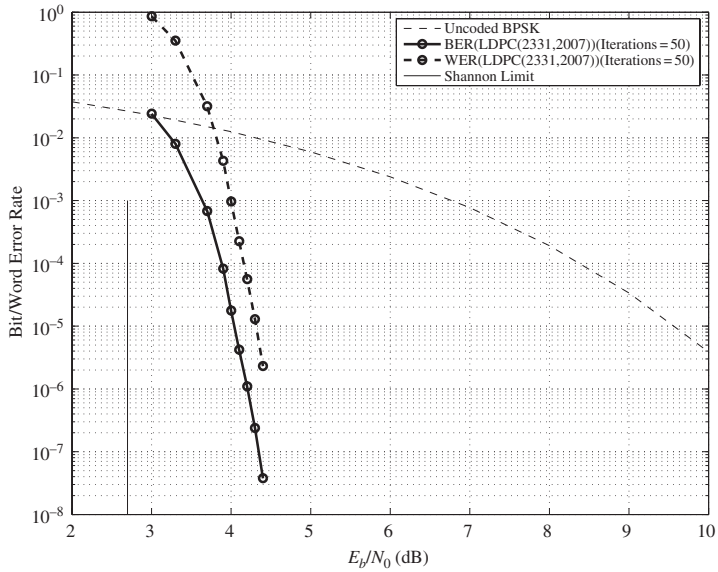


Figure 11.15 The error performance of the (2331,2007) QC-LDPC code given in Example 11.13.

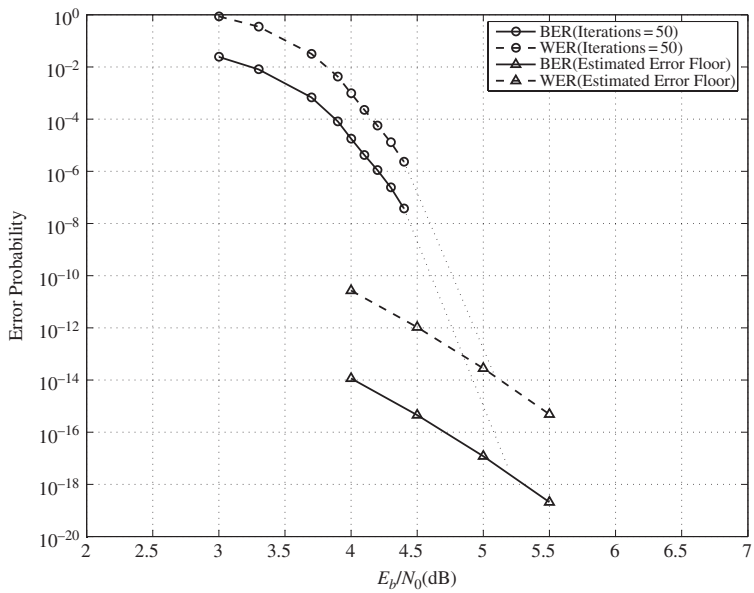


Figure 11.16 The estimated error floor of the (2331,2007) QC-LDPC code given in Example 11.13.

$\mathcal{L}_1, \ldots, \mathcal{L}_{K-1}$. Let β be a primitive element of GF(q). For $0 \leq i < K$, the line $\mathcal{L}_i$ consists of q points of the following form:

$$\mathcal{L}_i = \{\alpha^0, \alpha^0 + \beta^0\alpha^{j_i}, \alpha^0 + \beta\alpha^{j_i}, \ldots, \alpha^0 + \beta^{q-2}\alpha^{j_i}\}, \tag{11.19}$$

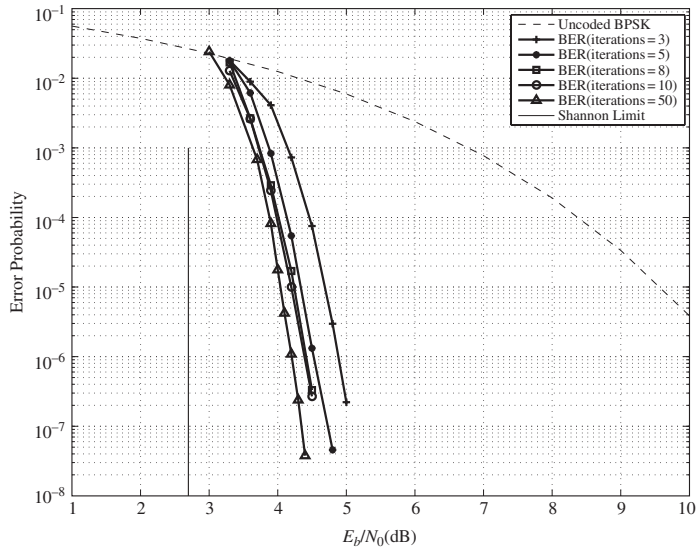


Figure 11.17 The rate of decoding convergence of the (2331,2007) QC-LDPC code given in Example 11.13.

where (1) the point α^{j_i} is linearly independent of the point α^0 and (2) the points $\alpha^{j_0}, \alpha^{j_1}, \ldots, \alpha^{j_{K-1}}$ lie on separate lines, i.e., for $k \neq i$, $\alpha^{j_k} \neq \alpha^0 + \beta^l \alpha^{j_i}$ with $0 \leq l < q-1$. For $0 \leq t < q^m - 1$ and $0 \leq i < K$, $\alpha^t \mathcal{L}_i$ is a line passing through the point α^t. Then the K lines $\alpha^t \mathcal{L}_0, \alpha^t \mathcal{L}_1, \ldots, \alpha^t \mathcal{L}_{K-1}$ form an intersecting bundle of lines at point α^t, denoted $\mathbf{I}(\alpha^t)$.

From the intersecting bundle of lines at point α^0, we form the following $K \times q$ matrix over $\mathrm{GF}(q^m)$ such that the q entries of the ith row are the q points on the ith line $\mathcal{L}_i$ of $\mathbf{I}(\alpha^0)$:

$$\mathbf{W}^{(8)}_{qc,\mathrm{disp}} = \begin{bmatrix} \mathcal{L}_0 \\ \mathcal{L}_1 \\ \vdots \\ \mathcal{L}_{K-1} \end{bmatrix} = \begin{bmatrix} \alpha^0 & \alpha^0 + \beta^0 \alpha^{j_0} & \alpha^0 + \beta \alpha^{j_0} & \cdots & \alpha^0 + \beta^{q-2} \alpha^{j_0} \\ \alpha^0 & \alpha^0 + \beta^0 \alpha^{j_1} & \alpha^0 + \beta \alpha^{j_1} & \cdots & \alpha^0 + \beta^{q-2} \alpha^{j_1} \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ \alpha^0 & \alpha^0 + \beta^0 \alpha^{j_{K-1}} & \alpha^0 + \beta \alpha^{j_{K-1}} & \cdots & \alpha^0 + \beta^{q-2} \alpha^{j_{K-1}} \end{bmatrix}. \tag{11.20}$$

Label the rows and columns of $\mathbf{W}^{(8)}$ from 0 to $K-1$ and 0 to $q-2$, respectively. $\mathbf{W}^{(8)}$ has the following structural properties: (1) all the entries of each row of $\mathbf{W}^{(8)}$ are nonzero elements of $\mathrm{GF}(q^m)$; (2) except for the 0th column, all the entries in each column are distinct nonzero elements in $\mathrm{GF}(q^m)$; (3) any two rows have identical entries at the 0th position and differ in all the other $q-1$ positions; (4) any two columns differ in all the K positions; and (5) for $0 \leq i, j < K$, $i \neq j$, $0 \leq k$, and $l < q^m - 1$, $\alpha^k \mathcal{L}_i$ and $\alpha^l \mathcal{L}_j$ are different lines and they differ in at least $q-1$ positions. Properties (1) and (5) imply that $\mathbf{W}^{(8)}$ satisfies α-multiplied row constraints 1 and 2. Hence, $\mathbf{W}^{(8)}$ can be used as the base matrix for array dispersion to construct an RC-constrained array of CPMs.

By dispersing each entry of $\mathbf{W}^{(8)}$ into a $(q^m-1)\times(q^m-1)$ CPM over GF(2), we obtain an RC-constrained $K\times q$ array $\mathbf{H}^{(8)}_{qc,\mathrm{EG},1}$ of $(q^m-1)\times(q^m-1)$ CPMs. It is a $K(q^m-1)\times q(q^m-1)$ matrix over GF(2) with column and row weights K and q, respectively. For $m>2$, K is much larger than q. In this case, there are more rows than columns in $\mathbf{H}^{(8)}_{qc,\mathrm{EG},1}$. Let $\mathbf{H}^{(8)}_{qc,\mathrm{EG},2}$ be the transpose of $\mathbf{H}^{(8)}_{qc,\mathrm{EG},1}$, i.e.,

$$\mathbf{H}^{(8)}_{qc,\mathrm{EG},2}=[\mathbf{H}^{(8)}_{qc,\mathrm{EG},1}]^{\mathrm{T}}. \tag{11.21}$$

Then $\mathbf{H}^{(8)}_{qc,\mathrm{EG},2}$ is an RC-constrained $q\times K$ array of $(q^m-1)\times(q^m-1)$ CPMs over GF(2). It is a $q(q^m-1)\times K(q^m-1)$ matrix over GF(2) with column and row weights q and K, respectively. For $m>2$, $\mathbf{H}^{(8)}_{qc,\mathrm{EG},2}$ has more columns than rows. Both arrays, $\mathbf{H}^{(8)}_{qc,\mathrm{EG},1}$ and $\mathbf{H}^{(8)}_{qc,\mathrm{EG},2}$, can be used for constructing QC-LDPC codes.

For $1\le g\le K$ and $1\le r\le q$, let $\mathbf{H}^{(8)}_{qc,\mathrm{EG},1}(g,r)$ be a $g\times r$ subarray of $\mathbf{H}^{(8)}_{qc,\mathrm{EG},1}$. The null space of $\mathbf{H}^{(8)}_{qc,\mathrm{EG},1}(g,r)$ gives a binary QC-LDPC code $\mathcal{C}^{(8)}_{qc,\mathrm{EG},1}$, whose Tanner graph has a girth of at least 6. For $1\le g\le q$ and $1\le r\le K$, let $\mathbf{H}^{(8)}_{qc,\mathrm{EG},2}(g,r)$ be a $g\times r$ subarray of $\mathbf{H}^{(8)}_{qc,\mathrm{EG},2}$. Then the null space of $\mathbf{H}^{(8)}_{qc,\mathrm{EG},2}(g,r)$ gives a QC-LDPC code $\mathcal{C}^{(8)}_{qc,\mathrm{EG},2}$ whose Tanner graph has a girth of at least 6. For $m>2$, using $\mathbf{H}^{(8)}_{qc,\mathrm{EG},2}$ allows us to construct longer and higher-rate codes. The above construction results in another class of Euclidean-geometry QC-LDPC codes.

Example 11.14. From the three-dimensional Euclidean EG(3,2^3) over GF(2^3), (11.19)–(11.21), we can construct an 8×72 array $\mathbf{H}^{(8)}_{qc,\mathrm{EG},2}$ of 511×511 CPMs. For the pairs of integers (3, 6), (4, 16), (4, 20) and (4, 32), we take the corresponding subarrays from $\mathbf{H}^{(8)}_{qc,\mathrm{EG},2}$, $\mathbf{H}^{(8)}_{qc,\mathrm{EG},2}(3,6)$, $\mathbf{H}^{(8)}_{qc,\mathrm{EG},2}(4,16)$, $\mathbf{H}^{(8)}_{qc,\mathrm{EG},2}(4,20)$, and $\mathbf{H}^{(8)}_{qc,\mathrm{EG},2}(4,32)$. These subarrays are 1533×3066, 2044×8176, 2044×10220, and 2044×16352 matrices with column- and row-weight pairs (3, 6), (4, 16), (4, 20), and (4, 32). The null spaces of these four matrices give (3066,1544), (8176,6162), (10220,8206), and (16352,14338) EG-QC-LDPC codes with rates 0.5036, 0.7537, 0.8029, and 0.8768, respectively. The performances of these codes over the binary-input AWGN channel decoded with the SPA using 100 iterations are shown in Figure 11.18. Consider the (8176,6162) code. At a BER of 10^{-6}, it performs 1.2 dB from the Shannon limit for rate 0.7537. For the (16352,14338) code, it performs 0.8 dB from the Shannon limit for rate 0.8768 at a BER of 10^{-6}.

In forming the base matrix $\mathbf{W}^{(8)}$, we use the bundle of lines intersecting at the point α^0. However, we can use the bundle of lines intersecting at any point α^j in EG(m,q). Of course, we can mask subarrays of $\mathbf{H}^{(8)}_{qc,\mathrm{EG},1}$ and $\mathbf{H}^{(8)}_{qc,\mathrm{EG},2}$ to construct both regular and irregular QC-LDPC codes. We can also construct RC-constrained arrays of CPMs and QC-LDPC codes based on bundles of intersecting lines in projective geometries in a similar way.

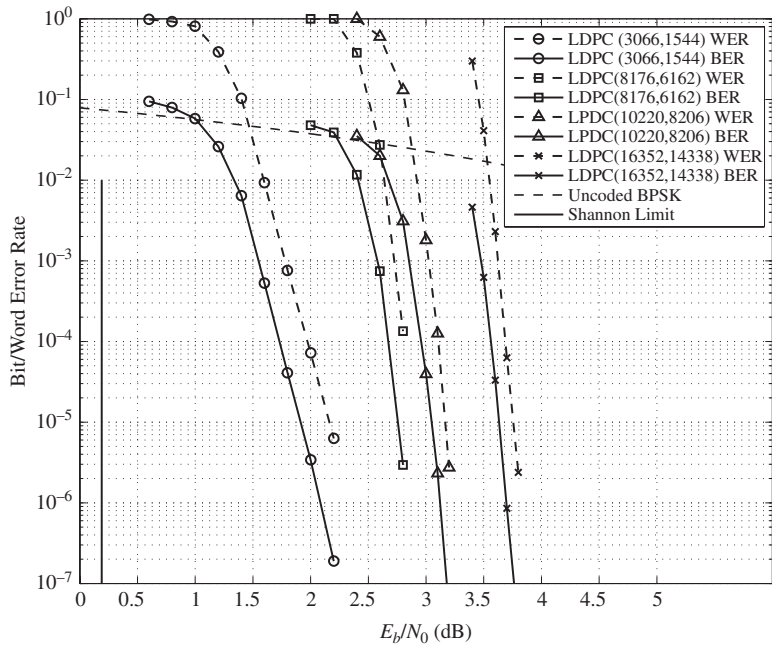


Figure 11.18 Error performances of the QC-LDPC codes given in Example 11.14.

11.9 A Class of Structured RS-Based LDPC Codes

Section 11.6 presented a method for constructing QC-LDPC codes based on the additive group of a prime field using additive matrix dispersions of the field elements. This method can be generalized to construct structured LDPC codes from any field. However, the codes constructed are not quasi-cyclic unless the field is a prime field. In this section, we apply the method presented in Section 11.6 to construct a class of LDPC codes based on extended RS codes with two information symbols. The codes constructed perform very well over the binary-input AWGN channel with iterative decoding using the SPA. One such code has been chosen as the IEEE 802.3 standard code for the 10G BASE-T Ethernet.

Consider the Galois field GF(q), where q is a power of a prime p. Let α be a primitive element of GF(q). Then, $\alpha^{-\infty} = 0$, $\alpha^0 = 1$, α , α^2, ..., α^{q-2} give all the elements of GF(q) and they form the additive group of GF(q) under the addition operation of GF(q). For each element α^i with $i = -\infty, 0, 1, \ldots, q-2$, define a q-tuple over GF(2),

$$\mathbf{z}(\alpha^i) = (z_{-\infty}, z_0, z_1, \ldots, z_{q-2}), \tag{11.22}$$

whose components correspond to the q elements $\alpha^{-\infty} = 0$, $\alpha^0 = 1$, α, α^2, ..., α^{q-2} of GF(q), where the ith component $z_i = 1$ and all the other $q-1$ components are set to zero. This q-tuple $\mathbf{z}(\alpha^i)$ over GF(2) is called the location-vector of α^i with respect to the additive group of GF(q). For simplicity, we call $\mathbf{z}(\alpha^i)$ the

A-location-vector of α^i (note that this is just a generalization of the $\mathcal{A}$-location-vector of an element of a prime field GF(p) defined in Section 11.6). For $i = -\infty$, the A-location-vector of $\alpha^{-\infty}$ is $\mathbf{z}(\alpha^{-\infty}) = (1, 0, 0, \ldots, 0)$.

For any element δ in GF(q), the sums $\delta + \alpha^{-\infty}, \delta + \alpha^0, \delta + \alpha, \ldots, \delta + \alpha^{q-2}$ are distinct and they give all the q elements of GF(q). Form the following $q \times q$ matrix over GF(2) with the A-location-vectors of $\delta + \alpha^{-\infty}, \delta + \alpha^0, \delta + \alpha, \ldots, \delta + \alpha^{q-2}$ as the rows:

$$A = \begin{bmatrix} \mathbf{z}(\delta + \alpha^{-\infty}) \\ \mathbf{z}(\delta + \alpha^0) \\ \mathbf{z}(\delta + \alpha) \\ \vdots \\ \mathbf{z}(\delta + \alpha^{q-2}) \end{bmatrix}. \tag{11.23}$$

Then A is a $q \times q$ permutation matrix (PM). This PM is called the q-fold matrix dispersion of δ with respect to the additive group of GF(q). The q-fold matrix dispersion of the 0 element of GF(q) is simply a $q \times q$ identity matrix.

Consider the $(q-1,2,q-2)$ cyclic RS code $\mathcal{C}$ over GF(q). As shown in Section 11.3, the two $(q-1)$-tuples over GF(q)

$$\mathbf{b} = (1, \alpha, \alpha^2, \ldots, \alpha^{q-2}) \tag{11.24}$$

and

$$\mathbf{c} = (1, 1, \ldots, 1) \tag{11.25}$$

are two codewords in $\mathcal{C}$ with weight $q-1$. Note that the $q-1$ components of $\mathbf{b}$ are the $q-1$ nonzero elements of GF(q). For $0 \le i < q-1$, the $(q-1)$-tuple

$$\alpha^i \mathbf{b} = (\alpha^i, \alpha^{i+1}, \ldots, \alpha^{q-2+i}) \tag{11.26}$$

is also a codeword in $\mathcal{C}$, where the power of each component in $\alpha^i\mathbf{b}$ is reduced by modulo-$(q-1)$. The codeword $\alpha^i\mathbf{b}$ can be obtained by cyclically shifting every component of $\mathbf{b}$ i places to the left. It is clear that, for $i = 0, \alpha^0\mathbf{b} = \mathbf{b}$. It is also clear that the $q-1$ components of $\alpha^i\mathbf{b}$ are still distinct and that they give the $q-1$ nonzero elements of GF(q). If we extend the codeword $\alpha^i\mathbf{b}$ by adding an overall parity-check symbol to its left, we obtain a codeword in the extended $(q,2,q-1)$ RS code $\mathcal{C}_e$ over GF(q) (see Section 3.4). Since the sum of the $q-1$ nonzero elements of GF(q) is equal to the 0 element of GF(q), the overall parity-check symbol of $\alpha^i\mathbf{b}$ is 0. Hence, for $0 \le i < q-1$,

$$\mathbf{w}_i = (0, \alpha^i\mathbf{b}) = (0, \alpha^i, \alpha^{i+1}, \ldots, \alpha^{q-2+i}) \tag{11.27}$$

is a codeword in the extended $(q,2,q-1)$ RS code over GF(q) with weight $q-1$. Let $\mathbf{w}_{-\infty} = (0, 0, \ldots, 0)$ be the all-zero q-tuple over GF(q). This all-zero q-tuple is the zero codeword of $\mathcal{C}_e$.

Since $q = 0$ modulo-q, the overall parity-check symbol of the codeword $\mathbf{c} = (1, 1, \ldots, 1)$ in $\mathcal{C}$ is 1. On extending $\mathbf{c}$ by adding its overall parity-check symbol, we obtain a q-tuple $\mathbf{c}_e = (1, 1, \ldots, 1)$, which consists of q 1-components. This q-tuple

$\mathbf{c}_e$ is a codeword of weight q in the extended $(q,2,q-1)$ RS code $\mathcal{C}_e$ over GF(q). For $k = -\infty, 0, 1, \ldots, q-2$,

$$\alpha^k \mathbf{c}_e = (\alpha^k, \alpha^k, \ldots, \alpha^k) \tag{11.28}$$

is also a codeword in the extended $(q,2,q-1)$ RS code $\mathcal{C}_e$ over GF(q). For $k = -\infty$, $\alpha^{-\infty}\mathbf{c}_e = (0, 0, \ldots, 0)$ is the zero codeword in $\mathcal{C}_e$. Note that $\mathbf{w}_{-\infty} = \alpha^{-\infty}\mathbf{c}_e$. For $k \neq -\infty$, $\alpha^k \mathbf{c}_e$ is a codeword in $\mathcal{C}_e$ with weight q. For $i, k = -\infty, 0, 1, \ldots, q-2$, the sum

$$\mathbf{w}_i + \alpha^k \mathbf{c}_e \tag{11.29}$$

is a codeword in $\mathcal{C}_e$. From the structures of $\mathbf{w}_i$ and $\alpha^k \mathbf{c}_e$ given by (11.27) and (11.28), we readily see that, for $i, k \neq -\infty$, the weight of the codeword $\mathbf{w}_i + \alpha^k \mathbf{c}_e$ is $q-1$. From the above analysis, we see that the extended $(q,2,q-1)$ RS code $\mathcal{C}_e$ has $q-1$ codewords with weight q, $(q-1)q$ codewords with weight $q-1$, and one codeword with weight zero.

Form a $q \times q$ matrix over GF(q) with the codewords $\mathbf{w}_{-\infty}, \mathbf{w}_0, \mathbf{w}_1, \ldots, \mathbf{w}_{q-2}$ of $\mathcal{C}_e$ as the rows:

$$\mathbf{W}^{(9)} = \begin{bmatrix} \mathbf{w}_{-\infty} \\ \mathbf{w}_0 \\ \mathbf{w}_1 \\ \vdots \\ \mathbf{w}_{q-2} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & \cdots & 0 \\ 0 & 1 & \alpha & \cdots & \alpha^{(q-2)} \\ 0 & \alpha & \alpha^2 & \cdots & \alpha^{(q-1)} \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & \alpha^{(q-2)} & \alpha^{(q-1)} & \cdots & \alpha^{2(q-2)} \end{bmatrix}, \tag{11.30}$$

where the power of each nonzero entry is reduced by modulo-$(q-1)$. We readily see that $[\mathbf{W}^{(9)}]^{\mathrm{T}} = \mathbf{W}^{(9)}$. The matrix $\mathbf{W}^{(9)}$ has the following structural properties: (1) except for the first row, each row consists of the q elements of GF(q); (2) except for the first column, every column consists of the q elements of GF(q); (3) any two rows differ in exactly $q-1$ positions; and (4) any two columns differ in exactly $q-1$ positions. Given the facts that the minimum distance of the extended a $(q,2,q-1)$ RS code $\mathcal{C}_e$ is $q-1$ and $\mathbf{w}_i + \alpha^k \mathbf{c}_e$ for $i, k = -\infty, 0, 1, \ldots, q-2$, is a codeword in $\mathcal{C}_e$, we can easily see that the following two lemmas hold.

Lemma 11.6. For any row $\mathbf{w}_i$ in $\mathbf{W}^{(9)}$, $\mathbf{w}_i + \alpha^k \mathbf{c}_e \neq \mathbf{w}_i + \alpha^l \mathbf{c}_e$ for $k \neq l$.

Lemma 11.7. For two different rows $\mathbf{w}_i$ and $\mathbf{w}_j$ of $\mathbf{W}^{(9)}$, $\mathbf{w}_i + \alpha^k \mathbf{c}_e \neq \mathbf{w}_j + \alpha^l \mathbf{c}_e$ differ in at least $q-1$ positions.

Lemmas 11.6 and 11.7 are simply generalizations of Lemmas 11.3 and 11.4 from a prime field GF(p) to a field GF(q), where q is a power of p. The conditions on the rows of $\mathbf{W}^{(9)}$ given by Lemmas 11.6 and 11.7 are referred to as additive row constraints 1 and 2.

On replacing each entry of $\mathbf{W}^{(9)}$ by its additive q-fold matrix dispersion as given by (11.23), we obtain the following $q \times q$ array of PMs over GF(2):

$$\mathbf{H}^{(9)}_{rs,\text{disp}} = \begin{bmatrix} \mathbf{A}_{-\infty,-\infty} & \mathbf{A}_{-\infty,0} & \mathbf{A}_{-\infty,1} & \cdots & \mathbf{A}_{-\infty,q-2} \\ \mathbf{A}_{0,-\infty} & \mathbf{A}_{0,0} & \mathbf{A}_{0,1} & \cdots & \mathbf{A}_{0,q-2} \\ \mathbf{A}_{1,-\infty} & \mathbf{A}_{1,0} & \mathbf{A}_{1,1} & \cdots & \mathbf{A}_{0,q-2} \\ \vdots & \vdots & \vdots & & \vdots \\ \mathbf{A}_{q-2,-\infty} & \mathbf{A}_{q-2,0} & \mathbf{A}_{q-2,1} & \cdots & \mathbf{A}_{q-2,q-2} \end{bmatrix}, \tag{11.31}$$

where the PMs in the top row and leftmost column are $q \times q$ identity matrices. For $0 \le i < q-1$, since the q entries in the ith row $\mathbf{w}_i$ of $\mathbf{W}^{(9)}$ are q different elements of GF(q), the q PMs in the ith row of the array $\mathbf{H}^{(9)}_{rs,\text{disp}}$ are distinct. Also, for $j \neq -\infty$, the q PMs in the jth column of $\mathbf{H}^{(9)}_{rs,disp}$ are distinct. Since any two rows (or two columns) of $\mathbf{W}^{(9)}$ differ in $q-1$ places, any two rows (or two columns) of $\mathbf{H}^{(9)}_{rs,\text{disp}}$ have one and only one position where they have identical PMs. $\mathbf{H}^{(9)}_{rs,disp}$ is called the additive q-fold array dispersion of $\mathbf{W}^{(9)}$.

Array $\mathbf{H}^{(9)}_{rs,\text{disp}}$ is a $q^2 \times q^2$ matrix over GF(2) with both column and row weights q. It follows from additive row constraints 1 and 2 on the base matrix $\mathbf{W}^{(9)}$ that $\mathbf{H}^{(9)}_{rs,\text{disp}}$, being a $q^2 \times q^2$ matrix over GF(2), satisfies the RC-constraint. The ith row $(\mathbf{A}_{i,-\infty}\ \ \mathbf{A}_{i,0}\ \ \mathbf{A}_{i,1}\ \ \ldots\ \ \mathbf{A}_{i,q-2})$ of $\mathbf{H}^{(9)}_{rs,\text{disp}}$ is a $q \times q^2$ matrix GF(2), which is obtained by replacing each entry of the following $q \times q$ matrix over GF(q) by its A-location-vector:

$$\mathbf{R}_i = \begin{bmatrix} \mathbf{w}_i + \alpha^{-\infty}\mathbf{c}_e \\ \mathbf{w}_i + \alpha^{0}\mathbf{c}_e \\ \mathbf{w}_i + \alpha^{1}\mathbf{c}_e \\ \vdots \\ \mathbf{w}_i + \alpha^{q-2}\mathbf{c}_e \end{bmatrix}. \tag{11.32}$$

Since $\alpha^{-\infty}\mathbf{c}_e$, $\alpha^0\mathbf{c}_e$, $\alpha^1\mathbf{c}_e, \ldots, \alpha^{q-2}\mathbf{c}_e$ form a $(q,1,q)$ subcode $\mathcal{C}_{e,\text{sub}}$ of the extended $(q,2,q-1)$ RS code $\mathcal{C}_e$, the rows of $\mathbf{R}_i$ are simply a coset of $\mathcal{C}_{e,\text{sub}}$ in $\mathcal{C}_e$. $\mathbf{H}^{(9)}_{rs,\text{disp}}$ is the same array of PMs as was constructed in [6].

For any pair (g,r) of integers with $1 \le g, r \le q$, take a $g \times r$ subarray $\mathbf{H}^{(9)}_{rs,\text{disp}}(g,r)$ from $\mathbf{H}^{(9)}_{rs,\text{disp}}$. $\mathbf{H}^{(9)}_{rs,\text{disp}}(g,r)$ is a $gq \times rq$ matrix over GF(2) with column and row weights g and r, respectively. Since $\mathbf{H}^{(9)}_{rs,\text{disp}}$ satisfies the RC-constraint, so does $\mathbf{H}^{(9)}_{rs,\text{disp}}(g,r)$. The null space of $\mathbf{H}^{(9)}_{rs,\text{disp}}(g,r)$ gives a (g,r)-regular LDPC code $\mathcal{C}^{(9)}_{rs,\text{disp}}$ whose Tanner graph has a girth of at least 6. The minimum distance of this code is at least $g+2$ for even g and $g+1$ for odd g. For a given finite field, the above construction gives a family of structured LDPC codes. Of course, subarrays of $\mathbf{H}^{(9)}_{rs,\text{disp}}$ can be masked to construct many more LDPC codes, regular or irregular. If q is a power of 2, LDPC codes with lengths that are powers of 2 or multiples of an 8-bit byte can be constructed. In many

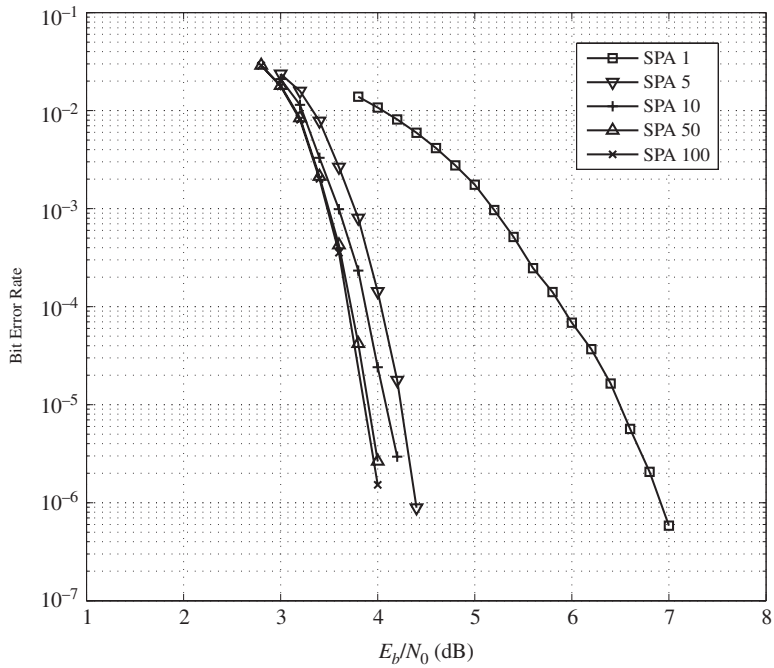


Figure 11.19 The rate of decoding convergence of the (2048,1723) LDPC code given in Example 11.15.

practical applications in communication and storage systems, codes with lengths that are powers of 2 or multiples of an 8-bit byte are preferred due to the data-frame requirements.

Example 11.15. Let GF(2^6) be the code-construction field. From the extended (64,2,63) RS code over GF(2^6), we can construct a 64×64 array $\mathbf{H}^{(9)}_{rs,\text{disp}}$ of 64×64 PMs. Choose $g = 6$ and $r = 32$. Take the 6×32 subarray $\mathbf{H}^{(9)}_{rs,\text{disp}}(6, 32)$ at the upper-left corner of $\mathbf{H}^{(9)}_{rs,\text{disp}}$. This subarray is a 384×2048 matrix over GF(2) with column and row weights 6 and 32, respectively. The null space of this matrix gives a (6,32)-regular (2048,1723) LDPC code with minimum distance at least 8. This code has been chosen as the IEEE 802.3 standard code for the 10G BASE-T Ethernet. The bit-error performances of this code over the binary-input AWGN channel decoded using the SPA with 1, 5, 10, 50, and 100 iterations are shown in Figure 11.19. We see that decoding of this code converges very fast. The performance curves of this code with 50 and 100 iterations basically overlap each other. The error floor of this code is below the BER of 10^{-12}.

Problems

11.1 Consider the (63,2,62) RS code over GF(2^6). From this code and (11.5), a 63×63 array $\mathbf{H}^{(1)}$ of 63×63 CPMs can be constructed. Take a 4×28

subarray $\mathbf{H}^{(1)}(4,28)$ from $\mathbf{H}^{(1)}$, avoiding zero matrices of $\mathbf{H}^{(1)}$. Construct a QC-LDPC code using $\mathbf{H}^{(1)}(4,8)$ as the parity-check matrix and compute its error performance over the binary-input AWGN channel using the SPA with 10 and 50 iterations.

11.2 Take a 15×30 subarray $\mathbf{H}^{(1)}(15,30)$ from the 63×63 array $\mathbf{H}^{(1)}$ of 63×63 CPMs constructed in Problem 11.1. Design a 15×30 matrix $\mathbf{Z}(15,30)$ over GF(2) with column and row weights 3 and 6, respectively. Using $\mathbf{H}^{(1)}(15,30)$ as the base array and $\mathbf{Z}(15,30)$ as the masking matrix, construct a QC-LDPC code and compute its error performance using the SPA with 50 iterations.

11.3 Show that an RC-constrained $q \times q$ array of $(q-1) \times (q-1)$ CPMs can be constructed using the minimum-weight codewords of the extended $(q,2,q-1)$ RS code over GF(q) (see the discussion of extended RS codes in Chapter 3).

11.4 Let GF(2^7) be the code-construction field. Using (11.7) and (11.8), a 127×127 array $\mathbf{H}^{(2)}_{qc,\text{disp}}$ of 127×127 CPMs can be constructed. Take an 8×16 subarray $\mathbf{H}^{(2)}_{8,16}$ from $\mathbf{H}^{(2)}$. Construct an 8×16 masking matrix $\mathbf{Z}(8,16) = [G_0 G_1]$ that consists of two 8×8 circulants $\mathbf{G}_0$ and $\mathbf{G}_1$ with generators $\mathbf{g}_0 = (1011000)$ and $\mathbf{g}_1 = (01010100)$, respectively. Masking $\mathbf{H}^{(2)}_{8,16}$ with $\mathbf{Z}_{8,16}$ results in a masked array $\mathbf{M}^{(2)}_{8,16}$ that is a 1016×2032 matrix over GF(2) with column and row weights 3 and 6, respectively. Determine the QC-LDPC code given by the null space of $\mathbf{M}^{(2)}_{8,16}$ and compute its bit- and word-error performances over the binary-input AWGN channel using BPSK transmission decoded with 50 iterations of the SPA.

11.5 Prove Lemmas 11.3 and 11.4.

11.6 Prove that the array $\mathbf{H}^{(6)}_{qc,\text{disp}}$ given by (11.16) satisfies the RC-constraint.

11.7 Use the prime field GF(53) and the method presented in Section 11.6 to construct a 53×53 array $\mathbf{H}$ of 53×53 CPMs. Take a 6×48 subarray $\mathbf{H}(6,48)$ from $\mathbf{H}^{(6)}_{qc,\text{disp}}$. Determine the QC-LDPC code with $\mathbf{H}^{(6)}_{qc,\text{disp}}(6,48)$ as the parity-check matrix and compute its error performance over the binary-input AWGN channel using the SPA with 50 iterations.

11.8 This is a continuation of Problem 11.7. Take a 26×52 subarray $\mathbf{H}^{(6)}_{qc,\text{disp}}(26,52)$ from the array $\mathbf{H}^{(6)}_{qc,\text{disp}}$ constructed in Problem 11.7. Construct an irregular masking matrix $\mathbf{Z}(26,52)$ based on the variable- and check-node degree distributions given in Example 11.2. Masking $\mathbf{H}^{(6)}(26,52)$ with $\mathbf{Z}(26,52)$ results in a masked array $\mathbf{M}^{(6)}(26,52)$. Determine the column and row weight distributions of the masking matrix $\mathbf{Z}(26,52)$. Determine the irregular QC-LDPC code given by the null space of $\mathbf{M}^{(6)}(26,52)$ and compute its bit- and word-error performances over the binary-input AWGN channel using BPSK transmission decoded with 5, 10, and 50 iterations of the SPA, respectively.

11.9 Prove Lemma 11.5.

11.10 Show that QC-LDPC codes can also be constructed from an intersecting bundle of lines in a projective geometry using a method similar to that given in Section 11.8. Choose a projective geometry, construct a code, and compute the bit- and word-error performances of the code constructed.

References

[1] R. C. Bose and D. K. Ray-Chaudhuri, "On a class of error correcting binary group codes," *Information and Control*, vol. 3, no. 3, pp. 68–79, March 1960.

[2] A. Hocquenghem, "Codes correcteurs d'erreurs," *Chiffres*, vol. 2, pp. 147–156, 1959.

[3] I. S. Reed and G. Solomon, "Polynomial codes over certain finite fields," *J. Soc. Indust. Appl. Math.*, Vol. 8, pp. 300–304, June 1960.

[4] L. Chen, J. Xu, I. Djurdjevic, and S. Lin, "Near-Shannon limit quasi-cyclic low-density parity-check codes," *IEEE Trans. Communications*, vol. 52, no. 7, pp. 1038–1042, July 2004.

[5] L. Chen, L. Lan, I. Djurdjevic, S. Lin, and K. Abdel-Ghaffar, "An algebraic method for constructing quasi-cyclic LDPC codes," *Proc. Int. Symp. on Information Theory and Its Applications (ISITA)*, Parma, October 2004, pp. 535–539.

[6] I. Djurdjevic, J. Xu, K. Abdel-Ghaffar, and S. Lin, "A class of low-density parity-check codes constructed based on Reed–Solomon Codes with two information symbols," *IEEE Communications Lett.*, vol. 7, no. 7, pp. 317–319, July 2003.

[7] J. L. Fan, "Array codes as low-density parity-check codes," *Proc. 2nd Int. Symp. on Turbo Codes and Related Topics*, Brest, September 2000, pp. 543–546.

[8] L. Lan, L.-Q. Zeng, Y. Y. Tai, S. Lin, and K. Abdel-Ghaffar, "Constructions of quasi-cyclic LDPC codes for AWGN and binary erasure channels based on finite fields and affine permutations," *Proc. IEEE Int. Symp. on Information Theory*, Adelaide, September 2005, pp. 2285–2289.

[9] L. Lan, L.-Q. Zeng, Y. Y. Tai, L. Chen, S. Lin, and K. Abdel-Ghaffar, "Construction of quasi-cyclic LDPC codes for AWGN and binary erasure channels: a finite field approach," *IEEE Trans. Information Theory*, vol. 53, no. 7, pp. 2429–2458, July 2007.

[10] S. M. Song, B. Zhou, S. Lin, and K. Abdel-Ghaffar, "A unified approach to the construction of binary and nonbinary quasi-cyclic LDPC codes based on finite fields," *IEEE Trans. Communications*, vol. 57, no. 1, pp. 84–93, January 2009.

[11] Y. Y. Tai, L. Lan, L.-Q. Zheng, S. Lin, and K. Abdel-Ghaffar, "Algebraic construction of quasi-cyclic LDPC codes for the AWGN and erasure channels," *IEEE Trans. Communications*, vol. 54, no. 10, pp. 1765–1774, October 2006.

[12] Z.-W. Li, L. Chen, L.-Q. Zeng, S. Lin, and W. Fong, "Efficient encoding of quasi-cyclic low-density parity-check codes," *IEEE Trans. Communications*, vol. 54, no. 1, pp. 71–81, January 2006.

[13] S. Lin and D. J. Costello, Jr., *Error Control Coding: Fundamentals and Applications*, 2nd edn., Upper Saddle River, NJ, Prentice-Hall, 2004.

12 LDPC Codes Based on Combinatorial Designs, Graphs, and Superposition

Combinatorial designs [1–8] form an important branch in combinatorial mathematics. In the late 1950s and during the 1960s, special classes of combinatorial designs, such as *balanced incomplete block designs*, were used to construct error-correcting codes, especial majority-logic-decodable codes. More recently, combinatorial designs were successfully used to construct structured LDPC codes [9–12]. LDPC codes of practical lengths constructed from several classes of combinatorial designs were shown to perform very well over the binary-input AWGN channel with iterative decoding.

Graphs form another important branch in combinatorial mathematics. They were also used to construct error-correcting codes in the early 1960s, but not very successfully. Only a few small classes of majority-logic-decodable codes were constructed. However, since the rediscovery of LDPC codes in the middle of the 1990s, graphs have become an important tool for constructing LDPC codes. One example is to use protographs for constructing iteratively decodable codes as described in Chapters 6 and 8.

This chapter presents several methods for constructing LDPC codes based on special types of combinatorial designs and graphs.

12.1 Balanced Incomplete Block Designs and LDPC Codes

Balanced incomplete block designs (BIBDs) form an important class of combinatorial designs. A special subclass of BIBDs can be used to construct RC-constrained matrices or arrays of CPMs from which LDPC codes can be constructed. This section gives a brief description of BIBDs. For an in-depth understanding of this subject, readers are referred to [1–8].

Let $\mathcal{X} = \{x_1, x_2, \ldots, x_m\}$ be a set of m objects. A BIBD $\mathcal{B}$ of $\mathcal{X}$ is a collection of n g-subsets of $\mathcal{X}$, denoted $B_1, B_2, \ldots, B_n$, called *blocks*, which have the following structural properties: (1) each object x_i appears in exactly r of the n blocks and (2) every two objects appear together in exactly λ of the n blocks. Since a BIBD is characterized by five parameters, m, n, g, r, and λ, it is also called an (m, n, g, r, λ)-BIBD. For the special case with $\lambda = 1$, each pair of objects in $\mathcal{X}$ appears in *one and only one block*. Consequently, any two blocks have *exactly one* object in common. BIBDs of this special type will be used for constructing LDPC codes whose Tanner graphs have girths of at least 6.

Instead of listing all the blocks, an (m, n, g, r, λ)-BIBD $\mathcal{B}$ of $\mathcal{X}$ can be efficiently described by an $m \times n$ matrix $\mathbf{H}_{\text{BIBD}} = [h_{i,j}]$ over GF(2) as follows: (1) the rows of $\mathbf{H}_{\text{BIBD}}$ correspond to the m objects of $\mathcal{X}$; (2) the columns of $\mathbf{H}_{\text{BIBD}}$ correspond to the n blocks of the design $\mathcal{B}$; and (3) the entry $h_{i,j}$ at the ith row and jth column is set to "1" if and only if the ith object x_i of $\mathcal{X}$ is contained in the jth block B_j of the design $\mathcal{B}$; otherwise, it is set to "0." This matrix over GF(2) is called the *incidence matrix* of the design $\mathcal{B}$. It follows from the properties of an (m, n, g, r, λ)-BIBD $\mathcal{B}$ that the incidence matrix $\mathbf{H}_{\text{BIBD}}$ of $\mathcal{B}$ has the following structural properties: (1) every column has weight g; (2) every row has weight r; and (3) any two columns (or two rows) have exactly λ 1-components in common.

For the special case with $\lambda = 1$, the incidence matrix $\mathbf{H}_{\text{BIBD}}$ of an $(m, n, g, r, 1)$-BIBD $\mathcal{B}$ satisfies the RC constraint and hence its null space gives an LDPC code whose Tanner graph has a girth of at least 6.

As an example, consider a set $\mathcal{X} = \{x_1, x_2, x_3, x_4, x_5, x_6, x_7\}$ of seven objects. The blocks

$$\begin{array}{lll} B_1 = \{x_1, x_2, x_4\}, & B_2 = \{x_2, x_3, x_5\}, & B_3 = \{x_3, x_4, x_6\}, \\ B_4 = \{x_4, x_5, x_7\}, & B_5 = \{x_5, x_6, x_1\}, & B_6 = \{x_6, x_7, x_2\}, \\ B_7 = \{x_7, x_1, x_3\}, & & \end{array}$$

form a (7, 7, 3, 3, 1)-BIBD $\mathcal{B}$ of $\mathcal{X}$. Every block consists of three objects, each object appears in three blocks, and every two objects appear together in one and only one block. The incidence matrix of this (7, 7, 3, 3, 1)-BIBD is a 7×7 matrix over GF(2) given as follows:

$$\mathbf{H}_{\text{BIBD}} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}.$$

Note that $\mathbf{H}_{\text{BIBD}}$ is also a circulant. Its null space gives a (3,3)-regular (7,3) cyclic LDPC code with minimum distance at least 4. However, since the vector sum of columns 1, 2, 4, and 5 of $\mathbf{H}_{\text{BIBD}}$ gives a zero column vector, the minimum distance of the code is then exactly 4.

12.2 Class-I Bose BIBDs and QC-LDPC Codes

Combinatorial design is an old and rich subject in combinatorial mathematics. Over the years, many classes of BIBDs have been constructed using various methods. Extensive coverage of these designs can be found in [4]. In this section and the next, two classes of BIBDs with $\lambda = 1$ are presented. These classes of BIBDs with $\lambda = 1$ were constructed by Bose [1] from finite fields and can be used to construct

QC-LDPC codes. In the following, we present the construction of these designs without providing the proofs. For proofs, readers are referred to [1].

12.2.1 Class-I Bose BIBDs

Let t be a positive integer such that $12t+1$ is a prime. Then there exists a prime field $\mathrm{GF}(12t+1) = \{0, 1, \ldots, 12t\}$. Let the elements of $\mathrm{GF}(12t+1)$ represent a set $\mathcal{X}$ with $12t+1$ objects for which a BIBD is to be constructed. Suppose $\mathrm{GF}(12t+1)$ has a primitive element α such that the condition

$$\alpha^{4t} - 1 = \alpha^c \tag{12.1}$$

holds, where c is an odd non-negative integer less than $12t+1$. Under such a condition on $\mathrm{GF}(12t+1)$, Bose [1] showed that there exists an $(m, n, g, r, 1)$-BIBD with $m = 12t+1$, $n = t(12t+1)$, $g = 4$, $r = 4t$, and $\lambda = 1$. This BIBD is referred to as a *class-I Bose BIBD*.

Since α is a primitive element of $\mathrm{GF}(12t+1)$, $\alpha^{-\infty} = 0, \alpha^0 = 1, \alpha, \ldots, \alpha^{12t-1}$ form the $12t+1$ elements of $\mathrm{GF}(12t+1)$ and $\alpha^{12t} = 1$. Note that the $12t+1$ powers of α simply give the $12t+1$ integral elements, $0, 1, \ldots, 12t+1$, of $\mathrm{GF}(12t+1)$. To form a class-I Bose BIBD, denoted $\mathcal{B}^{(1)}$, we first form t *base blocks* [1], which are given as follows: for $0 \le i < t$,

$$B_{i,0} = \{\alpha^{-\infty}, \alpha^{2i}, \alpha^{2i+4t}, \alpha^{2i+8t}\}. \tag{12.2}$$

For each base block $B_{i,0}$, we form $12t+1$ blocks, $B_{i,0}, B_{i,1}, \ldots, B_{i,12t}$, by adding each element of $\mathrm{GF}(12t+1)$ in turn to the elements in $B_{i,0}$. Then, for $0 \le j \le 12t$,

$$B_{i,j} = \{j + \alpha^{-\infty}, j + \alpha^{2i}, j + \alpha^{2i+4t}, j + \alpha^{2i+8t}\}, \tag{12.3}$$

where addition is modulo-$(12t+1)$ addition. The $12t+1$ blocks, $B_{i,0}, B_{i,1}, \ldots, B_{i,12t}$, are called *co-blocks* of the base block $B_{i,0}$ and they form a *translate class*, denoted T_i. The $t(12t+1)$ blocks in the t translate classes, $T_0, T_1, \ldots, T_{t-1}$, form a class-I $(12t+1, t(12t+1), 4, 4t, 1)$ Bose BIBD $\mathcal{B}^{(1)}$. Table 12.1 gives a list of ts such that $12t+1$ is a prime and the prime field $\mathrm{GF}(12t+1)$ has a primitive element α that satisfies the condition given by (12.1).

12.2.2 Type-I Class-I Bose BIBD-LDPC Codes

For the jth block $B_{i,j}$ in the ith translate class T_i of a class-I Bose BIBD with $0 \le i < t$ and $0 \le j \le 12t$, we define a $(12t+1)$-tuple over $\mathrm{GF}(2)$,

$$\mathbf{v}_{i,j} = (v_{i,j,0}, v_{i,j,1}, \ldots, v_{i,j,12t}), \tag{12.4}$$

whose components correspond to the $12t+1$ elements of $\mathrm{GF}(12t+1)$, where the kth component $v_{i,j,k} = 1$ if k is an element in $B_{i,j}$, otherwise $v_{i,j,k} = 0$. This $(12t+1)$-tuple $\mathbf{v}_{i,j}$ is simply the *incidence vector of the block* $B_{i,j}$ and it has weight 4. For $0 \le j \le 12t$, it follows from (12.3) that the incidence vector $\mathbf{v}_{i,j+1}$ of the $(j+1)$th block $B_{i,j+1}$ in T_i is the right cyclic-shift of the incidence vector $\mathbf{v}_{i,j}$ of the jth block $B_{i,j}$ in T_i. Note that $\mathbf{v}_{i,12t+1} = \mathbf{v}_{i,0}$. For $0 \le i < t$, form a

$(12t+1) \times (12t+1)$ circulant $\mathbf{G}_i$ with the incidence vector $\mathbf{v}_{i,0}$ of the base block $B_{i,0}$ of T_i as the first column and its $12t$ downward cyclic-shifts as the other $12t$ columns. The $12t+1$ columns of $\mathbf{G}_i$ are simply the *transposes* of the incidence vectors of the $12t+1$ blocks in the ith translate class T_i of the class-I $(12t+1, t(12t+1), 4, 4t, 1)$ Bose BIBD $\mathcal{B}^{(1)}$. The column and row weights of $\mathbf{G}_i$ are both 4. Form the following $(12t+1) \times t(12t+1)$ matrix over GF(2):

$$\mathbf{H}^{(1)}_{\text{BIBD}} = \begin{bmatrix} \mathbf{G}_0 & \mathbf{G}_1 & \cdots & \mathbf{G}_{t-1} \end{bmatrix}, \tag{12.5}$$

which consists of a row of t circulants. $\mathbf{H}^{(1)}_{\text{BIBD}}$ is the incidence matrix of the class-I $(12t+1, t(12t+1), 4, 4t, 1)$ Bose BIBD $\mathcal{B}^{(1)}$ given above. Since $\lambda = 1$, $\mathbf{H}^{(1)}_{\text{BIBD}}$ satisfies the RC-constraint. It has column and row weights 4 and $4t$, respectively.

For $1 \le k \le t$, let $\mathbf{H}^{(1)}_{\text{BIBD}}(k)$ be a subarray of $\mathbf{H}^{(1)}_{\text{BIBD}}$ that consists of k circulants of $\mathbf{H}^{(1)}_{\text{BIBD}}$. $\mathbf{H}^{(1)}_{\text{BIBD}}(k)$ is a $(12t+1) \times k(12t+1)$ matrix with column and row weights 4 and $4k$, respectively. The null space of $\mathbf{H}^{(1)}_{\text{BIBD}}(k)$ gives a $(4,4k)$-regular QC-LDPC code of length $k(12t+1)$ with rate at least $(k-1)/k$. The above construction gives a class of *type-I QC-BIBD-LDPC codes* with various lengths and rates.

Example 12.1. For $t = 15$, $12t+1 = 181$ is a prime. Then there exists a prime field $\text{GF}(181) = \{0, 1, \ldots, 180\}$. This field has a primitive element $\alpha = 2$ that satisfies the condition (12.1) with $c = 13$ (see Table 12.1). From this field, we can construct a class-I

Table 12.1. A list of ts for which $12t+1$ is a prime and the condition given by (12.1) holds

t	Field	(α,c)
1	GF(13)	(2,1)
6	GF(73)	(5,33)
8	GF(97)	(5,27)
9	GF(109)	(6,71)
15	GF(181)	(2,13)
19	GF(229)	(6,199)
20	GF(241)	(7,191)
23	GF(277)	(5,209)
28	GF(337)	(10,129)
34	GF(409)	(21,9)
35	GF(421)	(2,167)
38	GF(457)	(13,387)
45	GF(541)	(2,7)
59	GF(709)	(2,381)
61	GF(733)	(6,145)

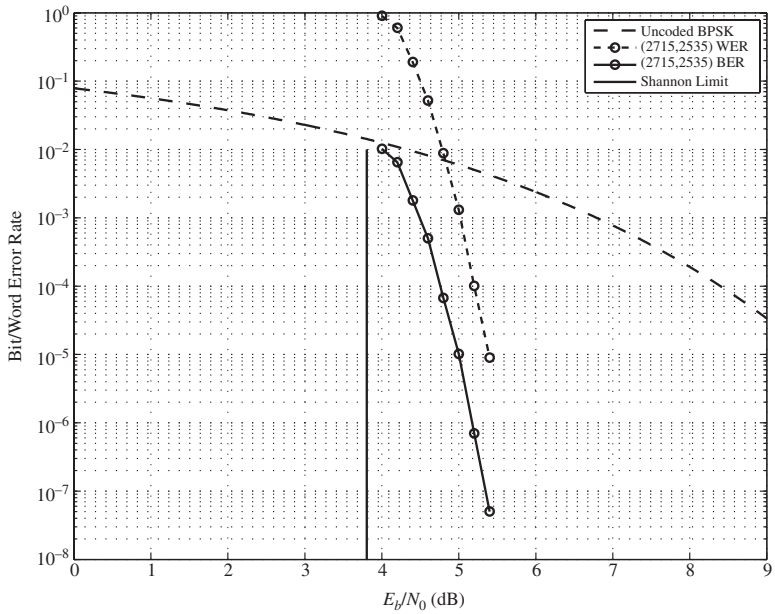


Figure 12.1 The error performance of the (2715,2535) type-I QC-BIBD-LDPC code given in Example 12.1.

(181, 2715, 4, 60, 1) Bose BIBD. The incidence matrix of this BIBD is

$$\mathbf{H}_{\text{BIBD}}^{(1)} = \begin{bmatrix} \mathbf{G}_0 & \mathbf{G}_1 & \cdots & \mathbf{G}_{14} \end{bmatrix},$$

which consists of 15 circulants of size 181×181, each with both column weight and row weight 4. $\mathbf{H}_{\text{BIBD}}^{(1)}$ is a 181×2175 matrix with column and row weights 4 and 60, respectively. Suppose we choose $\mathbf{H}_{\text{BIBD}}^{(1)}$ as the parity-check matrix for code generation. Then the null space of $\mathbf{H}_{\text{BIBD}}^{(1)}$ gives a (4,60)-regular (2715,2535) type-I QC-BIBD-LDPC code with rate 0.934, whose Tanner graph has a girth of at least 6.

Assume BPSK transmission over the binary-input AWGN channel. The error performance of this code with iterative decoding using the SPA with 100 iterations is shown in Figure 12.1. At a BER of 10^{-6}, it performs 1.3 dB from the Shannon limit.

12.2.3 Type-II Class-I Bose BIBD-LDPC Codes

For $0 \le i < t$, if we decompose each circulant $\mathbf{G}_i$ in $\mathbf{H}_{\text{BIBD}}^{(1)}$ given by (12.5) into a column of four $(12t+1) \times (12t+1)$ CPMs using the row decomposition presented in Section 10.5 (see (10.22)), we obtain the following $4 \times t$ array of $(12t+1) \times (12+1)$ CPMs:

$$\mathbf{H}_{\text{BIBD,decom}}^{(1)} = \begin{bmatrix} \mathbf{A}_{0,0} & \mathbf{A}_{0,1} & \cdots & \mathbf{A}_{0,t-1} \\ \mathbf{A}_{1,0} & \mathbf{A}_{1,1} & \cdots & \mathbf{A}_{1,t-1} \\ \mathbf{A}_{2,0} & \mathbf{A}_{2,1} & \cdots & \mathbf{A}_{2,t-1} \\ \mathbf{A}_{3,0} & \mathbf{A}_{3,1} & \cdots & \mathbf{A}_{3,t-1} \end{bmatrix}. \tag{12.6}$$

For $3 \leq k \leq 4$ and $4 \leq r \leq t$, let $\mathbf{H}^{(1)}_{\text{BIBD,decom}}(k,r)$ be a $k \times r$ subarray of $\mathbf{H}^{(1)}_{\text{BIBD,decom}}$. $\mathbf{H}^{(1)}_{\text{BIBD,decom}}(k,r)$ is a $k(12t+1) \times r(12t+1)$ matrix over GF(2) with column and row weights k and r, respectively. The null space of $\mathbf{H}^{(1)}_{\text{BIBD,decom}}(k,r)$ gives a (k,r)-regular QC-BIBD-LDPC code. The above construction gives another class of QC-BIBD-LDPC codes. If we choose $k = 3$ and $r = 3l$ with $l = 2, 3, 4, 5, \ldots$, we can construct a sequence of $(3,3l)$-regular QC-BIBD-LDPC codes with rates equal (or close) to $1/2, 2/3, 3/4, 4/5, \ldots$ If we choose $k = 4$ and $r = 4l$ with $l = 2, 3, 4, 5, \ldots$, we can construct a sequence of $(4,4l)$-regular QC-BIBD-LDPC codes with rates equal (or close) to 1/2,2/3,3/4, and $4/5, \ldots$

Suppose we take a $4 \times 4l$ subarray $\mathbf{H}^{(1)}_{\text{BIBD,decom}}(4,4l)$ from $\mathbf{H}^{(1)}_{\text{BIBD,DECOM}}$ with $4l \leq t$ and $1 \leq l$. Divide this array into l 4×4 subarrays, $\mathbf{H}^{(1)}_0(4,4), \mathbf{H}^{(1)}_1(4,4), \ldots, \mathbf{H}^{(1)}_{l-1}(4,4)$. For $0 \leq i < l$, mask the 4×4 subarray $\mathbf{H}^{(1)}_i(4,4)$ with the following 4×4 circulant masking matrix:

$$\mathbf{Z}_i(4,4) = \begin{bmatrix} 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \end{bmatrix}. \tag{12.7}$$

Let $\mathbf{M}^{(1)}_i(4,4) = \mathbf{Z}_i(4,4) \circledast \mathbf{H}^{(1)}_i(4,4)$ for $0 \leq i < l$. $\mathbf{M}^{(1)}_i(4,4)$ is a masked 4×4 array in which each column (or row) contains one zero matrix and three CPMs of size $(12t+1) \times (12t+1)$. Form the following $4 \times 4l$ masked array of circulant permutation and zero matrices of size $(12t+1) \times (12t+1)$:

$$\mathbf{M}^{(1)}_{\text{BIBD,decom}}(4,4l) = \begin{bmatrix} \mathbf{M}^{(1)}_0(4,4) & \mathbf{M}^{(1)}_1(4,4) & \cdots & \mathbf{M}^{(1)}_{l-1}(4,4) \end{bmatrix}. \tag{12.8}$$

This array is a $4(12t+1) \times 4l(12t+1)$ matrix over GF(2) with column and row weights 3 and $3l$. The null space of this masked matrix gives a $(3,3l)$-regular QC-BIBD-LDPC code. Since every 4×4 subarray $\mathbf{H}^{(1)}_i(4,4)$ is masked with the same masking matrix, the above masking is referred to as *uniform circulant masking*. Of course, the subarrays, $\mathbf{H}^{(1)}_0(4,4), \mathbf{H}^{(1)}_1(4,4), \ldots, \mathbf{H}^{(1)}_{l-1}(4,4)$, can be masked with different masking circulant matrices. With non-uniform circulant masking of the subarrays, we obtain a $4 \times 4l$ masked array $\mathbf{M}^{(1)}_{\text{BIBD,decom}}(4,4l)$ with multiple column weights but constant row weight. Then the null space of this irregular array gives an irregular QC-BIBD-LDPC code.

Example 12.2. Consider the array $\mathbf{H}^{(1)}_{\text{BIBD}}$ constructed in Example 12.1. Decompose each circulant $\mathbf{G}_i$ in this array into a column of four 181×181 CPMs. The decomposition results in a 4×15 array $\mathbf{H}^{(1)}_{\text{BIBD,decom}}$ of 181×181 CPMs. Suppose we take the 4×12 subarray $\mathbf{H}^{(1)}_{\text{BIBD,decom}}(4,12)$ from $\mathbf{H}^{(1)}_{\text{BIBD,decom}}$, say the first 12 columns of $\mathbf{H}^{(1)}_{\text{BIBD,decom}}$. $\mathbf{H}^{(1)}_{\text{BIBD,decom}}(4,12)$ is a 724×2172 matrix over GF(2) with column and row weights 4 and 12, respectively. Divide $\mathbf{H}^{(1)}_{\text{BIBD,decom}}(4,12)$ into three 4×4 subarrays and then mask

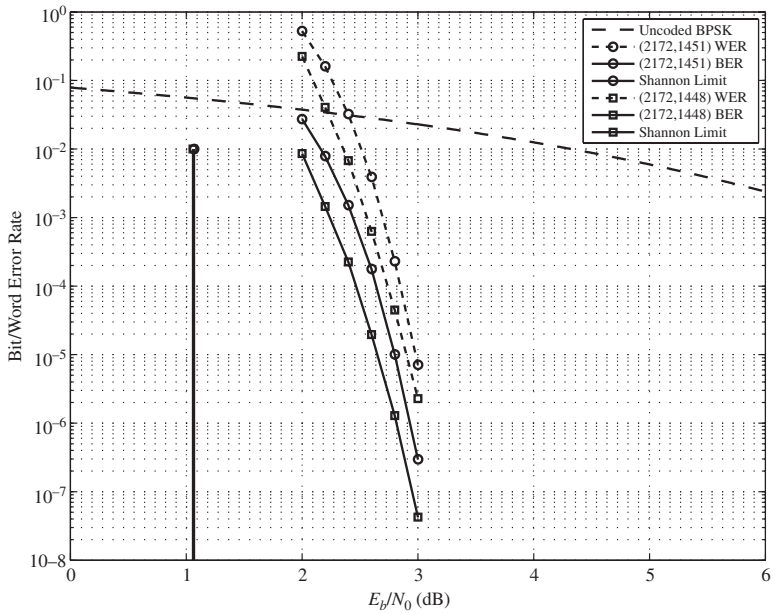


Figure 12.2 The error performances of the (2172,1451) and (2172,1448) codes given in Example 12.2.

each of these 4×4 subarrays with the circulant matrix given by (12.7). This results in a regular 4×12 masked array $\mathbf{M}^{(1)}_{\text{BIBD,decom}}(4, 12)$ of 181×181 circulant permutation and zero matrices. $\mathbf{M}^{(1)}_{\text{BIBD,decom}}(4, 12)$ is a 724×2172 matrix with column and row weights 3 and 9, respectively. The null spaces of $\mathbf{H}^{(1)}_{\text{BIBD,decom}}(4, 12)$ and $\mathbf{M}^{(1)}_{\text{BIBD,decom}}(4, 12)$ give (2172,1451) and (2172,1448) QC-BIBD-LDPC codes with rates 0.668 and 0.667, respectively. Their error performances with iterative decoding using the SPA with 100 iterations are shown in Figure 12.2.

Example 12.3. Let $t = 28$. Then $12t + 1 = 337$ is a prime and there is a prime field GF(337). From Table 12.1, we see that this prime field satisfies the condition given by (12.1). Using this prime field, we can construct a class-I (m, n, g, r, 1) Bose BIBD with $m = 337$, $n = 9436$, $g = 4$, $r = 112$, and $\lambda = 1$. This BIBD consists of 28 translate classes of blocks. Using the incidence vectors of the blocks in these translate classes, we can form the following row of 28 337×337 circulants, each having column weight and row weight 4: $\mathbf{H}^{(1)}_{\text{BIBD}} = \left[\mathbf{G}_0\ \mathbf{G}_1\ \cdots\ \mathbf{G}_{27}\right]$. Decompose each circulant $\mathbf{G}_i$ in $\mathbf{H}^{(1)}_{\text{BIBD}}$ into a column of four 337×337 CPMs by row decomposition. The decomposition results in a 4×28 array $\mathbf{H}^{(1)}_{\text{BIBD,decom}}$ of 337×337 CPMs. Suppose we take a 3×6 subarray $\mathbf{H}^{(1)}_{\text{BIBD,decom}}(3, 6)$ from $\mathbf{H}^{(1)}_{\text{BIBD,decom}}$, say the first three rows of the first six columns of $\mathbf{H}^{(1)}_{\text{BIBD,decom}}$. $\mathbf{H}^{(1)}_{\text{BIBD,decom}}(3, 6)$ is a 1011×2022 matrix over GF(2) with column and row weights 3 and 6, respectively. The null space of this matrix gives a (3,6)-regular (2022,1013) type-II QC-BIBD-LDPC code with rate 0.501. The error performance of

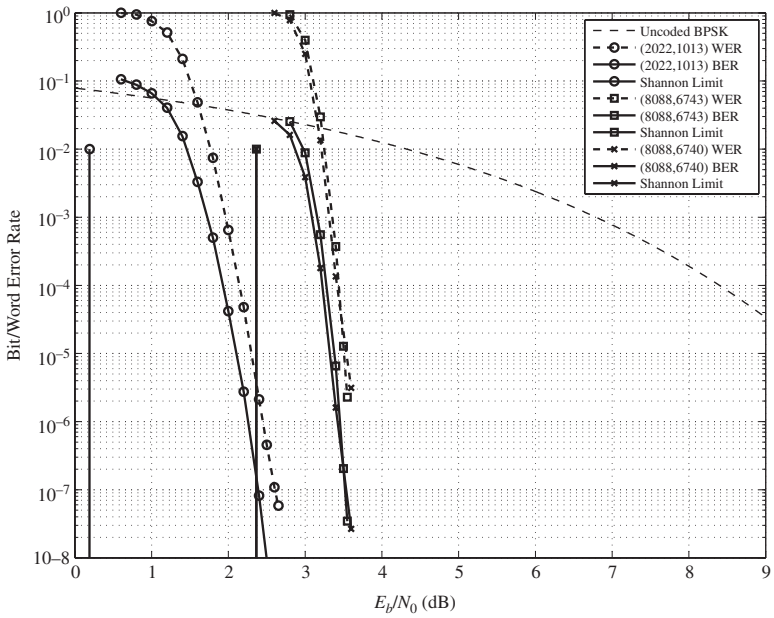


Figure 12.3 Error performances of the (2022,1013), (8088,6743), and (8088,6740) codes given in Example 12.3.

this code over the binary-input AWGN channel with iterative decoding using the SPA with 100 iterations is shown in Figure 12.3. At a BER of 10^{-6}, it performs 2 dB from the Shannon limit. It has no error floor down to the BER of 10^{-8}.

If we take a 4×24 subarray $\mathbf{H}^{(1)}_{\text{BIBD,decom}}(4, 24)$ from $\mathbf{H}^{(1)}_{\text{BIBD,decom}}$, say the first 24 columns of $\mathbf{H}^{(1)}_{\text{BIBD,decom}}$, the null space of $\mathbf{H}^{(1)}_{\text{BIBD,decom}}(4, 24)$ gives a (4,24)-regular (8088,6743) type-II QC-BIBD-LDPC code with rate 0.834 and estimated minimum distance 30. The error performance of this code over the binary-input AWGN channel with iterative decoding using the SPA with 100 iterations is also shown in Figure 12.3. At a BER of 10^{-6}, it performs 1.04 dB from the Shannon limit.

If we divide $\mathbf{H}^{(1)}_{\text{BIBD,decom}}(4, 24)$ into six 4×4 subarrays and mask each subarray with the masking matrix given by (12.7), we obtain a regular 4×24 masked array $\mathbf{M}^{(1)}_{\text{BIBD,decom}}(4, 24)$ with column and row weights 3 and 18, respectively. The null space of $\mathbf{M}^{(1)}_{\text{BIBD,decom}}(4, 24)$ gives a (3,18)-regular (8088,6740) type-II QC-BIBD-LDPC code with rate 0.833. The error performance of this code is also shown in Figure 12.3. At a BER of 10^{-6}, it performs 1.04 dB from the Shannon limit.

12.3 Class-II Bose BIBDs and QC-LDPC Codes

This section presents another class of Bose BIBDs constructed from prime fields. Two classes of QC-LDPC codes can be constructed from this class of Bose BIBDs.

12.3.1 Class-II Bose BIBDs

Let t be a positive integer such that $20t+1$ is a prime. Then there exists a prime field $\mathrm{GF}(20t+1) = \{0, 1, \ldots, 20t\}$ under modulo-$(20t+1)$ addition and multiplication. Let the elements of $\mathrm{GF}(20t+1)$ represent a set $\mathcal{X}$ of $20t+1$ objects. Suppose there exists a primitive element α for which the condition

$$\alpha^{4t} + 1 = \alpha^c \tag{12.9}$$

holds, where c is a positive odd integer less than $20t+1$. Bose [1] showed that under this condition there exists an $(m, n, g, r, 1)$-BIBD with $m=20t+1$, $n=t(20t+1)$, $g=5$, $r=5t$, and $\lambda=1$. For this BIBD, there are t base blocks, which are

$$\mathbf{B}_{i,0} = \{\alpha^{2i}, \alpha^{2i+4t}, \alpha^{2i+8t}, \alpha^{2i+12t}, \alpha^{2i+16t}\}, \tag{12.10}$$

with $0 \le i < t$. For each base block $\mathbf{B}_{i,0}$, we form $20t+1$ co-blocks, $\mathbf{B}_{i,0}, \mathbf{B}_{i,1}, \ldots, \mathbf{B}_{i,20t}$, by adding the $20t+1$ elements of $\mathrm{GF}(2t+1)$ in turn to the elements in $\mathbf{B}_{i,0}$. Then the jth co-block of $\mathbf{B}_{i,0}$ is given by

$$\mathbf{B}_{i,j} = \{j + \alpha^{2i}, j + \alpha^{2i+4t}, j + \alpha^{2i+8t}, j + \alpha^{2i+12t}, j + \alpha^{2i+16t}\}, \tag{12.11}$$

with $j \in \mathrm{GF}(20t+1)$. The $20t+1$ co-blocks of a base block $\mathbf{B}_{i,0}$ form a translate class T_i of the design. The $t(20t+1)$ blocks in the t translate classes $T_0, T_1, \ldots, T_{t-1}$ form a type-II $(20t+1, t(20t+1), 5, 5t, 1)$ Bose BIBD design, denoted $\mathcal{B}^{(2)}$. Table 12.2 gives a list of ts that satisfy the condition given by (12.9).

12.3.2 Type-I Class-II Bose BIBD-LDPC Codes

The incidence matrix of a class-II $(20t+1, t(20t+1), 5, 5t, 1)$ Bose BIBD, $\mathcal{B}^{(2)}$, can be arranged as a row of t circulants of size $(20t+1) \times (20t+1)$ as follows:

$$\mathbf{H}^{(2)}_{\mathrm{BIBD}} = \begin{bmatrix} \mathbf{G}_0 & \mathbf{G}_1 & \cdots & \mathbf{G}_{t-1} \end{bmatrix}, \tag{12.12}$$

where the ith circulant $\mathbf{G}_i$ is formed by the incidence vectors of the blocks in the ith translate class T_i of $\mathcal{B}^{(2)}$ arranged as columns in downward cyclic order. Each circulant $\mathbf{G}_i$ in $\mathbf{H}^{(2)}_{\mathrm{BIBD}}$ has both column weight and row weight 5. Therefore, $\mathbf{H}^{(2)}_{\mathrm{BIBD}}$ is an RC-constrained $(20t+1) \times t(20t+1)$ matrix over $\mathrm{GF}(2)$ with column and row weights 5 and $5t$, respectively.

For $1 \le k \le t$, let $\mathbf{H}^{(2)}_{\mathrm{BIBD}}(k)$ be a subarray of $\mathbf{H}^{(2)}_{\mathrm{BIBD}}$ that consists of k circulants of $\mathbf{H}^{(2)}_{\mathrm{BIBD}}$. $\mathbf{H}^{(2)}_{\mathrm{BIBD}}(k)$ is a $(20t+1) \times k(20t+1)$ matrix over $\mathrm{GF}(2)$ with column and row weights of 5 and $5k$, respectively. The null space of $\mathbf{H}^{(2)}_{\mathrm{BIBD}}(k)$ gives a $(5,5k)$-regular *type-I class-II QC Bose BIBD-LDPC code* of length $k(20t+1)$ with rate at least $(k-1)/k$. The above construction gives a class of QC-LDPC codes.

Example 12.4. Let $t = 21$. From Table 12.2, we see that a type-II (421, 8841, 5, 105, 1) Bose BIBD, $\mathcal{B}^{(2)}$, can be constructed from the prime field $\mathrm{GF}(421)$. $\mathcal{B}^{(2)}$ consists of 21 translate classes, each having 421 co-blocks. Using the incidence vectors of the blocks in these translate classes, we can form the incidence matrix

Table 12.2. A list of ts for which $20t + 1$ is a prime and the condition $\alpha^{4t} + 1 = \alpha^c$ holds

t	Field	(α,c)
2	GF(41)	(6,3)
3	GF(61)	(2,23)
12	GF(241)	(7,197)
14	GF(281)	(3,173)
21	GF(421)	(2,227)
30	GF(601)	(7,79)
32	GF(641)	(3,631)
33	GF(661)	(2,657)
35	GF(701)	(2,533)
41	GF(821)	(2,713)

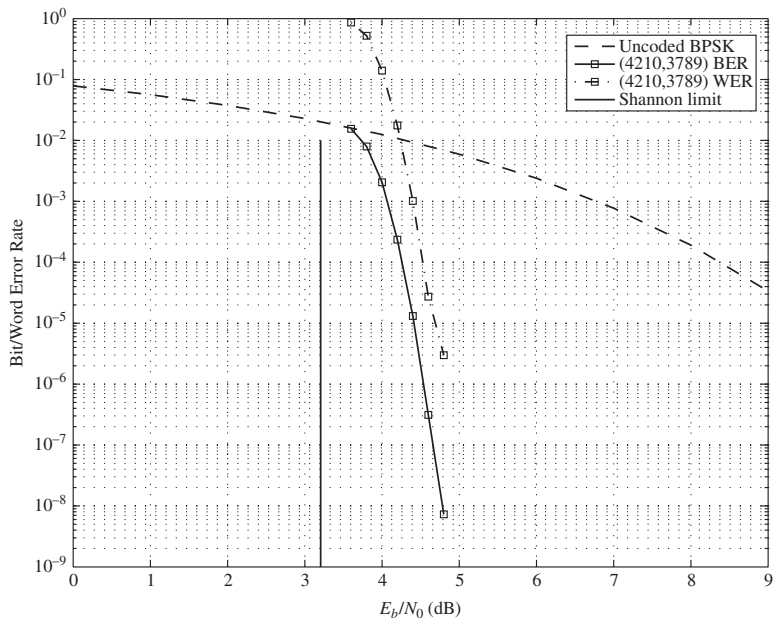


Figure 12.4 The error performance of the (4210,3789) QC-BIBD-LDPC code given in Example 12.4.

$\mathbf{H}^{(2)}_{\text{BIBD}} = \begin{bmatrix} \mathbf{G}_0 & \mathbf{G}_1 & \cdots & \mathbf{G}_{20} \end{bmatrix}$ of $\mathcal{B}^{(2)}$ that consists of 21 421×421 circulants, each with both column weight and row weight 5. Set $k = 10$. Take the first ten circulants of $\mathbf{H}^{(2)}_{\text{BIBD}}$ to form a subarray $\mathbf{H}^{(2)}_{\text{BIBD}}(10)$ of $\mathbf{H}^{(2)}_{\text{BIBD}}$. $\mathbf{H}^{(2)}_{\text{BIBD}}(10)$ is a 421×4210 matrix with column and row weights 5 and 50, respectively. The null space of $\mathbf{H}^{(2)}_{\text{BIBD}}(10)$ gives a (5,50)-regular (4210,3789) QC-BIBD-LDPC code with rate 0.9. The error performance of this code with iterative decoding using the SPA with 100 iterations is shown in Figure 12.4.

12.3.3 Type-II Class-II QC-BIBD-LDPC Codes

If we decompose each circulant $\mathbf{G}_i$ in $\mathbf{H}^{(2)}_{\text{BIBD}}$ given by (12.12) into a column of five $(20t+1)\times(20t+1)$ CPMs with row decomposition, we obtain an RC-constrained $5\times t$ array of $(20t+1)\times(20t+1)$ CPMs as follows:

$$\mathbf{H}^{(2)}_{\text{BIBD,decom}} = \begin{bmatrix} \mathbf{A}_{0,0} & \mathbf{A}_{0,1} & \cdots & \mathbf{A}_{0,t-1} \\ \mathbf{A}_{1,0} & \mathbf{A}_{1,1} & \cdots & \mathbf{A}_{1,t-1} \\ \mathbf{A}_{2,0} & \mathbf{A}_{2,1} & \cdots & \mathbf{A}_{2,t-1} \\ \mathbf{A}_{3,0} & \mathbf{A}_{3,1} & \cdots & \mathbf{A}_{3,t-1} \\ \mathbf{A}_{4,0} & \mathbf{A}_{4,1} & \cdots & \mathbf{A}_{4,t-1} \end{bmatrix}. \tag{12.13}$$

For $3\le k\le 5$ and $3\le r\le t$, let $\mathbf{H}^{(2)}_{\text{BIBD,decom}}(k,r)$ be a $k\times r$ subarray of $\mathbf{H}^{(2)}_{\text{BIBD,decom}}$. $\mathbf{H}^{(2)}_{\text{BIBD,decom}}(k,r)$ is a $k(20t+1)\times r(20t+1)$ matrix over GF(2) with column and row weights k and r, respectively. The null space of $\mathbf{H}^{(2)}_{\text{BIBD,decom}}(k,r)$ gives a (k,r)-regular QC-LDPC code of length $r(20t+1)$ with rate at least $(r-k)/r$. The above construction gives another class of QC-BIBD-LDPC codes.

Example 12.5. For $t=21$, consider the class-II (421, 8841, 5, 105, 1) Bose BIBD constructed from the prime field GF(421) given in Example 12.4. From the incidence matrix of this Bose BIBD and row decompositions of its constituent circulants, we obtain a 5×21 array $\mathbf{H}^{(2)}_{\text{BIBD,decom}}$ of 421×421 CPMs. Take a 4×20 subarray $\mathbf{H}^{(2)}_{\text{BIBD,decom}}(4,20)$ from $\mathbf{H}^{(2)}_{\text{BIBD,decom}}$. This subarray is a 1684×8420 matrix over GF(2) with column and row weights 4 and 20, respectively. The null space of this matrix gives a (4,20)-regular (8420,6739) QC-BIBD-LDPC code with rate 0.8004. The error performance of this code with iterative decoding using the SPA with 100 iterations is shown in Figure 12.5. At a BER of 10^{-6}, it performs 1.1 dB from the Shannon limit.

Let $r=5k$ such that $5k\le t$. Take a $5\times 5k$ subarray $\mathbf{H}^{(2)}_{\text{BIBD,decom}}(5,5k)$ from $\mathbf{H}^{(2)}_{\text{BIBD,decom}}$ and divide it into k 5×5 subarrays, $\mathbf{H}^{(2)}_0(5,5), \mathbf{H}^{(2)}_1(5,5), \ldots,$ $\mathbf{H}^{(2)}_{k-1}(5,5)$. Then

$$\mathbf{H}^{(2)}_{\text{BIBD,decom}}(5,5k) = \begin{bmatrix} \mathbf{H}^{(2)}_0(5,5) & \mathbf{H}^{(2)}_1(5,5) & \cdots & \mathbf{H}^{(2)}_{k-1}(5,5) \end{bmatrix}. \tag{12.14}$$

For $0\le i<k$, we can mask each constituent subarray $\mathbf{H}^{(2)}_i(5,5)$ of $\mathbf{H}^{(2)}_{\text{BIBD,decom}}(5,5k)$ with either of the following two circulant masking matrices:

$$\mathbf{Z}_1(5,5) = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \end{bmatrix}, \tag{12.15}$$

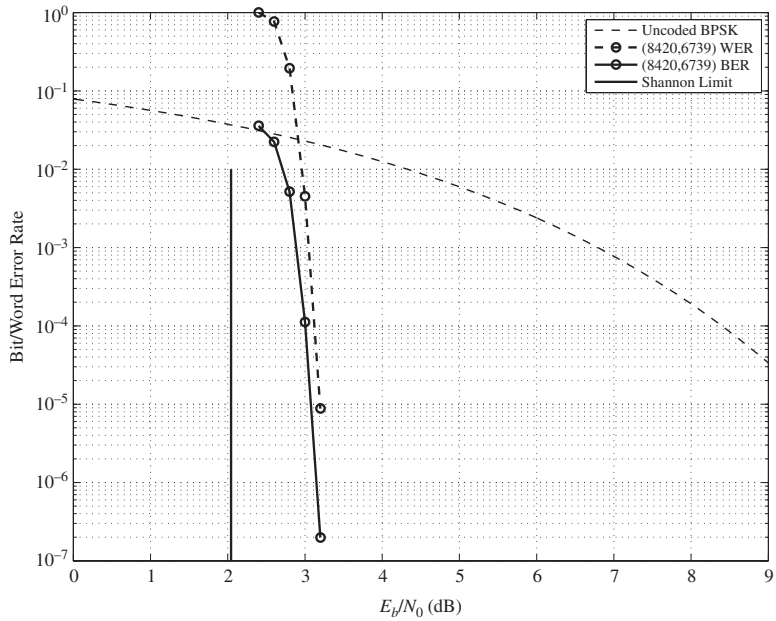


Figure 12.5 The error performance of the (8420,6739) QC-BIBD-LDPC code given in Example 12.5.

$$\mathbf{Z}_2(5,5) = \begin{bmatrix} 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 \end{bmatrix}. \tag{12.16}$$

For $0 \le i < k$, if we mask each constituent 5×5 subarray $\mathbf{H}_i^{(2)}(5,5)$ of $\mathbf{H}_{\text{BIBD,decom}}^{(2)}(5,5k)$ with $\mathbf{Z}_1(5,5)$ given by (12.15), we obtain a $5 \times 5k$ uniformly masked array of circulant permutation and zero matrices,

$$\mathbf{M}_{\text{BIBD,decom},1}^{(2)}(5,5k) = \begin{bmatrix} \mathbf{M}_{0,1}^{(2)}(5,5) & \mathbf{M}_{1,1}^{(2)}(5,5) & \cdots & \mathbf{M}_{k-1,1}^{(2)}(5,5) \end{bmatrix}, \tag{12.17}$$

where, for $0 \le i < k$, $\mathbf{M}_{i,1}^{(2)}(5,5) = \mathbf{Z}_1(5,5) \circledast \mathbf{H}_i^{(2)}(5,5)$. $\mathbf{M}_{\text{BIBD,decom},1}^{(2)}(5,5k)$ is a $5(20t+1) \times 5k(20t+1)$ matrix over GF(2) with column and row weights 3 and $3k$, respectively. The null space of $\mathbf{M}_{\text{BIBD,decom}}^{(2)}(5,5k)$ gives a (3,$3k$)-regular QC-BIBD-LDPC code.

For $0 \le i < k$, if we mask each constituent 5×5 subarray of $\mathbf{H}_{\text{BIBD,decom}}^{(2)}(5,5k)$ with $\mathbf{Z}_2(5,5)$ given by (12.16), we obtain a $5 \times 5k$ uniformly masked array of circulant permutation and zero matrices,

$$\mathbf{M}_{\text{BIBD,decom},2}^{(2)}(5,5k) = \begin{bmatrix} \mathbf{M}_{0,2}^{(2)}(5,5) & \mathbf{M}_{1,2}^{(2)}(5,5) & \cdots & \mathbf{M}_{k-1,2}^{(2)}(5,5) \end{bmatrix}, \tag{12.18}$$

where, for $0 \leq i < k$, $\mathbf{M}_{i,2}^{(2)}(5,5) = \mathbf{Z}_2(5,5) \circledast \mathbf{H}_i^{(2)}(5,5)$. $\mathbf{M}_{\text{BIBD,decom},2}^{(2)}(5,5k)$ is a $5(20t+1) \times 5k(20t+1)$ matrix over GF(2) with column and row weights 4 and $4k$, respectively. The null space of this matrix gives a (4,$4k$)-regular QC-BIBD-LDPC code.

The above maskings of the base array $\mathbf{H}_{\text{BIBD,decom}}^{(2)}(5,5k)$ are uniform maskings. Each constituent 5×5 subarray $\mathbf{H}_i^{(2)}(5,5)$ is masked by the same masking matrix. However, the constituent 5×5 subarrays of the base array $\mathbf{H}_{\text{BIBD,decom}}^{(2)}(5,5k)$ can be masked with different 5×5 circulant masking matrices with different column weights. This non-uniform masking results in a masked array $\mathbf{M}_{\text{BIBD,decom}}^{(2)}(5,5k)$ with multiple column weights. Besides the masking matrices given by (12.15) and (12.16), we define the following circulant masking matrix with column weight 2:

$$\mathbf{Z}_0(5,5) = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \end{bmatrix}. \tag{12.19}$$

Let k_0, k_1, k_2, and k_3 be four non-negative integers such that $k_0 + k_1 + k_2 + k_3 = k$. Suppose we mask the base array $\mathbf{H}_{\text{BIBD,decom}}^{(2)}(5,5k)$ as follows: (1) the first k_0 constituent 5×5 subarrays are masked with $\mathbf{Z}_0(5,5)$; (2) the next k_1 constituent 5×5 subarrays are masked with $\mathbf{Z}_1(5,5)$; (3) the next k_2 constituent 5×5 subarrays are masked with $\mathbf{Z}_2(5,5)$; and (4) the last k_3 constituent 5×5 subarrays are unmasked. This non-uniform masking results in a $5 \times 5k$ masked array $\mathbf{M}_{\text{BIBD,decom},3}^{(2)}(5,5k)$ with multiple column weights but constant row weight. The average column weight of the masked array is $(2k_0 + 3k_1 + 4k_2 + 5k_3)/k$. The masked array has a constant row weight, which is $2k_0 + 3k_1 + 4k_2 + 5k_3$. The null space of the non-uniformly masked array $\mathbf{M}_{\text{BIBD,decom},3}^{(2)}(5,5k)$ gives an irregular QC-BIBD-LDPC code. The error performance of this irregular code over the binary-input AWGN channel with iterative decoding depends on the choice of the parameters k_0, k_1, k_2, and k_3.

Example 12.6. Consider the 5×21 array $\mathbf{H}_{\text{BIBD,decom}}^{(2)}$ of 421×421 CPMs constructed from the class-II (421, 8841, 5, 105, 1) Bose BIBD using the prime field GF(421) with $t = 21$ given in Example 12.5. Take a 5×20 subarray $\mathbf{H}_{\text{BIBD,decom}}^{(2)}(5,20)$ from $\mathbf{H}_{\text{BIBD,decom}}^{(2)}$ and divide this subarray into four 5×5 subarrays, $\mathbf{H}_0^{(2)}(5,5)$, $\mathbf{H}_1^{(2)}(5,5)$, $\mathbf{H}_2^{(2)}(5,5)$, and $\mathbf{H}_3^{(2)}(5,5)$. Choose $k_0 = k_1 = k_2 = k_3 = 1$. We mask the first three 5×5 subarrays with $\mathbf{Z}_0(5,5)$, $\mathbf{Z}_1(5,5)$, and $\mathbf{Z}_2(5,5)$, respectively, and leave the fourth subarray $\mathbf{H}_3^{(2)}(5,5)$ unmasked. The masking results in a 5×20 masked array $\mathbf{M}_{\text{BIBD,decom},3}^{(2)}(5,20)$ of circulant permutation and zero matrices. This masked array has average column weight 3.5 and constant row weight 14. The null space of this masked array gives an irregular (8240,6315) QC-BIBD-LDPC code with rate 0.7664. The error performance of this code with iterative

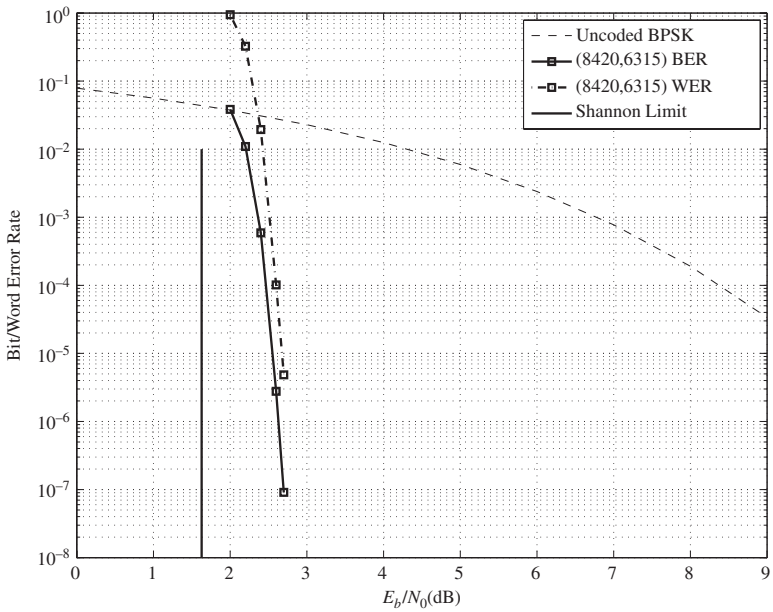


Figure 12.6 The error performance of the (8240,6315) QC-BIBD-LDPC code given in Example 12.6.

decoding using the SPA with 100 iterations is shown in Figure 12.6. At a BER of 10^{-6}, the code performs 1dB from the Shannon limit.

Remark. Before we conclude this section, we remark that any $k \times kl$ subarray $\mathbf{H}^{(e)}_{qc,\text{disp}}(k, kl)$ of the array $\mathbf{H}^{(e)}_{qc,\text{disp}}$ of CPMs with $1 \leq e \leq 6$ constructed in Chapter 11 can be divided into l subarrays of size $k \times k$. Each constituent $k \times k$ subarray of $\mathbf{H}^{(e)}_{qc,\text{disp}}(k, k)$ can be masked with a $k \times k$ circulant masking matrix with weight less than or equal to k. Then the masking results in a $k \times kl$ masked array $\mathbf{M}^{(e)}_{qc,\text{disp}}(k, kl)$. The null space of this masked array gives a regular QC-LDPC code if masking is uniform; otherwise, it is an irregular QC-LDPC code with non-uniform masking.

12.4 Construction of Type-II Bose BIBD-LDPC Codes by Dispersion

Since the class-I and -II Bose BIBDs given in Sections 12.2.1 and 12.3.1 are constructed using prime fields, RC-constrained arrays of CPMs in the forms given by (12.6) and (12.13) can also be obtained by the additive-dispersion technique presented in Section 11.6. For the explanation of array and code construction, we use the class-II Bose BIBDs given in Section 12.3.

Let t be a positive integer such that $20t + 1$ is a prime and the condition of (12.9) holds. Then there exists a class-II $(20t + 1, t(20t + 1), 5, 5t, 1)$ Bose BIBD constructed from the prime field $\text{GF}(20t + 1)$. Using this Bose BIBD, we form

the following $t \times 5$ matrix over GF$(20t+1)$ with the elements of the base blocks $B_{0,0}, B_{1,0}, \ldots, B_{t-1,0}$ (given by (12.10)) as rows:

$$\mathbf{W}_{\text{BIBD}} = \begin{bmatrix} \alpha^0 & \alpha^{4t} & \alpha^{8t} & \alpha^{12t} & \alpha^{16t} \\ \alpha^2 & \alpha^{2+4t} & \alpha^{2+8t} & \alpha^{2+12t} & \alpha^{2+16t} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \alpha^{2(t-1)} & \alpha^{2(t-1)+4t} & \alpha^{2(t-1)+8t} & \alpha^{2(t-1)+12t} & \alpha^{2(t-1)+16t} \end{bmatrix}. \tag{12.20}$$

It follows from the structural property of an $(m, n, g, r, 1)$-BIBD that $\mathbf{W}_{\text{BIBD}}$ satisfies additive row constraints 1 and 2 (given by Lemmas 11.3 and 11.4). Hence, $\mathbf{W}_{\text{BIBD}}$ can be used as a base matrix for dispersion.

On replacing each entry of $\mathbf{W}_{\text{BIBD}}$ by its additive $(20t+1)$-fold matrix dispersion, we obtain the following RC-constrained $t \times 5$ array of $(20t+1) \times (20t+1)$ CPMs over GF(2):

$$\mathbf{M}_{\text{BIBD,disp}} = \begin{bmatrix} \mathbf{M}_{0,0} & \mathbf{M}_{0,1} & \mathbf{M}_{0,2} & \mathbf{M}_{0,3} & \mathbf{M}_{0,4} \\ \mathbf{M}_{1,0} & \mathbf{M}_{1,1} & \mathbf{M}_{1,2} & \mathbf{M}_{1,3} & \mathbf{M}_{1,4} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \mathbf{M}_{t-1,0} & \mathbf{M}_{t-1,1} & \mathbf{M}_{t-1,2} & \mathbf{M}_{t-1,3} & \mathbf{M}_{t-1,4} \end{bmatrix}. \tag{12.21}$$

On taking the transpose of $\mathbf{M}_{\text{BIBD}}$, we obtain the following RC-constrained $5 \times t$ array of CPMs:

$$\mathbf{H}^{(3)}_{\text{BIBD,disp}} = \mathbf{M}^{\text{T}}_{\text{BIBD,disp}} = \begin{bmatrix} \mathbf{A}_{0,0} & \mathbf{A}_{0,1} & \cdots & \mathbf{A}_{0,t-1} \\ \mathbf{A}_{1,0} & \mathbf{A}_{1,1} & \cdots & \mathbf{A}_{1,t-1} \\ \vdots & \vdots & \vdots & \vdots \\ \mathbf{A}_{5,0} & \mathbf{A}_{5,1} & \cdots & \mathbf{A}_{5,t-1} \end{bmatrix}, \tag{12.22}$$

where $\mathbf{A}_{i,j} = [\mathbf{M}_{j,i}]^T$ with $0 \le i < 5$ and $0 \le j < t$. $\mathbf{H}^{(3)}_{\text{BIBD,disp}}$ and $\mathbf{H}^{(2)}_{\text{BIBD,decom}}$ are structurally the same.

The null space of any subarray of $\mathbf{H}^{(3)}_{\text{BIBD,disp}}$ gives a QC-BIBD-LDPC code. Masking a subarray of $\mathbf{H}^{(3)}_{\text{BIBD,disp}}$ also gives a QC-LDPC code.

Similarly, using additive dispersion, we can construct an RC-constrained $4 \times t$ array of $(12t+1) \times (12t+1)$ CPMs based on a class-I $(12t+1, t(12t+1), 4, 4t, 1)$ Bose BIBD constructed from the field GF$(12t+1)$, provided that $12t+1$ is a prime and condition (12.1) holds.

12.5 A Trellis-Based Construction of LDPC Codes

Graphs are not only useful for interpretation of iterative types of decoding as described in Chapter 5, but also form an important combinatorial tool for constructing iteratively decodable codes. A class of iteratively decodable codes constructed from a special type of graphs, called *protographs*, has been presented in Chapter 6. Constructions of LDPC codes with girth 6 or larger based on graphs

can be found in [13–15]. In this section, we present a trellis-based method to construct LDPC codes with large girth progressively from a bipartite graph with a small girth.

12.5.1 A Trellis-Based Method for Removing Short Cycles from a Bipartite Graph

Consider a *simple connected* bipartite graph $\mathcal{G}_0 = (V_1, V_2)$ with two disjoint sets of nodes, $V_1 = \{v_0, v_1, \ldots, v_{n-1}\}$ and $V_2 = \{c_0, c_1, \ldots, c_{m-1}\}$. Any edge in $\mathcal{G}_0$ connects a node in V_1 and a node in V_2. Since it is simple, there are no multiple edges between a node in V_1 and a node in V_2. Let λ_0 be the girth of $\mathcal{G}_0$. Form a trellis $\mathcal{T}$ of λ_0 sections with $\lambda_0 + 1$ levels of nodes, labeled from 0 to λ_0 [16]. Recall that the girth of a bipartite graph is even. For $0 \leq l \leq \lambda_0/2$, the nodes at the $2l$th level of $\mathcal{T}$ are the nodes in V_1, and the nodes at the $(2l+1)$th level of $\mathcal{T}$ are the nodes in V_2. Two nodes v_i and c_j at two consecutive levels are connected by a *branch* if and only if (v_i, c_j) is an edge in $\mathcal{G}_0$. Every section of $\mathcal{T}$ is simply a representation of the bipartite graph $\mathcal{G}_0$. Therefore, $\mathcal{T}$ is simply a repetition of $\mathcal{G}_0$ λ_0 times, and every section is the *mirror image* of the preceding section. For example, consider the bipartite graph shown in Figure 12.7. A four-section trellis constructed from this bipartite graph is shown in Figure 12.8.

The nodes at the 0th level of $\mathcal{T}$ are called the *initial nodes* and the nodes at the λ_0th level are called the *terminal nodes*. The initial nodes and terminal nodes of $\mathcal{T}$ are identical and they are nodes in V_1. For $0 \leq i < n$, an *elementary* (i, i)-path of $\mathcal{T}$ is defined as a sequence of λ_0 connected branches starting from the initial node v_i at the 0th level of $\mathcal{T}$ and ending at the terminal node v_i at the λ_0th level of $\mathcal{T}$ such that the following constraints are satisfied: (1) if (v_j, c_k) is a branch in the sequence, then (c_k, v_j) cannot be a branch in the sequence and vice versa; and (2) every branch in the sequence appears once and only once. Then an elementary (i, i)-path represents a cycle of length λ_0 in $\mathcal{G}_0$ starting and ending at node v_i; and, conversely, a cycle of length λ_0 in $\mathcal{G}_0$ starting and ending at node v_i is represented

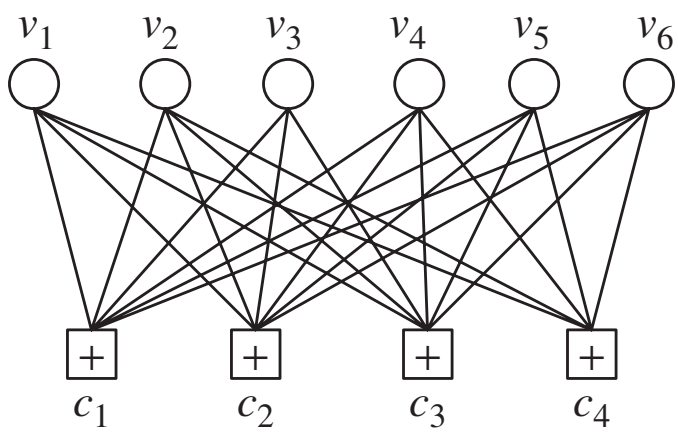


Figure 12.7 A simple bipartite graph with girth 4.

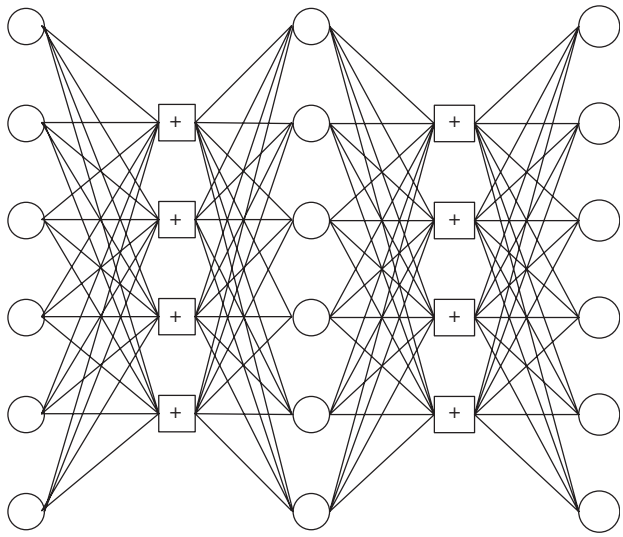


Figure 12.8 A four-section trellis constructed from the bipartite graph shown in Figure 12.7.

by an elementary (i, i)-path in $\mathcal{T}$. For an elementary (i, i)-path in $\mathcal{T}$, if we remove the last branch that connects a node c_k at the $(\lambda_0 - 1)$th level of $\mathcal{T}$ to the terminal node v_i at the λ_0th level of $\mathcal{T}$, we break all those cycles of length λ_0 in $\mathcal{G}_0$ that contain (c_k, v_i) as an edge. On this basis, we can develop a procedure to process $\mathcal{T}$ and to identify all the elementary paths in $\mathcal{T}$. Once all the elementary paths in $\mathcal{T}$ have been identified, we remove their last branches systematically to break all the cycles of length λ_0 in $\mathcal{G}_0$. Since branches in the last section of $\mathcal{T}$ are removed to break the cycles of length λ_0 in $\mathcal{G}_0$, the last section of $\mathcal{T}$ at the end of the branch-removal process gives a new bipartite graph $\mathcal{G}_1$ with a girth of $\lambda_0 + 2$.

To identify the elementary (i, i)-path for all i, we process the trellis $\mathcal{T}$ level by level [14]. Suppose we have processed the trellis $\mathcal{T}$ up to level l with $0 \leq l < \lambda_0$. For every node v_j (or c_j) at the lth level, we list all the *partial elementary paths* (PEPs) of length l that originate from the initial node v_i at the 0th level and terminate at v_j (or c_j), denoted by $\text{PEP}(i, j, l)$. We extend each partial elementary path on the list $\text{PEP}(i, j, l)$ to the $(l + 1)$th level of $\mathcal{T}$ through every branch diverging from v_j (or c_j) that does not appear in any partial elementary path on the list $\text{PEP}(i, j, l)$ before v_j (or c_j). For each node v_j (or c_j) at the $(l + 1)$th level of $\mathcal{T}$, we again form a list $\text{PEP}(i, j, l + 1)$ of all the partial elementary paths of length $l + 1$ that originate from the initial node v_i and terminate at v_j (or c_j). At the $(2m - 1)$th level of $\mathcal{T}$ with $0 < m < \lambda_0/2$, a partial elementary path originating from the initial node v_i at the 0th level of $\mathcal{T}$ and ending at a node c_j in the $(2m - 1)$th level of $\mathcal{T}$ cannot be extended to the node v_i at the $2m$th level through the branch (c_j, v_i), because this would create a cycle of length less than λ_0 that does not exist in $\mathcal{G}_0$ (since λ_0 is the shortest length of a cycle in $\mathcal{G}_0$). Continue the above *extend-and-list* (EAL) process until the $(\lambda_0 - 1)$th level of $\mathcal{T}$ is reached. For each note c_j at the $(\lambda_0 - 1)$th level of $\mathcal{T}$, all the partial elementary paths that originate from v_i and terminate at

c_j are extended through the branch (c_j, v_i), if it exists, to the terminal node v_i at the λ_0th level of $\mathcal{T}$. This extension gives the list $\text{PEP}(i, i, \lambda_0)$ of all the elementary (i, i)-paths in $\mathcal{T}$. The union of all the $\text{PEP}(i, i, \lambda_0)$ lists at the λ_0th level of $\mathcal{T}$ forms a table with all the elementary (i, i, λ_0)-paths in $\mathcal{T}$, which is denoted by $\text{EPT}(\mathcal{T})$ and called the *elementary path table* of $\mathcal{T}$.

The elementary paths of $\mathcal{T}$ give all the cycles of length λ_0 in $\mathcal{G}_0$. The next step is to break all the cycles of length λ_0 in $\mathcal{G}_0$ by removing the last branches of some or all of the elementary (i, i)-paths in $\text{EPT}(\mathcal{T})$. Let c_j be the node at the $(\lambda_0 - 1)$th level of $\mathcal{T}$ that has the largest degree and is connected to the terminal node v_i of the largest degree at the λ_0th level of $\mathcal{T}$. Remove the branch (c_j, v_i) from the last section of $\mathcal{T}$. Removal of (c_j, v_i) breaks all the elementary (i, i)-paths in $\mathcal{T}$ with (c_j, v_i) as the last branch and hence breaks all the cycles of length λ_0 in $\mathcal{G}_0$ with v_i as the starting and ending nodes that contain (c_j, v_i) as an edge. The degrees of v_i and c_j are reduced by 1. Removing the branch (c_j, v_i) from $\mathcal{T}$ also breaks all the other elementary (k, k)-paths in $\mathcal{T}$ that have (c_j, v_i) as a branch. These elementary (i, i)- and (k, k)-paths are then removed from $\text{EPT}(\mathcal{T})$. Removal of the branch (c_j, v_i) results in a new last section of $\mathcal{T}$. We repeat the above branch-removal process until all the elementary paths have been removed from $\text{EPT}(\mathcal{T})$. As a result, we break all the cycles of length λ_0 in the bipartite graph $\mathcal{G}_0$. Then the last section of the new trellis gives a pruned bipartite graph $\mathcal{G}_1$ with girth at least $\lambda_1 = \lambda_0 + 2$.

By applying the above cycle-removal process to the new bipartite graph $\mathcal{G}_1$, we can construct another new bipartite graph $\mathcal{G}_2$ with girth $\lambda_2 = \lambda_0 + 4$. We continue this process until we have obtained a bipartite graph with the desired girth or the degrees of some nodes in V (or S) have become too small.

12.5.2 Code Construction

The trellis-based cycle-removal process can be used either to construct LDPC codes or to improve the error performance of existing codes by increasing their girth [14]. We begin with a bipartite graph $\mathcal{G}_0$ with girth λ_0, which is either the Tanner graph of an existing LDPC code $\mathcal{C}_0$ or a chosen bipartite graph. We apply the cycle-removal process repeatedly. For $1 \leq k$, at the end of the kth application of the cycle-removal process, we obtain a new bipartite graph $\mathcal{G}_k = \left(V_1^{(k)}, V_2^{(k)}\right)$ with girth $\lambda_k = \lambda_0 + 2k$. Then we construct the incidence matrix $\mathbf{H}_k = [h_{i,j}]$ of $\mathcal{G}_k$ whose rows correspond to the nodes in $V_2^{(k)}$ and whose columns correspond to the nodes in $V_1^{(k)}$, where $h_{i,j} = 1$ if and only if the node c_i in $V_2^{(k)}$ and the node v_j in $V_1^{(k)}$ are connected by an edge in $\mathcal{G}_k$. Then the null space of $\mathbf{H}_k$ gives an LDPC code $\mathcal{C}_k$. Next, we compute the error performance of $\mathcal{C}_k$ with iterative decoding using the SPA. We compare the error performance of $\mathcal{C}_k$ with that of $\mathcal{C}_{k-1}$. If $\mathcal{C}_k$ performs better than $\mathcal{C}_{k-1}$, we continue the cycle-removal process; otherwise, we stop the cycle-removal process and $\mathcal{C}_{k-1}$ becomes the end code.

As the cycle-removal process continues, the degrees of variable nodes (also check nodes) of the resultant Tanner graph become smaller. At a certain point, when

there are too many nodes of small degrees, especially nodes of degree 2, the error performance of the resultant LDPC code starts to degrade and a high error floor appears. A general guide is that one should stop the cycle-removal process when the number of variable nodes with degree 2 becomes greater than the number of check nodes.

Example 12.7. Consider the three-dimensional Euclidean geometry EG(3,2^3). A 2-flat in EG(3,2^3) consists of 64 points (see Chapter 2). There are 511 2-flats in EG(3,2^3) not containing the origin of the geometry. The incidence vectors of these 511 2-flats not containing the origin form a single 511×511 circulant $\mathbf{G}$ with both column weight and row weight 64. The Tanner graph of $\mathbf{G}$ contains 3 605 616 cycles of length 4. We can decompose $\mathbf{G}$ into a 4×8 array $\mathbf{H}_0$ of 511×511 circulants with column and row decompositions (see Section 10.5), each with column weight and row weight 2. $\mathbf{H}_0$ is a matrix over GF(2) with column and row weights 8 and 16, respectively. The Tanner graph $\mathcal{G}_0$ of $\mathbf{H}_0$ has a girth of 4 and contains 13 286 cycles of length 4. The null space of $\mathbf{H}_0$ gives a (4088,2046) code $\mathcal{C}_0$ with rate 0.5009 whose error performance is shown in Figure 12.9. At a BER of 10^{-6}, it performs 3.4 dB from the Shannon limit. On starting from $\mathcal{G}_0$ and applying the cycle-removal process repeatedly, we obtain three bipartite graphs $\mathcal{G}_1$, $\mathcal{G}_2$, and $\mathcal{G}_3$ with girths 6, 8, and 10, respectively. The null spaces of the incidence matrices $\mathbf{H}_1$, $\mathbf{H}_2$, and $\mathbf{H}_3$ of $\mathcal{G}_1$, $\mathcal{G}_2$, and $\mathcal{G}_3$ give three rate-1/2 (4088,2044) LDPC codes, $\mathcal{C}_1$, $\mathcal{C}_2$, and $\mathcal{C}_3$, respectively. The error performances of these three LDPC codes are also shown in Figure 12.9. We see that the error performances of the codes are improved as the girth increases from 4 to 10. However, as the girth is increased from 10 to 12, the

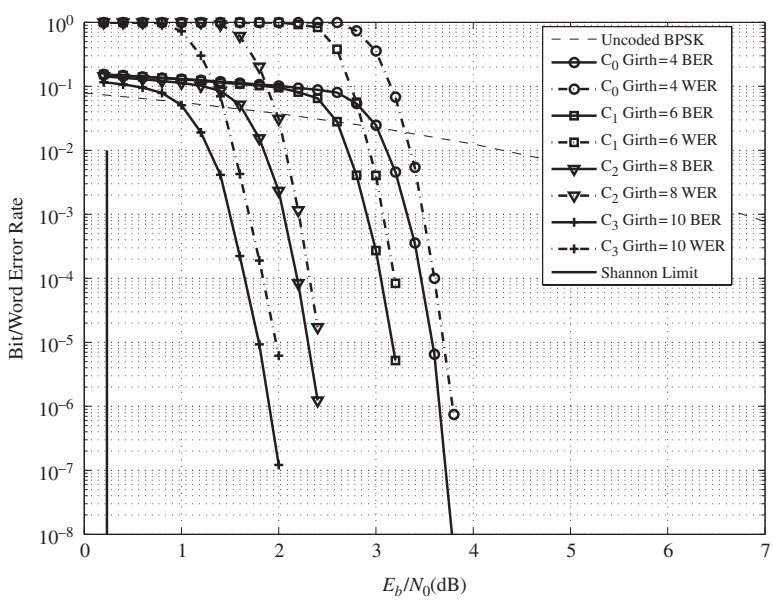


Figure 12.9 The error performances of the LDPC codes given in Example 12.7.

error performance of code $\mathcal{C}_4$ becomes poor and has a high error floor. So, $\mathcal{C}_3$ is the end code in our code-construction process. At a BER of 10^{-6}, $\mathcal{C}_3$ performs only 1.6 dB from the Shannon limit.

The above trellis-based method for constructing LDPC codes of moderate lengths with large girth is quite effective. However, it becomes ineffective when the bipartite graph to be processed becomes too big.

12.6 Construction of LDPC Codes Based on Progressive Edge-Growth Tanner Graphs

In the previous section, a trellis-based method for constructing LDPC codes with large girth was presented. The construction begins with a given simple connected bipartite (or Tanner) graph and then short cycles are progressively removed to obtain an end bipartite graph with a desired girth. From this end bipartite graph, we construct its incidence matrix and use it as the parity-check matrix to generate an LDPC code. In this section, a different graph-based method for constructing LDPC codes with large girth is presented. This construction method, proposed in [13], is simply the opposite of the trellis-based method. Construction begins with a set of n variable nodes and a set of m check nodes with no edges connecting nodes in one set to nodes in the other, i.e., a bipartite graph without edges. Then edges are progressively added to connect variable nodes and check nodes by applying a set of rules and a given variable-node degree profile (or sequence). Edges are added to a variable node one at a time using an *edge-selection procedure* until the number of edges added to a variable node is equal to its specified degree. This progressive addition of edges to a variable node is called *edge-growth*. Edge-growth is performed one variable node at a time. After the completion of edge-growth of one variable node, we move to the next variable node. Edge-growth moves from variable nodes of the smallest degree to variable nodes of the largest degree. When all the variable nodes have completed their edge-growth, we obtain a Tanner graph whose variable nodes have the specified degrees. The edge-selection procedure is devised to maximize the girth of the end Tanner graph.

Before we present the edge-growth procedure used to construct Tanner graphs with large girth, we introduce some concepts. Consider a bipartite graph $\mathcal{G} = (V_1, V_2)$ with variable-node set $V_1 = \{v_0, v_1, \ldots, v_{n-1}\}$ and check-node set $V_2 = \{c_0, c_1, \ldots, c_{m-1}\}$. Consider the variable node v_i. Let λ_i be the length of the shortest cycle that passes through (or contains) v_i. This length λ_i of the shortest cycle passing through v_i is called the *local girth* of v_i. Then the girth λ of $\mathcal{G}$ is given by $\lambda = \min\{\lambda_i : 0 \leq i < n\}$. The edge-growth procedure for constructing a Tanner graph is devised to maximize the local girth of every variable node (a greedy algorithm).

For a given variable node v_i in a Tanner graph $\mathcal{G}$, let $N_i^{(l)}$ denote the set of check nodes in $\mathcal{G}$ that are connected to v_i within a distance $2l + 1$. The distance between two nodes in a graph is defined as the length of a *shortest path* between the two nodes (see Chapter 2). The shortest paths that connect v_i to the check

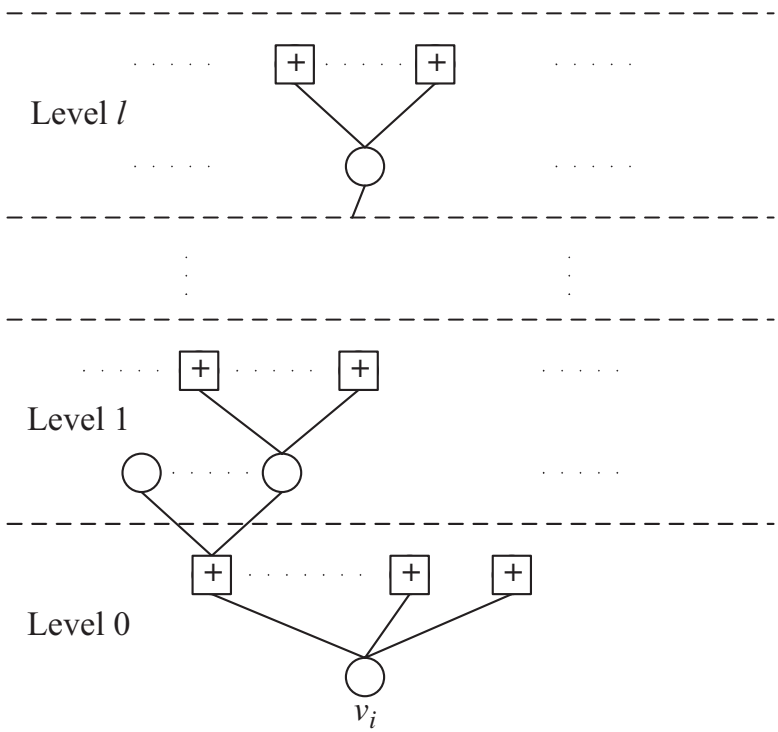


Figure 12.10 Tree representation of the neighborhood within depth l of a variable node.

nodes in $N_i^{(l)}$ can be represented by a tree $\mathcal{R}_i$ with v_i as the root as shown in Figure 12.10. This tree consists of $l+1$ levels and each level consists of two layers of nodes, a layer of variable nodes and a layer of check nodes. We label the levels from 0 to l. Each path in $\mathcal{R}_i$ consists of a sequence of alternate variable and check nodes; and each path in $\mathcal{R}_i$ terminates at a check node in $N_i^{(l)}$. Every node on a path in $\mathcal{R}_i$ appears once and only once. This path tree $\mathcal{R}_i$ with variable node v_i as its root can be constructed progressively. Starting from the variable node v_i, we transverse all the edges, $(v_i, c_{i_1}), (v_i, c_{i_2}), \ldots, (v_i, c_{i_{d_{v_i}}})$, that connect v_i to its d_{v_i} *nearest-neighbor* check nodes, $c_{i_1}, c_{i_2}, \ldots, c_{i_{d_{v_i}}}$. This results in the 0th level of the *path* tree $\mathcal{R}_i$. Next, we transverse all the edges that connect the check nodes in the 0th level to their respective nearest-neighbor variable nodes in the first level. Then we transverse all the edges that connect the variable nodes at the first layer of the first level of the tree $\mathcal{R}_i$ to their respective nearest-neighbor check nodes in the second layer of the first level. This completes the first level of the tree $\mathcal{R}_i$. The transversing process continues level by level until the lth level or a level from which the tree cannot grow further has been reached. It is clear that the distance between v_i and a variable node in the lth level is $2l$ and the distance between v_i and a check node in the lth level is $2l+1$. If there is an edge in $\mathcal{G}$ that connects a check node in the lth level of the tree $\mathcal{R}_i$ to the root variable node v_i, then adding this edge between the check node and v_i creates a cycle of length $2(l+1)$ in $\mathcal{G}$ that

passes through v_i. The set $N_i^{(l)}$ is referred to as the *neighborhood within depth* l of v_i. Let $\bar{N}_i^{(l)}$ be the *complementary set* of $N_i^{(l)}$, i.e., $\bar{N}_i^{(l)} = V_2 \backslash N_i^{(l)}$.

Using the concepts of the local girth of a variable node v_i, the neighborhood $N_i^{(l)}$ within depth l of v_i, and the tree representation of the paths that connect v_i to the check nodes in $N_i^{(l)}$, a progressive edge-growth (PEG) procedure was devised by Hu *et al.* [13] to construct Tanner graphs with relatively large girth. Suppose that, for the given number n of variable nodes, the given number m of check nodes, and the degree profile $D_v = (d_0, d_1, \ldots, d_{n-1})$ of variable nodes, we have completed the edge-growth of the first i variable-nodes, $v_0, v_1, \ldots, v_{i-1}$ with $1 \leq i \leq n$, using the PEG procedure. At this point, we have a *partially connected* Tanner graph. Next, we grow the edges incident from the ith variable node v_i to d_{v_i} check nodes. The growth is done one edge at a time. Suppose $k-1$ edges have been added to v_i with $1 \leq k \leq d_{v_i}$. Before the kth edge is added to v_i, we construct a path tree $\mathcal{R}_i$ with v_i as the root, as shown in Figure 12.10, based on the current partially connected Tanner graph, denoted $\mathcal{G}_{i,k-1}$. We keep growing the tree until it reaches a level, say the lth level, such that one of the following two situations occurs: (1) the tree cannot grow further but the cardinality $|N_i^{(l)}|$ of $N_i^{(l)}$ is smaller than m; or (2) $\bar{N}_i^{(l)} \neq \emptyset$ but $\bar{N}_i^{(l+1)} = \emptyset$. The first situation implies that not all check nodes can be reached from v_i under the current partially connected graph $\mathcal{G}_{i,k-1}$. In this case, we choose a check node c_j in $\bar{N}_i^{(l)}$ with the smallest degree and connect v_i and c_i with a new edge. This prevents creating an additional cycle passing through v_i. The second situation implies that all the check nodes are reachable from v_i. In this case, we choose the check node at the $(l+1)$th level that has the largest distance from v_i. Then add an edge between this chosen check node and v_i. Adding such an edge creates a cycle of length $2(l+2)$. By doing this, we maximize the local girth λ_i of v_i.

The PEG procedure for constructing a Tanner graph with maximized local girth can be put into an algorithm, called the *PEG algorithm* [13], as follows:

PEG Algorithm:
for $i = 0$ to $n-1$, **do**
begin
  for $k = 0$ to $d_{v_i} - 1$, **do**
  begin
    if $k = 0$, **then**
      $\mathcal{E}_i^{(0)} \leftarrow$edge$(v_i, c_j)$, where $\mathcal{E}_i^{(0)}$ is the first edge incident to v_i and c_j is a check node that has the lowest degree in the current partially connected Tanner graph.
    else
      grow a path tree $\mathcal{R}_i$ with v_i as the root to a level, say the lth level, based on the current partially connected Tanner graph such that either the cardinality of $N_i^{(l)}$ stops increasing (i.e., the path tree cannot grow further) but is less than m, or $\bar{N}_i^{(l)} \neq \emptyset$ but $\bar{N}_i^{(l+1)} = \emptyset$, then $\mathcal{E}_i^{(k)} \leftarrow (v_i, c_j)$, where $\mathcal{E}_i^{(k)}$ is the

kth edge incident to v_i and c_j is a check node chosen from $\bar{N}_i^{(l)}$ that has the lowest degree.

end

end

Unlike the trellis-based method for constructing Tanner graphs, the PEG algorithm cannot predict the girth of the end Tanner graph, so it is known only at the end of the implementation of the algorithm.

Example 12.8. The following degree distributions of variable and check nodes of a Tanner graph are designed for a rate-1/2 LDPC code of length 4088:

$$\tilde{\lambda}(X) = 0.451X + 0.3931X^2 + 0.1558X^9,$$
$$\tilde{\rho}(X) = 0.7206X^6 + 0.2994X^7.$$

From these degree distributions, using the PEG algorithm we constructed a rate-1/2 (4088,2044) irregular LDPC code whose Tanner graph has a girth of 8. The error performance of this code is shown in Figure 12.11.

The PEG algorithm given above can be improved to construct irregular LDPC codes with better performance in the high-SNR region without performance degradation in the low-SNR region. An improved PEG algorithm was presented in [15]. To present this improved PEG algorithm, we first introduce a new concept. A cycle

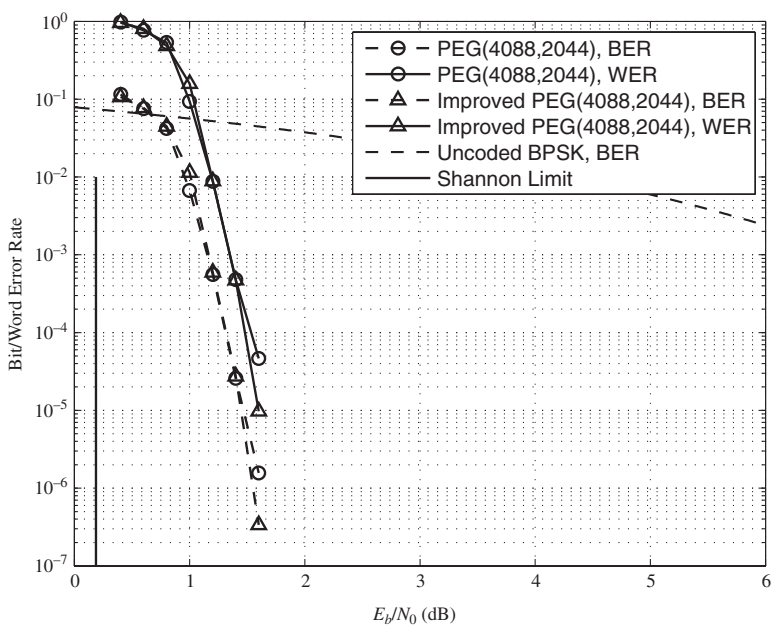


Figure 12.11 The error performance of the (4088,2044) irregular LDPC code given in Example 12.8.

$\mathcal{C}$ of length $2t$ in a Tanner graph $\mathcal{G}$ consists of t variable nodes and t check nodes. Let

$$\epsilon = \sum_{i=1}^{t}(d_i - 2), \tag{12.23}$$

where d_i is the degree of the ith variable node on $\mathcal{C}$. The sum ϵ is simply the number of edges that connect the cycle $\mathcal{C}$ to the rest of the Tanner graph, outside the cycle. This sum ϵ is a measure of the *connectivity* of cycle $\mathcal{C}$ to the rest of the Tanner graph. The larger this connectivity, the more messages from outside of the cycle $\mathcal{C}$ are available for the variable nodes on $\mathcal{C}$. For this reason, ϵ is called the *approximate cycle extrinsic message degree* (ACE) [17, 18].

The improved PEG algorithm presented in [15] is identical to the PEG algorithm except for the case in which $k \geq 1$, $\bar{N}_i^{(l)} \neq \emptyset$, and $\bar{N}_i^{(l+1)} = \emptyset$, when building the path tree $\mathcal{R}_i$ with v_i as the root. In this case, there may be more than one candidate check node with the smallest degree. Let $\mathcal{V}_{2,i}^{(l,k)}$ denote the set of candidate check nodes in $\bar{N}_i^{(l)}$ with the smallest degree. For any check node $c_j \in \mathcal{V}_{2,i}^{(l,k)}$, there is at least one path of length $2(l+1)+1$ between v_i and c_j, but no shorter path between them. Hence, the placement of edge $\mathcal{E}_i^{(k)}$ between v_i and c_j will create at least one new cycle of length $2(l+2)$, but no shorter cycles. In the PEG algorithm, a check node is chosen randomly from $\mathcal{V}_{2,i}^{(l,k)}$. However, in the improved PEG algorithm presented in [15], a check node $c_{\max}$ is chosen from $\mathcal{V}_{2,i}^{(l,k)}$ such that the new cycles created by adding an edge between v_i and $c_{\max}$ have the largest possible ACE. It was shown in [15] that this modified PEG algorithm results in LDPC codes with better error-floor performance than that of the irregular LDPC codes constructed with the original PEG algorithm presented above. Figure 12.11 also shows the performance of a (4088,2044) LDPC code constructed using the improved PEG algorithm presented in [15]. Clearly, the improvement is at the expense of additional computational complexity, but this applies once only, during the code design.

12.7 Construction of LDPC Codes by Superposition

This section presents a method for constructing long powerful LDPC codes from short and simple LDPC codes. This method includes the classic method for constructing product codes as a special case. We will show that the product of two LDPC codes gives an LDPC code with its minimum distance the product of the minimum distances of the two component codes.

12.7.1 A General Superposition Construction of LDPC Codes

Let $\mathbf{B} = [b_{i,j}]$ be a sparse $c \times t$ matrix over GF(2) that satisfies the RC constraint. The null space of $\mathbf{B}$ gives an LDPC code whose Tanner graph has a girth of at least 6. Let $\mathcal{Q} = \{\mathbf{Q}_1, \mathbf{Q}_2, \ldots, \mathbf{Q}_m\}$ be a class of sparse $k \times n$ matrices over GF(2) that has the following structural properties: (1) each member matrix in $\mathcal{Q}$

satisfies the RC-constraint; and (2) a matrix formed by any two member matrices in $\mathcal{Q}$ arranged either in a row or in a column satisfies the RC-constraint, which is called the *pair-wise RC-constraint*. The pair-wise RC-constraint implies that, for $1 \leq l \leq m$, a matrix formed by taking l member matrices in $\mathcal{Q}$ arranged either in a row or in a column also satisfies the RC-constraint. Since each member matrix in $\mathcal{Q}$ satisfies the RC-constraint, its null space gives an LDPC code.

If we replace each 1-entry in $\mathbf{B}$ by a member matrix in $\mathcal{Q}$ and a 0-entry by a $k \times n$ zero matrix, we obtain a $ck \times tn$ matrix $\mathbf{H}_{\text{sup}}$ over GF(2). For $\mathbf{H}_{\text{sup}}$ to satisfy the RC-constraint, the replacement of the 1-entries in $\mathbf{B}$ by the member matrices in $\mathcal{Q}$ is carried out under the rule that all the 1-entries in a column or in a row must be replaced by distinct member matrices in $\mathcal{Q}$. This replacement rule is called the *replacement constraint*. Since $\mathbf{B}$ and the member matrices in $\mathcal{Q}$ are sparse, $\mathbf{H}_{\text{sup}}$ is also a sparse matrix. $\mathbf{H}_{\text{sup}}$ is simply a $c \times t$ array of $k \times n$ submatrices, each either a member matrix in $\mathcal{Q}$ or a $k \times n$ zero matrix. Since $\mathbf{B}$ satisfies the RC constraint, there are no four 1-entries at the four corners of a rectangle in $\mathbf{B}$. This implies that there are no four member matrices in $\mathcal{Q}$ at the four corners of a rectangle in $\mathbf{H}_{\text{sup}}$, viewed as a $c \times t$ array of $k \times n$ submatrices. Then it follows from the pair-wise RC-constraint on the member matrices in $\mathcal{Q}$ and the constraint on the replacement of the 1-entries in $\mathbf{B}$ by the member matrices in $\mathcal{Q}$ that $\mathbf{H}_{\text{sup}}$ satisfies the RC-constraint. Hence, the null space of $\mathbf{H}_{\text{sup}}$ gives an LDPC code $\mathcal{C}_{\text{sup}}$ of length tn with rate at least $(tn - ck)/(tn)$.

The above construction of LDPC codes is referred to as the *superposition construction* [19,20]. The parity-check matrix $\mathbf{H}_{\text{sup}}$ is obtained by superimposing the member matrices in $\mathcal{Q}$ onto the matrix $\mathbf{B}$. The subscript "sup" of $\mathbf{H}_{\text{sup}}$ stands for "superposition." The $\mathbf{B}$ matrix is referred to as the *base matrix* for superposition, and the member matrices in $\mathcal{Q}$ are called the *constituent matrices*. If the base matrix $\mathbf{B}$ has constant column and row weights $w_{b,c}$ and $w_{b,r}$, respectively, and each constituent matrix in $\mathcal{Q}$ has the same constant column and row weights $w_{\text{con},c}$ and $w_{\text{con},r}$, respectively, then $\mathbf{H}_{\text{sup}}$ has constant column and row weights $w_{b,c}w_{\text{con},c}$ and $w_{b,r}w_{\text{con},r}$, respectively. In this case, the null space of $\mathbf{H}_{\text{sup}}$ gives a regular LDPC code with minimum distance at least $w_{b,c}w_{\text{con},c} + 1$. The subscripts "$b$," "$c$," "$r$," and "con" of $w_{b,c}$, $w_{b,r}$, $w_{\text{con},c}$, and $w_{\text{con},r}$ stand for "base," "column," "row," and "constituent," respectively.

If all the constituent matrices in $\mathcal{Q}$ are arrays of CPMs of the same size, then $\mathbf{H}_{\text{sup}}$ is an array of circulant permutation and zero matrices. In this case, the null space of $\mathbf{H}_{\text{sup}}$ gives a QC-LDPC code. Any array $\mathbf{H}^{(e)}_{qc,\text{disp}}$ with $1 \leq e \leq 6$ constructed in Chapter 11 can be divided into subarrays to form constituent matrices in $\mathcal{Q}$.

It is clear that the number of constituent matrices in $\mathcal{Q}$ must be large enough that the parity-check matrix $\mathbf{H}_{\text{sup}}$ can be constructed in such a way as to satisfy the replacement constraint. The assignment of constituent matrices in $\mathcal{Q}$ to the 1-entries in the base matrix $\mathbf{B}$ to satisfy the replacement constraint is equivalent to *coloring the edges* of the Tanner graph of $\mathbf{B}$ (a bipartite graph) such that no two adjacent edges have the same color [21,22]. It is known in graph theory that, for any bipartite graph, the minimum number of colors that is sufficient to achieve

the coloring constraint is equal to the maximum node degree of the graph [22]. Suppose $\mathbf{B}$ has constant column and row weights $w_{b,c}$ and $w_{b,r}$, respectively. Let $w_{b,\max} = \max\{w_{b,c}, w_{b,r}\}$. Then $w_{b,\max}$ constituent matrices in $\mathcal{Q}$ will suffice to satisfy the replacement constraint.

If $\mathbf{B}$ is a $t \times t$ circulant with row weight $w_{b,r}$, then the replacement of the 1-entries in $\mathbf{B}$ can be carried out in a cyclic manner. First, we replace the $w_{b,r}$ 1-entries in the top row of $\mathbf{B}$ by $w_{b,r}$ distinct constituent matrices in $\mathcal{Q}$ and the $t - w_{b,r}$ 0-entries by $t - w_{b,r}$ zero matrices of size $k \times n$. This results in a row of t submatrices of size $k \times n$. Then this row and its $t - 1$ right cyclic-shifts (with each $k \times n$ submatrix as a shifting unit) give a $tk \times tn$ superimposed matrix $\mathbf{H}_{\text{sup}}$ that satisfies the replacement constraint. This replacement of 1-entries in $\mathbf{B}$ by the constituent matrices in $\mathcal{Q}$ is called *cyclic replacement*. If $\mathbf{B}$ is an array of circulants, then cyclic replacement is applied to each circulant in $\mathbf{B}$.

If each constituent matrix in $\mathcal{Q}$ is a row of permutation (or circulant permutation) matrices, then, in replacing the 1-entries in $\mathbf{B}$ by constituent matrices in $\mathcal{Q}$, only the requirement that all the 1-entries in a column of $\mathbf{B}$ be replaced by distinct constituent matrices in $\mathcal{Q}$ is sufficient to guarantee that $\mathbf{H}_{\text{sup}}$ satisfies the RC constraint. This is due to the fact that a row of permutation matrices will satisfy the RC-constraint, no matter whether the permutation matrices in the row are all distinct or not.

If each constituent matrix in $\mathcal{Q}$ is a permutation matrix, then it is not necessary to follow replacement rules at all. In this case, if the base matrix $\mathbf{B}$ satisfies the RC-constraint, the superimposed matrix $\mathbf{H}_{\text{sup}}$ also satisfies the RC-constraint.

12.7.2 Construction of Base and Constituent Matrices

RC-constrained base matrices and pair-wise RC-constrained constituent matrices can be constructed using finite geometries, finite fields, or BIBDs.

Consider the m-dimensional Euclidean geometry EG(m,q) over GF(q). As shown in Section 10.1 (or Section 2.7.1), $K_c = q^{m-1}$ circulants of size $(q^m - 1) \times (q^m - 1)$ can be constructed from the type-1 incidence vectors of the lines in EG(m,q) not passing through the origin of the geometry. Each of these circulants has weight q (i.e., both the column weight and the row weight are q). These circulants satisfy the pair-wise RC-constraints. One or a group of these circulants arranged in a row (see (10.12)) can be used as a base matrix. If the weight q of a circulant is too large, it can be decomposed into a row of column descendant circulants of the same size with smaller weights by column splitting as shown in Section 10.5. These column descendant circulants also satisfy the pair-wise RC-constraint. One or a group of these column descendant circulants arranged in a row can be used as a base matrix.

The circulants constructed from EG(m,q) can be decomposed into a class of column (or row) descendant circulants by column (or row) splitting. These column (or row) descendants can be grouped to form a class of pair-wise RC-constraint constituent matrices for superposition. This is best explained by an example.

Example 12.9. Consider the two-dimensional Euclidean geometry EG(2,3) over GF(3). The type-1 incidence vectors of the lines in EG(2,3) not passing through the origin form a single 8×8 circulant matrix $\mathbf{B}$ with both column weight and row weight 3 that satisfies the RC constraint. We use this circulant as the base matrix for superposition code construction. Next we consider the three-dimensional Euclidean geometry EG($3,2^3$) over GF(2^3). Nine 511×511 circulants, $\mathbf{G}_1, \mathbf{G}_2, \ldots, \mathbf{G}_9$, can be constructed via the type-1 incidence vectors of the lines in EG($3,2^3$) not passing through the origin (see Example 10.2). Each of these circulants has both column weight and row weight 8. For $0 \leq i \leq 9$, we decompose $\mathbf{G}_i$ into eight 511×511 CPMs, $\mathbf{G}_{i,1}, \mathbf{G}_{i,2}, \ldots, \mathbf{G}_{i,8}$, by column decomposition. Using these eight CPMs, we form four 511×1002 matrices, $\mathbf{Q}_{i,1} = [\mathbf{G}_{i,1}\mathbf{G}_{i,2}]$, $\mathbf{Q}_{i,2} = [\mathbf{G}_{i,3}\mathbf{G}_{i,4}]$, $\mathbf{Q}_{i,3} = [\mathbf{G}_{i,5}\mathbf{G}_{i,6}]$, and $\mathbf{Q}_{i,4} = [\mathbf{G}_{i,7}\mathbf{G}_{i,8}]$. Each $\mathbf{Q}_{i,j}$, $1 \leq j \leq 4$, has column and row weights 1 and 2, respectively. Then $\mathcal{Q} = \{\mathbf{Q}_{i,1}, \mathbf{Q}_{i,2}, \mathbf{Q}_{i,3}, \mathbf{Q}_{i,4} \colon 1 \leq i \leq 9\}$ forms a class of constituent matrices for superposition code construction. To construct the superimposed parity-check matrix $\mathbf{H}_{\text{sup}}$, for $1 \leq i \leq 8$, we replace the three 1-entries of the ith row by three constituent matrices from the group $\{\mathbf{Q}_{i,1}, \mathbf{Q}_{i,2}, \mathbf{Q}_{i,3}, \mathbf{Q}_{i,4}\}$. In the replacement, the three 1-entries in a column must be replaced by three constituent matrices in $\mathcal{Q}$ with three different first indices. For $0 \leq i \leq 8$, the three 1-entries in the ith row of $\mathbf{B}$ are replaced by three constituent matrices with the same first indices i. The replacement results in an RC-constrained 4088×8176 superimposed matrix $\mathbf{H}_{\text{sup}}$ with column and row weights 3 and 6, respectively. It is an 8×16 array of 48 511×511 CPMs and 80 511×511 zero matrices. The null space of $\mathbf{H}_{\text{sup}}$ gives a (3,6)-regular (8176,4088) QC-LDPC code with rate 1/2, whose Tanner graph has a girth of at least 6. The error performance of this code with iterative decoding using the SPA with 100 iterations is shown in Figure 12.12. At a BER of 10^{-6}, it performs 1.5 dB from the Shannon limit.

For $1 \leq e \leq 6$, any RC-constrained array $\mathbf{H}^{(e)}_{qc,\text{disp}}$ of CPMs constructed as in Sections 11.3–11.7 can be partitioned into subarrays of the same size to form a class of constituent matrices for superposition code construction. Suppose the base matrix $\mathbf{B} = [b_{i,j}]$ is an RC-constrained $c \times t$ matrix over GF(2) with column and row weights $w_{b,c}$ and $w_{b,r}$, respectively. Choose two positive integers k and n such that ck and tn are smaller than the number of rows and the number of columns of $\mathbf{H}^{(e)}_{qc,\text{disp}}$, respectively. Take a $ck \times tn$ subarray $\mathbf{H}^{(e)}_{qc,\text{disp}}(ck, tn)$ from $\mathbf{H}^{(e)}_{qc,\text{disp}}$. Divide the $\mathbf{H}^{(e)}_{qc,\text{disp}}(ck, tn)$ horizontally into c subarrays, $\mathbf{Q}_1, \mathbf{Q}_2, \ldots, \mathbf{Q}_c$, where $\mathbf{Q}_i$ consists of the ith group of k consecutive rows of CPMs of $\mathbf{H}^{(e)}_{qc,\text{disp}}(ck, tn)$. For $1 \leq i \leq c$, divide $\mathbf{Q}_i$ vertically into t subarrays, $\mathbf{Q}_{i,1}, \mathbf{Q}_{i,2}, \ldots, \mathbf{Q}_{i,t}$, of the same size $k \times n$. Then the set of $k \times n$ subarrays of $\mathbf{H}^{(e)}_{qc,\text{disp}}(ck, tn)$ given by

$$\mathcal{Q} = \{\mathbf{Q}_{i,j} \colon 1 \leq i \leq c, \quad 1 \leq j \leq t\} \tag{12.24}$$

can be used as constituent matrices for superposition code construction with a $c \times t$ base matrix $\mathbf{B}$. Note that the member matrices in $\mathcal{Q}$ do not exactly satisfy the pair-wise RC-constraint. However, the member matrices in $\mathcal{Q}$ have the following

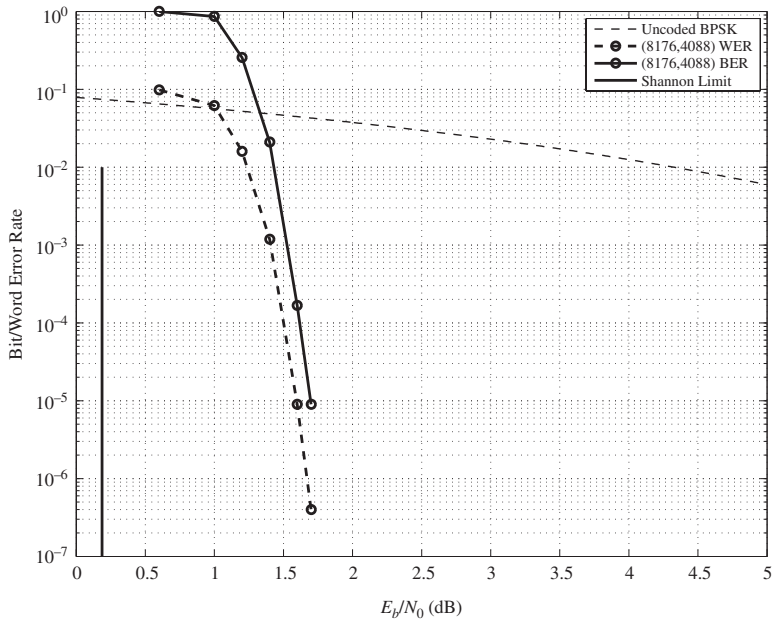


Figure 12.12 The error performance of the (8176,4088) QC-LDPC code given in Example 12.9.

RC-constraint properties: (1) for $j_1 \neq j_2$ any two matrices $\mathbf{Q}_{i,j_1}$ and $\mathbf{Q}_{i,j_2}$ with the same first index i arranged in a row satisfy the RC-constraint; and (2) for $i_1 \neq i_2$ any two matrices $\mathbf{Q}_{i_1,j}$ and $\mathbf{Q}_{i_2,j}$ with the same second index j arranged in a column satisfy the RC-constraint. In replacing the 1-entries of the base matrix $\mathbf{B}$ by the constituent matrices in $\mathcal{Q}$, the replacement is carried out as follows: if the entry $b_{i,j}$ at the ith row and jth column of $\mathbf{B}$ is a 1-entry, it is replaced by the constituent matrix $\mathbf{Q}_{i,j}$ in $\mathcal{Q}$, whereas if $b_{i,j} = 0$, it is replaced by a $k \times n$ array of zero matrices. This replacement results in an RC-constrained $ck \times tn$ array $\mathbf{H}_{\text{sup}}$ of circulant permutation and zero matrices. As a matrix over GF(2), it has column and row weights $kw_{b,c}$ and $nw_{b,r}$, respectively. The null space of $\mathbf{H}_{\text{sup}}$ gives a regular QC-LDPC code whose Tanner graphs has a girth of at least 6.

Example 12.10. Consider the two-dimensional Euclidean geometry EG(2,2^2) over GF(2^2). Using the type-1 incidence vectors of the lines in EG(2,2^2) not passing through the origin, we can construct a single 15×15 circulant over GF(2) with both column weight and row weight 4. We use this circulant as the base matrix $\mathbf{B}$ for superposition code construction. To construct constituent matrices to replace the 1-entries in $\mathbf{B}$, we use the prime field GF(127) to construct a 127×127 array $\mathbf{H}^{(6)}_{qc,\text{disp}}$ of 127×127 CPMs (see Section 11.6). Take a 15×120 subarray $\mathbf{H}^{(6)}_{qc,\text{disp}}(15, 120)$ from $\mathbf{H}^{(6)}_{qc,\text{disp}}$. Choose $k = 1$ and $n = 8$. Using the method described above, we partition $\mathbf{H}^{(6)}_{qc,\text{disp}}(15, 120)$ into the following class of 1×8 subarrays of 127×127 CPMs: $\mathcal{Q} = \{\mathbf{Q}_{i,j} : 1 \leq i \leq 15, 1 \leq j \leq 15\}$. We replace the 1-entries in $\mathbf{B}$ by the constituent arrays in $\mathcal{Q}$ using the replacement rule as described above. The replacement results in a 15×120 superimposed array $\mathbf{H}_{\text{sup}}$ of

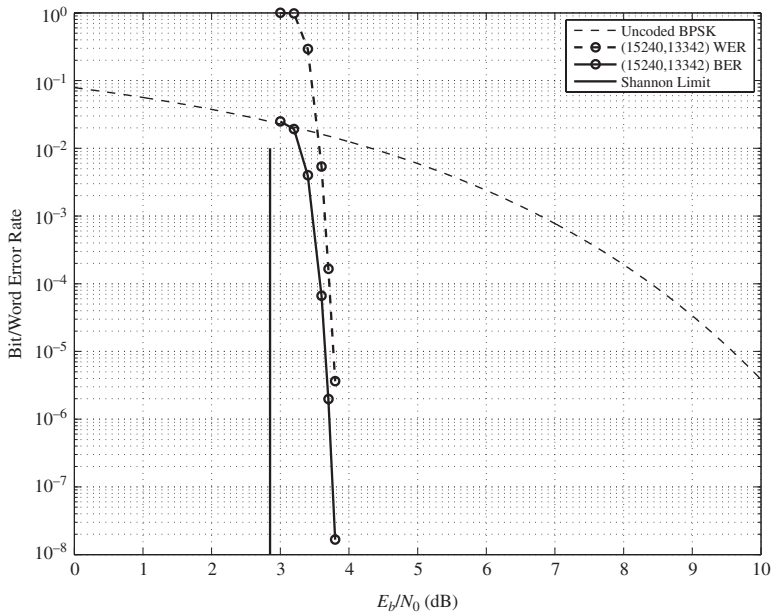


Figure 12.13 The error performance of the (15240,13342) QC-LDPC code given in Example 12.10.

circulant permutation and zero matrices of size 127×127. $\mathbf{H}_{\text{sup}}$ is a 1905×15240 matrix over GF(2) with column and row weights 4 and 32, respectively. The null space of this matrix gives a (4,32)-regular (15240,13342) QC-LDPC code with rate 0.875 whose Tanner graph has a girth of at least 6. The error performance of this code with iterative decoding using the SPA with 50 iterations is shown in Figure 12.13. At a BER of 10^{-6}, the code performs 0.86 dB from the Shannon limit. It has a beautiful waterfall error performance.

The circulants constructed using class-I or class-II Bose BIBDs given in Section 12.2 can also be used to construct base matrices for the superposition construction of LDPC codes. For example, let $t = 1$. There is a class-I $(13, 13, 4, 4, 1)$ Bose BIBD whose incidence matrix consists of a single 13×13 circulant $\mathbf{G}$ over GF(2) with both column weight and row weight 4. This circulant can be used as a base matrix for superposition to construct LDPC codes. If we decompose $\mathbf{G}$ into two column descendants, $\mathbf{G}_1$ and $\mathbf{G}_2$, with column splitting such that $\mathbf{G}_1$ is a circulant with both column weight and row weight 3 and $\mathbf{G}_2$ is a CPM. The circulant $\mathbf{G}_1$ can be used as a base matrix for superposition construction of LDPC codes. Suppose we construct a class $\mathcal{Q}$ of pair-wise RC-constrained constituent matrices for which each member constituent matrix consists of a row of l CPMs of size $n \times n$. On replacing each 1-entry of $\mathbf{G}_1$ by a constituent matrix in $\mathcal{Q}$ under the replacement constraint, we obtain a $13 \times 13l$ RC-constrained array $\mathbf{H}_{\text{sup}}$ of $n \times n$ circulant permutation and zero matrices. This array $\mathbf{H}_{\text{sup}}$ is a $13n \times 13ln$ matrix over GF(2) with column and row weights 3 and $3l$, respectively. The null space of $\mathbf{H}_{\text{sup}}$ gives a $(3,3l)$-regular QC-LDPC code of length $13ln$.

12.7.3 Superposition Construction of Product LDPC Codes

For $i = 1$ and 2, let $\mathcal{C}_i$ be an (n_i,k_i) LDPC code with minimum distance d_i that is given by the null space of an $m_i \times n_i$ RC-constrained parity-check matrix $\mathbf{H}_i$ over GF(2). For $i = 2$, express $\mathbf{H}_2$ in terms of its columns,

$$\mathbf{H}_2 = \begin{bmatrix} \mathbf{h}_1^{(2)} & \mathbf{h}_2^{(2)} & \cdots & \mathbf{h}_{n_2}^{(2)} \end{bmatrix}.$$

For $1 \le j \le n_2$, we form the following $m_2 n_1 \times n_1$ matrix over GF(2) by shifting the jth column $\mathbf{h}_j^{(2)}$ downward n_1 times:

$$\mathbf{H}_j^{(2)} = \begin{bmatrix} \mathbf{h}_j^{(2)} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \mathbf{h}_j^{(2)} & \cdots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{h}_j^{(2)} \end{bmatrix}, \tag{12.25}$$

where $\mathbf{0}$ is a column vector of length m_2. Each row of $\mathbf{H}_j^{(2)}$ has at most one 1-component and no two columns have any 1-component in common. It is clear that $\mathbf{H}_j^{(2)}$ satisfies the RC constraint. It follows from the structure of $\mathbf{H}_j^{(2)}$ with $1 \le j \le n_2$ that the $m_2 n_1 \times n_2 n_1$ matrix

$$\mathbf{H}_{2,\text{int}} = \begin{bmatrix} \mathbf{H}_1^{(2)} & \mathbf{H}_2^{(2)} & \cdots & \mathbf{H}_{n_2}^{(2)} \end{bmatrix}$$

is simply obtained by interleaving the columns of $\mathbf{H}^{(2)}$ with a span (or interleaving depth) of n_1. The subscript "int" of $\mathbf{H}_{2,\text{int}}$ stands for "interleaving." Since $\mathbf{H}_2$ satisfies the RC-constraint, it is obvious that $\mathbf{H}_{2,\text{int}}$ also satisfies the RC-constraint. We also note that, for $1 \le j \le n_2$, any row from $\mathbf{H}_1$ and any row from $\mathbf{H}_j^{(2)}$ have no more than one 1-component in common. Since $\mathbf{H}_1$ satisfies the RC-constraint, the matrix formed by $\mathbf{H}_1$ and $\mathbf{H}_j^{(2)}$ arranged in a column satisfies the RC-constraint.

Form the following $(n_2+1) \times n_2$ base matrix for superposition code construction:

$$\mathbf{B} = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \\ \hline 1 & 1 & 1 & \cdots & 1 \end{bmatrix}. \tag{12.26}$$

$\mathbf{B}$ consists of two submatrices, upper and lower ones. The upper submatrix is an $n_2 \times n_2$ identity matrix and the lower submatrix is a $1 \times n_2$ row matrix with n_2 1-components. Let

$$\mathcal{Q} = \{\mathbf{H}_1, \mathbf{H}_1^{(2)}, \mathbf{H}_2^{(2)}, \ldots, \mathbf{H}_{n_2}^{(2)}\}$$

be the class of constituent matrices for superposition code construction.

On replacing each 1-entry in the upper $n_2 \times n_2$ identity matrix of $\mathbf{B}$ by $\mathbf{H}_1$ and the jth 1-entry in the lower submatrix of $\mathbf{B}$ by the $\mathbf{H}_j^{(2)}$ in $\mathcal{Q}$ for $1 \le j \le n_2$, we

obtain the following $(m_1 n_2 + m_2 n_1) \times n_1 n_2$ matrix over GF(2):

$$\mathbf{H}_{\text{sup},p} = \left[\begin{array}{ccccc} \mathbf{H}_1 & 0 & 0 & \cdots & 0 \\ 0 & \mathbf{H}_1 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & \mathbf{H}_1 \\ \hline \mathbf{H}_1^{(2)} & \mathbf{H}_2^{(2)} & \mathbf{H}_3^{(2)} & \cdots & \mathbf{H}_{n_2}^{(2)} \end{array}\right]. \tag{12.27}$$

The matrix $\mathbf{H}_{\text{sup,p}}$ consists of two parts. The upper part of $\mathbf{H}_{\text{sup,p}}$ consists of an $n_2 \times n_2$ array of $m_1 \times n_1$ submatrices with $\mathbf{H}_1$ on its main diagonal and zero matrices elsewhere. The lower part of $\mathbf{H}_{\text{sup},p}$ is simply the matrix $\mathbf{H}_{2,\text{int}}$. It follows from the RC constraint structure of $\mathbf{H}_1$, $\mathbf{H}_2$, $\mathbf{H}_j^{(2)}$ with $1 \leq j \leq n_2$, and $\mathbf{H}_{2,\text{int}}$ that $\mathbf{H}_{\text{sup},p}$ satisfies the RC-constraint.

The null space of $\mathbf{H}_{\text{sup},p}$ gives an LDPC code $\mathcal{C}_{\text{sup},p}$ of length $n_1 n_2$ whose Tanner graph has a girth of at least 6. From the structure of $\mathbf{H}_{\text{sup},p}$, we can readily prove that $\mathbf{H}_{\text{sup},p}$ is a parity-check matrix of the direct product of $\mathcal{C}_1$ and $\mathcal{C}_2$, with $\mathcal{C}_1$ and $\mathcal{C}_2$ as the row and column codes, respectively. Therefore, $\mathcal{C}_{\text{sup},p} = \mathcal{C}_1 \times \mathcal{C}_2$ and the minimum distance of $\mathcal{C}_{\text{sup},p}$ is $d_{\text{sup},p} = d_1 \times d_2$, the product of the minimum distances of $\mathcal{C}_1$ and $\mathcal{C}_2$. The subscript "p" of $\mathbf{H}_{\text{sup},p}$ and $\mathcal{C}_{\text{sup},p}$ stands for "product." If $\mathcal{C}_1$ and $\mathcal{C}_2$ are both QC-LDPC codes, the encoding of $\mathcal{C}_{\text{sup},p}$ can be done with simple shift-registers by encoding the row code and the column code separately. Taking the product of more than two codes can be carried out recursively.

Three methods can be used to decode a product LDPC code $\mathcal{C}_{\text{sup},p}$ with two component codes, $\mathcal{C}_1$ and $\mathcal{C}_2$. The first method decodes the product LDPC code $\mathcal{C}_{\text{sup},p}$ with iterative decoding based on the parity-check matrix $\mathbf{H}_{\text{sup},p}$ of the code given by (12.27). The second method decodes the row code $\mathcal{C}_1$ and then the column code $\mathcal{C}_2$ iteratively, like turbo decoding. The third method is a hybrid decoding. First, we perform turbo decoding based on the two component codes. After a preset number of iterations of turbo decoding, we switch to iterative decoding of the product code $\mathcal{C}_{\text{sup},p}$ based on the parity-check matrix $\mathbf{H}_{\text{sup},p}$ given by (12.27).

Example 12.11. Let $\mathcal{C}_1$ be the (1023,781) cyclic EG-LDPC code constructed from the two-dimensional Euclidean geometry EG(2,2^5) over GF(2^5). The parity-check matrix $\mathbf{H}_1$ of $\mathcal{C}_1$ is a 1023×1023 circulant with both column weight and row weight 32 (see Section 10.1.1 for its construction). The minimum distance of $\mathcal{C}_1$ is exactly 33. Let $\mathcal{C}_2$ be the (32,31) single parity-check code with minimum distance 2. The parity-check matrix $\mathbf{H}_2$ of $\mathcal{C}_2$ is simply a row of 32 1-components. From (12.25), we see that, for $1 \leq j \leq 32$, $\mathbf{H}_j^{(2)}$ is a 1023×1023 identity matrix. It follows from (12.27) that we can form the parity-check matrix $\mathbf{H}_{\text{sup},p}$ of the product LDPC code $\mathcal{C}_{\text{sup},p} = \mathcal{C}_1 \times \mathcal{C}_2$. The null space of $\mathbf{H}_{\text{sup},p}$ gives a (32736,24211) product LDPC code with rate 0.74 with minimum distance 66. The error performance of this product LDPC code with iterative decoding based on its parity-check matrix $\mathbf{H}_{\text{sup},p}$ using the SPA with 100 iterations is shown in Figure 12.14. We see that

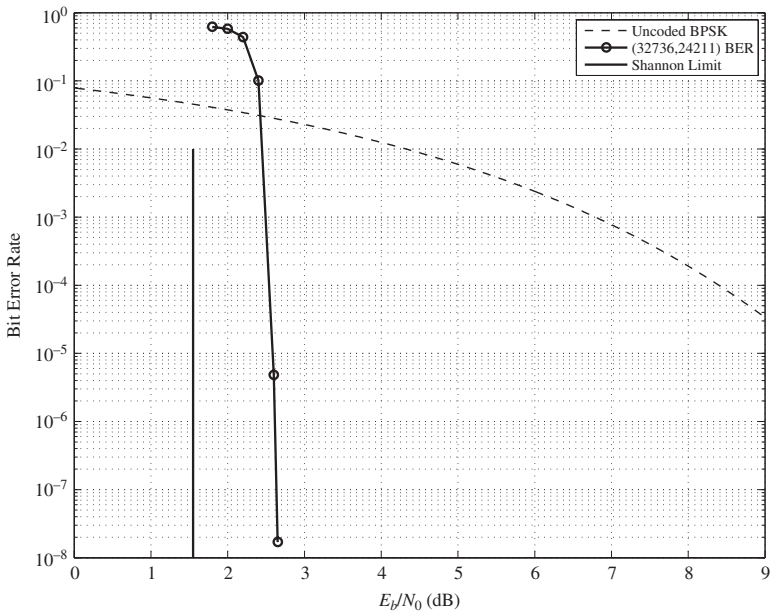


Figure 12.14 The error performance of the (32736,24211) product LDPC code given in Example 12.11.

the code has a beautiful straight-down waterfall error performance. Since the code has a very large minimum distance, it should have a very low error floor.

12.8 Two Classes of LDPC Codes with Girth 8

For $1 \leq e \leq 6$, consider an RC-constrained array $\mathbf{H}^{(e)}_{qc,\text{disp}}$ of $(q-1) \times (q-1)$ CPMs constructed using the Galois field GF(q) (see Sections 11.3–11.7). Let $\mathbf{H}^{(e)}_{qc,\text{disp}}(2,r)$ be a $2 \times r$ subarray of $\mathbf{H}^{(e)}_{qc,\text{disp}}$. The associated Tanner graph of this $2 \times r$ subarray is not only free of cycles of length 4 but is also free of cycles of length 6, since forming a cycle of length 6 requires at least three rows of CPMs. This cycle structure can be used to construct a class of (3,r)-regular QC-LDPC codes whose Tanner graphs have a girth of 8, using the superposition code-construction method.

Take a $2k \times r$ subarray $\mathbf{H}^{(e)}_{qc,\text{disp}}(2k,r)$ subarray from $\mathbf{H}^{(e)}_{qc,\text{disp}}$ with $2k$ and r smaller than the numbers of rows and columns of $\mathbf{H}^{(e)}_{qc,\text{disp}}$, respectively. Assume that $\mathbf{H}^{(e)}_{qc,\text{disp}}(2k,r)$ contains no zero submatrix. Slice $\mathbf{H}^{(e)}_{qc,\text{disp}}(2k,r)$ horizontally into k subarrays of size $2 \times r$, $\mathbf{H}^{(e)}_{1,qc,\text{disp}}(2,r), \mathbf{H}^{(e)}_{2,qc,\text{disp}}(2,r), \ldots, \mathbf{H}^{(e)}_{k,qc,\text{disp}}(2,r)$. Let

$$\mathcal{Q} = \{\mathbf{H}^{(e)}_{1,qc,\text{disp}}(2,r), \mathbf{H}^{(e)}_{2,qc,\text{disp}}(2,r), \ldots, \mathbf{H}^{(e)}_{k,qc,\text{disp}}(2,r)\} \tag{12.28}$$

be the class of constituent matrices for superposition code construction. Form a $(k+1) \times k$ base matrix $\mathbf{B}$ of the form given by (12.26). Next, we replace each 1-entry of the upper $k \times k$ identity submatrix of $\mathbf{B}$ by a constituent matrix in $\mathcal{Q}$ and each 1-entry of the lower submatrix of $\mathbf{B}$ by an $r(q-1) \times r(q-1)$ identity matrix. This replacement gives the following $(2k+r)(q-1) \times kr(q-1)$ matrix over GF(2):

$$\mathbf{H}_{\text{sup},p} = \left[\begin{array}{cccc} \mathbf{H}^{(e)}_{1,qc,\text{disp}}(2,r) & \mathbf{O} & \cdots & \mathbf{O} \\ \mathbf{O} & \mathbf{H}^{(e)}_{2,qc,\text{disp}}(2,r) & \cdots & \mathbf{O} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{O} & \mathbf{O} & \cdots & \mathbf{H}^{(e)}_{k,qc,\text{disp}}(2,r) \\ \hline \mathbf{I}_{r(q-1)\times r(q-1)} & \mathbf{I}_{r(q-1)\times r(q-1)} & \cdots & \mathbf{I}_{r(q-1)\times r(q-1)} \end{array}\right], \tag{12.29}$$

where $\mathbf{O}$ is a $2 \times r$ array of $(q-1) \times (q-1)$ zero matrices. The $r(q-1) \times r(q-1)$ identity matrix $\mathbf{I}_{r(q-1)\times r(q-1)}$ can be viewed as an $r \times r$ array of $(q-1) \times (q-1)$ identity and zero matrices, with the $(q-1) \times (q-1)$ identity matrices on the main diagonal of the array and zero matrices elsewhere. Then $\mathbf{H}_{\text{sup},p}$ is a $(2k+r) \times kr$ array of $(q-1) \times (q-1)$ circulant permutation and zero matrices. $\mathbf{H}_{\text{sup},p}$ has column and row weights 3 and r, respectively.

Note that the Tanner graph of the base matrix $\mathbf{B}$ (see (12.26)) is cycle-free. Given the cycle structure of each $2 \times r$ array $\mathbf{H}^{(e)}_{i,qc,\text{disp}}(2,r)$ in $\mathbf{H}_{\text{sup},p}$ and the fact that the Tanner graph of an identity matrix is cycle-free, we can readily prove that the Tanner graph of $\mathbf{H}_{\text{sup},p}$ has a girth of exactly 8. Therefore, the null space of $\mathbf{H}_{\text{sup},p}$ gives a (3,r)-regular QC-LDPC code $\mathcal{C}_{\text{sup},p}$, whose Tanner graph has a girth of 8. From the structure of $\mathbf{H}_{\text{sup},p}$ given by (12.29), we can easily see that no seven or fewer columns of $\mathbf{H}_{\text{sup},p}$ can be added to a zero column vector. Hence, the minimum distance of $\mathcal{C}_{\text{sup},p}$ is at least 8. The above superposition code construction gives a class of QC-LDPC codes. Actually, $\mathcal{C}_{\text{sup},p}$ is a generalized product code with k row codes and one column code. The ith row code is given by the null space of the $2 \times r$ array $\mathbf{H}^{(e)}_{i,qc,\text{disp}}(2,r)$ of $(q-1) \times (q-1)$ CPMs and the column code is simply the $(k, k-1)$ *single-parity-check code* (SPC) whose parity-check matrix $\mathbf{H}_{\text{spc}} = [1 \quad 1 \quad \cdots \quad 1]$ consists a row of k 1-components.

Example 12.12. Consider the 127×127 array $\mathbf{H}^{(1)}_{qc,\text{disp}}$ of 127×127 circulant permutation and zero matrices constructed from the field GF(2^7) using the method given in Section 11.3, where the zero matrices lie on the main diagonal of $\mathbf{H}^{(1)}_{qc,\text{disp}}$. Set $k = r = 6$. Take a 12×6 subarray $\mathbf{H}^{(1)}_{qc,\text{disp}}(12,6)$ from $\mathbf{H}^{(1)}_{qc,\text{disp}}$, avoiding the zero submatrices on the main diagonal of $\mathbf{H}^{(1)}_{qc,\text{disp}}$. Slice $\mathbf{H}^{(1)}_{qc,\text{disp}}(12,6)$ into six 2×6 subarrays, $\mathbf{H}^{(1)}_{1,qc,\text{disp}}(2,6), \mathbf{H}^{(1)}_{1,qc,\text{disp}}(2,6), \ldots, \mathbf{H}^{(1)}_{6,qc,\text{disp}}(2,6)$. We use these 2×6 subarrays for superposition code construction. Construct a 7×6 base matrix $\mathbf{B}$ of the form given by (12.26). Using the above superposition construction method, we construct an

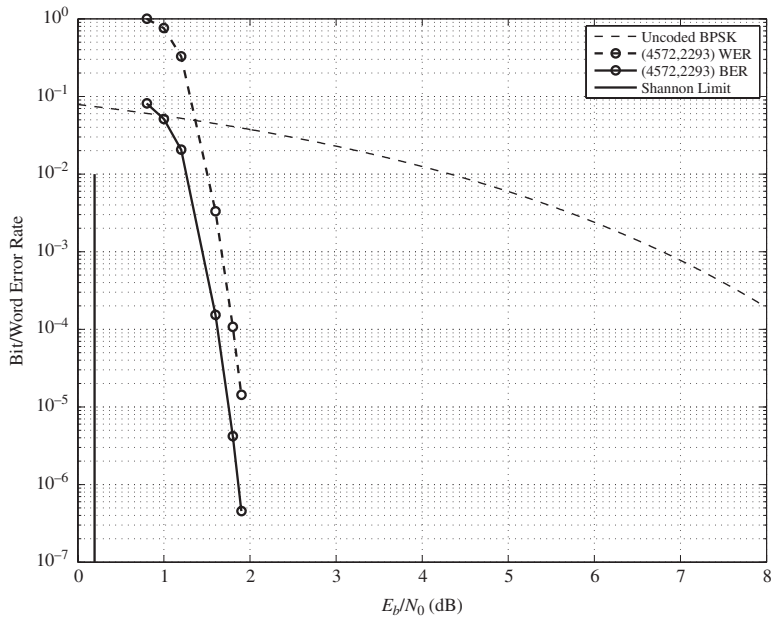


Figure 12.15 The error performance of the (4572,2307) QC-LDPC code given in Example 12.12.

18×36 array $\mathbf{H}_{\text{sup},p}$ of 127×127 circulant permutation and zero matrices. $\mathbf{H}_{\text{sup},p}$ is a 2286×4572 matrix with column and row weights 3 and 6, respectively. The null space of $\mathbf{H}_{\text{sup},p}$ gives a (3,6)-regular (4572,2307) QC-LDPC code with rate 0.5045, whose Tanner graph has a girth of 8. The error performance of this code with iterative decoding using the SPA with 100 iterations is shown in Figure 12.15. At a BER of 10^{-6}, it performs 1.6 dB from the Shannon limit.

The superposition construction given by (12.27) can be applied repeatedly to construct a large sparse matrix from a single small and simple matrix. A very simple case is to take the product of a short single parity-check code (SPC) with itself repeatedly. Let $\mathcal{C}_{\text{spc}}$ be an $(n, n-1)$ SPC with parity-check matrix $\mathbf{H}_{\text{spc}} = [1 \quad 1 \quad \cdots \quad 1]$ that consists a row of n 1-components. First, we use $\mathbf{H}_{\text{spc}}$ as both $\mathbf{H}_1$ and $\mathbf{H}_2$ to construct a superimposed matrix $\mathbf{H}_{\text{sup},p}$ based on (12.27). $\mathbf{H}_{\text{sup},p}$ is called the *second power* of $\mathbf{H}_{\text{spc}}$, denote $\mathbf{H}^2_{\text{spc}}$. $\mathbf{H}^2_{\text{spc}}$ has column and row weights 2 and n, respectively. We can easily check that the girth of the Tanner graph of $\mathbf{H}^2_{\text{spc}}$ is 8. Next, we use $\mathbf{H}^2_{\text{spc}}$ as $\mathbf{H}_1$ and $\mathbf{H}_{\text{spc}}$ as $\mathbf{H}_2$ to construct the third power $\mathbf{H}^3_{\text{spc}}$ of $\mathbf{H}_{\text{spc}}$ using (12.27). We can easily prove that the girth of the Tanner graph of $\mathbf{H}^3_{\text{spc}}$ is again 8. $\mathbf{H}^3_{\text{spc}}$ has column and row weights 3 and n, respectively. Repeating the above recursive construction forms the kth power $\mathbf{H}^k_{\text{spc}}$ of $\mathbf{H}_{\text{spc}}$, which has column and row weights k and n, respectively. The girth of $\mathbf{H}^k_{\text{spc}}$ is again 8. $\mathbf{H}^k_{\text{spc}}$ can be used either as a base matrix for superposition construction of an LDPC code, or as the parity-check matrix to generate an LDPC code $\mathcal{C}^k_{\text{spc}}$ of length n^k with rate $((n-1)/n)^k$, girth 8, and minimum distance 2^k.

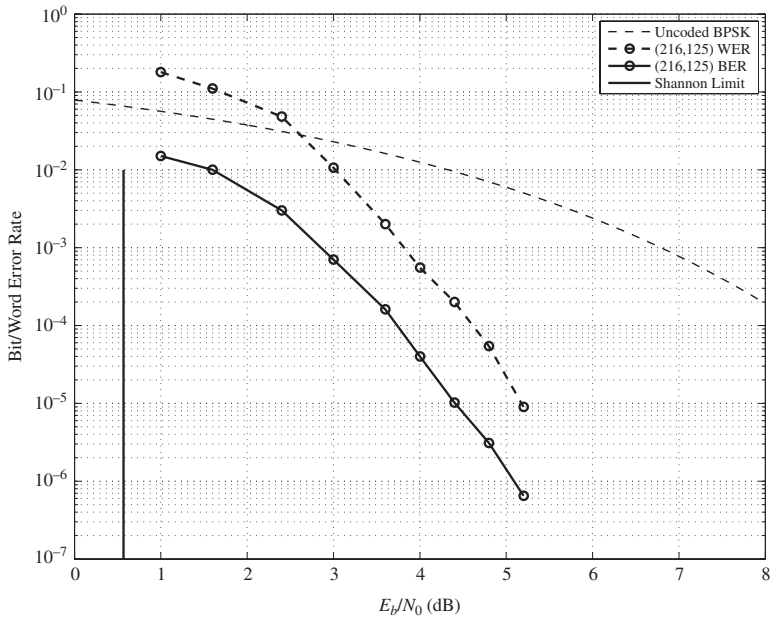


Figure 12.16 The error performance of the (216,125) product LDPC code given in Example 12.13.

Example 12.13. Suppose we use the (6,5) SPC for product LDPC code construction. The parity-check matrix of this SPC is $\mathbf{H}_{\text{spc}} = [1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1]$. Using (12.27) twice, we form the third power $\mathbf{H}_{\text{spc}}^3$ of $\mathbf{H}_{\text{spc}}$, which is a 128×216 matrix over GF(2) with column and row weights 3 and 6, respectively. The null space of $\mathbf{H}_{\text{spc}}^3$ gives a (216,125) product LDPC code with rate 0.579 and minimum distance 8, whose error performance with iterative decoding using the SPA (100 iterations) is shown in Figure 12.16.

Problems

12.1 For $t = 9$, there exists a class-I $(109, 981, 4, 36, 1)$ Bose BIBD. Utilizing this BIBD, a row of nine 109×109 circulants, $\mathbf{H}_{\text{BIBD}}^{(1)} = [\mathbf{G}_0 \quad \mathbf{G}_1 \quad \cdots \quad \mathbf{G}_8]$, can be constructed. Each circulant $\mathbf{G}_i$ has column weight and row weight 4. The null space of $\mathbf{H}_{\text{BIBD}}^{(1)}$ gives a type-I, class-I Bose BIBD-QC-LDPC code of length 981. Determine the code and compute its bit and word-error performances over the AWGN channel with iterative decoding using the SPA for 10 and 50 iterations, respectively.

12.2 Consider the class-I $(109, 981, 4, 36, 1)$ Bose BIBD constructed in Problem 12.1. Set $k = 4$ and $r = 8$. Construct a 4×8 array $\mathbf{H}_{\text{BIBD,decom}}^{(1)}(4, 8)$ of 109×109 CPMs. Determine the type-II class-1 Bose BIBD-QC-LDPC code given by the null space of $\mathbf{H}_{\text{BIBD,decom}}^{(1)}(4, 8)$. Compute its bit- and word-error performances over the

AWGN channel with iterative decoding using the SPA for 10 and 50 iterations, respectively.

12.3 Consider the 4×8 array $\mathbf{H}^{(1)}_{\text{BIBD,decom}}(4,8)$ of 109×109 CPMs constructed in Problem 12.2. This array can be divided into two 4×4 subarrays of 109×109 CPMs. By masking each of these two 4×4 subarray with the masking matrix given by (12.7), we obtain a 4×8 masked array $\mathbf{M}^{(1)}_{\text{BIBD,decom}}(4,8)$ of 109×109 circulant permutation and zero matrices. Determine the QC-LDPC code given by the null space of $\mathbf{M}^{(1)}_{\text{BIBD,decom}}(4,8)$ and compute its bit- and word-error performances over the AWGN channel with iterative decoding using the SPA for 10 and 50 iterations.

12.4 For $t = 41$, there exists a class-II $(821, 33661, 5, 205, 1)$ Bose BIBD. From this BIBD, a row of 41 821×821 circulants, $\mathbf{H}^{(2)}_{\text{BIBD}} = [\mathbf{G}_0 \quad \mathbf{G}_2 \quad \cdots \quad \mathbf{G}_{40}]$, can be constructed. Each circulant $\mathbf{G}_i$ has column weight and row weight 5. For $0 \leq i \leq 40$, decompose each circulant $\mathbf{G}_i$ into a column of five 821×821 CPMs with row decomposition. This results in a 5×41 array $\mathbf{H}^{(2)}_{\text{BIBD,decom}}$ of 821×821 CPMs. Take a 5×10 subarray $\mathbf{H}^{(2)}_{\text{BIBD,decom}}(5,10)$ from $\mathbf{H}^{(2)}_{\text{BIBD,decom}}$. Divide this 5×10 subarray $\mathbf{H}^{(2)}_{\text{BIBD,decom}}(5,10)$ into two 5×5 subarrays, $\mathbf{H}^{(2)}_1(5,5)$ and $\mathbf{H}^{(2)}_2(5,5)$. Mask each of these two 5×5 subarrays with the 5×5 masking matrix $\mathbf{Z}_1(5,5)$ given by (12.15). This results in a 5×10 masked array $\mathbf{M}^{(2)}_{\text{BIBD,decom}}(5,10)$. This masked array is an RC-constrained 4105×8210 matrix over GF(2) with column and row weights 3 and 6, respectively. Determine the QC-LDPC code given by the null space of $\mathbf{M}^{(2)}_{\text{BIBD,decom}}(5,10)$ and compute its bit- and word-error performances over the AWGN channel using the SPA with 10 and 50 iterations.

12.5 Consider the q^m points of the m-dimensional Euclidean geometry EG(m,q) over GF(q) as a collection of q^m objects. Given the structural properties of the lines in EG(m,q), show that there exists a BIBD with $\lambda = 1$ for a collection of q^m objects. Give the parameters of this BIBD.

12.6 For the m-dimensional projective geometry PG(m,q) over GF(q), show that there exists a BIBD with $\lambda = 1$ for a collection of $(q^{m+1} - 1)/(q - 1)$ objects.

12.7 Using the lines of the two-dimensional Euclidean geometry EG(2,2^4) over GF(2^4) that do not pass the origin of EG(2,2^5), a 255×255 circulant $\mathbf{G}$ with both column weight and row weight 16 can be constructed. The Tanner graph $\mathcal{G}$ of $\mathbf{G}$ has a girth of 6 (it is free of cycles of length 4). Decompose $\mathbf{G}$ with column decomposition into a 255×510 matrix $\mathbf{H}_0$ that consists of two circulants, $\mathbf{G}_0$ and $\mathbf{G}_1$, each having both column weight and row weight 8.

(a) Determine the number of cycles of length 6 in the Tanner graph $\mathbf{G}_0$ of $\mathbf{H}_0$. Compute the bit- and word-error performances of the code $\mathcal{C}_0$ given by the null space of $\mathbf{H}_0$.

(b) Use the trellis-based cycle-removal technique given in Section 12.5 to remove the cycles of length 6 in the Tanner graph $\mathcal{G}_0$ of $\mathbf{H}_0$. The cycle-removal process results in a Tanner graph $\mathcal{G}_1$ that is free of cycles of length 6. From this Tanner graph, construct an LDPC code $\mathcal{C}_1$ and compute its bit- and word-error performances.

(c) Remove the cycles of length 8 from the Tanner graph $\mathcal{G}_1$. This results in a new Tanner graph $\mathcal{G}_2$ that has a girth of at least 10. From $\mathcal{G}_2$, construct a new LDPC code $\mathcal{C}_2$ and compute its bit- and word-error performances.

12.8 Use the PEG algorithm to construct a rate-1/2 (2044,1022) irregular LDPC code based on the degree distributions given in Example 12.8. Compute the bit- and word-error performances of the code constructed. Also construct a rate-1/2 (2044,1022) irregular LDPC code with the improved PEG algorithm using ACE and compute its bit- and word-error performances.

12.9 Consider the three-dimensional Euclidean geometry EG(3,3) over GF(3). From the lines of EG(3,3) not passing through the origin, four 26×26 circulants over GF(2) can be constructed, each having both column weight and row weight 3. Take two of these circulants and arrange them in a row to form a 26×52 RC-constrained matrix $\mathbf{B}$ with column and row weights 3 and 6, respectively. $\mathbf{B}$ can be used as a base matrix for superposition construction of LDPC codes. Suppose we replace each 1-entry of $\mathbf{B}$ by a distinct 126×126 circulant permutation matrix and each 0-entry by a 126×126 zero matrix. This replacement results in an RC-constrained 26×52 array $\mathbf{H}_{\text{sup}}$ of 126×126 circulant permutation and zero matrices. $\mathbf{H}_{\text{sup}}$ is a 4056×8112 matrix over GF(2) with column and row weights 3 and 6, respectively. Determine the code given by the null space of $\mathbf{H}_{\text{sup}}$ and compute its bit- and word-error performances over the AWGN channel using the SPA with 10 and 50 iterations.

12.10 Let $\mathcal{C}_1$ be the (63,37) cyclic EG-LDPC code constructed using the two-dimensional Euclidean geometry EG(2,2^3) over GF(2^3). The parity-check matrix $\mathbf{H}_1$ of $\mathcal{C}_1$ is a 63×63 circulant with both column weight and row weight 8. The minimum distance is exactly 9. Let $\mathcal{C}_2$ be the (8,7) single parity-check code whose parity-check matrix $\mathbf{H}_2 = [1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1]$ is a row of eight 1s. The parity-check matrix $\mathbf{H}_{\text{sup},p}$ of the product of $\mathcal{C}_1$ and $\mathcal{C}_2$ is of the form given by (12.27). The product code $\mathcal{C}_{\text{sup},p}$ of $\mathcal{C}_1$ and $\mathcal{C}_2$ is a (504,259) LDPC code with minimum distance 18. Compute the bit- and word-error performances of $\mathcal{C}_{\text{sup},p}$ over the AWGN channel using the SPA with 50 iterations.

References

[1] R. C. Bose, "On the construction of balanced incomplete block designs," *Ann. Eugenics*, vol. 9, pp. 353–399, 1939.

[2] I. F. Blake and R. C. Mullin, *The Mathematical Theory of Coding*, New York, Academic Press, 1975.

[3] R. D. Carmichael, *Introduction to Theory of Groups of Finite Orders*, Boston, MA, Gin & Co., 1937.

[4] C. J. Colbourn and J. H. Dintz (eds.), *The Handbook of Combinatorial Designs*, Boca Raton, FL, CRC Press, 1996.

[5] D. J. Finney, *An Introduction to the Theory of Experimental Design*, Chicago, IL, University of Chicago Press, 1960.

[6] M. Hall, Jr., *Combinatorial Theory*, 2nd edn., New York, Wiley, 1986.

[7] H. B. Mann, *Analysis and Design of Experiments*, New York, Dover Publications, 1949.

[8] H. J. Ryser, *Combinatorial Mathematics*, New York, Wiley, 1963.

[9] B. Ammar, B. Honary, Y. Kou, J. Xu, and S. Lin, "Construction of low density parity-check codes based on balanced incomplete designs," *IEEE Trans. Information Theory*, vol. 50, no. 6, pp. 1257–1268, June 2004.

[10] S. Johnson and S. R. Weller, "Regular low-density parity-check codes form combinatorial designs," *Proc. 2001 IEEE Information Theory Workshop*, Cairns, Australia, September 2001, pp. 90–92.

[11] L. Lan, Y. Y. Tai, S. Lin, B. Memari, and B. Honary, "New constructions of quasi-cyclic LDPC codes based on special classes of BIBDs for the AWGN and binary erasure channels," *IEEE Trans. Communications*, vol. 56, no. 1, pp. 39–48, January 2008.

[12] B. Vasic and O. Milenkovic, "Combinatorial construction of low density parity-check codes for iterative decoding," *IEEE Trans. Information Theory*, vol. 50, no. 6, pp. 1156–1176, June 2004.

[13] X.-Y. Hu, E. Eleftheriou, and D. M. Arnold, "Regular and irregular progressive edge-growth Tanner graphs," *IEEE Trans. Information Theory*, vol. 51, no. 1, pp. 386–398, January 2005.

[14] L. Lan, Y. Y. Tai, L. Chen, S. Lin, and K. Abdel-Ghaffar, "A trellis-based method for removal cycles from bipartite graphs and construction of low density parity check codes," *IEEE Communications Lett.*, vol. 8, no. 7, pp. 443–445, July 2004.

[15] H. Xiao and A. H. Banihashemi, "Improved progressive-edge-growth (PEG) construction of irregular LDPC codes," *IEEE Communications Lett.*, vol. 8, no. 12, pp. 715–717, December 2004.

[16] S. Lin and D. J. Costello, Jr., *Error Control Coding: Fundamentals and Applications*, Upper Saddle River, NJ, Prentice-Hall, 2004.

[17] T. Tian, C. Jones, J. D. Villasenor, and R. D. Wesel, "Construction of irregular LDPC codes with low error floors," *Proc. IEEE Int. Conf. Communications*, vol. 5, Anchorage, AK, May 2003, pp. 3125–3129.

[18] T. Tian, C. Jones, J. D. Villasenor, and R. D. Wesel, "Selective avoidance of cycles in irregular LDPC code construction," *IEEE Trans. Communications*, vol. 52, no. 8, pp. 1242–1247, August 2004.

[19] J. Xu and S. Lin, "A combinatoric superposition method for constructing low-density parity-check codes," *Proc. Int. Symp. on Information Theory*, vol. 30, Yokohama, June–July 2003.

[20] J. Xu, L. Chen, L.-Q. Zeng, L. Lan, and S. Lin, "Construction of low-density parity-check codes by superposition," *IEEE Trans. Communications*, vol. 53, no. 2, pp. 243–251, February 2005.

[21] N. Deo, *Graph Theory and Applications to Engineering and Computer Engineering*, Englewood cliffs, NJ, Prentice-Hall, 1974.

[22] D. B. West, *Introduction to Graph Theory*, 2nd edn., Upper Saddle River, NJ, Prentice-Hall, 2001.

13 LDPC Codes for Binary Erasure Channels

Many channels, such as wireless, magnetic recording, and jammed channels, tend to suffer from time intervals during which their reliability deteriorates significantly, to a degree that compromises data integrity. In some scenarios, receivers are able to detect the presence of these time intervals and may choose, accordingly, to "erase" some (or all of the) symbols received during such intervals. This technique causes symbol losses at known locations. This chapter is devoted to LDPC codes for correcting (or recovering) transmitted symbols that have been erased, called *erasures*. The simplest channel model with erasures is the *binary erasure channel* over which a transmitted bit is either correctly received or erased. There are two basic types of binary erasure channel, *random* and *burst*. Over a random *binary erasure channel* (BEC), erasures occur at random locations, each with the same probability of occurrence, whereas over a *binary burst erasure channel* (BBEC), erasures cluster into bursts. In this chapter, we first show that the LDPC codes constructed in Chapters 10–12, besides performing well over the AWGN channel, also perform well over the BEC. Then, we construct LDPC codes for correcting bursts of erasures. A list of references on LDPC codes for the binary erasure channels is given at the end of this chapter.

13.1 Iterative Decoding of LDPC Codes for the BEC

For transmission over the BEC, a transmitted symbol, 0 or 1, is either correctly received with probability $1-p$ or erased with probability p, called the *erasure probability*, as shown in Figure 13.1. The channel output alphabet consists of three symbols, namely 0, 1, and "?," where the symbol "?" denotes a transmitted symbol that has been erased, called an *erasure*.

Consider an LDPC code $\mathcal{C}$ of length n given by the null space of an $m \times n$ sparse matrix $\mathbf{H}$ over GF(2). Let $\mathbf{v} = (v_0, v_1, \ldots, v_{n-1})$ be an n-tuple over GF(2). Then $\mathbf{v}$ is a codeword in $\mathcal{C}$ if and only if $\mathbf{v}\mathbf{H}^{\mathrm{T}} = \mathbf{0}$. Suppose a codeword $\mathbf{v} = (v_0, v_1, \ldots, v_{n-1})$ in $\mathcal{C}$ is transmitted and $\mathbf{r} = (r_0, r_1, \ldots, r_{n-1})$ is the corresponding received sequence. Let $\mathcal{E} = \{j_1, j_2, \ldots, j_t\}$ be the set of locations in $\mathbf{r}$ with $0 \le j_0 < j_1 < \cdots < j_t < n$, where the transmitted symbols are erased, i.e., $r_{j_1}, r_{j_2}, \ldots, r_{j_t} =$?. The set $\mathcal{E}$ displays the pattern of the erased symbols (or erasure locations) in $\mathbf{r}$ and hence is called an *erasure pattern*. Let $\{0, 1, \ldots, n-1\}$ be the index set of the components of a codeword in $\mathcal{C}$. Define $\bar{\mathcal{E}} \triangleq \{0, 1, \ldots, n-1\} \backslash \mathcal{E}$. Then $\bar{\mathcal{E}}$ is the set

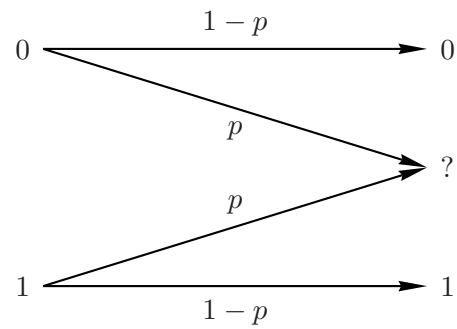


Figure 13.1 A model of the BEC.

of positions in $\mathbf{r}$ where the transmitted symbols are correctly received, i.e., $r_i = v_i$ for $i \in \bar{\mathcal{E}}$. Decoding $\mathbf{r}$ involves determining the value of v_{j_l} (or r_{j_l}) for each $j_l \in \mathcal{E}$. An erasure pattern $\mathcal{E}$ is said to be *recoverable* (or *correctable*) if the value of each erased transmitted symbol v_{j_l} with $j_l \in \mathcal{E}$ can be correctly determined.

In the following, we describe a simple iterative decoding method for correcting erasures. Label the rows of the parity-check matrix $\mathbf{H}$ of $\mathcal{C}$ from 0 to $m-1$. For $0 \le i < m$, let

$$\mathbf{h}_i = (h_{i,0}, h_{i,1}, \ldots, h_{i,n-1}) \tag{13.1}$$

denote the ith row of $\mathbf{H}$. For a received sequence $\mathbf{r} = (r_0, r_1, \ldots, r_{n-1})$ to be a codeword, it must satisfy the parity-check constraint

$$s_i = r_0 h_{i,0} + r_1 h_{i,1} + \cdots + r_{n-1} h_{i,n-1} = 0, \tag{13.2}$$

for $i = 0, 1, \ldots, m-1$, which is called a *check-sum*. The received symbol r_j in the above check-sum is said to be *checked* by the row $\mathbf{h}_i$, if $h_{i,j} = 1$. In this case, if $r_j = ?$ and all the other received symbols checked by $\mathbf{h}_i$ are not erasures (i.e., they are correctly received), then (13.2) contains only one unknown, r_j. Consequently r_j can be determined from

$$r_j = \sum_{k=0, k\neq j}^{n-1} r_k h_{i,k}. \tag{13.3}$$

For each erased position j_l in an erasure pattern $\mathcal{E} = \{j_1, j_2, \ldots, j_t\}$, if there exists a row $\mathbf{h}_i$ in $\mathbf{H}$ that checks only the erased symbol r_{j_l} and not any of the other $t-1$ erased symbols with locations in $\mathcal{E}$, then it follows from (13.3) that the value of the erased symbol $j_l \in \mathcal{E}$ can be correctly determined from the correctly received symbols that are checked by $\mathbf{h}_i$. Such an erasure pattern is said to be *recoverable in one step* (or *one iteration*). However, there are erasure patterns that are not recoverable in one step, but recoverable in *multiple steps iteratively*. Given an erasure pattern $\mathcal{E}$, we first determine the values of those erased symbols that can be recovered in one step by using (13.3). Then, we remove the recovered erased

symbols from $\mathcal{E}$. This results in a new erasure pattern $\mathcal{E}_1$ of smaller size. Next, we determine the values of erased symbols in $\mathcal{E}_1$ that are recoverable using (13.3). On removing the recovered erased symbols from $\mathcal{E}_1$, we obtain an erasure pattern $\mathcal{E}_2$ of size smaller than that of $\mathcal{E}_1$. We repeat the above process until either all the erased symbols in $\mathcal{E}$ have been recovered, or an erasure pattern $\mathcal{E}_m$ is obtained such that no erasure in $\mathcal{E}_m$ can be recovered using (13.3). In the latter case, some erasures in $\mathcal{E}$ cannot be recovered. The set of erasure locations in $\mathcal{E}_m$ is said to form a *stopping set* [1,2] (see Chapters 5 and 8) that stops the recovery process.

For a given erasure pattern $\mathcal{E}$, let $\mathbf{H}_{\mathcal{E}}$ be the submatrix of $\mathbf{H}$ whose columns correspond to the locations of erased symbols given in $\mathcal{E}$. Let $\mathbf{r}_{\mathcal{E}}$ denote the subsequence of $\mathbf{r}$ that consists of the erased symbols in $\mathbf{r}$. Then the above iterative process for recovering erasures in a received sequence $\mathbf{r}$ can be formulated as an algorithm. To initialize the recovering process, we set $k = 0$ and $\mathcal{E}_0 = \mathcal{E}$. Then we execute the following steps iteratively:

1. Determine $\mathcal{E}_k$ and form $\mathbf{r}_{\mathcal{E}}$. If $\mathcal{E}_k$ is empty, stop decoding, otherwise go to Step 2.
2. Form $\mathbf{H}_{\mathcal{E}_k}$ on the basis of $\mathbf{r}_{\mathcal{E}}$.
3. Find the set $\mathcal{R}_k$ of rows in $\mathbf{H}_{\mathcal{E}}$ such that each row in $\mathcal{R}_k$ contains a single 1-component. If $\mathcal{R}_k \neq \emptyset$, determine the erasures in $\mathcal{E}_k$ that are checked by the rows in $\mathcal{R}_k$. Determine values of these erasures by application of (13.3) based on the rows of $\mathbf{H}$ that correspond to the rows in $\mathcal{R}_k$. Then go to Step 4. If $\mathcal{R}_k = \emptyset$, then stop the decoding.
4. Remove the locations of the erasures recovered at Step 3 from $\mathcal{E}_k$. Set $k = k + 1$ and go to Step 1.

If the decoding stops at Step 1, either there is no erasure in $\mathbf{r}$ or all the erasures in the erasure pattern $\mathcal{E}$ have been recovered and decoding is successful. If decoding stops at Step 3, some erasures in $\mathcal{E}$ cannot be recovered.

The above decoding algorithm for recovering erased symbols was first proposed in [3] and then in [1]. It is equivalent to the iterative erasure filling algorithm presented in Chapter 5. Since the parity-check matrix $\mathbf{H}$ of an LDPC code is a sparse matrix, the weight of each row is relatively small compared with the length of the code and hence the number of terms in the sum given by (13.3) is small. Therefore, hardware implementation of the above iterative decoding algorithm can be very simple.

13.2 Random-Erasure-Correction Capability

The performance of an LDPC code over the BEC is determined by the stopping sets and the degree distributions of the nodes of its Tanner graph $\mathcal{G}$. Let $\mathcal{V}$ be a set of variable nodes in $\mathcal{G}$, and $\mathcal{S}$ be a set of check nodes in $\mathcal{G}$ that are adjacent to the nodes in $\mathcal{V}$, i.e., each check node in $\mathcal{S}$ is connected to *at least one* variable node in $\mathcal{V}$. The nodes in $\mathcal{S}$ are called the *neighbors* of the nodes in $\mathcal{V}$. A set $\mathcal{V}$ of variable nodes in $\mathcal{G}$ is called a *stopping set* of $\mathcal{G}$, if each check node in the neighbor set $\mathcal{S}$ of $\mathcal{V}$ is

connected to *at least two variable nodes* in $\mathcal{V}$. If an erasure pattern $\mathcal{E}$ corresponds to a stopping set in the Tanner graph $\mathcal{G}$ of an LDPC code, then a row of the parity-check matrix $\mathbf{H}$ of the code that checks an erasure in $\mathcal{E}$ also checks at least one other erasure in $\mathcal{E}$. In this case, the sum given by (13.2) contains at least two unknowns (two erasures). As a result, no erasure in $\mathcal{E}$ can be determined with (13.3).

A set $\mathcal{V}$ of variables nodes in $\mathcal{G}$ may contain more than one stopping set. The union of two stopping sets is also a stopping set and the union of all stopping sets contained in $\mathcal{V}$ gives the *maximum stopping set* of $\mathcal{V}$. A set $\mathcal{V}_{\text{ssf}}$ of variable nodes in $\mathcal{G}$ is said to be *stopping-set-free* (SSF), if it does not contain any stopping set. It is clear that, for a given erasure pattern $\mathcal{E}$, the erasures that are contained in the maximum stopping set of $\mathcal{E}$ cannot be recovered. It is also clear that any erasure pattern $\mathcal{E}$ is recoverable if it is SSF. Let $\mathcal{V}_{\min}$ be a stopping set of *minimum size* in the Tanner graph $\mathcal{G}$ of an LDPC code, called a *minimum stopping set*. If the code symbols corresponding to the variable nodes in $\mathcal{V}_{\min}$ are erased during the transmission, then $\mathcal{V}_{\min}$ forms an irrecoverable erasure pattern of minimum size. Therefore, for random-erasure-correction (REC) with the above iterative decoding algorithm, it is desired to construct codes with the largest possible minimum stopping sets in their Tanner graphs. The error-floor performance of an LDPC code with the above iterative decoding depends on the size and number of minimum stopping sets in the Tanner graph $\mathcal{G}$ of the code, or more precisely, on the stopping-set distribution of $\mathcal{G}$. A good LDPC code for REC must have no or very few small stopping sets in its Tanner graph.

Consider a (g,r)-regular LDPC code given by an RC-constrained parity-check matrix $\mathbf{H}$ with column and row weights g and r, respectively. The size of a minimum possible stopping set in its Tanner graph is $g+1$. To prove this, let $\mathcal{E} = \{j_1, j_2, \ldots, j_t\}$ be an erasure pattern that consists of t erasures with $0 \leq t \leq g$. Consider the erasure at a location j_l in $\mathcal{E}$. Since the column weight of $\mathbf{H}$ is g, there exists a set of g rows, $\Lambda_l = \{\mathbf{h}_{i_1}, \mathbf{h}_{i_2}, \ldots, \mathbf{h}_{i_g}\}$, such that all the rows in Λ_l check the erased symbol r_{j_l}. The other $t-1$ erasures in $\mathcal{E}$ can be checked by at most $t-1$ rows in Λ_l. Therefore, there is at least one row in Λ_l that checks only the erased symbol r_{j_l} and $r-1$ other correctly received symbols. Since $\mathbf{H}$ satisfies the RC-constraint, no two erasures in $\mathcal{E}$ can be checked simultaneously by two rows in Λ_l. The above two facts guarantee that every erasure in $\mathcal{E}$ can be recovered. Therefore, the size of a minimum possible stopping set of the Tanner graph of an LDPC code given by the null space of an RC-constrained (g,r)-regular parity-check matrix $\mathbf{H}$ is at least $g+1$. Now suppose that an erasure pattern $\mathcal{E} = \{j_1, j_2, \ldots, j_{g+1}\}$ with $g+1$ erasures occurs. Consider the g rows in Λ_l that check the erased symbol r_{j_l}. If, in the worst case, each of the other g erasures is checked separately by the g rows in Λ_l, then each row in Λ_l checks two erasures. In this case, $\mathcal{E}$ corresponds to a stopping set of size $g+1$. Therefore, the size of a minimum possible stopping set of the Tanner graph of a (g,r)-regular LDPC code given by the null space of an RC-constrained parity-check matrix $\mathbf{H}$ is $g+1$.

For (g,r)-regular LDPC codes, it has been proved in [2,4] that the sizes of minimum stopping sets of their Tanner graphs with girths 4, 6, and 8 are 2, $g+1$, and

$2g$, respectively. Hence, for correcting random erasures with an LDPC code using iterative decoding, the most critical cycles in the code's Tanner graph are cycles of length 4. Therefore, in construction of LDPC codes for the BEC, cycles of length 4 must be avoided in their Tanner graphs. It is proved in [5] that a code with minimum distance $d_{\min}$ contains stopping sets of size $d_{\min}$. In fact, the *support* of a codeword (defined as the locations of nonzero components of a codeword) forms a stopping set of size equal to the weight of the codeword [5]. Therefore, in the construction of an LDPC code for correcting random erasures, we need to keep its minimum distance large.

13.3 Good LDPC Codes for the BEC

The cyclic EG-LDPC codes in Section 10.1.1 based on Euclidean geometries have minimum stopping sets of large sizes in their Tanner graphs. Consider the cyclic EG-LDPC code $\mathcal{C}_{\mathrm{EG},c,k}$ of length $q^m - 1$ produced by the null space of the parity-check matrix $\mathbf{H}^{(1)}_{\mathrm{EG},c,k}$ given by (10.7) constructed on the basis of the m-dimensional Euclidean geometry EG(m,q) over GF(q). $\mathbf{H}^{(1)}_{\mathrm{EG},c,k}$ consists of a column of k circulants of size $(q^m - 1) \times (q^m - 1)$ over GF(2), each having both column weight and row weight q. Since $\mathbf{H}^{(1)}_{\mathrm{EG},c,k}$ satisfies the RC-constraint and has column weight $g = kq$, it follows that the size of a minimum stopping set of the Tanner graph of $\mathcal{C}_{\mathrm{EG},c,k}$ is $kq + 1$ with $1 \leq k < q^{m-1}$. Using the iterative decoding algorithm presented in Section 13.1, the code is capable of correcting any erasure pattern with kq or fewer random erasures. Consider the special case for which $m = 2$ and $q = 2^s$. The cyclic EG-LDPC code $\mathcal{C}_{\mathrm{EG},c,1}$ based on the two-dimensional Euclidean geometry EG(2,2^s) over GF(2^s) has length $2^{2s} - 1$ and minimum distance $2^s + 1$. The parity-check matrix $\mathbf{H}^{(1)}_{\mathrm{EG},c,1}$ of this code consists of a single $(2^{2s} - 1) \times (2^{2s} - 1)$ circulant with both column weight and row weight 2^s. Hence, the size of a minimum stopping set in the Tanner graph of this code is $2^s + 1$. Since cyclic EG-LDPC codes have minimum stopping sets of large sizes in their Tanner graphs, they should perform well over the BEC.

Many experimental results have shown that the structured LDPC codes constructed in Chapters 10–12 perform well not only over the AWGN channel but also over the BEC with the iterative decoding presented in Section 13.1. In the following, we will use structured LDPC codes from different classes to demonstrate this phenomenon. For each code, we give its performance and compare it with the Shannon limit in terms of the *unresolved* (or *unrecovered*) *erasure bit rate* (UEBR). For transmitting information at a rate R (information bits per channel usage) over the BEC, the Shannon limit is $1 - R$. The implication of this Shannon limit is that for erasure probability p smaller than $1 - R$, information can be transmitted reliably over the BEC by using a sufficiently long code with rate R, and, conversely, reliable transmission is not possible if the erasure probability p is larger than the Shannon limit.

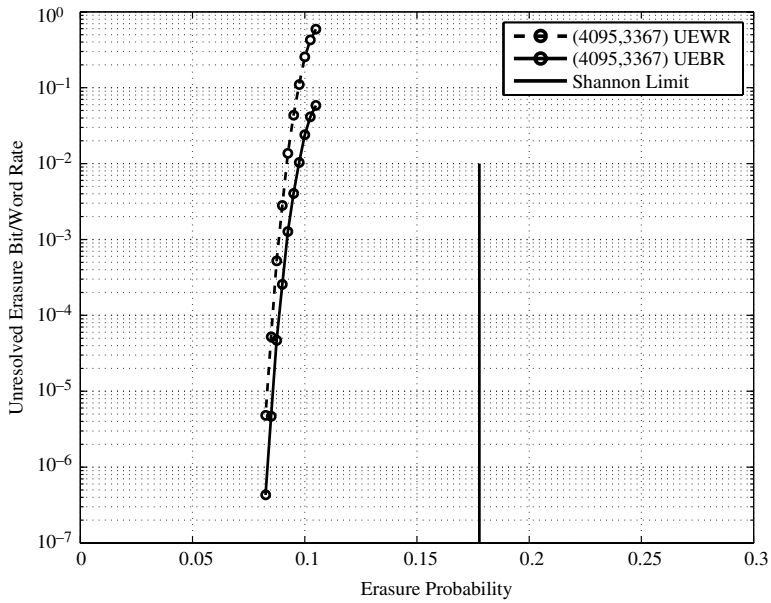


Figure 13.2 The error performance of the (4095,3367) cyclic EG-LDPC code over the BEC. (UEWR stands for unresolved erasure word rate.)

Example 13.1. Consider the (4095,3367) cyclic EG-LDPC code of rate 0.822 with minimum distance 65 based on the two-dimensional Euclidean geometry EG(2,2^6) over GF(2^6) given in Example 10.1. The parity-check matrix of this code is a 4095×4095 circulant over GF(2) with both column weight and row weight 64. Since the parity-check matrix of this code satisfies the RC-constraint, the size of a minimum stopping set in its Tanner graph is 65. Therefore, any erasure pattern with 64 or fewer random erasures is recoverable with the iterative decoding algorithm given in Section 13.1. The Shannon limit for rate 0.822 is $1 - 0.822 = 0.178$. The error performance of this code over the BEC with iterative decoding is shown in Figure 13.2. At a UEBR of 10^{-6}, the code performs 0.095 from the Shannon limit. As shown in Figure 10.1, this code also performs very well over the AWGN channel. So, this code performs well both over the BEC and over the AWGN channel.

Example 13.2. Consider the (8192,7171) QC-LDPC code based on the (256,2,255) RS code over the prime field GF(257) given in Example 11.1. The code has rate $R = 0.8753$. Figure 11.1 shows that the code performs very well over the binary-input AWGN channel. At a BER of 10^{-6}, it performs only 1 dB from the Shannon limit. The error performance of this code over the BEC with iterative decoding is shown in Figure 13.3. At a UEBR of 10^{-6}, the code performs 0.040 from the Shannon limit, 0.1247. From Figures 11.1 and 13.3, we see that the code performs well both over the AWGN and over the binary random erasure channels.

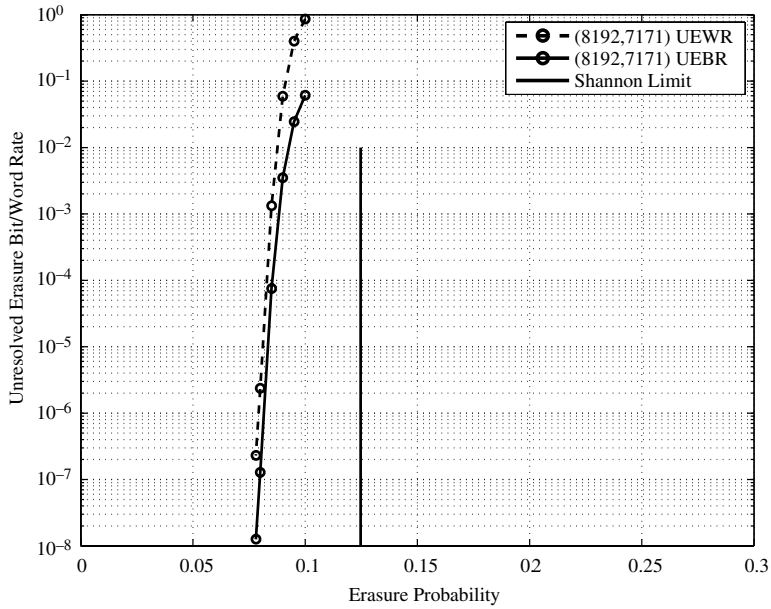


Figure 13.3 The error performance of the (8192,7171) QC-LDPC code over the BEC.

Example 13.3. Consider the (5256,4823) QC-LDPC code constructed using the additive group of the prime field GF(73) given in Example 11.11. This code has rate 0.918. Figure 13.4 shows its error performance over the BEC with iterative decoding. We see that, at a UEBR of 10^{-6}, the gap between the code performance and the Shannon limit, 0.0824, is less than 0.03. Also from Figure 13.4, we see that the code performs smoothly down to a UEBR of 10^{-8} without showing any error floor. As shown in Figure 11.11, the code also performs well over the binary-input AWGN channel, and iterative decoding of this code converges very fast. Its estimated error floor for the AWGN channel is below 10^{-25}, as shown in Figure 11.12.

Example 13.4. Consider the 256×256 array $\mathbf{H}^{(1)}_{qc,\text{disp}}$ of circulant permutation and zero matrices over GF(2) based on the (256,2,255) RS code over GF(257) given in Example 11.1. Take an 8×128 subarray $\mathbf{H}^{(1)}_{qc,\text{disp}}(8,128)$ from $\mathbf{H}^{(1)}_{qc,\text{disp}}$, avoiding the zero matrices on the main diagonal of $\mathbf{H}^{(1)}_{qc,\text{disp}}$. We use this subarray as the base array for masking. Design an 8×128 masking matrix $\mathbf{Z}(8,128)$ that consists of a row of 16 8×8 circulants, $[\mathbf{G}_0 \quad \mathbf{G}_1 \quad \cdots \quad \mathbf{G}_{15}]$, whose generators are given in Table 13.1. Since each generator has weight 4, the masking matrix $\mathbf{Z}(8,128)$ has column and row weights 4 and 64, respectively. By masking $\mathbf{H}^{(1)}_{qc,\text{disp}}(8,128)$ with $\mathbf{Z}(8,128)$, we obtain an 8×128 masked array $\mathbf{M}^{(1)}(8,128) = \mathbf{Z}(8,128) \circledast \mathbf{H}^{(1)}_{qc,\text{disp}}(8,128)$ of circulant permutation and zero matrices. $\mathbf{M}^{(1)}(8,128)$ is a 2048×32768 matrix over GF(2) with column and row weights 4 and 64, respectively. The null space of $\mathbf{M}^{(1)}(8,128)$ gives a (32768,30721) QC-LDPC code with rate 0.9375. The performances of this code over the AWGN channel and the BEC

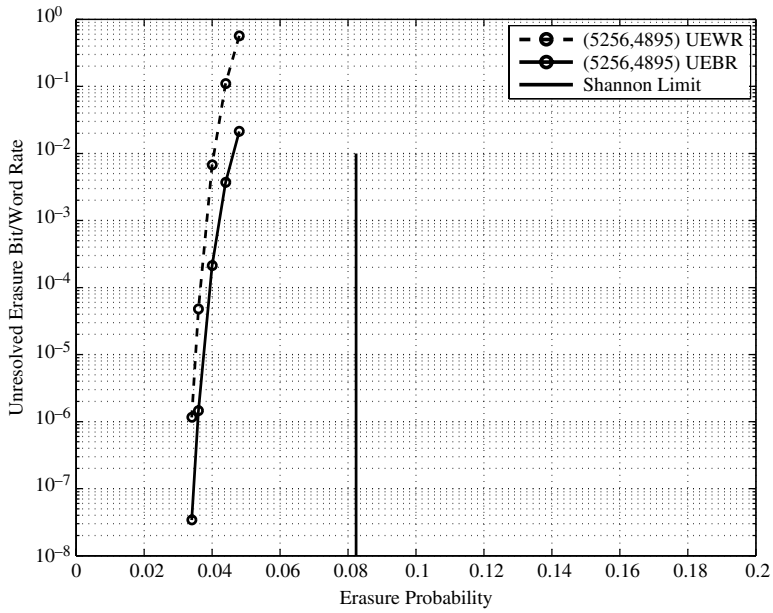


Figure 13.4 The error performance of the (5256,4823) QC-LDPC code over the BEC.

Table 13.1. The generators of the masking matrix $\mathbf{Z}(8, 128)$ of Example 13.4

$\mathbf{g}_0 = (10100101)$	$\mathbf{g}_1 = (01101010)$
$\mathbf{g}_2 = (10101100)$	$\mathbf{g}_3 = (10100110)$
$\mathbf{g}_4 = (01011100)$	$\mathbf{g}_5 = (10111000)$
$\mathbf{g}_6 = (01010110)$	$\mathbf{g}_7 = (01110010)$
$\mathbf{g}_8 = (10010110)$	$\mathbf{g}_9 = (01011010)$
$\mathbf{g}_{10} = (00011110)$	$\mathbf{g}_{11} = (11000110)$
$\mathbf{g}_{12} = (00111010)$	$\mathbf{g}_{13} = (01011010)$
$\mathbf{g}_{14} = (00111010)$	$\mathbf{g}_{15} = (11001100)$

are shown in Figures 13.5 and 13.6. From Figure 13.5, we see that, at a BER of 10^{-6}, the code performs only 0.65 dB from the Shannon limit for the binary-input AWGN channel. From Figure 13.6, we see that, at a UEBR of 10^{-6}, the code performs 0.017 from the Shannon limit, 0.0625, for the BEC. Again this long QC-LDPC code performs well both over the AWGN and over the binary random erasure channels.

Example 13.5. In this example, we consider the (4,24)-regular (8088,6743) type-II QC-BIBD-LDPC code with rate 0.834 constructed in Example 12.3. The error performance of this code over the binary-input AWGN channel, decoded with the SPA, is shown in Figure 12.3. At a BER of 10^{-6}, it performs 1.05 dB from the Shannon limit for the AWGN channel. The error performance of this code over the BEC, decoded with the iterative

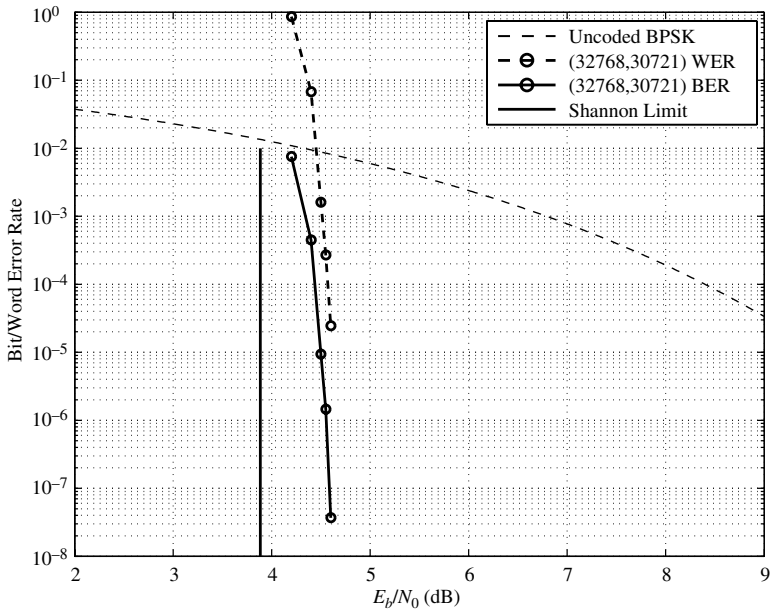


Figure 13.5 The error performance of the (32786,30721) QC-LDPC code given in Example 13.4 over the binary-input AWGN channel.

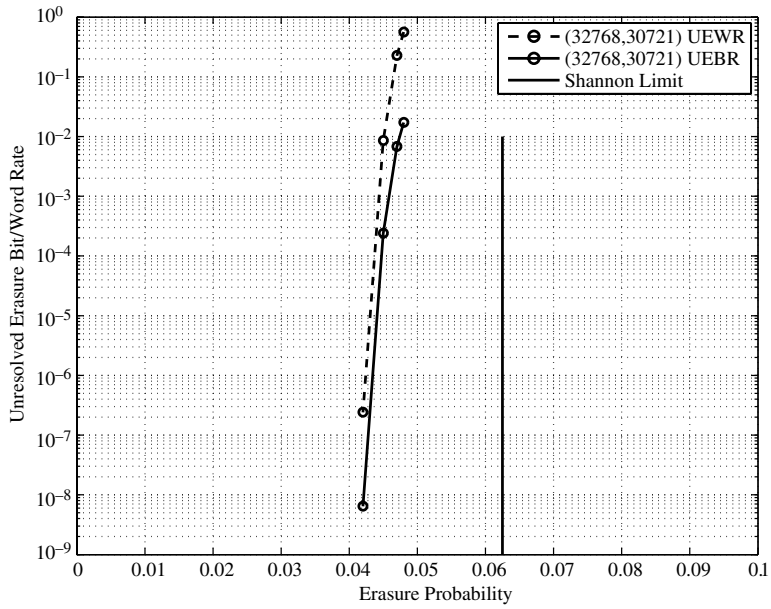


Figure 13.6 The error performance of the (32786,30721) QC-LDPC code given in Example 13.4 over the BEC.

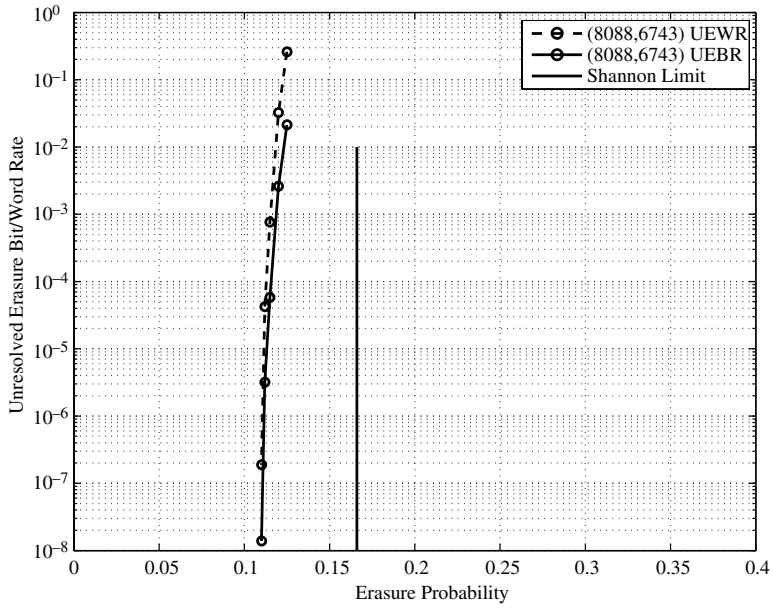


Figure 13.7 The error performance of the (8088,6743) QCBIBD-LDPC code given in Example 13.5 over the BEC.

decoding algorithm given in Section 13.1, is shown in Figure 13.7. For code rate 0.834, the Shannon limit for the BEC is 0.166. From Figure 13.7, we see that, at a UEBR of 10^{-6}, the code performs 0.055 from the Shannon limit, 0.166, for the BEC.

In [6–9], many other structured LDPC codes constructed from finite geometries, finite fields, and combinatorial designs have been shown to perform well both over the AWGN and over binary random erasure channels.

13.4 Correction of Erasure-Bursts

An erasure pattern $\mathcal{E}$ is called an *erasure-burst* of length b if the erasures in $\mathcal{E}$ are confined to b *consecutive locations*, the first and last of which are erasures. Erasure-bursts occur often in recording, jammed, and some fading channels. This section is concerned with the correction of erasure-bursts with LDPC codes using iterative decoding. LDPC codes for correcting erasure-bursts over the BBEC were first investigated in [10–13] and later in [6–9,14] using a different approach that leads to construction of good erasure-burst-correction LDPC codes algebraically. The erasure-burst-correction algorithm, code designs, and constructions presented in this and the next two sections follow the approach given in [7,9].

Let $\mathbf{v} = (v_0, v_1, \ldots, v_{n-1})$ be a nonzero n-tuple over GF(2). The first (leftmost) 1-component of $\mathbf{v}$ is called the *leading* 1 of $\mathbf{v}$ and the last (rightmost) 1-component of $\mathbf{v}$ is called the *trailing* 1 of $\mathbf{v}$. If $\mathbf{v}$ has only a single 1-component, then the leading 1 and trailing 1 of $\mathbf{v}$ are the same. A *zero-span* of $\mathbf{v}$ is defined as *a sequence of*

consecutive zeros between two 1-components. The zeros to the right of the trailing 1 of $\mathbf{v}$ together with the zeros to the left of the leading 1 of $\mathbf{v}$ also form a zero-span, called the *end-around zero-span*. The number of zeros in a zero-span is called the *length* of the zero-span. A zero-span of zero length is called a *null* zero-span. Consider the 16-tuple over GF(2), $\mathbf{v} = (0001000010010000)$. It has three zero-spans with lengths 4, 2, and 7, respectively, where the zero-span of length 7 is an end-around zero-span.

Consider a (g,r)-regular LDPC code $\mathcal{C}$ given by the null space of an RC-constrained $m \times n$ parity-check matrix $\mathbf{H}$ over GF(2) with column and row weights g and r, respectively. Label the rows and columns of $\mathbf{H}$ from 0 to $m-1$ and 0 to $n-1$, respectively. For $0 \leq j < n$, there is a set of g rows,

$$\Lambda_j = \{\mathbf{h}_{i_1}, \mathbf{h}_{i_2}, \ldots, \mathbf{h}_{i_g}\},$$

in $\mathbf{H}$ with $0 \leq i_1 < i_2, \ldots, i_g < m$, such that each row $\mathbf{h}_{i_l}$ in Λ_j has a 1-component in the jth column of $\mathbf{H}$ (or at the jth position of $\mathbf{h}_{i_l}$). For each row $\mathbf{h}_{i_l}$ in Λ_j, we find its zero-span starting from the $(j+1)$th column (or the $(j+1)$th position of $\mathbf{h}_{i_l}$) to the next 1-component and compute its length. If the 1-component of a row $\mathbf{h}_{i_l}$ in Λ_j at position j is the trailing 1 of the row, we determine its end-around zero-span. Among the zero-spans of the g rows in Λ_j with a 1-component in the jth column of $\mathbf{H}$, a zero-span of *maximum length* is called the *zero-covering-span* of the jth column of $\mathbf{H}$.

Let σ_j be length of the zero-covering-span of the jth column of $\mathbf{H}$. The n-sequence of integers,

$$(\sigma_0, \sigma_1, \ldots, \sigma_{n-1}), \tag{13.4}$$

is called the *profile* of column zero-covering-spans of the parity-check matrix $\mathbf{H}$. The column zero-covering-span of *minimum length* is called the *zero-covering-span of the parity-check matrix* $\mathbf{H}$. The length of the zero-covering-span of $\mathbf{H}$ is given by

$$\sigma = \min\{\sigma_j \colon 0 \leq j < n\}. \tag{13.5}$$

By employing the concept of zero-covering-spans of columns of the parity-check matrix $\mathbf{H}$ of an LDPC code $\mathcal{C}$, a simple erasure-burst decoding algorithm can be devised. The erasure-burst-correction capability of code $\mathcal{C}$ is determined by the length σ of the zero-covering-span of its parity-check matrix $\mathbf{H}$.

Example 13.6. Consider the (3,3)-regular cyclic (7,3) LDPC code given by the null space of the following 7×7 circulant over GF(2):

$$\mathbf{H} = \begin{bmatrix} \mathbf{h}_0 \\ \mathbf{h}_1 \\ \mathbf{h}_2 \\ \mathbf{h}_3 \\ \mathbf{h}_4 \\ \mathbf{h}_5 \\ \mathbf{h}_6 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix}. \tag{13.6}$$

We can easily check that the length of each column zero-covering-span is 3. Hence, the profile of column zero-covering-spans of $\mathbf{H}$ is $(3, 3, 3, 3, 3, 3, 3)$ and the length of the zero-covering-span of the parity-check matrix $\mathbf{H}$ is $\sigma = 3$.

Consider a (g,r)-regular LDPC code $\mathcal{C}$ over GF(2) given by the null space of an $m \times n$ parity-check matrix $\mathbf{H}$ with column and row weights g and r, respectively, for which the profile of the column zero-covering-span is $(\sigma_0, \sigma_1, \ldots, \sigma_{n-1})$. Suppose this code is used for correcting erasure-bursts. Let $\mathbf{v} = (v_0, v_1, \ldots, v_{n-1})$ be the transmitted codeword and $\mathbf{r} = (r_0, r_1, \ldots, r_{n-1})$ the corresponding received sequence. For $0 \leq j < n$, if j is the starting position of an erasure-burst pattern $\mathcal{E}$ of length at most $\sigma_j + 1$ that occurs during the transmission of $\mathbf{v}$, then there is a row $\mathbf{h}_i = (h_{i,0}, h_{i,1}, \ldots, h_{i,n-1})$ in $\mathbf{H}$ for which the jth component $h_{i,j}$ is "1" and that is followed by a zero-span of length σ_j. Since the burst starts at position j and has length at most $\sigma_j + 1$, all the received symbols checked by $\mathbf{h}_i$, other than the jth received symbol r_j, are known. Consequently, the value of the jth erased symbol r_j (or the transmitted symbol v_j) can be determined from (13.3). Replacing the erased symbol at the jth position by its value results in a shorter erasure-burst pattern $\mathcal{E}\backslash\{j\}$. The erasure-recovering procedure can be repeated until all erased symbols in $\mathcal{E}$ have been determined.

The above iterative algorithm for correcting erasure-bursts that is based on the profile of column zero-covering-spans of the parity-check matrix of an LDPC code can be formulated as follows:

---

1. Check the received sequence $\mathbf{r}$. If there are erasures, determine the starting position of the erasure-burst in $\mathbf{r}$, say j, and go to Step 2. If there is no erasure in $\mathbf{r}$, stop decoding.
2. Determine the length of the erasure-burst, say b. If $b \leq \sigma_j + 1$, go to Step 3; otherwise stop decoding.
3. Determine the value of the erasure at position j using (13.3) and go to Step 1.

It follows from (13.5) that any erasure-burst of length $\sigma + 1$ or less is guaranteed to be recoverable regardless of its starting position. Therefore, $\sigma + 1$ is a lower bound on the erasure-burst-correction capability of an LDPC code using the iterative decoding algorithm given above. The simple (7,3) cyclic LDPC code given in Example 13.1 is capable of correcting any erasure-burst of length 4 or less.

From the definition of the zero-covering-span of the parity-check matrix $\mathbf{H}$ of an LDPC code and (13.5), the Tanner graph of the code has the following structural property: for an LDPC code whose parity-check matrix $\mathbf{H}$ has a zero-covering-span of length σ, no $\sigma + 1$ or fewer consecutive variable nodes in its Tanner graph form a stopping set and hence any $\sigma + 1$ consecutive variable nodes form a *stopping-set-free zone*.

The erasure-burst-correction capability l_b of an (n,k) LDPC code (or any (n,k) linear block code) of length n and dimension k is upper bounded by $n - k$, the number of parity-check symbols of the code. This can be easily proved as follows.

Suppose the code is capable of correcting all the erasure-bursts of length $n-k+1$ with the iterative decoding algorithm presented above. Then it must have a parity-check matrix with a zero-covering-span of length $n-k$. This implies that the parity-check matrix of the code has at least $n-k+1$ linearly independent rows. Consequently, the rank of the parity-check matrix is at least $n-k+1$, which contradicts the fact that the rank of any parity-check matrix of an (n,k) linear block code is $n-k$. To measure the efficiency of erasure-burst-correction of an (n,k) LDPC code with the iterative decoding algorithm presented above, we define its *erasure-burst-correction efficiency* as

$$\eta = \frac{l_b}{n-k}. \tag{13.7}$$

Clearly η is upper bounded by 1. An LDPC code is said to be *optimal* for erasure-burst-correction if its erasure-burst efficiency is equal to 1.

The concepts and iterative decoding algorithm developed for a regular LDPC code can be applied to irregular LDPC codes and, in fact, can be applied to any linear block codes. For an LDPC code, the same parity-check matrix $\mathbf{H}$ can be used for iterative decoding both over the AWGN and over the binary erasure channels.

13.5 Erasure-Burst-Correction Capabilities of Cyclic Finite-Geometry and Superposition LDPC Codes

13.5.1 Erasure-Burst-Correction with Cyclic Finite-Geometry LDPC Codes

The erasure-burst-correction capabilities of cyclic LDPC codes based on two-dimensional Euclidean (or projective) geometries can be easily determined. Consider a cyclic EG-LDPC code $\mathcal{C}_{\mathrm{EG},c,1}$ based on the two-dimensional Euclidean geometry EG(2,q) over GF(q) (see Section 10.1). Take a line $\mathcal{L}$ in EG(2,q) not passing through the origin, and form its incidence vector $\mathbf{v}_{\mathcal{L}}$, a (q^2-1)-tuple over GF(2). A parity-check matrix $\mathbf{H}^{(1)}_{\mathrm{EG},c,1}$ of $\mathcal{C}_{\mathrm{EG},c,1}$ is a $(q^2-1)\times(q^2-1)$ circulant, which is formed with $\mathbf{v}_{\mathcal{L}}$ as the first (or top) row and its q^2-2 cyclic-shifts as the other rows. Since the incidence vector $\mathbf{v}_{\mathcal{L}}$ has q 1-components, it has q zero-spans. Determine the maximum zero-span of $\mathbf{v}_{\mathcal{L}}$ and its length. Note that the maximum zero-span of $\mathbf{v}_{\mathcal{L}}$ is unique, otherwise $\mathbf{H}^{(1)}_{\mathrm{EG},c,1}$ would not satisfy the RC constraint. Since all the other rows of $\mathbf{H}^{(1)}_{\mathrm{EG},c,1}$ are cyclic-shifts of $\mathbf{v}_{\mathcal{L}}$, their maximum zero-spans have the same length as that of the maximum zero-span of $\mathbf{v}_{\mathcal{L}}$. The starting positions of the maximum zero-spans of $\mathbf{v}_{\mathcal{L}}$ and its q^2-2 cyclic-shifts are all different and they spread from 0 to $n-1$. Consequently, the lengths of the zero-covering-spans of the q^2-1 columns of the parity-check matrix $\mathbf{H}^{(1)}_{\mathrm{EG},c,1}$ of $\mathcal{C}_{\mathrm{EG},c,1}$ are all equal to the length of the maximum zero-span of $\mathbf{v}_{\mathcal{L}}$. Hence, the length σ of the zero-covering-span of the parity-check matrix $\mathbf{H}^{(1)}_{\mathrm{EG},c,1}$ of $\mathcal{C}_{\mathrm{EG},c,1}$ is equal to the length of the maximum zero-span of $\mathbf{v}_{\mathcal{L}}$. Table 13.2 gives the lengths of zero-covering-spans σ and actual erasure-burst-correction capabilities l_b of a list of cyclic EG-LDPC codes.

Table 13.2. Erasure-burst-correction capabilities of some-finite-geometry cyclic LDPC codes (l_b is the actual erasure-burst-correction capability)

Geometry	Codes	Size of minimal stopping set	σ	l_b
EG(2,2^4)	(255,175)	17	54	70
EG(2,2^5)	(1023,781)	33	121	157
EG(2,2^6)	(4095,3367)	65	309	367
EG(2,2^7)	(16383,14197)	129	561	799
PG(2,2^4)	(273,191)	18	61	75
PG(2,2^5)	(1057,813)	34	90	124
PG(2,2^6)	(4161,3431)	66	184	303
PG(2,2^7)	(16513,14326)	130	1077	1179

Computing the length of the zero-covering-span of the parity-check matrix of a cyclic PG-LDPC code constructed from the two-dimensional projective geometry PG(2,q) given in Section 10.6 can be done in the same manner. The lengths of the zero-covering-spans σ of the parity-check matrices and the actual erasure-burst-correction capabilities l_b of some cyclic PG-LDPC codes are also given in Table 13.2.

13.5.2 Erasure-Burst-Correction with Superposition LDPC Codes

Consider the superposition product $C_{\text{sup},p} = C_1 \times C_2$ of an (n_1,k_1) LDPC code C_1 and an (n_2,k_2) LDPC code C_2 given by the null spaces of an $m_1 \times n_1$ and an $m_2 \times n_2$ low-density parity-check matrix, $\mathbf{H}_1$ and $\mathbf{H}_2$, respectively. Let σ_1 and σ_2 be the lengths of the zero-covering-spans of $\mathbf{H}_1$ and $\mathbf{H}_2$, respectively. Consider the parity-check matrix $\mathbf{H}_{\text{sup},p}$ of the superposition product code $C_{\text{sup},p}$ given by (12.27). On examining the structures of $\mathbf{H}_{\text{sup},p}$ and $\mathbf{H}_j^{(2)}$ given by (12.25), we can readily see that, for each column of $\mathbf{H}_{\text{sup},p}$, there is a 1-component in the lower part of $\mathbf{H}_{\text{sup},p}$ that is followed by a span of at least $\sigma_2(n_1 - 1)$ consecutive zeros. This implies that the length $\sigma_{\text{sup},p}$ of the zero-covering-span of $\mathbf{H}_{\text{sup},p}$ is at least $\sigma_2(n_1 - 1)$. Consequently, using $\mathbf{H}_{\text{sup},p}$ for decoding, the superposition product LDPC code $C_{\text{sup},p}$ is capable of correcting any erasure-burst of length up to at least $\sigma_2(n_1 - 1) + 1$ with the erasure-burst decoding algorithm presented in Section 13.4. The superposition product LDPC code $C_{\text{sup},p}$ generated by the parity-check matrix $\mathbf{H}_{\text{sup},p}$ given by (12.27) has C_1 as the row code and C_2 as the column code. If we switch the roles of C_1 and C_2, with C_2 as the row code and C_1 as the column code, then the length of the zero-covering-span of the parity-check matrix of the resultant product code $C^*_{\text{sup},p} = C_2 \times C_1$ is at least $\sigma_1(n_2 - 1)$. In this case, the superposition product LDPC code $C^*_{\text{sup},p}$ is capable of correcting any erasure-burst of length up to at least $\sigma_1(n_2 - 1) + 1$. Summarizing the above results, we can conclude that the erasure-burst-correction capability of the superposition product

of two LDPC codes, C_1 and C_2, is lower bounded as follows:

$$b_{\text{sup},p} = \max(\sigma_1(n_2 - 1) + 1, \sigma_2(n_1 - 1) + 1). \tag{13.8}$$

Consider the parity-check matrix $\mathbf{H}_{\text{sup},p}$ of a product QC-LDPC code $\mathcal{C}_{\text{sup},p}$ given by (12.29), constructed by superposition. The lower part of the matrix consists of a row of k identity matrices of size $r(q-1) \times r(q-1)$. On examining the structure of $\mathbf{H}_{\text{sup},p}$, we find that, for each column of $\mathbf{H}_{\text{sup},p}$, there is a 1-component in the lower part of $\mathbf{H}_{\text{sup},p}$ that is followed by a span of $r(q-1)-1$ zeros and then a 1-component. This zero-span is actual the span of the $r(q-1)-1$ zeros between a 1-component in a $r(q-1) \times r(q-1)$ identity matrix and the corresponding 1-component in the next $r(q-1) \times r(q-1)$ identity matrix of the lower part of $\mathbf{H}_{\text{sup},p}$, including the end-around case. Consequently, the length of the zero-covering-span of $\mathbf{H}_{\text{sup},p}$ is at least $r(q-1)-1$ (actually it is the exact length of the zero-covering-span of the parity-check matrix $\mathbf{H}_{\text{sup},p}$). Therefore, the product QC-LDPC code $\mathcal{C}_{\text{sup},p}$ given by the null space of $\mathbf{H}_{\text{sup},p}$ given by (12.29) is capable of correcting any erasure-burst of length up to at least $r(q-1)$ with the iterative decoding algorithm presented in the previous section.

Example 13.7. Consider the (3,6)-regular (4572,2307) product QC-LDPC code with rate 0.5015 given in Example 12.12 that was constructed using the superposition method presented in Section 12.7.4. The length of the zero-covering-span of the parity-check matrix $\mathbf{H}_{\text{sup},p}$ of this code is $(6 \times 127) - 1 = 761$. Then 762 is a lower bound on the erasure-burst-correction capability of the code. However, by computer search, we find that the actual erasure-burst-correction capability is $l_b = 1015$. In Figure 12.15, we showed that this code performs well over the AWGN channel with iterative decoding using the SPA. The performance of this code over the BEC with iterative decoding presented in Section 13.1 is shown in Figure 13.8. At a UEBR of 10^{-6}, we see that the code performs 0.11 from the Shannon limit, 0.4985. So, the code performs universally well over all three types of channels, AWGN, binary random, and erasure-burst channels.

13.6 Asymptotically Optimal Erasure-Burst-Correction QC-LDPC Codes

Asymptotically optimal QC-LDPC codes can be constructed by masking the arrays of CPMs constructed in Chapter 11 using a special class of masking matrices.

Let k, l, and s be three positive integers such that $2 \leq k, l$ and $1 \leq s < k$. Form l k-tuples over GF(2), $\mathbf{u}_0, \mathbf{u}_1, \ldots, \mathbf{u}_{l-1}$, where (1) $\mathbf{u}_0 = (1, 0, 0, \ldots, 0)$ consists of a single 1-component at the leftmost position followed by $k-1$ consecutive zeros; (2) $\mathbf{u}_{l-1} = (0, \ldots, 0, 1, \ldots, 1)$ consists of a sequence of s consecutive zeros followed by $k-s$ consecutive 1s; and (3) the other k-tuples, $\mathbf{u}_1$ to $\mathbf{u}_{l-2}$, are zero k-tuples, i.e., $\mathbf{u}_1 = \mathbf{u}_2 = \cdots = \mathbf{u}_{l-2} = (0, 0, \ldots, 0)$. Define the following kl-tuple over GF(2):

$$\mathbf{u} = (\mathbf{u}_0, \mathbf{u}_1, \ldots, \mathbf{u}_{l-1}). \tag{13.9}$$

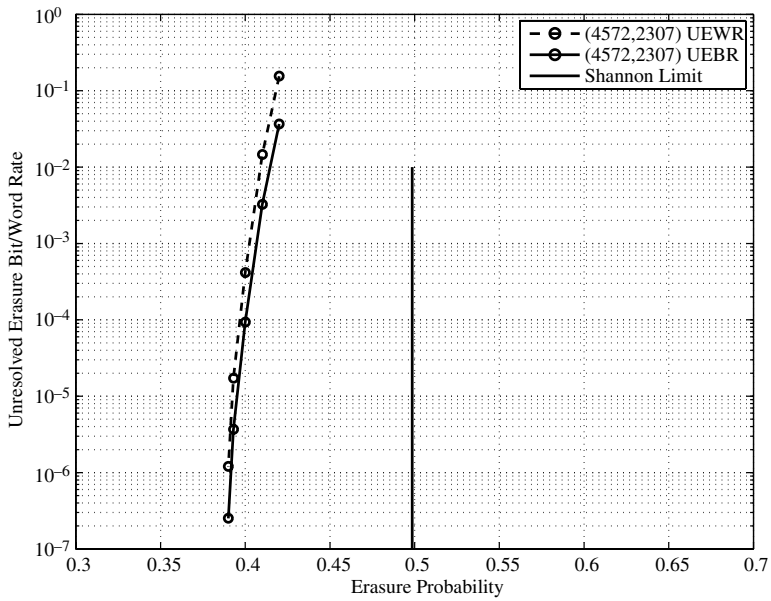


Figure 13.8 The error performance of the (4572,2307) QC-LDPC code over the BEC channel.

Then the kl-tuple $\mathbf{u}$ has a single 1-component at the left end and $k-s$ 1-components at the right end. It has weight $k-s+1$. This kl-tuple has one and only one zero-span of length $k(l-1)+s-1$.

Form a $kl \times kl$ circulant $\mathbf{G}$ with $\mathbf{u}$ as the generator. Then $\mathbf{G}$ has the following structural properties: (1) its column and row weights are both $k-s+1$; and (2) each row (or column) has a unique zero-span of length $k(l-1)+s-1$, including the end-around zero-span. For $t \geq 1$, we form the following $kl \times klt$ matrix over GF(2):

$$\mathbf{Z}(kl, klt) = [\mathbf{G} \quad \mathbf{G} \quad \cdots \quad \mathbf{G}], \tag{13.10}$$

which consists of a row of t $\mathbf{G}$-matrices. This matrix $\mathbf{Z}(kl, klt)$ has constant column and row weights $k-s+1$ and $t(k-s+1)$, respectively. It has the following structural properties: (1) each column is a downward cyclic-shift of the column on its left (including the transition across the boundary of two neighboring $\mathbf{G}$-matrices) and the first column is the downward cyclic-shift of the last column; (2) each row is the right cyclic-shift of the row above it and the first row is the right cyclic-shift of the last row; (3) each row consists of t zero-spans (including the end-around zero-span), each of length $k(l-1)+s-1$; and (4) each column has a unique zero-span of length $k(l-1)+s-1$. It follows from the above structural properties of $\mathbf{Z}(kl, klt)$ that the length of the zero-covering-span of each column of $\mathbf{Z}(kl, klt)$ is $k(l-1)+s-1$. Note that there are zero-spans in some rows of $\mathbf{Z}(kl, klt)$ that run across the boundary of two neighboring $\mathbf{G}$-matrices. Consequently, the length of the zero-covering-span of $\mathbf{Z}(kl, klt)$ is $\sigma = k(l-1)+s-1$.

Example 13.8. Let $k = 4$, $l = 3$, $t = 2$, and $s = 2$. Construct three 4-tuples over GF(2), $\mathbf{u}_0 = (1000)$, $\mathbf{u}_1 = (0000)$, and $\mathbf{u}_2 = (0011)$. Form the following 12-tuple over GF(2):

$$\mathbf{u} = (\mathbf{u}_0, \mathbf{u}_1, \mathbf{u}_2) = (100000000011).$$

Use $\mathbf{u}$ as the generator to form a 12×12 circulant $\mathbf{G}$ with both column weight and row weight 3. Then use two $\mathbf{G}$-matrices to form the following matrix $\mathbf{Z}(12, 24)$ with column and row weights 3 and 6, respectively:

$$\mathbf{Z}(12,24) = \begin{bmatrix}
1&0&0&0&0&0&0&0&0&0&1&1&1&0&0&0&0&0&0&0&0&0&1&1\\
1&1&0&0&0&0&0&0&0&0&0&1&1&1&0&0&0&0&0&0&0&0&0&1\\
1&1&1&0&0&0&0&0&0&0&0&0&1&1&1&0&0&0&0&0&0&0&0&0\\
0&1&1&1&0&0&0&0&0&0&0&0&0&1&1&1&0&0&0&0&0&0&0&0\\
0&0&1&1&1&0&0&0&0&0&0&0&0&0&1&1&1&0&0&0&0&0&0&0\\
0&0&0&1&1&1&0&0&0&0&0&0&0&0&0&1&1&1&0&0&0&0&0&0\\
0&0&0&0&1&1&1&0&0&0&0&0&0&0&0&0&1&1&1&0&0&0&0&0\\
0&0&0&0&0&1&1&1&0&0&0&0&0&0&0&0&0&1&1&1&0&0&0&0\\
0&0&0&0&0&0&1&1&1&0&0&0&0&0&0&0&0&0&1&1&1&0&0&0\\
0&0&0&0&0&0&0&1&1&1&0&0&0&0&0&0&0&0&0&1&1&1&0&0\\
0&0&0&0&0&0&0&0&1&1&1&0&0&0&0&0&0&0&0&0&1&1&1&0\\
0&0&0&0&0&0&0&0&0&1&1&1&0&0&0&0&0&0&0&0&0&1&1&1
\end{bmatrix}. \tag{13.11}$$

We can readily see that each row of $\mathbf{Z}(12, 24)$ has two zero-spans of length 9. The profile of the column zero-covering-span is $(9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9)$. The length of the zero-covering-span of $\mathbf{Z}(12, 24)$ is $\sigma = 9$.

Take a $kl \times klt$ subarray $\mathbf{H}(kl, klt)$ from any RC-constrained array $\mathbf{H}$ of $(q-1) \times (q-1)$ CPMs constructed in Chapter 11 as the base array for masking. If $\mathbf{H}$ contains zero matrices (such as $\mathbf{H}^{(1)}_{qc,\text{disp}}$), $\mathbf{H}(kl, klt)$ is taken avoiding the zero matrices in $\mathbf{H}$. On masking $\mathbf{H}(kl, klt)$ with $\mathbf{Z}(kl, klt)$, we obtain an RC-constrained $kl \times klt$ masked array $\mathbf{M}(kl, klt) = \mathbf{Z}(kl, klt) \circledast \mathbf{H}(kl, klt)$ of $(q-1) \times (q-1)$ circulant permutation and zero matrices. In each column of $\mathbf{M}(kl, klt)$ (as an array), there is a circulant permutation matrix in a row followed by $k(l-1) + s - 1$ consecutive $(q-1) \times (q-1)$ zero matrices and ending with a CPM. $\mathbf{M}(kl, klt)$ is an RC-constrained $kl(q-1) \times klt(q-1)$ matrix over GF(2) with column and row weights $k - s + 1$ and $t(k - s + 1)$, respectively. The length of the zero-covering-span of each column of $\mathbf{M}(kl, klt)$ is at least $[k(l-1) + s - 1](q-1)$. Consequently, the length of the zero-covering-span of $\mathbf{M}(kl, klt)$ is at least $[k(l-1) + s - 1](q-1)$. The null space of $\mathbf{M}(kl, klt)$ gives a $(k - s + 1, t(k - s + 1))$-regular QC-LDPC code $\mathcal{C}_{qc,\text{mas}}$ of length $klt(q-1)$, rate $(t-1)/t$, and minimum distance at least $k - s + 2$, whose Tanner graph has a girth of at least 6. The code is capable of correcting any erasure-burst of length up to

$$[k(l-1) + s - 1](q-1) + 1. \tag{13.12}$$

The number of parity-check bits of $\mathcal{C}_{qc,\text{mas}}$ is at most $kl(q-1)$. The erasure-burst-correction efficiency is

$$\eta \geq \frac{[k(l-1)+s-1](q-1)+1}{kl(q-1)}. \tag{13.13}$$

For large q, k, l and $k-s=2$ or 3 (or $k-s$ small relative to k), the above lower bound on η is close to 1. For $k-s=2$, $\mathcal{C}_{qc,\text{mas}}$ is a (3,3t)-regular QC-LDPC code, and for $k-s=3$, $\mathcal{C}_{qc,\text{mas}}$ is a (4,4t)-regular QC-LDPC code.

If the base array $\mathbf{H}(kl, klt)$ contains zero matrices, the above construction simply gives a near-regular code. The erasure-burst-correction of the code is still lower bounded by (13.12). If the base array $\mathbf{H}(kl, klt)$ is taken from the array $\mathbf{H}^{(6)}_{qc,\text{disp}}$ constructed from a prime field GF(p) (see Section 11.6), then $\mathbf{H}(kl, klt)$ is a $kl \times klt$ array of $p \times p$ circulant permutation matrices. In this case, the code $\mathcal{C}_{qc,\text{disp}}$ given by the null space of $\mathbf{M}(kl, klt)$ is capable of correcting any erasure-burst of length up to $[k(l-1)+s-1]p+1$.

Also, subarrays of the arrays of permutation matrices constructed on the basis of the decomposition of finite Euclidean geometries given in Section 10.3 (see (10.16)) can be used as base arrays for masking to construct asymptotically optimal LDPC codes for correcting erasure-bursts.

Example 13.9. Consider the RC-constrained 72×72 array $\mathbf{H}^{(1)}_{qc,\text{disp}}$ of 72×72 circulant permutation and zero matrices based on the (72,2,71) RS code over the prime field GF(73) (see Example 11.2). Set $k=4$, $l=2$, $t=8$, and $s=1$. Then $kl=8$ and $klt=64$. Take the first eight rows of $\mathbf{H}^{(1)}_{qc,\text{disp}}$ and remove the first and last seven columns. This results in an 8×64 subarray $\mathbf{H}^{(1)}_{qc,\text{disp}}(8,64)$ of CPMs (no zero matrices). This subarray $\mathbf{H}^{(1)}_{qc,\text{disp}}(8,64)$ will be used as the base array for masking to construct a QC-LDPC code for correcting erasure-bursts over the BEC. To construct an 8×64 masking matrix $\mathbf{Z}(8,64)$, we first form two 4-tuples over GF(2), $\mathbf{u}_0 = (1000)$ and $\mathbf{u}_1 = (0111)$. Concatenate these two 4-tuples to form an 8-tuple over GF(2), $\mathbf{u} = (\mathbf{u}_0\mathbf{u}_1) = (10000111)$. Use $\mathbf{u}$ as the generator to form the following 8×8 circulant over GF(2):

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

On the basis of (13.10), we construct an 8×64 masking matrix $\mathbf{Z}(8,64)$ that consists of a row of eight $\mathbf{G}$-matrices, i.e., $\mathbf{Z}(8,64) = [\mathbf{GGGGGGGG}]$. The length of the zero-covering-span of $\mathbf{Z}(8,64)$ is 4. The column and row weights of $\mathbf{Z}(8,64)$ are 4 and 32,

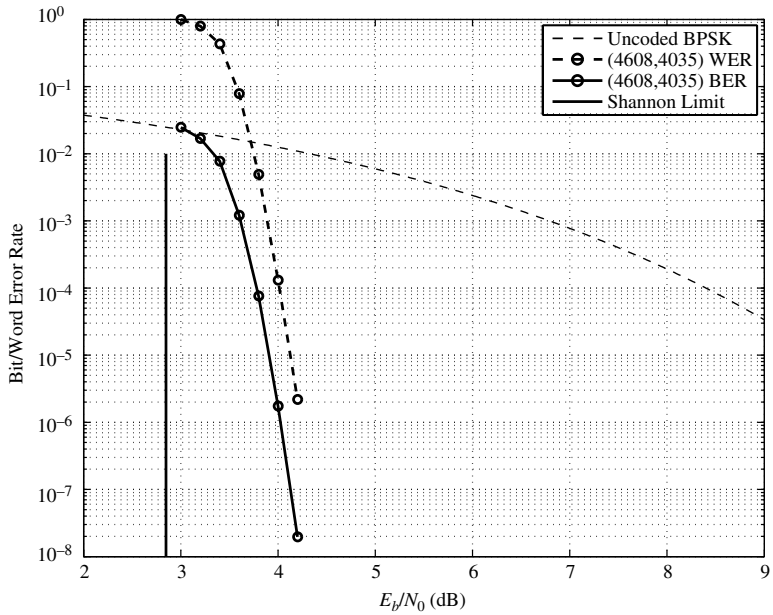


Figure 13.9 The error performance of the (4608,4035) QC-LDPC code over the binary-input AWGN channel given in Example 13.9.

respectively. By masking $\mathbf{H}^{(1)}_{qc,\text{disp}}(8,64)$ with $\mathbf{Z}(8,64)$, we obtain an RC-constrained 8×64 masked array $\mathbf{M}^{(1)}(8,64) = \mathbf{Z}(8,64) \circledast \mathbf{H}^{(1)}_{qc,\text{disp}}(8,64)$. This masked array is a 576×4608 matrix over GF(2) with column and row weights 4 and 32, respectively. The length σ of the zero-covering-span of $\mathbf{M}^{(1)}(8,64)$ is $4 \times 72 = 288$. The null space of $\mathbf{M}^{(1)}(8,64)$ gives a (4608,4035) QC-LDPC code with rate 0.8757 that is capable of correcting any erasure-burst of length up to $\sigma + 1 = 289$. By computer search, we find that the code can actually correct any erasure-burst of length up to 375. Thus, the erasure-burst-correction efficiency is 0.654. This (4608,4035) QC-LDPC code also performs well over the AWGN channel and the BEC as shown in Figures 13.9 and 13.10. For the AWGN, it performs 1.15 dB from the Shannon limit at a BER of 10^{-8}. For the BEC, it performs 0.045 from the Shannon limit, 0.1243.

Example 13.10. In this example, we construct a long QC-LDPC code with high erasure-burst-correction efficiency, which also performs well both over the AWGN channel and over the BEC. The array for code construction is the 256×256 array $\mathbf{H}^{(1)}_{qc,\text{disp}}$ of 256×256 circulant permutation and zero matrices constructed on the basis of the (256,2,255) RS code over GF(257) given in Example 11.1. Set $k = 4$, $l = 8$, $t = 8$, and $s = 1$. Then $kl = 32$ and $klm = 256$. Take a 32×256 subarray $\mathbf{H}^{(1)}_{qc,\text{disp}}(32,256)$ from $\mathbf{H}^{(1)}_{qc,\text{disp}}$, say the first 32 rows of $\mathbf{H}^{(1)}_{qc,\text{disp}}$, and use it as the base array for masking. Each of the first 32 columns of $\mathbf{H}^{(1)}_{qc,\text{disp}}(32,256)$ contains a single 256×256 zero matrix. Next, we construct a 32-tuple $\mathbf{u} = (\mathbf{u}_0, \mathbf{u}_1, \ldots, \mathbf{u}_7)$, where $\mathbf{u}_0 = (1000)$, $\mathbf{u}_7 = (0111)$, and $\mathbf{u}_1 = \cdots = \mathbf{u}_6 = (0000)$.

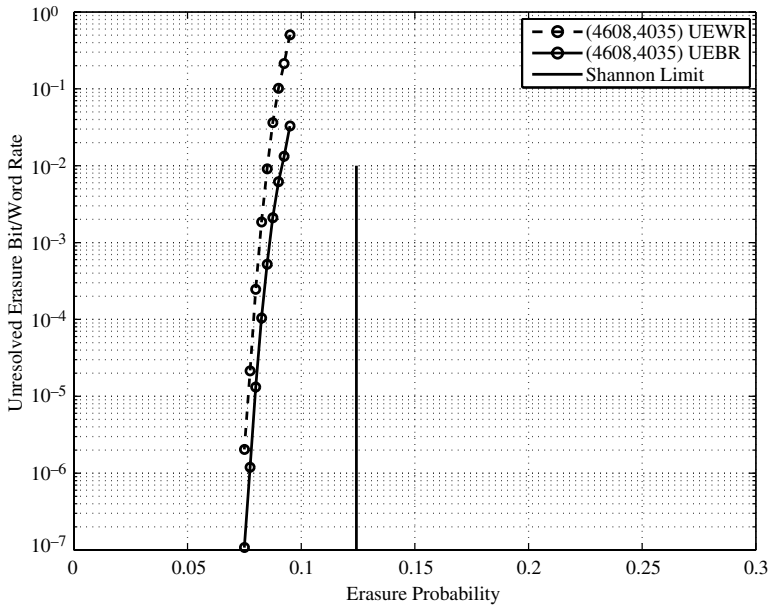


Figure 13.10 The error performance of the (4608,4035) QC-LDPC code over the BEC given in Example 13.9.

Construct a 32×32 circulant $\mathbf{G}$ with $\mathbf{u}$ as the generator. Both the column weight and the row weight of $\mathbf{G}$ are 4. Then form a 32×256 masking matrix $\mathbf{Z}(32, 256)$ with eight $\mathbf{G}$-matrices arranged in a row, i.e., $\mathbf{Z}(32, 258) = [\mathbf{GGGGGGGG}]$, which has column and row weights 4 and 32, respectively. By masking $\mathbf{H}^{(1)}_{qc,\text{disp}}(32, 256)$ with $\mathbf{Z}(32, 256)$, we obtain a 32×256 masked matrix $\mathbf{M}^{(1)}(32, 256) = \mathbf{Z}(32, 256) \circledast \mathbf{H}^{(1)}_{qc,\text{disp}}(32, 256)$, which is an 8192×65536 matrix with two different column weights, 3 and 4, and two different row weights, 31 and 32. The length of the zero-covering-span σ of $\mathbf{M}^{(1)}(32, 256)$ is $[k(l-1) + s - 1](q-1) = [4(8-1) + 1 - 1] \times 256 = 7168$. The null space of $\mathbf{M}^{(1)}(32, 256)$ gives a (65536,57345) near-regular QC-LDPC code with rate 0.875. The code is capable of correcting any erasure-burst of length at least up to 7169. Its erasure-burst-correction efficiency is lower bounded by 0.8752. The error performances of this code over the binary-input AWGN channel and BEC are shown in Figures 13.11 and 12, respectively. For the AWGN channel, the code performs 0.6 dB from the Shannon limit at a BER of 10^{-6}. For the BEC, it performs 0.03 from the Shannon limit, 0.125.

13.7 Construction of QC-LDPC Codes by Array Dispersion

A subarray of the array $\mathbf{H}$ of circulant permutation matrices constructed in Chapters 11 and 12 can be dispersed into a larger array with a lower density to construct new QC-LDPC codes. In this section, we present a *dispersion technique* [9] by which to construct a large class of QC-LDPC codes. Codes constructed by this

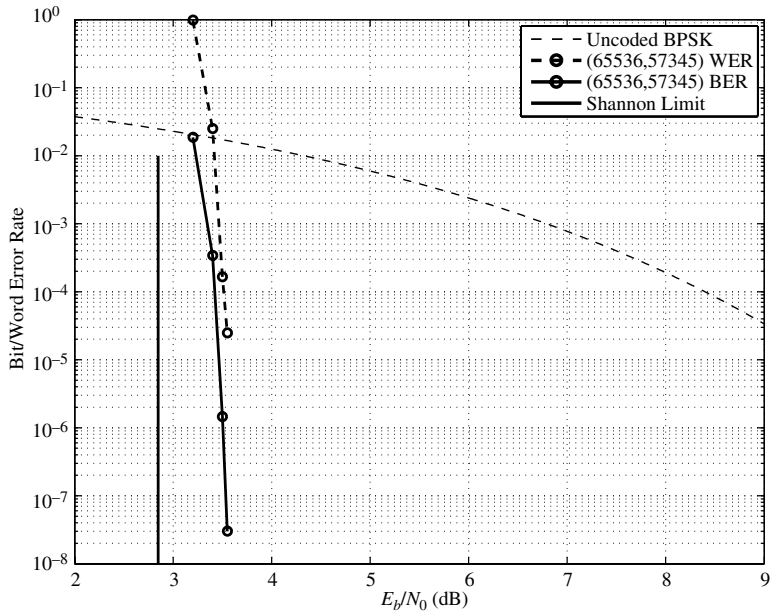


Figure 13.11 The error performance of the (65536,57345) QC-LDPC code over the AWGN channel given in Example 13.10.

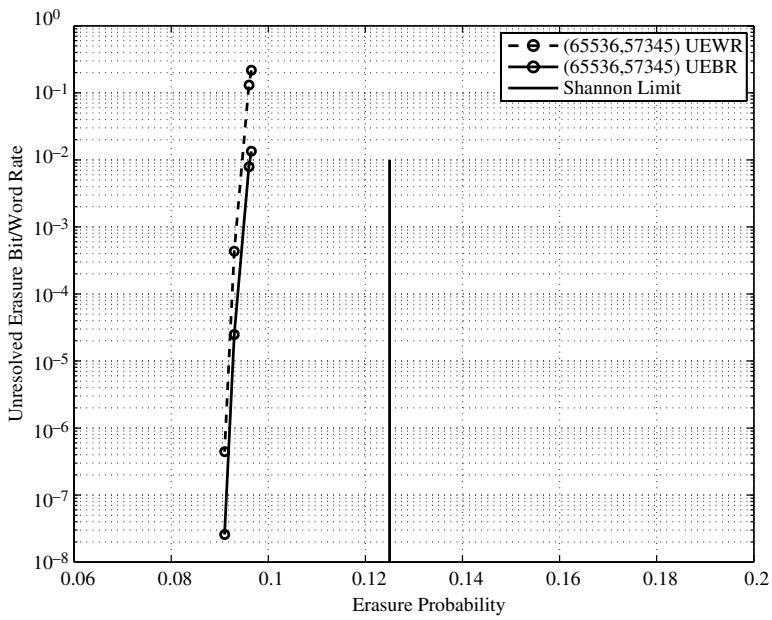


Figure 13.12 The error performance of the (65536,57345) QC-LDPC code over the BEC given in Example 13.10.

dispersion technique not only have good *erasure-burst*-correction capabilities but also perform well over the AWGN and binary random erasure channels.

Let $\mathbf{H}$ be an $c \times n$ RC-constrained array of $(q-1) \times (q-1)$ CPMs as constructed in either Chapter 11 or Chapter 12. The values of c and n depend on the method that is used to construct $\mathbf{H}$. For example, if $\mathbf{H}$ is the array $\mathbf{H}^{(1)}_{qc,\text{disp}}$ given by (11.6) constructed from the $(q-1,\ 2,\ q-2)$ RS code over GF(q), then $c = n = q-1$. Let k and t be two positive integers such that $1 < k$, $2 < t \leq c$, and $kt \leq n$, and let $\mathbf{H}(t, kt)$ be a $t \times kt$ subarray of $\mathbf{H}$. We assume that $\mathbf{H}(t, kt)$ does not contain any zero submatrix of $\mathbf{H}$ (if any exist). Divide $\mathbf{H}(t, kt)$ into k $t \times t$ subarrays, $\mathbf{H}_1(t,t), \mathbf{H}_2(t,t), \ldots, \mathbf{H}_k(t,t)$, such that

$$\mathbf{H}(t, kt) = [\mathbf{H}_1(t,t) \quad \mathbf{H}_2(t,t) \quad \ldots \quad \mathbf{H}_k(t,t)], \tag{13.14}$$

where for $1 \leq j \leq k$, the jth $t \times t$ subarray $\mathbf{H}_j(t,t)$ is expressed in the following form:

$$\mathbf{H}_j(t,t) = \begin{bmatrix} \mathbf{A}^{(j)}_{0,0} & \mathbf{A}^{(j)}_{0,1} & \cdots & \mathbf{A}^{(j)}_{0,t-1} \\ \mathbf{A}^{(j)}_{1,0} & \mathbf{A}^{(j)}_{1,1} & \cdots & \mathbf{A}^{(j)}_{1,t-1} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{A}^{(j)}_{t-1,0} & \mathbf{A}^{(j)}_{t-1,1} & \cdots & \mathbf{A}^{(j)}_{t-1,t-1} \end{bmatrix}, \tag{13.15}$$

where each $\mathbf{A}^{(j)}_{i,l}$ with $0 \leq i, l < t$ is a $(q-1) \times (q-1)$ CPM over GF(2). Since $\mathbf{H}(t, kt)$ satisfies the RC-constraint, each subarray $\mathbf{H}_j(t,t)$ also satisfies the RC-constraint.

Cut $\mathbf{H}_j(t,t)$ into two *triangles*, namely *upper* and *lower triangles*, along its main diagonal, where the lower triangle contains the CPMs on the main diagonal of $\mathbf{H}_j(t,t)$. Form two $t \times t$ arrays of circulant permutation and zero matrices as follows:

$$\mathbf{H}_{j,U}(t,t) = \begin{bmatrix} \mathbf{O} & \mathbf{A}^{(j)}_{0,1} & \mathbf{A}^{(j)}_{0,2} & \cdots & \mathbf{A}^{(j)}_{0,t-1} \\ \mathbf{O} & \mathbf{O} & \mathbf{A}^{(j)}_{1,2} & \cdots & \mathbf{A}^{(j)}_{1,t-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \mathbf{O} & \mathbf{O} & \mathbf{O} & \cdots & \mathbf{A}^{(j)}_{t-2,t-1} \\ \mathbf{O} & \mathbf{O} & \mathbf{O} & \cdots & \mathbf{O} \end{bmatrix} \tag{13.16}$$

and

$$\mathbf{H}_{j,L}(t,t) = \begin{bmatrix} \mathbf{A}^{(j)}_{0,0} & \mathbf{O} & \mathbf{O} & \cdots & \mathbf{O} \\ \mathbf{A}^{(j)}_{1,0} & \mathbf{A}^{(j)}_{1,1} & \mathbf{O} & \cdots & \mathbf{O} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \mathbf{A}^{(j)}_{t-2,0} & \mathbf{A}^{(j)}_{t-2,1} & \mathbf{A}^{(j)}_{t-2,2} & \cdots & \mathbf{O} \\ \mathbf{A}^{(j)}_{t-1,0} & \mathbf{A}^{(j)}_{t-1,1} & \mathbf{A}^{(j)}_{t-1,2} & \cdots & \mathbf{A}^{(j)}_{t-1,t-1} \end{bmatrix}, \tag{13.17}$$

where $\mathbf{O}$ is a $(q-1) \times (q-1)$ zero matrix. From (13.16), we see that the upper triangle of the $t \times t$ array $\mathbf{H}_{j,U}(t,t)$ above the main diagonal is identical to the upper triangle of $\mathbf{H}_j(t,t)$ above the main diagonal; and the rest of the submatrices

in $\mathbf{H}_{j,U}(t,t)$ are zero matrices. From (13.17), we see that the lower triangle of $\mathbf{H}_{j,L}(t,t)$ including the submatrices on the main diagonal is identical to that of $\mathbf{H}_j(t,t)$; and the submatrices above the main diagonal are zero matrices. Since $\mathbf{H}_j(t,t)$ satisfies the RC constraint, it is clear that $\mathbf{H}_{j,U}(t,t)$ and $\mathbf{H}_{j,L}(t,t)$ also satisfy the RC-constraint.

For $1 \leq j \leq k$ and $2 \leq l$, we form the following $l \times l$ array of $t \times t$ subarrays:

$$\mathbf{H}_{j,l\text{-f,disp}}(lt,lt) = \begin{bmatrix} \mathbf{H}_{j,L}(t,t) & \mathbf{O} & \mathbf{O} & \cdots & \mathbf{O} & \mathbf{H}_{j,U}(t,t) \\ \mathbf{H}_{j,U}(t,t) & \mathbf{H}_{j,L}(t,t) & \mathbf{O} & \cdots & \mathbf{O} & \mathbf{O} \\ \mathbf{O} & \mathbf{H}_{j,U}(t,t) & \mathbf{H}_{j,L}(t,t) & \cdots & \mathbf{O} & \mathbf{O} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \mathbf{O} & \mathbf{O} & \mathbf{O} & \cdots & \mathbf{H}_{j,U}(t,t) & \mathbf{H}_{j,L}(t,t) \end{bmatrix}, \quad (13.18)$$

where $\mathbf{O}$ is a $t \times t$ array of $(q-1) \times (q-1)$ zero matrices. From (13.18), we see that each row of $\mathbf{H}_{j,l\text{-f,disp}}(lt,lt)$ is a right cyclic-shift of the row above it and the first row is the right cyclic-shift of the last row. Also, the $t \times t$ subarrays, $\mathbf{H}_{j,L}(t,t)$ and $\mathbf{H}_{j,U}(t,t)$, in $\mathbf{H}_{j,l\text{-f,disp}}(lt,lt)$ are separated by a *span* of $l-2$ $t \times t$ zero subarrays, including the *end-around* case with $\mathbf{H}_{j,L}(t,t)$ as the *starting* subarray and $\mathbf{H}_{j,U}(t,t)$ as the *ending* subarray. From (13.16)–(13.18), we readily see that the CPMs in each row (or each column) of $\mathbf{H}_{j,l\text{-f,disp}}(lt,lt)$ together form the jth subarray $\mathbf{H}_j(t,t)$ of the $t \times kt$ array $\mathbf{H}(t,kt)$ given by (13.14). $\mathbf{H}_{j,l\text{-f,disp}}(lt,lt)$ is called the *l-fold dispersion* of $\mathbf{H}_j(t,t)$, where the subscripts, "l-f" and "disp" of $\mathbf{H}_{j,l\text{-f,disp}}(lt,lt)$ stand for "l-fold" and "dispersion," respectively. $\mathbf{H}_{j,l\text{-f,disp}}(lt,lt)$ is an $lt(q-1) \times lt(q-1)$ matrix over GF(2) with both column weight and row weight t. Since $\mathbf{H}_j(t,t)$, as a $t(q-1) \times t(q-1)$ matrix, satisfies the RC-constraint, it follows readily from (13.18) that the l-fold dispersion of $\mathbf{H}_j(t,t)$, as an $lt(q-1) \times lt(q-1)$ matrix, also satisfies the RC-constraint. Since any two subarrays in $\mathbf{H}(t,kt)$ given by (13.14) jointly satisfy the RC-constraint, their l-fold dispersions jointly satisfy the RC constraint.

Now we view $\mathbf{H}_{j,l\text{-f,disp}}(lt,lt)$ as an $lt \times lt$ array of $(q-1) \times (q-1)$ circulant permutation and zero matrices. From the structures of $\mathbf{H}_{j,U}(t,t)$, $\mathbf{H}_{j,L}(t,t)$ and $\mathbf{H}_{j,l\text{-f,disp}}(lt,lt)$ given by (13.16), (13.17), and (13.18), respectively, we readily see that each row (or each column) of $\mathbf{H}_{j,l\text{-f,disp}}(lt,lt)$ contains a single span of $(l-1)t$ zero matrices of size $(q-1) \times (q-1)$ between two CPMs, including the end-around case. For $0 \leq s < t$, by replacing the s CPMs right after the single span of zero matrices by s $(q-1) \times (q-1)$ zero matrices, we obtain a new $lt \times lt$ array $\mathbf{H}_{j,l\text{-f,disp},s}(lt,lt)$ of circulant permutation and zero matrices. $\mathbf{H}_{j,l\text{-f,disp},s}(lt,lt)$ is called the *s-masked* and *l-fold dispersion* of $\mathbf{H}_j(t,t)$. Each row of $\mathbf{H}_{j,l\text{-f,disp},s}(lt,lt)$ contains a single span of $(l-1)t+s$ zero matrices of size $(q-1) \times (q-1)$. The number s is called the *masking parameter*.

On replacing each $t \times t$ subarray in the $t \times kt$ array $\mathbf{H}(t,kt)$ of (13.14) by its s-masked l-fold dispersion, we obtain the following $lt \times klt$ array of $(q-1) \times (q-1)$ circulant permutation and zero matrices over GF(2):

$$\mathbf{H}_{l\text{-f,disp},s}(lt,lkt) = [\mathbf{H}_{1,l\text{-f,disp},s}(lt,lt) \quad \mathbf{H}_{2,l\text{-f,disp},s}(lt,lt) \quad \cdots \quad \mathbf{H}_{k,l\text{-f,disp},s}(lt,lt)]. \quad (13.19)$$

$\mathbf{H}_{l\text{-f,disp},s}(lt, klt)$ is referred to as the *s-masked* and *l-fold dispersion* of the array $\mathbf{H}(t, kt)$ given by (13.14). As an $lt \times klt$ array of circulant permutation and zero matrices, each row of $\mathbf{H}_{l\text{-f,disp},s}(lt, klt)$ contains k spans of zero matrices, each consisting of $(l-1)t+s$ zero matrices of size $(q-1)\times(q-1)$, including the end-around case. $\mathbf{H}_{l\text{-f,disp},s}(lt, klt)$ is an $lt(q-1) \times klt(q-1)$ matrix over GF(2) with column and row weights $t-s$ and $k(t-s)$, respectively. It satisfies the RC-constraint.

The null space over GF(2) of $\mathbf{H}_{l\text{-f,disp},s}(lt, klt)$ gives a $(t-s,k(t-s))$-regular QC-LDPC code $\mathcal{C}_{l\text{-f,disp},s}$ of length $klt(q-1)$ with rate at least $(k-1)/k$, whose Tanner graph has a girth of at least 6. The above construction by multi-fold array dispersion gives a large class of QC-LDPC codes. This multi-fold array dispersion allows us to construct long codes of various rates. There are five degrees of freedom in the code construction, namely, q, k, l, s, and t. The parameters k and t are limited by n, i.e., $kt \leq n$. To avoid having a column weight of $\mathbf{H}_{l\text{-f,disp},s}(kt, klt)$ less than 3, we need to choose s and t such that $t-s \geq 3$. However, there is no limitation on l. Therefore, for given q, k, s, and t, we can construct very long codes over the same field GF(2) by varying l. A special subclass of QC-LDPC codes is obtained by choosing s and t such that $t-s=3$. This gives a class of $(3,3k)$-regular QC-LDPC codes. On setting $k = 2, 3, 4, 5, \ldots$, we obtain a sequence of codes with rational rates equal to (or at least equal to) $1/2, 2/3, 3/4, 4/5, \ldots$ If we choose t and s such that $t-s=4$, then we obtain a class of $(4,4k)$-regular QC-LDPC codes.

Next, we show that the QC-LDPC codes constructed by array dispersion given above are effective for correcting erasure-bursts. Consider the s-masked and l-fold dispersed $lt \times klt$ array $\mathbf{H}_{l\text{-f,disp},s}(lt, klt)$ of $(q-1) \times (q-1)$ circulant permutation and zero matrices over GF(2) given by (13.19). For $1 \leq j \leq k$, the jth subarray $\mathbf{H}_{j,l\text{-f,disp},s}(lt, lt)$ of $\mathbf{H}_{l\text{-f,disp},s}(lt, klt)$ is an $lt \times lt$ array of $(q-1) \times (q-1)$ circulant permutation and zero matrices. From the structure of the arrays, $\mathbf{H}_{j,U}(t,t)$, $\mathbf{H}_{j,L}(t,t)$, and $\mathbf{H}_{j,l\text{-f,disp}}(lt, lt)$ given by (13.16), (13.17), and (13.18), respectively, we readily see that, for each column of $\mathbf{H}_{j,l\text{-f,disp},s}(lt, lt)$, there is a row with a CPM in that column, which is followed horizontally along the row by a span of $(l-1)t+s$ consecutive $(q-1) \times (q-1)$ zero matrices before it is ended with another CPM in the same row, including the end-around case. Now we consider $\mathbf{H}_{j,l\text{-f,disp},s}(lt, lt)$ as an $lt(q-1) \times lt(q-1)$ matrix over GF(2). Then, for each column of $\mathbf{H}_{j,l\text{-f,disp},s}(lt, lt)$, there is a row with a nonzero entry in that column, which is followed horizontally along the row by a span of at least $[(l-1)t+s](q-1)$ zero entries before it is ended with another nonzero entry in the same row, including the end-around case.

Since all the subarrays,

$$\mathbf{H}_{1,l\text{-f,disp},s}(lt, lt), \mathbf{H}_{2,l\text{-f,disp},s}(lt, lt), \ldots, \mathbf{H}_{k,l\text{-f,disp},s}(lt, lt),$$

of the s-masked l-fold dispersed array $\mathbf{H}_{l\text{-f,disp},s}(lt, klt)$ have identical structure, for each column of the array $\mathbf{H}_{l\text{-f,disp},s}$ there is a row with a CPM in that column, which is followed horizontally along the row across all the boundaries of neighboring subarrays by a span of $(l-1)t+s$ consecutive $(q-1) \times (q-1)$ zero matrices,

including the end-around case. Now view $\mathbf{H}_{l\text{-f,disp},s}(lt, klt)$ as an $lt(q-1) \times klt(q-1)$ matrix over GF(2). Then, for each column in $\mathbf{H}_{l\text{-f,disp},s}(lt, klt)$, there is a row with a nonzero entry in that column, which is followed horizontally along the row by a span of at least $[(l-1)t+s](q-1)$ zero entries before it is ended with another nonzero entry in the same row, including the end-around case. Therefore, the length of the zero-covering-span of each column of $\mathbf{H}_{l\text{-f,disp},s}(lt, klt)$ is at least $[(l-1)t+s](q-1)$ and hence the length of the zero-covering-span of the s-masked and l-fold dispersed $lt(q-1) \times klt(q-1)$ matrix $\mathbf{H}_{l\text{-f,disp},s}(lt, klt)$ over GF(2) is lower bounded by $[(l-1)t+s)](q-1)$.

Consider the QC-LDPC code $\mathcal{C}_{l\text{-f,disp},s}$ given by the null space of the $lt(q-1) \times klt(q-1)$ matrix $\mathbf{H}_{l\text{-f,disp},s}(lt, klt)$ over GF(2). Then, with the simple iterative decoding algorithm presented in Section 13.4, the code $\mathcal{C}_{l\text{-f,disp},s}$ is capable of correcting any erasure-burst of length up to $[(l-1)t+s](q-1)+1$. Since the row rank of $\mathbf{H}_{l\text{-f,disp},s}(lt, klt)$ is at most $lt(q-1)$, the erasure-burst-correction efficiency η is at least

$$\frac{[(l-1)t+s](q-1)+1}{lt(q-1)} \approx \frac{(l-1)t+s}{lt}. \tag{13.20}$$

For large l, the erasure-burst-correction efficiency of $\mathcal{C}_{l\text{-f,disp},s}$ approaches unity. Therefore, the class of QC-LDPC codes constructed by array dispersion is asymptotically optimal for correcting busts of erasures.

Example 13.11. Suppose we construct a 63×63 array $\mathbf{H}^{(1)}_{qc,\text{disp}}$ of 63×63 circulant permutation and zero matrices based on the (63,2,62) RS code over GF(2^6) using the construction method presented in Section 11.3 (see (11.6)). Set $k = 2$, $l = 8$, $s = 5$, and $t = 8$. Take an 8×16 subarray $\mathbf{H}(8, 16)$ from $\mathbf{H}^{(1)}_{qc,\text{disp}}$, avoiding zero matrices on the main diagonal of $\mathbf{H}^{(1)}_{qc,\text{disp}}$. The 5-masked and 8-fold dispersion of $\mathbf{H}(8, 16)$ gives a 64×128 array $\mathbf{H}_{8\text{-f,disp},s}(64, 128)$ of 63×63 circulant permutation and zero matrices. It is a 4032×8064 matrix over GF(2) with column and row weights 3 and 6, respectively. The null space of this matrix gives an (8064,4032) QC-LDPC code with rate 0.5. This code is capable of correcting any erasure-burst of length up to at least 3844. Hence the erasure-burst-correction efficiency of this code is at least 0.9533.

If we keep $k = 2$, $t = 8$, and $s = 5$, and let $l = 16$, 32, and 64, we obtain three long codes with rates 0.5. They are (16128,8064), (32256,16128), and (64512,32256) QC-LDPC codes. They can correct erasure-bursts of lengths up to at least 7876, 15939, and 32067, respectively. Their erasure-burst-correction efficiencies are 0.9767, 0.9883, and 0.9941.

For small l, say $l = 2$, QC-LDPC codes constructed by array dispersion also perform well over the AWGN channel. This is illustrated in the next example.

Example 13.12. Utilizing the four-dimensional Euclidean geometry EG(4,2^2) over GF(2^2), (11.19), (11.20), and (11.21), we can form a 4×84 array $\mathbf{H}^{(8)}_{qc,\text{EG},2}$ array of 255×255 CPMs. Set $k = 5$, $l = 2$, $s = 0$, and $t = 4$. Take a 4×20 subarray $\mathbf{H}(4, 20)$

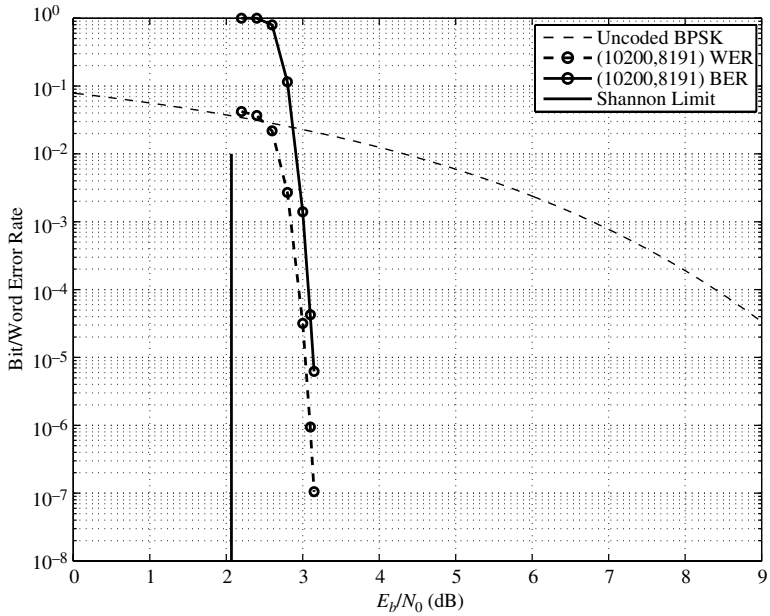


Figure 13.13 The error performance of the (10200,8191) QC-LDPC code over the binary-input AWGN channel given in Example 13.12.

from $\mathbf{H}^{(8)}_{qc,\mathrm{EG},2}$. The 0-masked and 2-fold dispersion of $\mathbf{H}(4,20)$ gives an 8×40 array $\mathbf{H}_{2\text{-f},\mathrm{disp},0}(8,40)$ of 255×255 circulant permutation and zero matrices. It is a 2040×10200 matrix over GF(2) with column and row weights 4 and 20, respectively. The null space of $\mathbf{H}_{2\text{-f},\mathrm{disp},0}$ gives a (10200,8191) QC-LDPC code with rate 0.803; the lower bound on its erasure-burst-correction capability is 1021. However, by computer search, it is found that the code can actually correct any erasure-burst of length up to 1366. The performances of this code over the AWGN and binary random erasure channels decoded with iterative decoding are shown in Figures 13.13 and 13.14, respectively. We see that the (10200,8191) QC-LDPC code performs well over both channels. For the AWGN channel, the code performs 1 dB from the Shannon limit at a BER of 10^{-6}. For the BEC, the code performs 0.055 from the Shannon limit, 0.197, at a UEBR of 10^{-6}.

13.8 Cyclic Codes for Correcting Bursts of Erasures

Except for the cyclic EG- and PG-LDPC codes presented in Sections 10.1.1 and 10.6, no other well-known cyclic codes perform well with iterative decoding over either the AWGN or random erasure channels. However, cyclic codes are very effective for correcting bursts of erasures with the simple iterative decoding algorithm presented in Section 13.4.

Consider an (n,k) cyclic code $\mathcal{C}$ over GF(q) with generator polynomial

$$\mathbf{g}(X) = g_0 + g_1 X + \cdots + g_{n-k-1} X^{n-k-1} + X^{n-k},$$

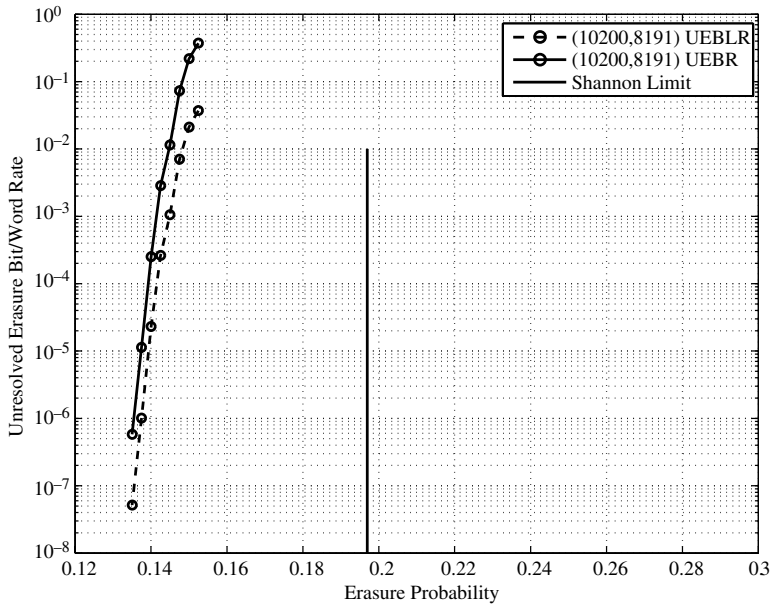


Figure 13.14 The error performance of the (10200,8191) QC-LDPC code over the BEC given in Example 13.12.

where $g_0 \neq 0$ and $g_i \in \mathrm{GF}(q)$ (see (3.48)). Its parity-check polynomial is given by

$$\mathbf{h}(X) = h_0 + h_1 X + \cdots + h_{k-1}X^{k-1} + X^k = \frac{X^n - 1}{g_0^{-1} X^{n-k}\mathbf{g}(X^{-1})}, \tag{13.21}$$

where $h_0 \neq 0$, $h_i \in \mathrm{GF}(q)$, and g_0^{-1} is the multiplicative inverse of g_0. Let $\mathcal{C}_d$ be the dual code of $\mathcal{C}$. Then the parity-check polynomial $\mathbf{h}(X)$ of $\mathcal{C}$ is the generator polynomial of $\mathcal{C}_d$, which is an $(n,n-k)$ cyclic code over $\mathrm{GF}(q)$. The n-tuple over $\mathrm{GF}(q)$ corresponding to $\mathbf{h}(X)$,

$$\mathbf{h} = (h_0, h_1, \ldots, h_{k-1}, 1, 0, 0, \ldots, 0), \tag{13.22}$$

is called the *parity vector* of the cyclic code $\mathcal{C}$ and is a codeword in the dual code $\mathcal{C}_d$ of $\mathcal{C}$. The rightmost $n-k-1$ components of $\mathbf{h}$ are zeros. Therefore, $\mathbf{h}$ has a zero-span of length $n-k-1$. The following lemma proves that the length of this zero-span of $\mathbf{h}$ is the longest.

Lemma 13.1. *The maximum length of a zero-span in the parity-vector* $\mathbf{h}$ *of an (n,k) cyclic code over GF(q) is* $n-k-1$.

Proof. First we know that $\mathbf{h}$ has a zero-span of length $n-k-1$ that consists of the $n-k-1$ zeros at the rightmost $n-k-1$ positions of $\mathbf{h}$. For $k < n-k$, the lemma is obviously true. Hence, we need only prove the lemma for $k \geq n-k$. Let $\mathbf{z}_0$ denote the zero-span that consists of the rightmost $n-k-1$ zeros of $\mathbf{h}$.

Suppose there is a zero-span $\mathbf{z}$ of length $\lambda \geq n-k$ in $\mathbf{h}$. Since h_0 and $h_k = 1$ are nonzero, this zero-span $\mathbf{z}$ starts at position i for some i such that $1 \leq i \leq k-\lambda$. We cyclically shift $\mathbf{h}$ until the zero-span $\mathbf{z}$ has been shifted to the rightmost λ positions of a new n-tuple $\mathbf{h}^* = (h_0^*, h_1^*, \ldots, h_{n-\lambda-1}^*, 0, 0, \ldots, 0)$, where h_0^* and $h_{n-\lambda-1}^*$ are nonzero. This new n-tuple $\mathbf{h}^*$ and its λ cyclic-shifts form a set of $\lambda + 1 > n-k$ linearly independent vectors and they are codewords in the dual code $\mathcal{C}_d$ of $\mathcal{C}$. However, this contradicts the fact that $\mathcal{C}_d$, of dimension $n-k$, has at most $n-k$ linearly independent codewords. This proves the lemma. □

Form an $n \times n$ matrix over GF(q) with the parity vector $\mathbf{h}$ and its $n-1$ cyclic-shifts as rows as follows:

$$\mathbf{H}_n = \begin{array}{c} n-k \left\{ \right. \\ \\ \\ \\ \\ \\ \\ \\ k \left\{ \right. \\ \\ \\ \\ \end{array} \left[\begin{array}{cccccccccc} h_0 & h_1 & h_2 & \cdots & h_{k-1} & 1 & 0 & 0 & \cdots & 0 \\ 0 & h_0 & h_1 & h_2 & \cdots & h_{k-1} & 1 & 0 & \cdots & 0 \\ 0 & 0 & h_0 & h_1 & h_2 & \cdots & h_{k-1} & 1 & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & 0 & h_0 & h_1 & h_2 & \cdots & h_{k-1} & 1 \\ \hdashline 1 & 0 & 0 & \cdots & 0 & h_0 & h_1 & h_2 & \cdots & h_{k-1} \\ h_{k-1} & 1 & 0 & 0 & \cdots & 0h_0 & h_1 & h_2 & \cdots & h_{k-2} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ h_1 & h_0 & \cdots & h_{k-1} & 1 & 0 & 0 & \cdots & 0 & h_0 \end{array} \right]. \tag{13.23}$$

Since $\mathcal{C}_d$, the dual code of $\mathcal{C}$, is cyclic and the first row $(h_0, h_1, \ldots, h_{k-1}, 1, 0, 0, \ldots, 0)$ of $\mathbf{H}_n$ is a codeword in $\mathcal{C}_d$, all the rows of $\mathbf{H}_n$ are codewords in $\mathcal{C}_d$. Furthermore, the first $n-k$ rows of $\mathbf{H}_n$ are linearly independent and they form a full rank parity-check matrix, $\mathbf{H}_{n-k}$, which is often used to define $\mathcal{C}$ (see (3.30)), i.e., $\mathcal{C}$ is the null space of $\mathbf{H}_{n-k}$. The matrix $\mathbf{H}_n$ is simply a *redundant expansion* of the parity-check matrix $\mathbf{H}_{n-k}$. Therefore, $\mathbf{H}_n$ is also a parity-check matrix of $\mathcal{C}$.

From $\mathbf{H}_n$, we see that each row has a zero-span of length $n-k-1$ between $h_k = 1$ and h_0 (including the end-around case), each starting at a different position. This implies that every column of $\mathbf{H}_n$ has a zero-covering-span of length $n-k-1$ and hence the length of the zero-covering-span of $\mathbf{H}_n$ is $n-k-1$. Suppose a codeword $\mathbf{v}$ in $\mathcal{C}$ is transmitted over a q-ary erasure-burst channel. Let $\mathbf{r} = (r_0, r_1, \ldots, r_{n-1})$ be the corresponding received sequence with an erasure-burst pattern $\mathcal{E}$ of length $n-k$ or less. Suppose the starting position of the erasure-burst $\mathcal{E}$ is j with $0 \leq j < n$. Then, there exists a row $\mathbf{h}_i = (h_{i,0}, h_{i,1}, \ldots, h_{i,n-1})$ in $\mathbf{H}_n$ for which the jth component $h_{i,j}$ is equal to the 1-element of GF(q) (see $\mathbf{H}_n$ given by (13.23)) and followed by a zero-span of length $n-k-1$. Therefore, the row checks only

the erasure at the jth position but not other erasures in $\mathcal{E}$. On setting the inner product $\langle \mathbf{r}, \mathbf{h}_i \rangle = 0$, we have the following equation:

$$h_{i,0}r_0 + h_{i,1}r_1 + \cdots + h_{i,n-1}r_{n-1} = 0,$$

which contains only one unknown r_j, the erased symbol at the jth position. From this equation, we can determine the value of the jth transmitted symbol as follows:

$$v_j = r_j = -h_{i,j}^{-1} \sum_{l=0, l \neq j}^{n-1} r_l h_{i,l}. \tag{13.24}$$

Once the symbol v_j has been recovered, the index j is removed from the erasure pattern $\mathcal{E}$. The procedure can be repeated to recover all the erased symbols in $\mathcal{E}$ iteratively using the decoding algorithm presented in Section 13.4.

Since the length of the zero-covering-span of $\mathbf{H}_n$ is $n - k - 1$, $\mathcal{C}$ is capable of correcting any erasure-burst of length up to $n - k$, which is the limit of the erasure-burst-correction capability of any (n,k) linear block code, binary or nonbinary. Therefore, using the expanded parity-check matrix $\mathbf{H}_n$ and the simple iterative decoding algorithm presented in Section 13.4, any (n,k) cyclic code is optimal in terms of correcting a single erasure-burst over a span of n code symbols. RS codes are effective not only at correcting random erasures but also at correcting bursts of erasures.

See also [15–23].

Problems

13.1 Compute the error performance of the (8176,7156) QC-EG-LDPC code given in Example 10.10 over the binary random erasure channel with the iterative decoding given in Section 13.1. How far from the Shannon limit does the code perform at a UEBR of 10^{-6}?

13.2 Consider the three-dimensional Euclidean geometry EG(3,3) over GF(3). From the lines in EG(3,3) not passing through the origin, four 26×26 circulants over GF(2) can be constructed. Each of these circulants has both column weight and row weight 3. Determine the length of the zero-covering-span of each of these four circulants. Do all these four circulants have the same length of zero-covering-span?

13.3 Using one of the four 26×26 circulants, denoted $\mathbf{G}$, constructed in Problem 13.2, form a 26×52 matrix $\mathbf{Z}(26, 52) = [\mathbf{G}\mathbf{G}]$ over GF(2). Construct a 127×127 array $\mathbf{H}_{qc,\text{disp}}^{(6)}$ of 127×127 circulant permutation matrices based on the prime field GF(127) and the additive 127-fold matrix dispersion technique presented in Section 11.6. Take a 26×52 subarray $\mathbf{H}_{qc,\text{disp}}^{(6)}(26, 52)$ from the array $\mathbf{H}_{qc,\text{disp}}^{(6)}$. Masking $\mathbf{H}_{qc,\text{disp}}^{(6)}(26, 52)$ with $\mathbf{Z}(26, 52)$ results in a 26×52 masked array $\mathbf{M}^{(6)}(26, 52)$ of

circulant permutation and zero matrices. $\mathbf{M}^{(6)}(26, 52)$ is a 3302×6604 matrix over GF(2) with column and row weights 3 and 6, respectively.

(a) Determine the QC-LDPC code given by the null space of $\mathbf{M}^{(6)}(26, 52)$.
(b) Compute the bit- and block-error performance of the code given in (a) over the binary-input AWGN channel using the SPA with 50 iterations.
(c) Compute the error performance of the code given in (a) over the binary random erasure channel.
(d) Determine the length of the zero-covering-span of $\mathbf{M}^{(6)}(26, 52)$ and the erasure-burst-correction capability.

13.4 Consider the 127×127 array $\mathbf{H}^{(6)}_{qc,\text{disp}}$ of 127×127 CPMs constructed using GF(127) given in Problem 13.3. Set $t = 5$, $k = 4$, $l = 3$, and $s = 2$. Take a 5×20 subarray $\mathbf{H}^{(6)}_{qc,\text{disp}}(5, 20)$ from $\mathbf{H}^{(6)}_{qc,\text{disp}}$. Using the array dispersion technique given in Section 13.7, construct a 2-masked and 3-fold dispersion $\mathbf{H}^{(6)}_{\text{3-f,disp,2}}(15, 60)$ of $\mathbf{H}^{(6)}_{qc,\text{disp}}(5, 20)$, which is a 15×60 array of a 127×127 circulant permutation and zero matrix.

(a) Determine the QC-LDPC code given by the null space of the array $\mathbf{H}^{(6)}_{\text{3-f,disp,2}}(15, 60)$. What is its erasure-burst-correction capability and efficiency?
(b) Compute the bit- and word-error performance of the code given in (a) over the binary-input AWGN channel using the SPA with 50 iterations.
(c) Compute the error performance of the code given in (a) over the binary random erasure channel.

13.5 Prove that the maximum zero-span of the parity-vector $\mathbf{h}$ of an (n,k) cyclic code over GF(q) is unique.

References

[1] C. Di, D. Proietti. I. E. Teletar, T. J. Richardson, and R. L. Urbanke, "Finite length analysis of low-density parity-check codes on the binary erasure channels," *IEEE Trans. Information Theory*, vol. 48, no. 6, pp. 1576–1579, Jun 2002.

[2] A. Orlitsky, R. Urbanke, K. Viswanathan, and J. Zhang, "Stopping sets and the girth of Tanner graphs," *Proc. IEEE Int. Symp. Information Theory*, Lausanne, June 2002, p. 2.

[3] M. G. Luby, M. Mitzenmacker, M. A. Sokrollahi, and D. A. Spilman, "Efficient erasure correcting codes," *IEEE Trans. Information Theory*, vol. 47, no. 2, pp. 569–584, February 2001.

[4] A. Orlitsky, K. Viswanathan, and J. Zhang, "Stopping sets distribution of LDPC code ensemble," *IEEE Trans. Information Theory*, vol. 51, no. 3, pp. 929–953, March 2005.

[5] T. Tian, C. Jones, J. D. Villasenor, and R. D. Wesel, "Construction of irregular LDPC code with low error floors," *Proc. IEEE Int. Conf. Communications*, Anchorage, AK, May 2003, pp. 3125–3129.

[6] L. Lan, Y. Y. Tai, S. Lin, B. Memari, and B. Honary, "New construction of quasi-cyclic LDPC codes based on special classes of BIBD's for the AWGN and binary erasure channels," *IEEE Trans. Communications*, vol. 56, no. 1, pp. 39–48, January 2008.

[7] L. Lan, L.-Q. Zeng, Y. Y. Tai, L. Chen, S. Lin, and K. Abdel-Ghaffar, "Construction of quasi-cyclic LDPC codes for AWGN and binary erasure channels: a finite field approach," *IEEE Trans. Information Theory*, vol. 53, no.7, pp. 2429–2458, July 2007.

[8] S. Song, S. Lin, and K. Addel-Ghaffar, "Burst-correction decoding of cyclic LDPC codes," *Proc. IEEE Int. Symp. Information Theory*, Seattle, WA, July 9–14, 2006, pp. 1718–1722.

[9] Y. Y. Tai, L. Lan, L.-Q. Zeng, S. Lin, and K. Abdel-Ghaffar, "Algebraic construction of quasi-cyclic LDPC codes for the AWGN and erasure channels," *IEEE Trans. Communications*, vol. 54, no. 10, pp. 1765–1774, October 2006.

[10] J. Ha and S. W. McLaughlin, "Low-density parity-check codes over Gaussian channels with erasures," *IEEE Trans. Information Theory*, vol. 49, no. 7, pp. 1801–1809, July 2003.

[11] F. Peng, M. Yang, and W. E. Ryan, "Design and analysis of eIRA codes on correlated fading channels," *Proc. IEEE Global Telecommun. Conf.*, Dallas, TX, November–December 2004, pp. 503–508.

[12] M. Yang and W. E. Ryan, "Design of LDPC codes for two-state fading channel models," *Proc. 5th Int. Symp. Wireless Personal Multimedia Communications*, Honolulu, HI, October 2002, pp. 503–508.

[13] M. Yang and W. E. Ryan, "Performance of efficiently encodable low-density parity-check codes in noise bursts on the EPR4 channel," *IEEE Trans. Magnetics*, vol. 40, no. 2, pp. 507–512, March 2004.

[14] S. Song, S. Lin, K. Abdel-Ghaffar, and W. Fong, "Erasure-burst and error-burst decoding of linear codes," *Proc. IEEE Information Theory Workshop*, Lake Tahoe, CA, September 2–6, 2007, pp. 132–137.

[15] D. Burshtein and B. Miller, "An efficient maximum-likelihood decoding of LDPC codes over the binary erasure channel," *IEEE Trans. Information Theory*, vol. 50, no. 11, pp. 2837–2844, November 2004.

[16] S. Lin and D. J. Costello, Jr., *Error Control Coding: Fundamentals and Applications*, Upper Saddle River, NJ, Prentice-Hall, 2004.

[17] P. Oswald and A. Shokrollahi, "Capacity-achieving sequences for erasure channels," *IEEE Trans. Information Theory*, vol. 48, no. 12, pp. 3017–3028, December 2002.

[18] H. D. Pfister, I. Sason, and R. L. Urbanke, "Capacity-approaching ensembles for the binary erasure channel with bounded complexity," *IEEE Trans. Information Theory*, vol. 51, no. 7, pp. 2352–2379, July 2005.

[19] H. Pishro-Nik and F. Fekri, "On decoding of low-density parity-check codes over the binary erasure channel," *IEEE Trans. Information Theory*, vol. 50, vol 3, pp. 439–454, March 2004.

[20] T. J. Richardson and R. L. Urbanke, *Modern Coding Theory*, Cambridge, Cambridge University Press, 2008.

[21] M. Rashidpour, A. Shokrollahi, and S. H. Jamali, "Optimal regular LDPC codes for the binary erasure channel," *IEEE Communications Lett.*, vol. 9, no. 6, pp. 546–548, June 2005.

[22] H. Saeedi and A. H. Banihashimi, "Deterministic design of low-density parity-check codes for binary erasure channel," *Proc. IEEE Globecom*, San Francisco, CA, November 2006, pp. 1566–1570.

[23] B. N. Vellambi, and F. Fekri, "Results on the improved decoding algorithm for low-density parity-check codes over the binary erasure channels," *IEEE Trans. Information Theory*, vol. 53, no. 4, pp. 1510–1520, April 2007.

14 Nonbinary LDPC Codes

Although a great deal of research effort has been expended on the design, construction, encoding, decoding, performance analysis, and applications of binary LDPC codes in communication and storage systems, very little has been done on nonbinary LDPC codes in these respects. The first study of nonbinary LDPC codes was conducted by Davey and MacKay in 1998 [1]. In their paper, they generalized the SPA for decoding binary LDPC codes to decode q-ary LDPC codes, called *QSPA*. Later, in 2000, MacKay and Davey introduced a *fast-Fourier-transform* (FFT)-based QSPA to reduce the decoding computational complexity of QSPA [2]. This decoding algorithm is referred to as FFT-QSPA. MacKay and Davey's work on FFT-QSPA was further improved by Barnault and Declercq in 2003 [3] and Declercq and Fossorier in 2007 [4]. Significant works on the design, construction and analysis of nonbinary LDPC codes didn't appear until the middle of 2000. Results in these works are very encouraging. They show that nonbinary LDPC codes have a great potential to replace widely used RS codes in some applications in communication and storage systems. This chapter is devoted to nonbinary LDPC codes.

Just like binary LDPC codes, nonbinary LDPC codes can be classified into two major categories: (1) random-like nonbinary codes constructed by computer under certain design criteria or rules; and (2) structured nonbinary codes constructed on the basis of algebraic or combinatorial tools, such as finite fields and finite geometries. In this chapter, we focus on algebraic constructions of nonbinary LDPC codes. The design and construction of random-like nonbinary LDPC codes can be found in [1,5,7,8,11].

14.1 Definitions

Fundamental concepts, structural properties, and methods of construction, encoding, and decoding developed for binary LDPC codes in the previous chapters can be generalized to LDPC codes with symbols from nonbinary fields.

Let $\mathrm{GF}(q)$ be a Galois field with q elements, where q is a power of a prime. A q-ary *regular* LDPC code $\mathcal{C}$ of length n is given by the null space over $\mathrm{GF}(q)$ of a sparse parity-check matrix $\mathbf{H}$ over $\mathrm{GF}(q)$ that has the following structural

properties: (1) each row has weight r; and (2) each column has weight g, where r and g are small compared with the length of the code. Such a q-ary LDPC code is said to be (g,r)-regular. If the columns and/or rows of the parity-check matrix $\mathbf{H}$ have *varying* (*multiple*) weights, then the null space over GF(q) of $\mathbf{H}$ gives a q-ary *irregular* LDPC code. If $\mathbf{H}$ is an array of sparse circulants of the same size over GF(q), then the null space over GF(q) of $\mathbf{H}$ gives a q-ary quasi-cyclic (QC) LDPC code. If $\mathbf{H}$ consists of a single sparse circulant or a column of sparse circulants, then the null space over GF(q) of $\mathbf{H}$ gives a q-ary cyclic LDPC code. Encoding of q-ary cyclic and QC-LDPC codes can be implemented with shift-registers just like encoding of binary cyclic and QC codes using the circuits as presented in Section 3.2 and 3.6 with some modifications.

The Tanner graph of a q-ary LDPC code $\mathcal{C}$ given by the null space of a sparse $m \times n$ parity-check matrix $\mathbf{H} = [h_{i,j}]$ over GF(q) is constructed in the same way as that for a binary LDPC code. The graph has n variable nodes that correspond to the n code symbols of a code word in $\mathcal{C}$ and m check nodes that correspond to m check-sum constraints on the code symbols. The jth variable node v_j is connected to the ith check node c_i with an edge if and only if the jth code symbol v_j is contained in the ith check-sum c_i, i.e., if and only the entry $h_{i,j}$ at the intersection of the ith row and jth column of $\mathbf{H}$ is a nonzero element of GF(q). To ensure that the Tanner graph of the q-ary LDPC code $\mathcal{C}$ is free of cycles of length 4 (or has a girth of at least 6), we further impose the following constraint on the rows and columns of $\mathbf{H}$: no two rows (or two columns) of $\mathbf{H}$ have more than one position where they both have nonzero components. This constraint is referred to as the row–column (RC) constraint. This RC-constraint was imposed on the parity-check matrices of all the binary LDPC codes presented in previous chapters, regardless of their methods of construction. For a (g,r)-regular q-ary LDPC code $\mathcal{C}$, the RC-constraint on $\mathbf{H}$ also ensures that the minimum distance of the q-ary LDPC code $\mathcal{C}$ is at least $g + 1$, where g is the column weight of $\mathbf{H}$.

14.2 Decoding of Nonbinary LDPC Codes

The SPA for decoding binary LDPC codes in the probability domain (see Problem 5.10) can be generalized to decode q-ary LDPC codes. The first such generalization was presented by Davey and MacKay [1]. We call this SPA for decoding q-ary LDPC codes a q-ary SPA (QSPA).

14.2.1 The QSPA (Davey and MacKay [1])

Consider a q-ary LDPC code $\mathcal{C}$ given by the null space over GF(q) of the following $m \times n$ parity-check matrix over GF(q):

$$\mathbf{H} = \begin{bmatrix} \mathbf{h}_0 \\ \mathbf{h}_1 \\ \vdots \\ \mathbf{h}_{m-1} \end{bmatrix}$$
$$= \begin{bmatrix} h_{0,0} & h_{0,1} & \cdots & h_{0,n-1} \\ h_{1,0} & h_{1,1} & \cdots & h_{1,n-1} \\ \vdots & \vdots & \ddots & \vdots \\ h_{m-1,0} & h_{m-1,1} & \cdots & h_{m-1,n-1} \end{bmatrix}, \tag{14.1}$$

where, for $0 \le i < m$, the ith row is an n-tuple over GF(q), $\mathbf{h}_i = (h_{i,0}, h_{i,1}, \ldots, h_{i,n-1})$, and, for $0 \le j < n$, the jth column is an m-tuple over GF(q), $\mathbf{g}_j = (h_{0,j}, h_{1,j}, \ldots, h_{m-1,j})^{\mathrm{T}}$. Define the following two sets of indices for the nonzero components of $\mathbf{h}_i$ and $\mathbf{g}_j$: (1) for $0 \le j < n$,

$$M_j = \{i\colon 0 \le i < m, h_{i,j} \ne 0\}; \tag{14.2}$$

and (2) for $0 \le i < m$,

$$N_i = \{j\colon 0 \le j < n, h_{i,j} \ne 0\}. \tag{14.3}$$

From the definitions of M_j and N_j, it is clear that

$$\sum_{j=0}^{n-1} |M(j)| = \sum_{i=0}^{m-1} |N(i)|,$$

where $|M_j|$ and $|N_i|$ are the cardinalities of the index sets M_j and N_j, respectively. Each of the above two sums simply gives the total number of nonzero entries of the parity-check matrix $\mathbf{H}$ given by (14.1), denoted by D_H.

Let $\mathbf{v} = (v_0, v_1, \ldots, v_{n-1})$ be a codeword in $\mathcal{C}$. Then $\mathbf{v} \cdot \mathbf{H}^{\mathrm{T}} = \mathbf{0}$, which gives m constraints on the n code symbols of $\mathbf{v}$:

$$c_i = v_0 h_{i,0} + v_1 h_{i,1} + \cdots + v_{n-1} h_{i,n-1} = 0 \tag{14.4}$$

for $0 \le i < m$, which are called check-sums. If $h_{i,j} \ne 0$, the code symbol v_j is contained in the check-sum c_i. In this case, we say that v_j is checked by c_i (or by the ith row $\mathbf{h}_i$ of $\mathbf{H}$). It follows from the definition of the index set N_i that only the code symbols of $\mathbf{v}$ with indices in N_i are checked by the ith check-sum c_i. The rows of $\mathbf{H}$ (or the check-sums given by (14.4)) that check on the code symbol v_j are said to be *orthogonal* on v_j. It follows from the definition of M_j that only the rows of $\mathbf{H}$ with indices in M_j are orthogonal on v_j.

Suppose a codeword of $\mathcal{C}$, $\mathbf{v} = (v_0, v_1, \ldots, v_{n-1})$, is transmitted. Let $\mathbf{y} = (y_0, y_1, \ldots, y_{n-1})$ and $\mathbf{z} = (z_0, z_1, \ldots, z_{n-1})$ be the soft- and hard-decision received sequences, respectively, where, for $0 \le j < n$, the jth received symbol z_j is an element in GF(q). The syndrome of $\mathbf{z}$ is

$$\mathbf{s} = (s_0, s_1, \ldots, s_{m-1}) = \mathbf{z} \cdot \mathbf{H}^{\mathrm{T}},$$

where, for $0 \le i < m$,

$$s_i = z_0 h_{i,0} + z_1 h_{i,1} + \cdots + z_j h_{i,j} + \cdots + z_{n-1} h_{i,n-1},$$

which is the check-sum computed from the received sequence $\mathbf{z}$ and the ith row $\mathbf{h}_i$ of $\mathbf{H}$. Note that s_i corresponds to the check-sum c_i computed from $\mathbf{v}$ and the ith row $\mathbf{h}_i$ of $\mathbf{H}$. The hard-decision received sequence $\mathbf{z}$ is a codeword in $\mathcal{C}$ if and only if the check-sums $s_0, s_1, \ldots, s_{m-1}$ computed from $\mathbf{z}$ are all equal to zero, i.e., $s_i = c_i = 0$ for $0 \leq i < m$. We say that the check-sum s_i is satisfied if $s_i = c_i = 0$. The check-sums computed from $\mathbf{z}$ that contain z_j are also said to be orthogonal on z_j (or v_j). If $s_i \neq 0$, then some of the received symbols of $\mathbf{z}$ checked by the sum s_i are not equal to the transmitted code symbols, i.e., they are erroneous.

It follows from the construction of the Tanner graph of a q-ary LDPC code that if $h_{i,j}$ is a nonzero element in GF(q), then there is an edge connecting the jth variable node v_j to the ith check node c_i. This edge provides a communication link for message passing between the variable node v_j and check node c_i in iterative decoding of the received sequence $\mathbf{z}$ (or $\mathbf{y}$).

During the iterative decoding process with the QSPA, the reliability measures of the received symbols of the soft-decision received sequence $\mathbf{y}$ are updated at each iteration step. From these updated reliability measures, a new hard-decision received sequence is computed. Let $\mathbf{z}^{(k)} = (z_0^{(k)}, z_1^{(k)}, \ldots, z_{n-1}^{(k)})$ be the hard-decision received sequence computed at the end of the $(k-1)$th iteration of decoding for $k \geq 1$. If the syndrome

$$\mathbf{s}^{(k)} = (s_0^{(k)}, s_1^{(k)}, \ldots, s_{m-1}^{(k)}) = \mathbf{z}^{(k)} \cdot \mathbf{H}^{\mathrm{T}},$$

with

$$s_i^{(k)} = z_0^{(k)} h_{i,0} + z_1^{(k)} h_{i,1} + \cdots + z_{n-1}^{(k)} h_{i,n-1},$$

computed from $\mathbf{z}^{(k)}$ is a zero m-tuple, then the decoding process stops and $\mathbf{z}^{(k)}$ is taken as the decoded codeword. Otherwise, the decoding process continues until a preset maximum number $I_{\max}$ of iterations is reached. In this case, a decoding failure is declared. For $k = 0$, $\mathbf{z}^{(0)} = \mathbf{z}$.

The messages passed between the variable node v_j and the check node c_i over the edge (c_i, v_j) corresponding to the nonzero entry $h_{i,j}$ of $\mathbf{H}$ during the kth iteration of decoding using the QSPA are two types of probabilities, $q_{i,j}^{a,(k)}$ and $\sigma_{i,j}^{a,(k)}$. The probability $q_{i,j}^{a,(k)}$ is a message sent from the variable node v_j to the check node c_i. This message is the conditional probability that the jth symbol $z_j^{(k)}$ of $\mathbf{z}^{(k)}$ is equal to the symbol a of GF(q), given the information obtained via other check nodes with indices in $M_j \setminus i$. The probability $\sigma_{i,j}^{a,(k)}$ is a message sent from the ith check node c_i to the variable node v_j. This message is the probability that the check-sum $s_i^{(k)}$ computed from $\mathbf{z}^{(k)}$ and $\mathbf{h}_i$ is satisfied (i.e., $s_i^{(k)} = 0$) given that the jth symbol $z_j^{(k)}$ of $\mathbf{z}^{(k)}$ is set to symbol a of GF(q) and other symbols of $\mathbf{z}^{(k)}$ contained in the check-sum $s_i^{(k)}$ have a separable probability distribution, $\{q_{i,t}^{b_t,(k)} \colon t \in N_i \backslash j, b_t \in \mathrm{GF}(q)\}$.

Let $a_1, a_2, \ldots, a_q$ denote the q elements of GF(q). Let $P_j^{a_1}, P_j^{a_2}, \ldots, P_j^{a_q}$ be the prior probabilities of the jth received symbol z_j of $\mathbf{z}$ equal to $a_1, a_2, \ldots, a_q$,

respectively, for $0 \le j < n$. It is clear that $P_j^{a_1} + P_j^{a_2} + \cdots + P_j^{a_q} = 1$. The probability $\sigma_{i,j}^{a,(k)}$ is given by

$$\sigma_{i,j}^{a,(k)} = \sum_{\mathbf{z}^{(k)}, z_j^{(k)} = a} P(s_i^{(k)} = 0 | \mathbf{z}^{(k)}, z_j^{(k)} = a) \cdot \prod_{t \in N_i \setminus j} q_{i,t}^{b_t,(k)}, \tag{14.5}$$

where $P(s_i^{(k)} = 0 | \mathbf{z}^{(k)}, z_j^{(k)} = a) = 1$ when $z_j^{(k)}$ is fixed at the symbol a of GF(q) and $\mathbf{z}^{(k)}$ satisfies the ith check-sum (i.e., $s_i^{(k)} = 0$); otherwise $P(s_i^{(k)} = 0 | \mathbf{z}^{(k)}, z_j^{(k)} = a) = 0$. The computed values of $\sigma_{i,j}^{a,(k)}$ for $a \in \mathrm{GF}(q)$, $i \in M_j$, and $j \in N_i$ are then used to update the probability $q_{i,j}^{a,(k)}$ for the next decoding iteration as follows:

$$q_{i,j}^{a,(k+1)} = f_{i,j}^{(k+1)} P_j^a \prod_{t \in M_j \setminus i} \sigma_{t,j}^{a,(k)}, \tag{14.6}$$

where $f_{i,j}^{(k+1)}$ is chosen such that

$$q_{i,j}^{a_1,(k+1)} + q_{i,j}^{a_2,(k+1)} + \cdots + q_{i,j}^{a_q,(k+1)} = 1. \tag{14.7}$$

The QSPA, just like the binary SPA, consists of a sequence of decoding iterations. During each decoding iteration, the two sets of probability messages, $\{q_{i,j}^{a,(k)} \colon i \in M_j, j \in N_i, a \in \mathrm{GF}(q)\}$ and $\{\sigma_{i,j}^{a,(k)} \colon i \in M_j, j \in N_i, a \in \mathrm{GF}(q)\}$, update each other. The QSPA can be formulated as follows:

Initialization. Set $k = 0$ and the maximum number of iterations to $I_{\max}$. For every pair (i, j) of integers such that $h_{i,j} \neq 0$ with $0 \le i < m$ and $0 \le j < n$, set $q_{i,j}^{a_1,(0)} = P_j^{a_1}$ and $q_{i,j}^{a_2,(0)} = P_j^{a_2}$, $\ldots$, $q_{i,j}^{a_q,(0)} = P_j^{a_q}$.

1. (**Updating** $\sigma_{i,j}^{a,(k)}$) For $a \in \mathrm{GF}(q)$ and every pair (i, j) of integers such that $h_{i,j} \neq 0$, with $0 \le i < m$ and $0 \le j < n$, compute the probability $\sigma_{i,j}^{a,(k)}$ on the basis of (14.5). Go to Step 2.
2. (**Updating** $q_{i,j}^{a,(k)}$) For $a \in \mathrm{GF}(q)$ and every pair (i, j) of integers such that $h_{i,j} \neq 0$ with $0 \le i < m$ and $0 \le j < n$, compute $q_{i,j}^{a,(k+1)}$ using (14.6). Form $\mathbf{z}^{(k+1)} = (z_0^{(k+1)}, z_1^{(k+1)}, \ldots, z_{n-1}^{(k+1)})$, where

$$z_j^{(k+1)} = \arg\max_a P_j^a \prod_{i \in M_j} \sigma_{i,j}^{a,(k+1)}. \tag{14.8}$$

   Compute $\mathbf{s}^{(k+1)} = \mathbf{z}^{(k+1)} \mathbf{H}^{\mathrm{T}}$. If $\mathbf{s}^{(k+1)} = \mathbf{0}$ or the maximum number $I_{\max}$ of iterations is reached, go to Step 3. Otherwise, set $k := k + 1$ and go to Step 1.
3. (**Termination**) Stop the decoding process and output $\mathbf{z}^{(k+1)}$ as the decoded code word if $\mathbf{s}^{(k+1)} = 0$. If $\mathbf{s}^{(k+1)} \neq 0$, the presence of errors has been detected, in which case a decoding failure is declared.

For practical applications, q is chosen as a power of 2, say $q = 2^s$. In this case, every element a_t of GF(2^s) can be represented as an s-tuple $(a_{t,0}, a_{t,1}, \ldots, a_{t,s-1})$ over GF(2). In binary transmission, each code symbol v_j of a transmitted codeword $\mathbf{v} = (v_0, v_1, \ldots, v_{n-1})$ is expanded into an s-tuple, $(v_{j,0}, v_{j,1}, \ldots, v_{j,s-1})$, over GF(2). Then the binary representation of $\mathbf{v}$ is

$$\mathbf{v}^* = ((v_{0,0}, \ldots, v_{0,s-1}), (v_{1,0}, \ldots, v_{1,s-1}), \ldots, (v_{n-1,0}, \ldots, v_{n-1,s-1})).$$

The corresponding soft-decision and hard-decision received sequences are

$$\mathbf{y}^* = ((y_{0,0}, \ldots, y_{0,s-1}), (y_{1,0}, \ldots, y_{1,s-1}), \ldots, (y_{n-1,0}, \ldots, y_{n-1,s-1}))$$

and

$$\mathbf{z}^* = ((z_{0,0}, \ldots, z_{0,s-1}), (z_{1,0}, \ldots, z_{1,s-1}), \ldots, (z_{n-1,1}, \ldots, z_{n-1,s-1})),$$

respectively. Suppose the binary expansion $\mathbf{v}^*$ of codeword $\mathbf{v}$ is transmitted over the AWGN channel with two-sided power-spectral density $N_0/2$ using BPSK modulation with unit signal energy. Then the transmitted signal sequence can be represented by a bipolar sequence with $+1$ representing code symbol 1 and -1 representing code symbol 0. For $0 \le j < n$ and $0 \le l < s$, the probability of the received bit $z_{j,l}$ being "1" given the channel output $y_{j,l}$ is

$$p^1_{z_{j,l}} = \frac{1}{1 + \exp(4y_{j,l}/N_0)}. \tag{14.9}$$

Then the probability of the received bit $z_{j,l}$ being 0 is $p^0_{z_{j,l}} = 1 - p^1_{z_{j,l}}$. For $1 \le j < n$, the probability $P_j^{a_t}$ of the jth received symbol z_j of $\mathbf{z}$ being the element a_t of GF(q) with binary expansion $(a_{t,0}, a_{t,1}, \ldots, a_{t,s-1})$ is

$$P_j^{a_t} = \prod_{l=0}^{s-1} p^{a_{t,l}}_{z_{j,l}}. \tag{14.10}$$

The probabilities $P_j^{a_t}$ with $a_t \in$ GF(q) are then used as the prior probabilities at the initialization step of the QSPA described above.

For each nonzero entry $h_{i,j}$ in $\mathbf{H}$, the number of computations required to compute the probability messages passing between check node c_i and variable node v_j in each decoding iteration is on the order of q^2. Consequently, the number of computations required per iteration of QSPA is on the order of

$$\sum_{i=0}^{m-1} |N_i| q^2 = D_H q^2.$$

If $\mathbf{H}$ has a low density of nonzero entries, then the computational complexity of the QSPA required per iteration is dominated by the size q of the code-symbol alphabet GF(q).

For large q, the computational complexity of QSPA may become prohibitively large. To reduce the computational complexity, MacKay and Davey [2] proposed

a fast-Fourier-transform (FFT)-based method to compute the probability $\sigma_{i,j}^{a,(k)}$ given by (14.5). This FFT-based QSPA is referred to as the *FFT-QSPA*.

14.2.2 The FFT-QSPA

The FFT-QSPA [2] presented in this section reduces the complexity of computing the probability message $\sigma_{i,j}^{a,(k)}$ given by (14.5) for q a power of 2, say $q = 2^s$.

Consider the hard-decision decoded sequence $\mathbf{z}^{(k)} = (z_0^{(k)}, z_1^{(k)}, \ldots, z_{n-1}^{(k)})$ over GF(2^s) that is used to start the kth iteration of decoding using QSPA. This sequence satisfies the ith parity-check-sum $s_i^{(k)}$ if and only if

$$\sum_{t \in N_i} h_{i,t} z_t^{(k)} = 0. \tag{14.11}$$

Let $\tilde{z}_{i,t}^{(k)} = h_{i,t} z_t^{(k)}$. It is clear that $\tilde{z}_{i,t}^{(k)}$ is also an element of GF(2^s). Then, from (14.11), we have

$$\tilde{z}_{i,j}^{(k)} = \sum_{t \in N_i \setminus j} \tilde{z}_{i,t}^{(k)} \quad . \tag{14.12}$$

We associate each nonzero entry $h_{i,j}$ of $\mathbf{H}$ with two new quantities, $\tilde{q}_{i,j}^{\tilde{a},(k)}$ and $\tilde{\sigma}_{i,j}^{\tilde{a},(k)}$, both of which are probabilities of the symbol $\tilde{z}_{i,j}^{(k)}$ being the element $\tilde{a} = h_{i,j} a$ in GF(2^s). These two probabilities are defined as follows: for $\tilde{a} = h_{i,j} a$,

$$\tilde{q}_{i,j}^{\tilde{a},(k)} \triangleq q_{i,j}^{a,(k)} \tag{14.13}$$

and

$$\tilde{\sigma}_{i,j}^{\tilde{a},(k)} \triangleq \sigma_{i,j}^{a,(k)}. \tag{14.14}$$

Let $a_1, a_2, \ldots, a_{2^s}$ denote the 2^s elements of GF(2^s). Define two probability mass 2^s-tuples as follows:

$$\tilde{\mathbf{q}}_{i,j}^{(k)} \triangleq (\tilde{q}_{i,j}^{\tilde{a}_1,(k)}, \tilde{q}_{i,j}^{\tilde{a}_2,(k)}, \ldots, \tilde{q}_{i,j}^{\tilde{a}_{2^s},(k)}) \tag{14.15}$$

and

$$\tilde{\boldsymbol{\sigma}}_{i,j}^{(k)} \triangleq (\tilde{\sigma}_{i,j}^{\tilde{a}_1,(k)}, \tilde{\sigma}_{i,j}^{\tilde{a}_2,(k)}, \ldots, \tilde{\sigma}_{i,j}^{\tilde{a}_{2^s},(k)}), \tag{14.16}$$

where $\tilde{q}_{i,j}^{\tilde{a}_t,(k)}$ and $\tilde{\sigma}_{i,j}^{\tilde{a}_t,(k)}$ are probabilities indicating the symbol $\tilde{z}_{i,j}^{(k)} = h_{i,j} z_j^{(k)}$ being the symbol $\tilde{a}_t = h_{i,j} a_t$ of GF(2^s).

Consider two probability mass 2^s-tuples, $\mathbf{u}$ and $\mathbf{v}$. For $1 \le t \le 2^s$, let u^{a_t} and v^{a_t} be the probability components of $\mathbf{u}$ and $\mathbf{v}$ associated with the symbol a_t of GF(2^s), respectively. The *convolution* in GF(2^s) of $\mathbf{u}$ and $\mathbf{v}$ is a probability mass 2^s-tuple $\mathbf{w} = \mathbf{u} \otimes \mathbf{v}$, where $\otimes$ denotes the convolution product in GF(2^s), and

the probability component w^{a_t} of $\mathbf{w}$ associated with the symbol a_t of GF(2^s) is given by

$$w^{a_t} = \sum_{\substack{a_f, a_l \in \mathrm{GF}(2^s) \\ a_t = a_f + a_l}} u^{a_f} v^{a_l}, \tag{14.17}$$

where the addition $a_f + a_l$ is carried out in GF(2^s). It follows from (14.5) and (14.12)–(14.17) that the probability mass 2^s-tuple $\tilde{\boldsymbol{\sigma}}_{i,j}^{(k)}$ can be updated as follows:

$$\tilde{\boldsymbol{\sigma}}_{i,j}^{(k)} = \bigotimes_{t \in N_i \backslash j} \tilde{\mathbf{q}}_{i,t}^{(k)}. \tag{14.18}$$

The probability mass vector $\tilde{\boldsymbol{\sigma}}_{i,j}^{(k)}$ can be computed efficiently using the FFT as follows:

$$\tilde{\boldsymbol{\sigma}}_{i,j}^{(k)} = \mathrm{FFT}^{-1} \prod_{t \in N_i \backslash j} \mathrm{FFT}(\tilde{\mathbf{q}}_{i,t}^{(k)}), \tag{14.19}$$

where FFT^{-1} denotes the inverse of the FFT and $\prod$ is the term-by-term product.

In the following, we present an efficient way to compute $\tilde{\boldsymbol{\sigma}}_{i,j}^{(k)}$. The radix-2 FFT of a probability mass 2^s-tuple $\mathbf{u}$ is given by

$$\mathbf{v} = \mathrm{FFT}(\mathbf{u}) = \mathbf{u} \times_0 \mathbf{F} \times_1 \mathbf{F} \times \cdots \times_{s-1} \mathbf{F}, \tag{14.20}$$

where $\mathbf{F}$ is a 2×2 matrix over GF(2) given by

$$\mathbf{F} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}. \tag{14.21}$$

Represent the symbol a_t of GF(2^s) by an s-tuple over GF(2), $(a_{t,0}, a_{t,1}, \ldots, a_{t,s-1})$. Then $\mathbf{w} = \mathbf{u} \bigotimes_l \mathbf{F}$ is computed as follows. For $0 \le l < s$,

$$\begin{aligned} w^{(a_{t,0}, \ldots, a_{t,l-1}, 0, a_{t,l+1}, \ldots, a_{t,s-1})} &= u^{(a_{t,0}, \ldots, a_{t,l-1}, 0, a_{t,l+1}, \ldots, a_{t,s-1})} \\ &\quad + u^{(a_{t,0}, \ldots, a_{t,l-1}, 1, a_{t,l+1}, \ldots, a_{t,s-1})}, \end{aligned} \tag{14.22}$$

$$\begin{aligned} w^{(a_{t,0}, \ldots, a_{t,l-1}, 1, a_{t,l+1}, \ldots, a_{t,s-1})} &= u^{(a_{t,0}, \ldots, a_{t,l-1}, 0, a_{t,l+1}, \ldots, a_{t,s-1})} \\ &\quad - u^{(a_{t,0}, \ldots, a_{t,l-1}, 1, a_{t,l+1}, \ldots, a_{t,s-1})}. \end{aligned} \tag{14.23}$$

In each layer of the FFT given by (14.20), we compute the sum and difference of the probabilities of two field elements differing from each other by only one bit position. It is easy to check that the inverse transform matrix $\mathbf{F}^{-1}$ of $\mathbf{F}$ is

$$\mathbf{F}^{-1} = \frac{1}{2} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}. \tag{14.24}$$

Then, it follows from (14.20) that

$$\mathbf{u} = \mathrm{FFT}^{-1}(\mathbf{v}) = \mathbf{v} \times_0 \mathbf{F}^{-1} \times_1 \mathbf{F}^{-1} \times \cdots \times_{s-1} \mathbf{F}^{-1}, \tag{14.25}$$

Using (14.20)–(14.25) with $\mathbf{u} = \tilde{\mathbf{q}}_{i,t}^{(k)}$, we can efficiently compute $\mathrm{FFT}(\tilde{\mathbf{q}}_{i,t}^{(k)})$. From (14.19), we can compute

$$\tilde{\boldsymbol{\sigma}}_{i,j}^{(k)} = (\tilde{\sigma}_{i,j}^{\tilde{a}_1,(k)}, \tilde{\sigma}_{i,j}^{\tilde{a}_2,(k)}, \ldots, \tilde{\sigma}_{i,j}^{\tilde{a}_{2^s},(k)}). \tag{14.26}$$

Since $\tilde{a}_t = h_{i,j}a_t$, it follows from the definition of $\tilde{\sigma}_{i,j}^{\tilde{a}_t,(k)}$ that, for $1 \le t \le 2^s$, $\sigma_{i,j}^{a_t,(k)} = \tilde{\sigma}_{i,j}^{\tilde{a}_t,(k)}$. Summarizing the above developments, the probability messages, $\sigma_{i,j}^{a_t,(k)}$ for $1 \le t \le 2^s$, passed from check node c_i to variable node v_j can be updated in three steps:

1. Compute $\tilde{\mathbf{q}}_{i,j}^{(k)}$ from $\mathbf{q}_{i,j}^{(k)}$ according to $\tilde{q}_{i,j}^{\tilde{a}_t,(k)} = q_{i,j}^{a_t,(k)}$ if $\tilde{a}_t = h_{i,j}a_t$.
2. Compute $\tilde{\boldsymbol{\sigma}}_{i,j}^{(k)} = \mathrm{FFT}^{-1}(\prod_{t \in N_i \setminus j} \mathrm{FFT}(\tilde{\mathbf{q}}^{(k)} i, j))$.
3. Compute $\boldsymbol{\sigma}_{i,j}^{(k)}$ from $\tilde{\boldsymbol{\sigma}}_{i,j}^{(k)}$ according to $\sigma_{i,j}^{a_t,(k)} = \tilde{\sigma}_{i,j}^{\tilde{a}_t,(k)}$ if $\tilde{a}_t = h_{i,j}a_t$ for $1 \le t \le 2^s$.

Using the FFT, the number of computations required to compute the probability messages passed between a check node and an adjacent variable node is on the order of $q \log q$ with $q = 2^s$. Consequently, the number of computations required per iteration is on the order of

$$\sum_{i=0}^{m-1} |N_i| q \log q = D_H q \log q. \tag{14.27}$$

The FFT-QSPA presented above reduces the computational complexity of the QSPA presented in Section 14.2.1 by a factor of $q/\log q$. For large q, the FFT-QSPA reduces the computational complexity of the QSPA drastically. For example, let $q = 2^8$. The FFT-QSPA reduces the computational complexity of the QSPA by a factor of 32.

Note that the above FFT-QSPA devised by Davey and MacKay applies only to q that is a power of 2. This FFT-QSPA can be generalized to any q that is a power of a prime [4]. Since, for practical applications, q is commonly (if not always) chosen as a power of 2, we will not present the generalized FFT-QSPA presented in [4]. As in the binary case, reduced-complexity QSPA algorithms that trade off performance and complexity have been developed. However, these algorithms are simplified versions of the QSPA and, therefore, their computational complexity per iteration is still on the order of q^2, $O(q^2)$. Such is the case for the min-sum decoding over $\mathrm{GF}(q)$ presented in [22].

14.3 Construction of Nonbinary LDPC Codes Based on Finite Geometries

All the algebraic methods and techniques presented in Chapters 10–13 can be applied, with some modifications, for constructing LDPC codes over nonbinary fields. In this section, we start with constructions of nonbinary LDPC codes based

on the lines and flats of finite geometries. In presenting the constructions, we will follow the fundamental concepts and notations presented in Chapter 10.

14.3.1 A Class of q^m-ary Cyclic EG-LDPC Codes

Consider the m-dimensional Euclidean geometry EG(m,q) over GF(q). Recall that the Galois field GF(q^m) is a realization of the m-dimensional Euclidean geometry EG(m,q) (see Chapters 2 and 10). Let α be a primitive element of GF(q^m). Then the powers of α, $\alpha^{-\infty} = 0, \alpha^0 = 1, \alpha, \ldots, \alpha^{q^m-2}$, represent the q^m points of EG(m,q) and $\alpha^{-\infty} = 0$ represents the origin of EG(m,q). Let EG*(m,q) denote the subgeometry obtained from EG(m,q) by removing the origin and all the lines passing through the origin of EG(m,q). Then EG*(m,q) contains $n = q^m - 1$ non-origin points and

$$J_{0,\mathrm{EG}}(m,1) = (q^{m-1} - 1)(q^m - 1)/(q - 1)$$

lines (see (2.54)) not passing through the origin of EG(m,q).

Let $\mathcal{L} = \{\alpha^{j_1}, \alpha^{j_2}, \ldots, \alpha^{j_q}\}$ be a line in EG*(m,q) that consists of the points $\alpha^{j_1}, \alpha^{j_2}, \ldots, \alpha^{j_q}$, with $0 \le j_1, j_2, \ldots, j_q < q^m - 1$. Define the following $(q^m - 1)$-tuple over GF(q^m) based on the points of $\mathcal{L}$:

$$\mathbf{v}_{\mathcal{L}} = (v_0, v_1, \ldots, v_{q^m-2}), \tag{14.28}$$

whose $q^m - 1$ components, $v_0, v_1, \ldots, v_{q^m-2}$, correspond to the $q^m - 1$ non-origin points $\alpha^0, \alpha, \ldots, \alpha^{q^m-2}$ of EG*(m,q), where the j_1th, j_2th, $\ldots$, j_qth components are $v_{j_1} = \alpha^{j_1}, v_{j_2} = \alpha^{j_2}, \ldots, v_{j_q} = \alpha^{j_q}$, and other components are equal to the 0-element of GF(q^m). This $(q^m - 1)$-tuple $\mathbf{v}_{\mathcal{L}}$ over GF(q^m) is called the *type-1 q^m-ary incidence-vector* of line $\mathcal{L}$, in contrast to the type-1 binary incidence vector of a line in EG*(m,q) defined by (10.6) in Section 10.1. This vector displays the q points on line $\mathcal{L}$, with not only their locations but also their values represented by nonzero elements of GF(q^m). Consider the line $\alpha\mathcal{L} = \{\alpha^{j_1+1}, \alpha^{j_2+1}, \ldots, \alpha^{j_q+1}\}$. The type-1 q^m-ary incidence vector $\mathbf{v}_{\alpha\mathcal{L}}$ of the line $\alpha\mathcal{L}$ is the right cyclic-shift of the type-1 q^m-ary incidence vector $\mathbf{v}_{\mathcal{L}}$ of the line $\mathcal{L}$ multiplied by α.

Recall that, for any line $\mathcal{L}$ in EG*(m,q), the $q^m - 1$ lines $\mathcal{L}, \alpha\mathcal{L}, \ldots, \alpha^{q^m-2}\mathcal{L}$ form a cyclic class (see Sections 2.7.1 and 10.1) and the $J_{0,\mathrm{EG}}(m,1)$ lines in EG*(m,q) can be partitioned into $K_c = (q^{m-1} - 1)/(q - 1)$ cyclic classes, $S_1, S_2, \ldots, S_{K_c}$. For each cyclic class S_i of lines in EG*(m,q), we form a $(q^m - 1) \times (q^m - 1)$ matrix $\mathbf{H}_{q^m,c,i}$ over GF(q^m) with the type-1 q^m-ary incidence vectors $\mathbf{v}_{\mathcal{L}}, \mathbf{v}_{\alpha\mathcal{L}}, \ldots, \mathbf{v}_{\alpha^{q^m-2}\mathcal{L}}$ of the lines $\mathcal{L}, \alpha\mathcal{L}, \ldots, \alpha^{q^m-2}\mathcal{L}$ in S_i as rows arranged in cyclic order.The matrix $\mathbf{H}_{q^m,c,i}$ is a *special type of circulant* over GF(q^m) in which each row is the right cyclic-shift of the row above it multiplied by α, and the first row is the right cyclic-shift of the last row multiplied by α. Such a circulant is called an *α-multiplied circulant* over GF(q^m). Both the column weight and the row weight of $\mathbf{H}_{q^m,c,i}$ are q. This α-multiplied circulant $\mathbf{H}_{q^m,c,i}$ over GF(q^m) is simply the q^m-ary counterpart of the binary circulant $\mathbf{H}_{c,i}$ over GF(2) constructed on the basis of the type-1

binary incidence vectors of the lines in the cyclic class S_i of $\mathrm{EG}^*(m,q)$ that was defined in Section 10.1.1.

For $1 \leq k \leq K_c$, if we replace each $(q^m - 1) \times (q^m - 1)$ circulant $\mathbf{H}_{c,i}$ over GF(2) in the matrix $\mathbf{H}^{(1)}_{\mathrm{EG},c,k}$ over GF(2) given by (10.7) by the corresponding α-multiplied $(q^m - 1) \times (q^m - 1)$ circulant over $\mathrm{GF}(q^m)$, we obtain the following $k(q^m - 1) \times (q^m - 1)$ matrix over $\mathrm{GF}(q^m)$:

$$\mathbf{H}^{(1)}_{q^m,\mathrm{EG},c,k} = \begin{bmatrix} \mathbf{H}_{q^m,c,1} \\ \mathbf{H}_{q^m,c,2} \\ \vdots \\ \mathbf{H}_{q^m,c,k} \end{bmatrix}, \tag{14.29}$$

which consists of a column of k α-multiplied $(q^m - 1) \times (q^m - 1)$ circulants over $\mathrm{GF}(q^m)$. This matrix has column and row weights kq and q, respectively. From the definition of the type-1 q^m-ary incidence vector of a line in $\mathrm{EG}^*(m,q)$, it is clear that each nonzero entry of $\mathbf{H}^{(1)}_{q^m,\mathrm{EG},c,k}$ is a nonzero point of $\mathrm{EG}^*(m,q)$, represented by a nonzero element of $\mathrm{GF}(q^m)$. Since the rows of $\mathbf{H}^{(1)}_{q^m,\mathrm{EG},c,k}$ correspond to different lines in $\mathrm{EG}^*(m,q)$ and two lines have at most one point in common, no two rows (or two columns) of $\mathbf{H}^{(1)}_{q^m,\mathrm{EG},c,k}$ have more than one position where they have identical nonzero components. Consequently, $\mathbf{H}^{(1)}_{q^m,\mathrm{EG},c,k}$ satisfies the RC-constraint.

The null space of $\mathbf{H}^{(1)}_{q^m,\mathrm{EG},c,k}$ gives a q^m-ary cyclic EG-LDPC code $\mathcal{C}_{q^m,\mathrm{EG},c,k}$ over $\mathrm{GF}(q^m)$ of length $q^m - 1$ with minimum distance at least $kq + 1$, whose Tanner graph has a girth of at least 6. This q^m-ary cyclic EG-LDPC code is the q^m-ary counterpart of the binary cyclic EG-LDPC code $\mathcal{C}_{\mathrm{EG},c,k}$ given by the null space of the binary matrix $\mathbf{H}^{(1)}_{\mathrm{EG},c,k}$ given by (10.7). The generator polynomial $\mathbf{g}_{q^m}(X)$ of $\mathcal{C}_{q^m,\mathrm{EG},c,k}$ can be determined in exactly the same way as that for its binary counterpart $\mathcal{C}_{\mathrm{EG},c,k}$ (see (10.8)). The most interesting case is that $m = 2$. In this case, $\mathbf{H}^{(1)}_{q^2,\mathrm{EG},c,1}$ consists of a single α-multiplied $(q^2 - 1) \times (q^2 - 1)$ circulant over $\mathrm{GF}(q^2)$ that is constructed from the $q^2 - 1$ lines in the subgeometry $\mathrm{EG}^*(2,q)$ of the two-dimensional Euclidean geometry $\mathrm{EG}(2,q)$ over $\mathrm{GF}(q)$. The null space of $\mathbf{H}^{(1)}_{q^2,\mathrm{EG},c,1}$ gives a q^2-ary cyclic EG-LDPC code over $\mathrm{GF}(q^2)$ of length $q^2 - 1$ with minimum distance at least $q + 1$.

In the following, two examples are given to illustrate the above construction of q^m-ary cyclic LDPC codes. In these and subsequent examples in the rest of this chapter, we set q as a power of 2, say $q = 2^s$. In decoding, we use the FFT-QSPA with 50 iterations. For each constructed code, we compute its error performance over a binary-input AWGN channel using BPSK signaling and compare its word-error performance with that of an RS code of the same length, rate, and symbol alphabet decoded with the hard-decision (HD) Berlekamp–Massey (BM) algorithm [23,24] (or the Euclidean algorithm) (see Sections 3.3 and 3.4) and the algebraic soft-decision (ASD) Koetter–Vardy (KV) algorithm [25] (the most

well-known soft-decision decoding algorithm for RS codes). The ASD-KV algorithm for decoding an RS code consists of three steps: *multiplicity assignment*, *interpolation*, and *factorization*. The major part of the computational complexity (70%) involved in application of the ASD-KV algorithm comes from the interpolation step and is on the order of $\lfloor\lambda\rfloor^4 N^2$ [27,28], denoted $\mathcal{O}(\lfloor\lambda\rfloor^4 N^2)$, where N is the length of the code and λ is a complexity parameter that is determined by the *interpolation cost* of the *multiplicity matrix* constructed at the multiplicity-assignment step. As λ increases, the performance of the ASD-KV algorithm improves, but the computational complexity increases drastically. As λ approaches ∞, the performance of the ASD-KV algorithm reaches its limit. The parameter is called the *interpolation complexity coefficient*.

Example 14.1. Let the two-dimensional Euclidean geometry EG(2,2^3) over GF(2^3) be the code-construction geometry. The subgeometry EG*(2,2^3) of EG(2,2^3) consists of 63 lines not passing through the origin of EG(2,2^3). These lines form a single cyclic class. Let α be a primitive element of GF(2^6). Form an α-multiplied 63×63 circulant over GF(2^6) with the 64-ary incidence vectors of the 63 lines in EG*(2,2^3) as rows arranged in cyclic order. Both the column weight and row weight of this circulant are 8. Use this circulant as the parity-check matrix $\mathbf{H}_{2^6,\mathrm{EG},c,1}$ of a cyclic EG-LDPC code. The null space over GF(2^6) of this parity-check matrix gives a 64-ary (63,37) cyclic EG-LDPC code $\mathcal{C}_{2^6,\mathrm{EG},c,1}$ over GF(2^6) with minimum distance at least 9, whose Tanner graph has a girth of at least 6. The generator polynomial of this cyclic LDPC code is

$$\begin{aligned}\mathbf{g}_{2^6}(X) = {} & \alpha^{26} + \alpha^{24}X^2 + \alpha^{20}X^6 + \alpha^{16}X^{10} + \alpha^{14}X^{12} \\ & + \alpha^{13}X^{13} + \alpha^{12}X^{14} + \alpha^{11}X^{15} + \alpha^2 X^{24} + X^{26}.\end{aligned}$$

It has eight consecutive powers of α as roots, from α^2 to α^9. The BCH lower bound on the minimum distance (see Section 3.3) of this 64-ary cyclic LDPC code is 9, which is the same as the column weight of $\mathbf{H}^{(1)}_{2^6,\mathrm{EG},c,1}$ plus 1.

The symbol- and word-error performances of this code decoded with the FFT-QSPA using 50 iterations over the binary-input AWGN channel with BPSK transmission are shown in Figure 14.1, which also includes the word performances of the (63,37,27) RS code over GF(2^6) decoded with the HD-BM and ASD-KV algorithms, respectively. At a WER of 10^{-6}, the 64-ary (63,37) cyclic LDPC code achieves a coding gain of 2.6 dB over the (63,37,27) RS code over GF(2^6) decoded with the HD-BM algorithm, while achieving coding gains of 1.8 dB and 1.2 dB over the RS code decoded using the ASD-KV algorithm with interpolation complexity coefficients 4.99 and ∞, respectively. The FFT-QSPA decoding of the 64-ary (63,37) cyclic LDPC code also converges very fast, as shown in Figure 14.2. At a WER of 10^{-6}, the performance gap between 3 and 50 iterations is only 0.1 dB, while the performance gap between 2 and 50 iterations is 0.6 dB. We see that, even with three iterations of the FFT-QSPA, the 64-ary (63,37) QC-LDPC code still achieves a coding gain of 1.7 dB over the (63,37,27) RS code decoded with the ASD-KV algorithm with interpolation-complexity coefficient 4.99.

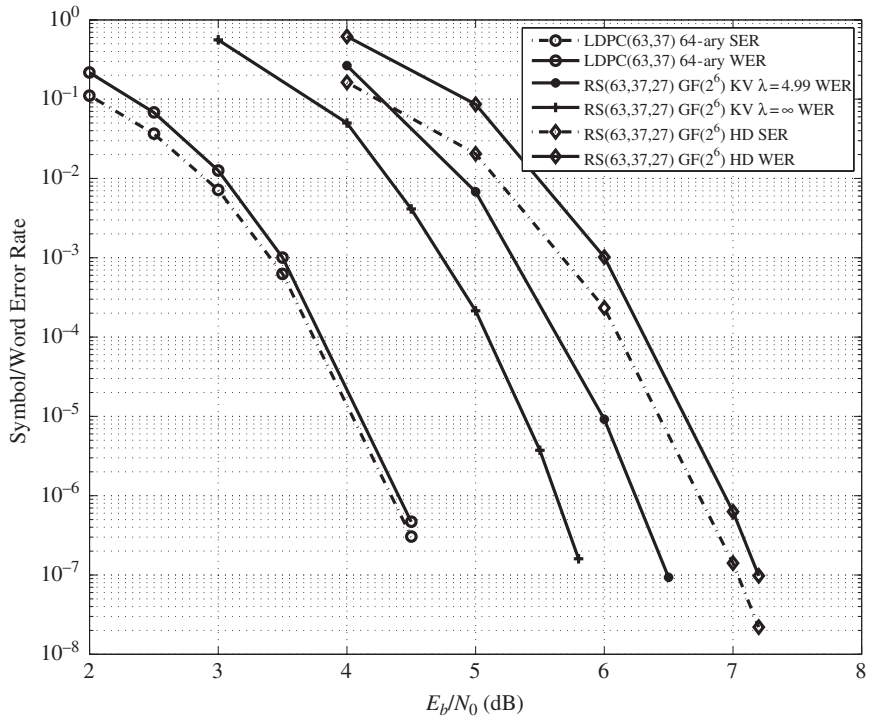


Figure 14.1 Symbol- and word-error performances of the 64-ary (63,37) cyclic EG-LDPC code decoded with 50 iterations of the FFT-QSPA and the word-error performances of the (63,37,27) RS code over GF(2^6) decoded with the ASD-KV algorithm with interpolation-complexity coefficients 4.99 and ∞.

To decode the 64-ary (63,37) cyclic LDPC code using the FFT-QSPA, the number of computations required per iteration is on the order of 193 536. With 3 and 50 iterations, the numbers of computations required are on the orders of 580 608 and 9 676 800, respectively. However, to decode the (63,37,27) RS code over GF(2^6) using the ASD-KV algorithm with interpolation-complexity coefficient 4.99, the number of computations needed in order to carry out the interpolation step is on the order of 1 016 064, which is greater than the 580 608 required with three iterations of the FFT-QSPA but less than the 9 676 800 required with 50 iterations of the FFT-QSPA. If we increase the interpolation coefficient to 9.99 in decoding the (63,37,27) RS code, the ASD-KV will achieve a 0.2 dB improvement in performance. Then the number of computations required to carry out the interpolation step would be 26 040 609, which is much larger than the 9 676 000 required with 50 iterations of the FFT-QSPA in decoding of the 64-ary (63,37) cyclic EG-LDPC code. In fact, for practical application, three iterations of the FFT-QSPA in decoding of the 64-ary (63,37) cyclic EG-LDPC code will be enough.

If 64-QAM is used for transmission, the symbol- and word-error performances of the 64-ary (63,37) cyclic EG-LDPC code are as shown in Figure 14.3. At a WER of 10^{-5}, the

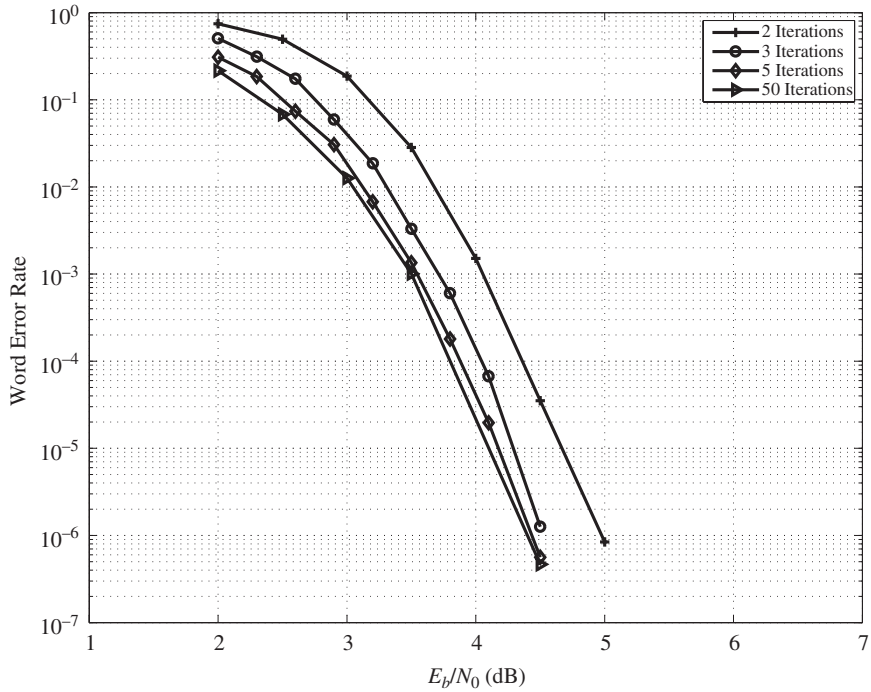


Figure 14.2 The rate of convergence in decoding of the 64-ary (63,37) cyclic EG-LDPC code using the FFT-QSPA with various numbers of iterations.

64-ary (63,37) cyclic EG-LDPC code achieves a coding gain of 3.3 dB over the (63,37,27) RS code decoded with the HD-BM algorithm.

Example 14.2. Let the two-dimensional Euclidean geometry EG(2,2^4) over GF(2^4) be the code-construction geometry. Let α be a primitive element of GF(2^8). On the basis of the 256-ary incidence vectors of the 255 lines in EG(2,2^4) not passing through the origin, we can form an α-multiplied 255×255 circulant over GF(2^8) whose column weight and row weight are both 16. The null space over GF(2^8) of this circulant gives a 256-ary (255,175) cyclic EG-LDPC code over GF(2^6) with minimum distance at least 17 whose Tanner graph has a girth of at least 6. The symbol- and word-error performances of this cyclic EG-LDPC code over a binary-input AWGN channel decoded using the FFT-QSPA with 50 iterations are shown in Figure 14.4, which also includes the symbol- and word-error performances of the (255,175,81) RS code over GF(2^8) decoded with the HD-BM algorithm. At a WER of 10^{-5}, the 256-ary (255,175) cyclic EG-LDPC code decoded with 50 iterations of the FFT-QSPA achieves a coding gain of 1.5 dB over the (255,175,81) RS code decoded with the HD-BM algorithm.

Figure 14.5 shows the word-error performances of the 256-ary (255,175) cyclic EG-LDPC code decoded with 3 and 50 iterations of the FFT-QSPA and the word-error

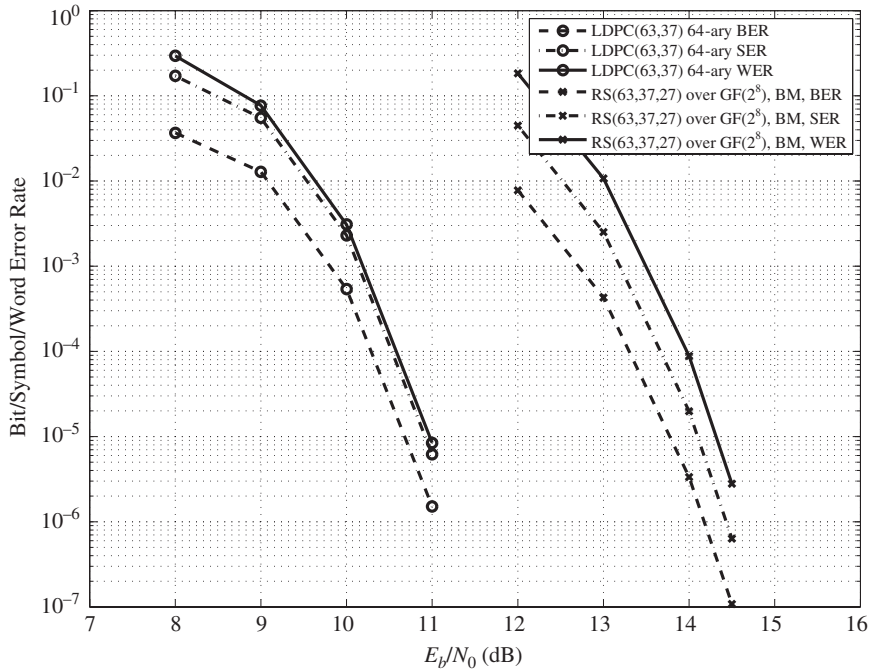


Figure 14.3 Symbol- and word-error performances of the 64-ary (63,37) cyclic EG-LDPC code over an AWGN channel with 64-QAM signaling.

performances of the (255,175,81) RS code decoded with the ASD-KV algorithm using interpolation-complexity coefficients 4.99 and ∞, respectively. We see that at a WER of 10^{-5}, the 256-ary cyclic EG-LDPC code with 50 iterations of the FFT-QSPA has coding gains of 1.1 dB and a 0.7 dB over its corresponding RS code decoded with the ASD-KV algorithm using interpolation-complexity coefficients 4.99 and ∞, respectively. With three iterations of the FFT-QSPA, the 256-ary cyclic EG-LDPC code achieves a coding gain of 1 dB over the corresponding RS code decoded using the ASD-KV algorithm with interpolation-complexity coefficient 4.99.

The number of computations required with three iterations in decoding of the 256-ary (255,175) cyclic EG-LDPC code with the FFT-QSPA is on the order of $3 \times (255 \times 16 \times 256 \times 8) = 25\,067\,520$. The number of computations required to carry out the interpolation step in decoding of the (255,175,81) RS code using the ASD-KV algorithm with interpolation-complexity coefficient 4.99 is on the order of 16 646 400.

One special feature of the q^m-ary cyclic EG-LDPC code constructed from the q^m-ary incidence vectors of the lines in the m-dimensional Euclidean geometry EG(m,q) over GF(q) is that the length $q^m - 1$ of the code is one less than the size q^m of the code alphabet, just like a primitive RS code over GF(q^m) [29].

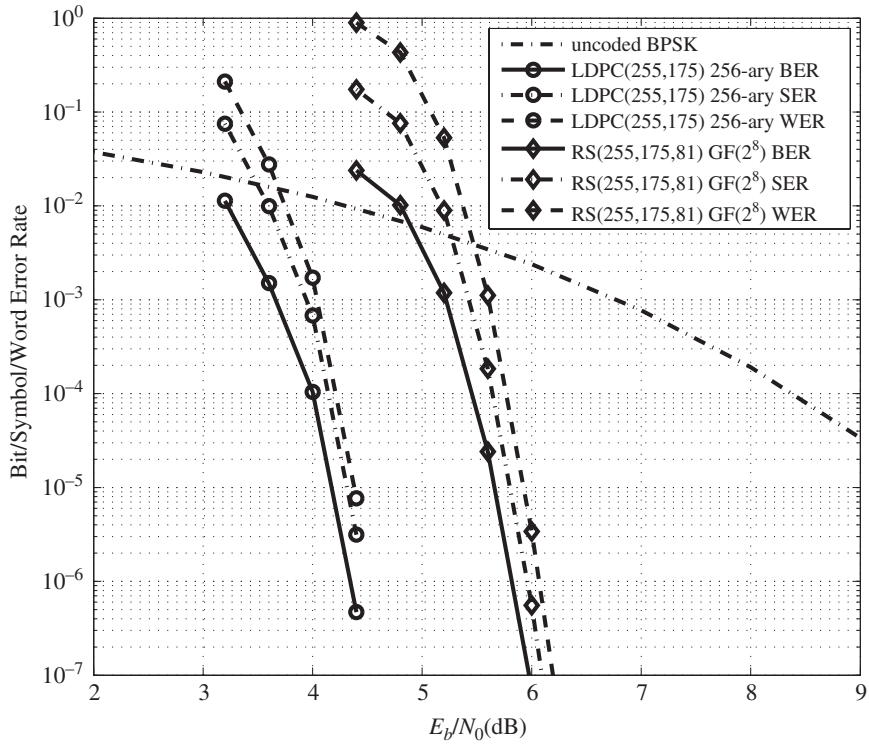


Figure 14.4 Bit-, symbol-, and word-error performances of the 256-ary (255,175) cyclic EG-LDPC code decoded with the FFT-QSPA using 50 iterations and the (255,175,81) RS code decoded with the HD-BM algorithm.

14.3.2 A Class of Nonbinary Quasi-Cyclic EG-LDPC Codes

For $1 \leq k \leq K_c$, let $\mathbf{H}^{(2)}_{q^m,\mathrm{EG},qc,k}$ be the transpose of the matrix $\mathbf{H}^{(1)}_{q^m,\mathrm{EG},c,k}$ given by (14.29), i.e.,

$$\begin{aligned}\mathbf{H}^{(2)}_{q^m,\mathrm{EG},c,k} &= \left[\mathbf{H}^{(1)}_{q^m,\mathrm{EG},qc,k}\right]^{\mathrm{T}} \\ &= \left[\mathbf{H}^{\mathrm{T}}_{q^m,c,1} \quad \mathbf{H}^{\mathrm{T}}_{q^m,c,2} \quad \cdots \quad \mathbf{H}^{\mathrm{T}}_{q^m,c,k}\right], \end{aligned} \tag{14.30}$$

where, for $1 \leq i < k$, $\mathbf{H}^{\mathrm{T}}_{q^m,c,i}$ is the transpose of the submatrix $\mathbf{H}_{q^m,c,i}$ in $\mathbf{H}^{(1)}_{q^m,\mathrm{EG},c,k}$. $\mathbf{H}^{(2)}_{q^m,\mathrm{EG},qc,k}$ consists of a row of k α-multiplied $(q^m-1)\times(q^m-1)$ circulants that is a $(q^m-1)\times k(q^m-1)$ matrix over $\mathrm{GF}(q^m)$ with column and row weights q and kq, respectively. Since $\mathbf{H}^{(1)}_{q^m,\mathrm{EG},c,k}$ satisfies the RC-constraint, $\mathbf{H}^{(2)}_{q^m,\mathrm{EG},qc,k}$ also satisfies the RC-constraint. The null space of $\mathbf{H}^{(2)}_{q^m,\mathrm{EG},qc,k}$ gives a q^m-ary QC-EG-LDPC code of length $k(q^m-1)$ with minimum distance at least $q+1$. For $m>2$, K_c is greater than unity. As a result, for $k>1$, the nonbinary QC-EG-LDPC code given by the null space of $\mathbf{H}^{(2)}_{q^m,\mathrm{EG},qc,k}$ has a length $k(q^m-1)$ that is longer than

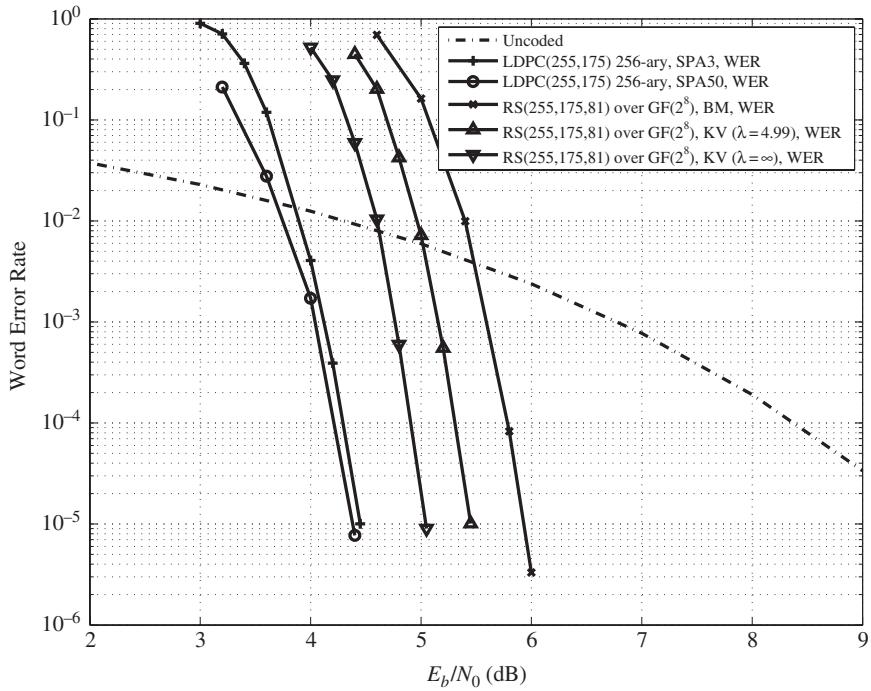


Figure 14.5 Word-error performances of the 256-ary (255,175) cyclic EG-LDPC code decoded with 3 and 50 iterations of the FFT-QSPA and the word-error performance of the (255,175,81) RS code decoded using the ASD-KV algorithm with interpolation-complexity coefficients 4.99 and ∞.

the size q^m of the code alphabet GF(q^m). The above construction allows us to construct longer nonbinary LDPC codes from a smaller nonbinary field.

Example 14.3. Let the three-dimensional Euclidean geometry EG(3,2^2) over GF(2^2) be the code-construction geometry. The subgeometry EG*(3,2^2) of EG(3,2^2) consists of 315 lines not passing through the origin of EG(3,2^2). These lines can be partitioned into five cyclic classes, each consisting of 63 lines not passing through the origin. Let α be a primitive element of GF(2^6). From the type-1 64-ary incidence vectors of the lines in these five cyclic classes, we can form five α-multiplied 63×63 circulants, $\mathbf{H}_{2^6,c,1}, \ldots, \mathbf{H}_{2^6,c,5}$ over GF(2^6), each having both column weight and row weight 4. Choose $k = 5$. On the basis of (14.30), we form the following 63×315 matrix over GF(2^6):

$$\mathbf{H}^{(2)}_{2^6,\mathrm{EG},qc,5} = \left[\mathbf{H}^{\mathrm{T}}_{2^6,c,1} \quad \mathbf{H}^{\mathrm{T}}_{2^6,c,2} \quad \mathbf{H}^{\mathrm{T}}_{2^6,c,3} \quad \mathbf{H}^{\mathrm{T}}_{2^6,c,4} \quad \mathbf{H}^{\mathrm{T}}_{2^6,c,5}\right],$$

which consists of a row of five α-multiplied 63×63 circulants over GF(2^6). The column and row weights of $\mathbf{H}^{(2)}_{2^6,\mathrm{EG},qc,5}$ are 4 and 20, respectively. The null space over GF(2^6) of $\mathbf{H}^{(2)}_{2^6,\mathrm{EG},qc,5}$ gives a 64-ary (315,265) QC-EG-LDPC code over GF(2^6) with rate 0.8412. The word-error performance of this 64-ary QC-EG-LDPC code over a

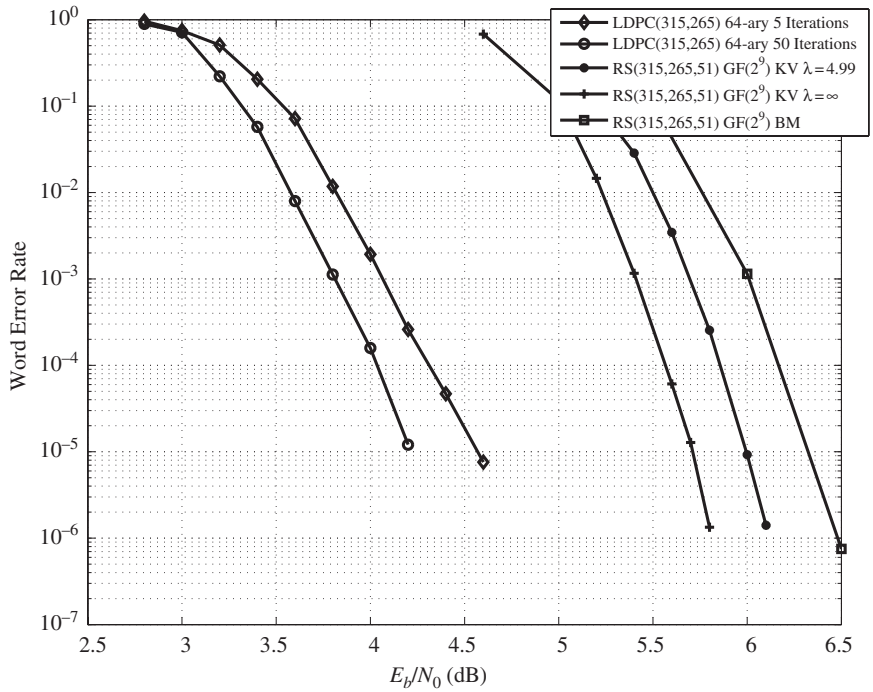


Figure 14.6 Word-error performances of the 64-ary (315,265) QC-EG-LDPC code over a binary-input AWGN channel decoded with 5 and 50 iterations of the FFT-QSPA.

binary-input AWGN channel decoded with 50 iterations of the FFT-QSPA is shown in Figure 14.6 which also includes the word performances of the (315,265,51) shortened RS code over GF(2^9) decoded with the HD-BM algorithm and the ASD-KV algorithm with interpolation-complexity coefficients 4.99 and ∞, respectively. At a WER of 10^{-5}, the 64-ary (315,265) QC-EG-LDPC code achieves a coding gain of 2.1 dB over the (315,265,51) shortened RS code decoded with the HD-BM algorithm, while achieving coding gains of 1.8 dB and 1.5 dB over the shortened RS code decoded using the ASD-KV algorithm with interpolation-complexity coefficients 4.99 and ∞, respectively. As shown in Figure 14.6, even with five iterations of the FFT-QSPA, the 64-ary (315,265) QC-EG-LDPC code achieves a coding gain of 1.5 dB over the (315,256,51) shortened RS code decoded using the ASD-KV algorithm with interpolation-complexity coefficient 4.99.

To decode the 64-ary (315,265) QC-EG-LDPC code with 5 and 50 iterations of the FFT-QSPA, the numbers of computations required are on the order of 2 381 400 and 23 814 000, respectively. However, the number of computations required in order to carry out the interpolation step in decoding of the (315,256,51) shortened RS code using the ASD-KV algorithm with interpolation-complexity coefficient 4.99 is on the order of 25 401 600, which is larger than both 2 381 400 and 23 814 000, the numbers of computations required in order to decode the 64-ary (315,265) QC-EG-LDPC code with 5 and 50 iterations of the FFT-QSPA, respectively.

14.3.3 A Class of Nonbinary Regular EG-LDPC Codes

The construction of binary regular LDPC codes based on the parallel bundles of lines in Euclidean geometries presented in Section 10.2 can be generalized to construct non-binary regular LDPC codes in a straightforward manner.

Consider the m-dimensional Euclidean geometry EG(m,q) over GF(q). Let $\mathcal{L}$ be a line in EG(m,q), passing or not passing through the origin of EG(m,q). Let α be a primitive element of GF(q^m). Define the following q^m-tuple over GF(q^m) based on the points on $\mathcal{L}$ as follows:

$$\mathbf{v}_{\mathcal{L}} = (v_{-\infty}, v_0, \ldots, v_{q^m-2}),$$

where $v_i = \alpha^i$ if α^i is a point on $\mathcal{L}$, otherwise $v_i = 0$. This q^m-tuple over GF(q^m) is referred to as the *type-2 q^m-ary incidence vector* of $\mathcal{L}$, in contrast to the binary type-2 incidence vector of $\mathcal{L}$ defined in Section 10.2 (see (10.13)).

As presented in Section 2.7.1 (and also in Section 10.2), the lines in EG(m,q) can be partitioned into $K_p = (q^m - 1)/(q - 1)$ parallel bundles, denoted $\mathcal{P}_1(m, 1), \mathcal{P}_2(m, 1), \ldots, \mathcal{P}_{K_p}(m, 1)$, each consisting of q^{m-1} parallel lines. For $1 \leq i \leq K_p$, form a $q^{m-1} \times q^m$ matrix $\mathbf{H}_{q^m,p,i}$ over GF(q^m) with the type-2 q^m-ary incidence vectors of the q^{m-1} parallel lines in the parallel bundle $\mathcal{P}_i(m, 1)$ as rows. It is clear that $\mathbf{H}_{q^m,p,i}$ has column and row weights 1 and q, respectively. Choose a positive integer k such that $1 \leq k \leq K_p$. Form a $kq^{m-1} \times q^m$ matrix over GF(q^m) as follows:

$$\mathbf{H}^{(3)}_{q^m,\mathrm{EG},p,k} = \begin{bmatrix} \mathbf{H}_{q^m,p,1} \\ \mathbf{H}_{q^m,p,2} \\ \vdots \\ \mathbf{H}_{q^m,p,k} \end{bmatrix}. \tag{14.31}$$

This matrix has column and row weights k and q, respectively. Since the rows of $\mathbf{H}^{(3)}_{q^m,\mathrm{EG},p,k}$ correspond to the lines in EG(m,q), $\mathbf{H}^{(3)}_{q^m,\mathrm{EG},p,k}$ satisfies the RC-constraint. The null space over GF(q^m) of $\mathbf{H}^{(3)}_{q^m,\mathrm{EG},p,k}$ gives a q^m-ary (k,q)-regular LDPC code over GF(q^m) of length q^m with minimum distance at least $k + 1$, whose Tanner graph has a girth of at least 6. The above construction gives a class of nonbinary regular LDPC codes.

Example 14.4. Consider the two-dimensional Euclidean geometry EG(2,2^4) over GF(2^4). This geometry consists of 272 lines, each consisting of 16 points. These lines can be partitioned into 17 parallel bundles, $\mathcal{P}_1(2, 1), \ldots, \mathcal{P}_{17}(2, 1)$, each consisting of 16 parallel lines. Take four parallel bundles of lines, say $\mathcal{P}_1(2, 1)$, $\mathcal{P}_2(2, 1)$, $\mathcal{P}_3(2, 1)$, and $\mathcal{P}_4(2, 1)$. Form a 64×256 matrix $\mathbf{H}^{(3)}_{2^8,\mathrm{EG},p,4}$ over GF(2^8) that has column and row weights 4 and 16, respectively. The null space over GF(2^8) of this matrix gives a 256-ary (4,16)-regular (256,203) EG-LDPC code over GF(2^8). The symbol- and word-error performances of this EG-LDPC code over the binary-input AWGN channel decoded with 5 and 50 iterations of the FFT-QSPA are shown in Figure 14.7. Also included in Figure 14.7 are the

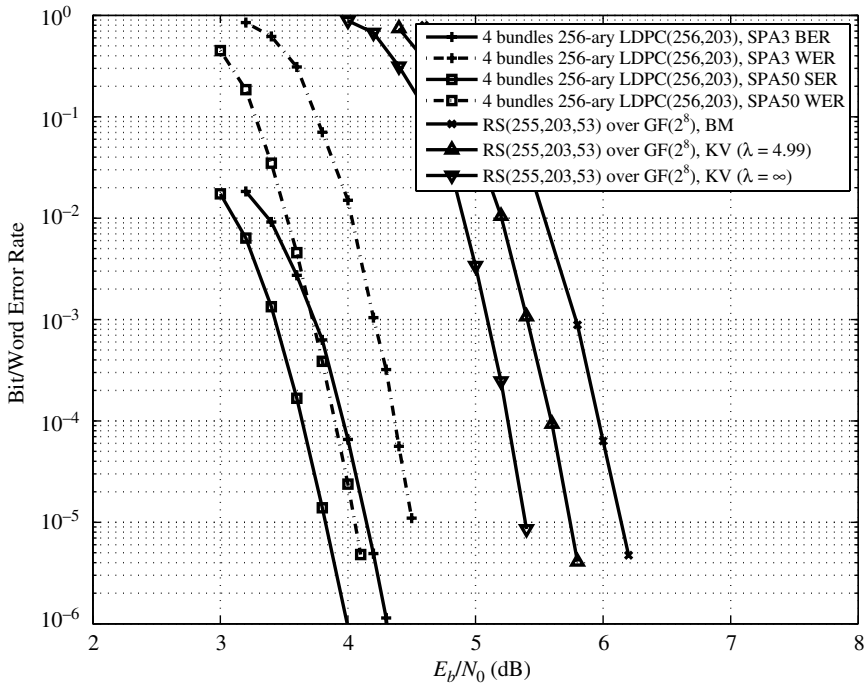


Figure 14.7 Symbol- and word-error performances of the 256-ary (256,203) EG-LDPC code given in Example 14.4.

word-error performances of the (255,203,53) RS code over GF(2^8) decoded with the HD-BM and ASD-KV algorithms. We see that the 256-ary (256,203) EG-LDPC code decoded with either 5 or 50 iterations of the FFT-QSPA achieves significant coding gains over the (255,203,53) RS code decoded with either the HD-BM algorithm or the ASD-KV algorithm.

14.3.4 Nonbinary LDPC Code Constructions Based on Projective Geometries

The two methods for constructing binary PG-LDPC codes presented in Section 10.6 can be generalized to construct nonbinary PG-LDPC codes in a straightforward manner.

Consider the m-dimensional projective geometry PG(m,q) over GF(q) with $m \geq 2$. This geometry consists of

$$n = (q^{m+1} - 1)/(q - 1)$$

points and

$$J_4 = J_{\mathrm{G}}(m, 1) = \frac{(q^m - 1)(q^{m+1} - 1)}{(q^2 - 1)(q - 1)}$$

lines (see (10.24) and (10.25)), each line consisting of $q + 1$ points.

Let α be a primitive element of GF(q^{m+1}). The n points of GF(q^m) can be represented by the elements $\alpha^0, \alpha, \ldots, \alpha^{n-1}$ of GF(q^{m+1}) (see Sections 2.7.2 and 10.6.1). Let $\mathcal{L}$ be a line in PG(m,q). The q^{m+1}-ary incidence vector of $\mathcal{L}$ is defined the following n-tuple over GF(q^{m+1}),

$$\mathbf{v}_{\mathcal{L}} = (v_0, v_1, \ldots, v_{n-1}),$$

whose components correspond to the n points, $\alpha^0, \alpha, \ldots, \alpha^{n-1}$ in PG(m,q), where for $0 \leq i < n$, $v_i = \alpha^i$ if and only if α^i is a point on $\mathcal{L}$, otherwise $v_i = 0$. It is clear that the weight of $\mathbf{v}_{\mathcal{L}}$ is $q+1$. The n-tuple over GF(q^{m+1}) obtained by cyclic-shifting $\mathbf{v}_{\mathcal{L}}$ one place to the right and multiplying every component of $\mathbf{v}_{\mathcal{L}}$ by α is also a q^{m+1}-ary incidence vector of a line in PG(m,q) (the power of α^{i+1} is taken modulo n).

As described in Section 2.6, for even m, the lines in PG(m,q) can be partitioned into

$$K_{c,\mathrm{PG}}^{(e)}(m,1) = \frac{q^m - 1}{q^2 - 1}$$

cyclic classes of size n (see (2.72)). For odd m, the lines in PG(m,q) can be partitioned into

$$K_{c,\mathrm{PG}}^{(o)}(m,1) = \frac{q(q^{m-1} - 1)}{q^2 - 1}$$

cyclic classes of size n (see (2.73)) and a single cyclic class of size

$$l_0 = (q^{m+1} - 1)/(q^2 - 1).$$

For each cyclic class S_i with $0 \leq i < K_{c,\mathrm{PG}}^{(e)}(m,1)$ (or $K_{c,\mathrm{PG}}^{(o)}(m,1)$), we form an $n \times n$ α-multiplied circulant $\mathbf{H}_{q^{m+1},c,i}$ over GF(q^{m+1}) with the q^{m+1}-ary incidence vectors of the n lines in S_i as rows arranged in cyclic order. Both the column weight and the row weight of $\mathbf{H}_{q^{m+1},c,i}$ are $q+1$. For $1 \leq k \leq K_{c,\mathrm{PG}}^{(e)}(m,1)$ (or $K_{c,\mathrm{PG}}^{(o)}(m,1)$), form the following $kn \times n$ matrix over GF(q^{m+1}):

$$\mathbf{H}_{q^{m+1},\mathrm{PG},c,k} = \begin{bmatrix} \mathbf{H}_{q^{m+1},c,0} \\ \mathbf{H}_{q^{m+1},c,1} \\ \vdots \\ \mathbf{H}_{q^{m+1},c,k-1} \end{bmatrix}, \tag{14.32}$$

which has column and row weights $k(q+1)$ and $q+1$, respectively. Since the rows of $\mathbf{H}_{q^{m+1},\mathrm{PG},c,k}$ correspond to the lines in PG(m,q) and two lines have at most one point in common, $\mathbf{H}_{q^{m+1},\mathrm{PG},c,k}$ satisfies the RC-constraint. Hence, the null space over GF(q^{m+1}) of $\mathbf{H}_{q^{m+1},\mathrm{PG},c,k}$ gives a q^{m+1}-ary cyclic PG-LDPC code. The above construction gives a class of nonbinary cyclic PG-LDPC codes.

Let $\mathbf{H}_{q^{m+1},\mathrm{PG},qc,k}$ be the transpose of $\mathbf{H}_{q^{m+1},\mathrm{PG},c,k}$, i.e.,

$$\mathbf{H}_{q^{m+1},\mathrm{PG},qc,k} = \mathbf{H}_{q^{m+1},PG,c,k}^{\mathrm{T}}. \tag{14.33}$$

$\mathbf{H}_{q^{m+1},\mathrm{PG},qc,k}$ is an $n \times kn$ matrix over $\mathrm{GF}(q^{m+1})$ with column and row weights $q+1$ and $k(q+1)$, respectively. The null space over $\mathrm{GF}(q^{m+1})$ of $\mathbf{H}_{q^{m+1},\mathrm{PG},qc,k}$ gives a q^{m+1}-ary QC-PG-LDPC code. The above construction gives a class of nonbinary QC-PG-LDPC codes.

Example 14.5. Let the two-dimensional projective geometry $\mathrm{PG}(2,2^3)$ over $\mathrm{GF}(2^3)$ be the code-construction geometry. This geometry has 73 points and 73 lines, each line consisting of nine points. The 73 lines of $\mathrm{PG}(2,2^3)$ form a single cyclic class. Let α be a primitive element of $\mathrm{GF}(2^9)$. Using the 2^9-ary incidence vectors of the lines in $\mathrm{PG}(2,2^3)$ as rows, we can form an α-multiplied 73×73 circulant over $\mathrm{GF}(2^9)$ with both column weight and row weight 9. The null space over $\mathrm{GF}(2^9)$ of this α-multiplied circulant gives a 2^9-ary (73,45) cyclic PG-LDPC code over $\mathrm{GF}(2^9)$ with minimum distance at least 10. The word-error performances of this code over the binary-input AWGN channel with BPSK transmission on decoding this code with 5 and 50 iterations of the FFT-QSPA are shown in Figure 14.8. Also included in Figure 14.8 are the word-error performances of the (73,45,39) shortened RS code over $\mathrm{GF}(2^9)$ decoded with the HD-BM and ASD-KV algorithms. At a WER of 10^{-5}, the 2^9-ary (73,45) cyclic PG-LDPC code achieves a coding gain of 2.6 dB over the (73,45,39) shortened RS code decoded with the HD-BM algorithm,

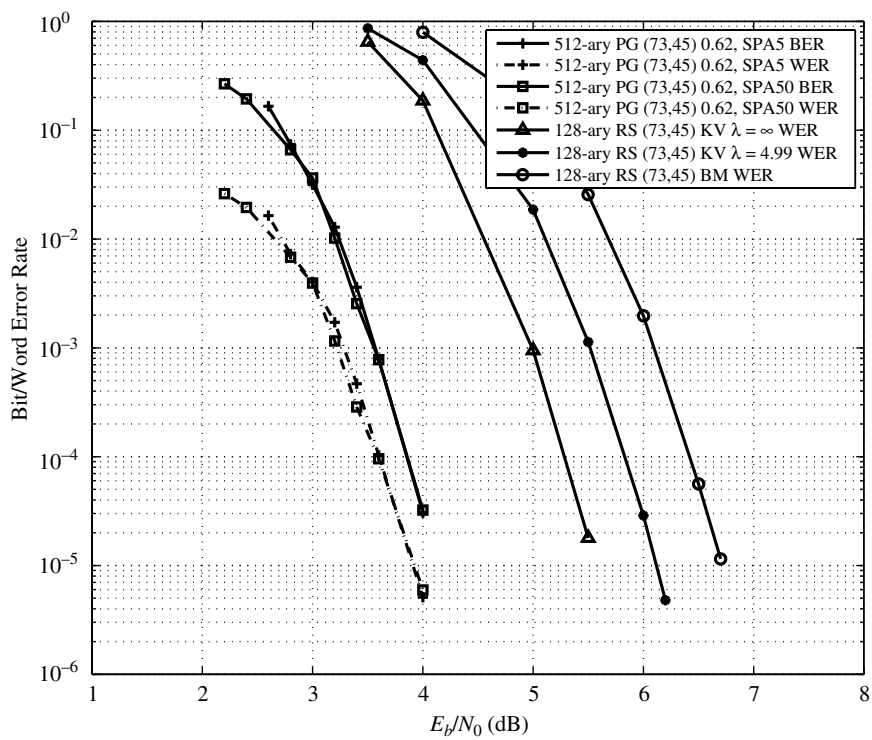


Figure 14.8 Word-error performances of the 2^9-ary (73,45) cyclic PG-LDPC code over a binary-input AWGN channel decoded with 5 and 50 iterations of the FFT-QSPA.

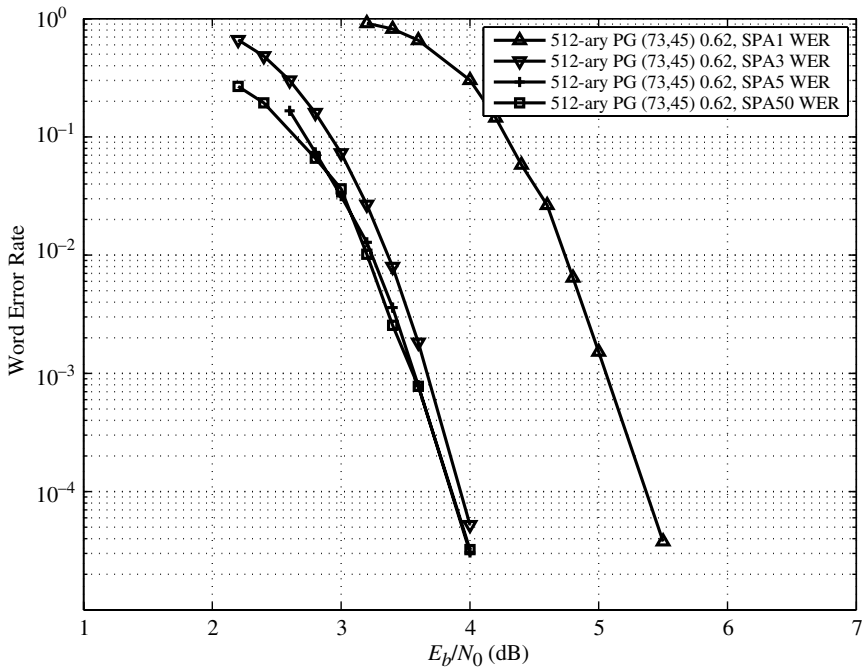


Figure 14.9 The rate of convergence in decoding of the 2^9-ary (73,45) cyclic PG-LDPC code with the FFT-QSPA.

while achieving coding gains of 2.0 dB and 1.4 dB over the RS code decoded using the ASD-KV algorithm with interpolation complexity coefficients 4.99 and ∞, respectively. Figure 14.9 shows that the FFT-QSPA decoding of the 2^9-ary (73,45) cyclic PG-LDPC code converges very fast. The word-error performance of the code with five iterations is almost the same as that with 50 iterations.

The numbers of computations required in decoding of the 2^9-ary (73,45) cyclic PG-LDPC code with 5 and 50 iterations are on the order of 15 107 715 and 151 077 150, respectively. However, the number of computations required in order to carry out the interpolation step in decoding the (73,45,39) shortened RS code with the ASD-KV algorithm with interpolation-complexity coefficient 4.99 is on the order of 1 364 224.

14.4 Constructions of Nonbinary QC-LDPC Codes Based on Finite Fields

In Chapter 11, methods for constructing binary QC-LDPC codes based on finite fields have been presented. These methods can be generalized to construct nonbinary QC-LDPC codes. The generalization is based on dispersing the elements of a nonbinary finite field GF(q) into *special circulant permutation matrices* over GF(q) in a way very similar to the binary matrix dispersions of elements of a finite field presented in Section 10.1.

14.4.1 Dispersion of Field Elements into Nonbinary Circulant Permutation Matrices

Consider the Galois field GF(q). Let α be a primitive element of GF(q). Then the q powers of α, $\alpha^{-\infty}, \alpha^0, \alpha, \ldots, \alpha^{q-2}$, form all the elements of GF(q). For each nonzero element α^i in GF(q) with $0 \leq i < q-1$, we form a $(q-1)$-tuple over GF(q), $\mathbf{z}(\alpha^i) = (z_0, z_1, \ldots, z_{q-2})$, whose components correspond to the $q-1$ nonzero elements, $\alpha^0, \alpha, \ldots, \alpha^{q-2}$, of GF($q$), where the ith component $z_i = \alpha^i$ and all the other components are equal to zero. This unit-weight $(q-1)$-tuple $\mathbf{z}(\alpha^i)$ over GF(q) is called the *q-ary location-vector* of α^i. The q-ary location-vector of the 0-element is defined as the all-zero $(q-1)$-tuple.

Let δ be a nonzero element of GF(q). Then the q-ary location-vector $\mathbf{z}(\alpha\delta)$ of the field element $\alpha\delta$ is the right cyclic-shift (one place to the right) of the q-ary location-vector of δ multiplied by α. Form a $(q-1) \times (q-1)$ matrix $\mathbf{A}$ over GF(q) with the q-ary location-vectors of $\delta, \alpha\delta, \ldots, \alpha^{q-1}\delta$ as rows. Matrix $\mathbf{A}$ is *a special type of CPM* over GF(q) for which each row is a right cyclic-shift of the row above it multiplied by α and the first row is the right cyclic-shift of the last row multiplied by α. Such a matrix is called a q-ary *α-multiplied CPM*. $\mathbf{A}$ is called the *q-ary $(q-1)$-fold matrix dispersion* of the field element δ of GF(q).

14.4.2 Construction of Nonbinary QC-LDPC Codes Based on Finite Fields

Let $\mathbf{W}$ be an $m \times n$ matrix over GF(q) given by (11.2) that satisfies α-multiplied row constraints 1 and 2. If we disperse every nonzero entry in $\mathbf{W}$ into an α-multiplied $(q-1) \times (q-1)$ CPM over GF(q) and a 0-entry into a $(q-1) \times (q-1)$ zero matrix, we obtain an $m \times n$ array of α-multiplied $(q-1) \times (q-1)$ circulant permutation and/or zero matrices over GF(q),

$$\mathbf{H}_{q,\text{disp}} = [\mathbf{A}_{i,j}]_{0 \leq i < m}^{0 \leq j < n}, \tag{14.34}$$

where $\mathbf{A}_{i,j}$ is either an α-multiplied $(q-1) \times (q-1)$ CPM or a $(q-1) \times (q-1)$ zero matrix. It can be proved that $\mathbf{H}_{q,\text{disp}}$ satisfies the RC-constraint. Just like in the binary case, we call $\mathbf{W}$ the base matrix for dispersion and $\mathbf{H}_{q,\text{disp}}$ the q-ary dispersion of $\mathbf{W}$.

For $1 \leq g \leq m$ and $1 \leq r \leq n$, let $\mathbf{H}_{q,\text{disp}}(g, r)$ be a $g \times r$ subarray of $\mathbf{H}_{q,\text{disp}}$. Then $\mathbf{H}_{q,\text{disp}}(g, r)$ is a $g(q-1) \times r(q-1)$ matrix over GF(q). The null space of $\mathbf{H}_{q,\text{disp}}(g, r)$ over GF(q) gives a q-ary QC-LDPC code over GF(q).

Any of the eight base matrices, $\mathbf{W}^{(1)}$ to $\mathbf{W}^{(8)}$, given in Chapter 11 can be dispersed into an RC-constrained array of α-multiplied $(q-1) \times (q-1)$ CPMs over GF(q). For $1 \leq i \leq 8$, let $\mathbf{H}_{q,\text{disp}}^{(i)}$ be the q-ary dispersion of the base matrix $\mathbf{W}^{(i)}$. Utilizing the q-ary arrays, $\mathbf{H}_{q,\text{disp}}^{(1)}$ to $\mathbf{H}_{q,\text{disp}}^{(8)}$, we can construct eight classes of q-ary QC-LDPC codes over GF(q). Furthermore, the masking technique presented in Section 10.4, the superposition technique presented in Section 12.7, and the array dispersion technique presented in Section 13.7 can also be used to construct q-ary QC-LDPC codes. In the following, we give several examples to illustrate various constructions of nonbinary QC-LDPC codes based on finite fields.

Example 14.6. From the (31,2,30) RS code over GF(2^5) and (11.5), we can construct a 31×31 base matrix $\mathbf{W}^{(1)}$ over GF(2^5) that satisfies α-multiplied row constraints 1 and 2, where α is a primitive element of GF(2^5). The dispersion of $\mathbf{W}^{(1)}$ gives a 31×31 array $\mathbf{H}^{(1)}_{2^5,\text{disp}}$ of α-multiplied 31×31 circulant permutation and zero matrices over GF(2^5) with the zero matrices on the main diagonal of $\mathbf{H}^{(1)}_{2^5,\text{disp}}$. Take the first four rows of $\mathbf{H}^{(1)}_{2^5,\text{disp}}$ and remove the first three columns. This results in a 4×28 subarray $\mathbf{H}^{(1)}_{2^5,\text{disp}}(4, 28)$ of $\mathbf{H}^{(1)}_{2^5,\text{disp}}$. The last submatrix of the first column of $\mathbf{H}^{(1)}_{2^5,\text{disp}}$ is a 31×31 zero matrix. $\mathbf{H}^{(1)}_{2^5,\text{disp}}(4, 28)$ is a 124×868 matrix over GF(2^5). The null space over GF(2^5) of $\mathbf{H}^{(1)}_{2^5,\text{disp}}(4, 28)$ gives a 32-ary near-regular (868,756) QC-LDPC code over GF(2^5) with rate 0.871. The symbol- and word-error performances of this code are shown in Figure 14.10. Also included in Figure 14.10 are the word-error performances of the (868,756,113) shortened RS code over GF(2^{10}) decoded with the HD-BM and ASD-KV algorithms with interpolation-complexity coefficients 4.99 and ∞, respectively. We see that, at a WER of 10^{-5}, the 32-ary (868,756) QC-LDPC code decoded with 50 iterations of the FFT-QSPA achieves a coding gain of 1.9 dB over the (868,756,113) shortened RS code decoded with the HD-BM algorithm, while achieving coding gains of 1.6 dB and 1.4 dB over the RS code decoded with the ASD-KV algorithm with interpolation-complexity coefficients 4.99 and ∞, respectively. The number of computations required in order to decode the

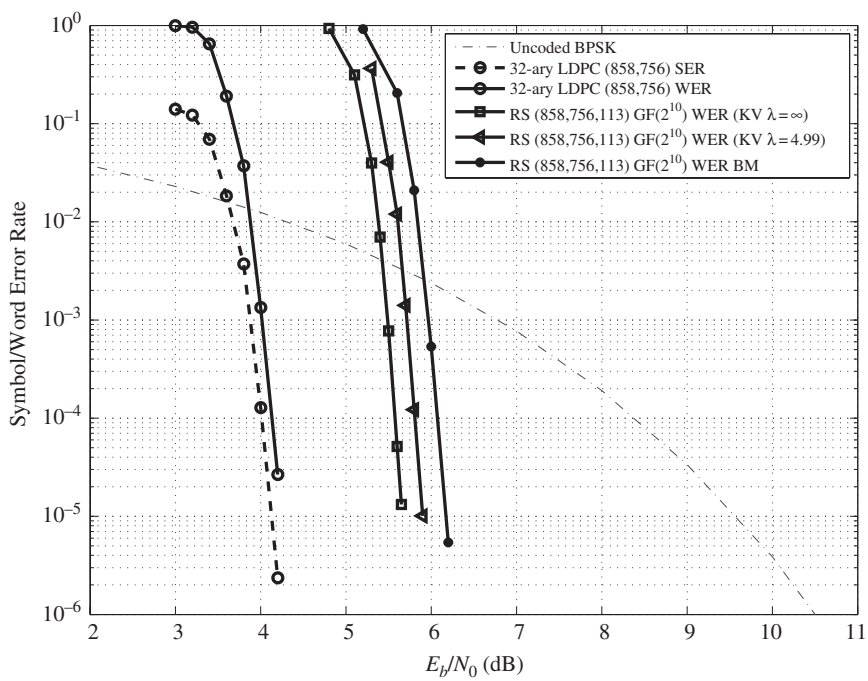


Figure 14.10 Symbol- and word-error performances of the 32-ary (868,756) QC-LDPC code given in Example 14.6 over a binary-input AWGN channel decoded with 50 iterations of the FFT-QSPA and the word-error performances of the (868,756,113) shortened RS code over GF(2^{10}) decoded with the HD-BM algorithm and the ASD-KV algorithm.

32-ary (868,756) QC-LDPC code with 50 iterations of the FFT-QSPA is on the order of 27 776 000; however, the number of computations required in order to carry out the interpolation step of the ASD-KV algorithm is on the order of 192 876 544.

14.4.3 Construction of Nonbinary QC-LDPC Codes by Masking

As shown in Chapters 10–13, masking can be applied to arrays of binary (circulant) permutation matrices to produce masked arrays with a low density of (circulant) permutation matrices. The null spaces of these masked arrays give binary LDPC codes whose Tanner graphs have fewer edges and short cycles. As a result, these binary LDPC codes may have very good error performances with reduced decoding complexity. Masking can also be applied to arrays of α-multiplied CPMs to produce high-performance non-binary QC-LDPC codes. This is illustrated in the next example.

Example 14.7. Let α be a primitive element of GF(2^6). Utilizing the (63,2,62) RS code over GF(2^6) and (11.5), we can construct a 63×63 matrix $\mathbf{W}^{(1)}$ over GF(2^6) that satisfies α-multiplied row constraints 1 and 2. On dispersing each nonzero entry in $\mathbf{W}^{(1)}$ into an α-multiplied 63×63 CPM over GF(2^6) and a 0-entry into a 63×63 zero matrix, we obtain a 63×63 RC-constrained array $\mathbf{H}^{(1)}_{2^6,\text{disp}}$ of α-multiplied circulant permutation and zero matrices, with the zero matrices lying on the main diagonal of $\mathbf{H}^{(1)}_{2^6,\text{disp}}$. Take an 8×16 subarray $\mathbf{H}^{(1)}_{2^6,\text{disp}}(8,16)$ from $\mathbf{H}^{(1)}_{2^6,\text{disp}}$, avoiding zero matrices. We use $\mathbf{H}^{(1)}_{2^6,\text{disp}}(8,16)$ as the base array for masking. Construct an 8×16 masking matrix $\mathbf{Z}(8,16)$ over GF(2) that consists of two 8×8 circulants, $\mathbf{G}_1$ and $\mathbf{G}_2$, in a row, each having both column weight and row weight 3. By computer search, we find two primitive generators (top rows) of the two circulants in $\mathbf{Z}(8,16)$, which are $\mathbf{g}_1 = (01011000)$ and $\mathbf{g}_2 = (00101010)$. By masking the base array $\mathbf{H}^{(1)}_{2^6,\text{disp}}(8,16)$ with $\mathbf{Z}(8,16)$, we obtain an 8×16 masked array $\mathbf{M}^{(1)}_{2^6}(8,16) = \mathbf{Z}(8,16) \otimes \mathbf{H}^{(1)}_{2^6,\text{disp}}(8,16)$ which is a 504×1008 matrix over GF(2^6) with column and row weights 3 and 6, respectively. The null space over GF(2^6) of $\mathbf{M}^{(1)}_{2^6}(8,16)$ gives a 64-ary (1008,504) QC-LDPC code over GF(2^6) with rate 0.5. The bit-, symbol- and word-error performances of this code decoded with 5 and 50 iterations of the FFT-QSPA are shown in Figure 14.11, which also includes the word-error performances of the (1008,504,505) shortened RS code over GF(2^6) decoded with the HD-BM algorithm and the ASD-KV algorithm with interpolation-complexity coefficients 4.99 and ∞, respectively. At a WER of 10^{-5}, the 64-ary (1008,504) QC-LDPC code achieves a coding gain of 4 dB over the (1008,504,505) RS code over GF(2^{10}) decoded with the HD-BM algorithm, while achieving coding gains of 3.3 dB and 2.8 dB over the shortened RS code decoded using the ASD-KV algorithm with interpolation-complexity coefficients 4.99 and ∞, respectively. Even with five iterations of the FFT-QSPA, the 64-ary (1008,504) QC-LDPC code has a coding gain of 2.3 dB over the RS code decoded using the ASD-KV algorithm with interpolation-complexity coefficient 4.99.

The numbers of computations required in decoding of the 64-ary (1008,504) QC-LDPC code with 5 and 50 iterations of the FFT-QSPA are on the order of 5 806 080 and 58 060 800, respectively. However,the number of computations required in order to

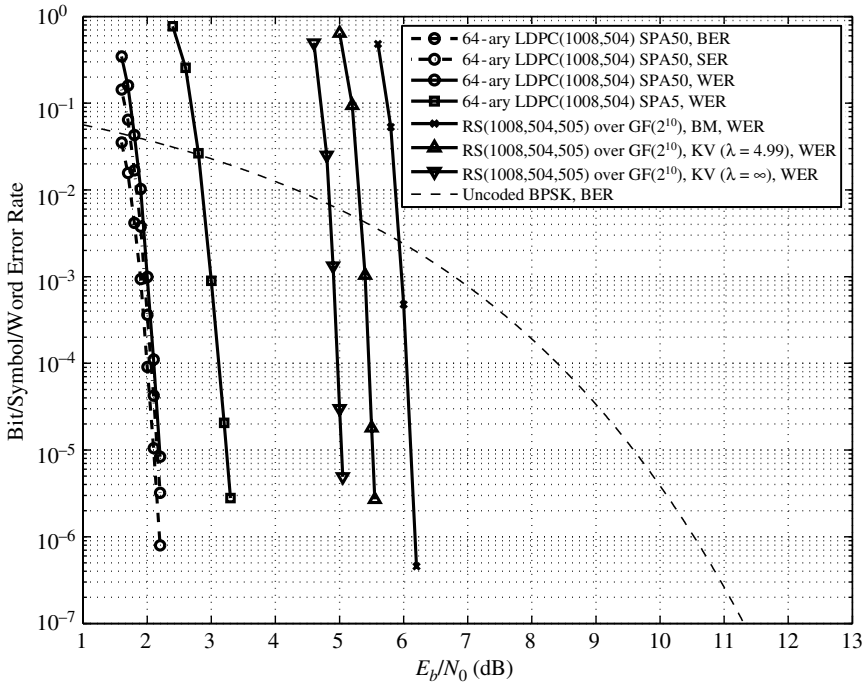


Figure 14.11 Error performances of the 64-ary (1008,504) QC-LDPC code given in Example 14.7 over a binary-input AWGN channel decoded with 5 and 50 iterations of the FFT-QSPA.

carry out the interpolation step of the ASD-KV algorithm with interpolation-complexity coefficient 4.99 in decoding of the (1008,504,505) RS code is on the order of 260 112 384.

14.4.4 Construction of Nonbinary QC-LDPC Codes by Array Dispersion

In Section 13.7, a technique, called *array dispersion*, has been presented for constructing binary QC-LDPC codes both for the AWGN and for erasure channels using arrays of binary CPMs. This technique can be generalized to construct nonbinary QC-LDPC codes. We begin with an array $\mathbf{H}_{q,\text{disp}}$ of α-multiplied CPMs over GF(q). Then take a subarray from $\mathbf{H}_{q,\text{disp}}$ and disperse it in exactly the same way as described in Section 13.7. The null space of the dispersed q-ary array gives a q-ary QC-LDPC code. We illustrate this construction by two examples.

Example 14.8. Consider the 31×31 array $\mathbf{H}^{(1)}_{2^5,\text{disp}}$ of α-multiplied 31×31 CPMs constructed in Example 14.6 using the (31,2,30) RS code over GF(2^5). Set $k = 2$, $l = 2$, $s = 2$ and $t = 5$. Take a 5×10 subarray $\mathbf{H}^{(1)}_{2^5,\text{disp}}(5, 10)$ from $\mathbf{H}^{(1)}_{2^5,\text{disp}}$, avoiding the zero matrices on the main diagonal of $\mathbf{H}^{(1)}_{2^5,\text{disp}}$. The 2-masked and 2-fold dispersion of $\mathbf{H}^{(1)}_{2^5,\text{disp}}(5, 10)$ gives a 10×20 array $\mathbf{H}^{(1)}_{2^5,\text{2-f.disp},2}(10, 20)$ of α-multiplied 31×31 circulant permutation and zero matrices over GF(2^5). $\mathbf{H}^{(1)}_{2^5,\text{2-f,disp},2}(10, 20)$ is a 310×620 matrix over GF(2^5)

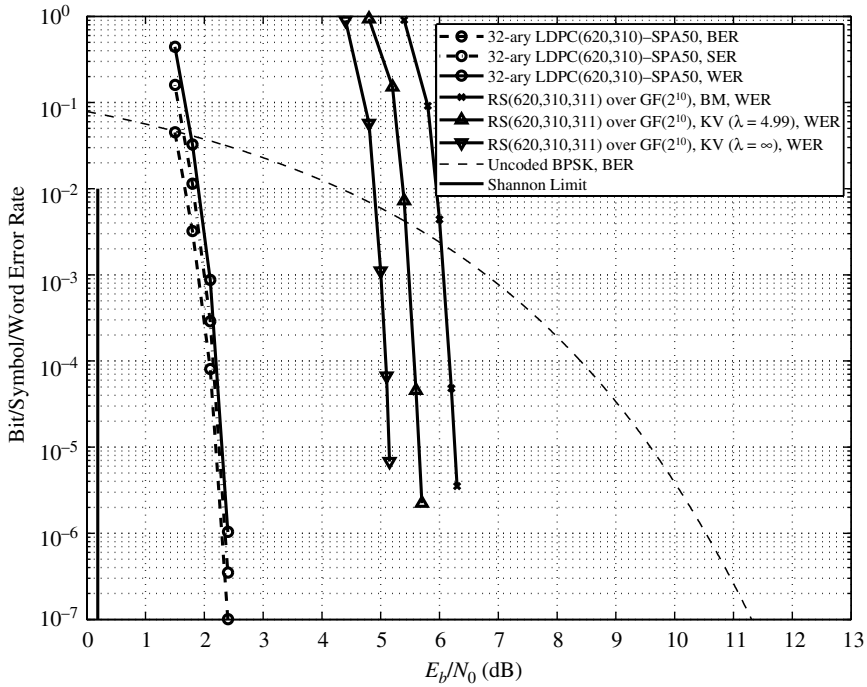


Figure 14.12 Bit-, symbol- and word-error performances of the 32-ary (620,310) QC-LDPC code over a binary-input AWGN channel decoded with 50 iterations of the FFT-QSPA.

with column and row weights 3 and 6, respectively. The null space of this matrix gives a 32-ary (3,6)-regular (620,310) QC-LDPC code over GF(2^5). The bit-, symbol-, and word-error performances of this code over a binary-input AWGN channel decoded with 50 iterations of the FFT-QSPA are shown in Figure 14.12, which also includes the word-error performances of the (620,310,311) shortened RS code over GF(2^{10}) decoded using the HD-BM algorithm and the ASD-KV algorithm with interpolation-complexity coefficients 4.99 and ∞, respectively. At a WER of 10^{-5}, the 32-ary (620,310) QC-LDPC code has a coding gains of 4 dB over the (620,310,311) RS code decoded with the HD-BM algorithm, while achieving coding gains of 3.2 dB and 2.8 dB over the RS code decoded using the ASD-KV with interpolation-complexity coefficients 4.99 and ∞, respectively.

The number of computations required in decoding the 32-ary (620,310) QC-LDPC code with 50 iterations of the FFT-QSPA is on the order of 14 880 000. However, the number of computations required in order to carry out the interpolation step of the ASD-KV algorithm with interpolation-complexity coefficient 4.99 in decoding of the (620,310,311) RS code is on the order of 101 606 400.

The 32-ary (620,310) QC-LDPC code over GF(2^5) is also capable of correcting any erasure-burst of length up to at least 218 symbols. The erasure-burst-correction efficiency of this code is at least 0.703.

14.5 Construction of QC-EG-LDPC Codes Based on Parallel Flats in Euclidean Geometries and Matrix Dispersion

In Section 14.3, methods for constructing nonbinary EG-LDPC codes based on two types of nonbinary incidence vectors of lines in Euclidean geometries have been presented. In this section, we show that matrices based on the points on parallel bundles of μ-flats in Euclidean geometries that satisfy α-multiplied row constraints 1 and 2 can be constructed. By dispersing the entries (represented by elements of finite fields) of these matrices into either α-multiplied CPMs or zero matrices, we obtain RC-constrained arrays of α-multiplied circulant permutation and/or zero matrices. From these arrays, we can construct nonbinary QC-EG-LDPC codes.

Consider the m-dimensional Euclidean geometry, EG(m,q), over GF(q). As shown in Section 2.7.1, for $0 \leq \mu \leq m$, there are

$$J_{\mathrm{EG}}(m,\mu) = q^{m-u} \prod_{i=1}^{\mu} (q^{m-i+1} - 1)/(q^{\mu-i+1} - 1)$$

μ-flats in EG(m,q) (see (2.49)), each consisting of q^{μ} points. These μ-flats can be partitioned into

$$K_{\mathrm{EG}}(m,\mu) = \prod_{i=1}^{\mu} (q^{m-i+1} - 1)/(q^{\mu-i+1} - 1)$$

parallel bundles of μ-flats (see (2.50)), each consisting of $q^{m-\mu}$ parallel μ-flats. The parallel μ-flats in a parallel bundle are *mutually disjoint* and contain all the q^m points of EG(m,q); each point appears once and only once on only one of the flats in the parallel bundle. See Section 2.71 for the detailed structure of Euclidean geometries over finite fields.

Let α be a primitive element of the Galois field GF(q^m). Then the powers of α, $\alpha^{-\infty} = 0, \alpha^0 = 1, \alpha^2, \ldots, \alpha^{q^m-2}$, give all the elements of GF(q^m) and they represent all the q^m points of EG(m,q). For $1 \leq \mu < m$, let $\mathcal{P}(m,\mu)$ be a parallel bundle of μ-flats. Let $\mathcal{F}_0^{(\mu)}, \mathcal{F}_1^{(\mu)}, \ldots, \mathcal{F}_{q^{m-\mu}-1}^{(\mu)}$ denote the $q^{m-\mu}$ parallel μ-flats in $\mathcal{P}(m,\mu)$, where $\mathcal{F}_0^{(\mu)}$ passes through the origin of EG(m,q). Suppose that $\mathcal{F}_0^{(\mu)}$ consists of the following set of points (represented by elements in GF(q^m)):

$$\mathcal{F}_0^{(\mu)} = \{\alpha^{j_0} = 0, \alpha^{j_1}, \ldots, \alpha^{j_{q^\mu-1}}\}. \tag{14.35}$$

Then for $1 \leq i < q^{m-\mu}$, the ith μ-flat $\mathcal{F}_i^{(\mu)}$ in $\mathcal{P}(m,\mu)$ consists of the following set of points:

$$\begin{aligned} \mathcal{F}_i^{(\mu)} &= \delta_i + \mathcal{F}_0^{(\mu)} \\ &= \{\delta_i, \delta_i + \alpha^{j_1}, \ldots, \delta_i + \alpha^{j_{q^\mu-1}}\}, \end{aligned} \tag{14.36}$$

where δ_i is a point *not in any other* μ-flat in $\mathcal{P}(m,\mu)$.

Let $n_1 = q^{\mu}$ and $n_2 = q^{m-\mu}$. Utilizing the parallel bundle $\mathcal{P}(m,\mu)$ of μ-flats in EG(m,q), we form the following $n_2 \times n_1$ matrix over GF(q^m):

$$\mathbf{W}_{q^m,\mathrm{EG},p,\mu} = \begin{bmatrix} 0 & \alpha^{j_1} & \cdots & \alpha^{j_{n_1-1}} \\ \delta_1 & \delta_1 + \alpha^{j_1} & \cdots & \delta_1 + \alpha^{j_{n_1-1}} \\ \vdots & \vdots & \ddots & \vdots \\ \delta_{n_2-1} & \delta_{n_2-1} + \alpha^{j_1} & \cdots & \delta_{n_2-1} + \alpha^{j_{n_1-1}} \end{bmatrix}, \tag{14.37}$$

where the entries of the ith row are the points of the ith μ-flat of $\mathcal{P}(m,\mu)$. $\mathbf{W}_{q^m,\mathrm{EG},p,\mu}$ has the following structural properties: (1) all the q^m entries are distinct and they form all the elements of GF(q^m), i.e., every element of GF(q^m) appears once and only once in the matrix; (2) any two rows (two columns) differ in all positions; and (3) the 0-entry appears at the upper-left corner of the matrix. From these structural properties, it can easily be proved that $\mathbf{W}_{q^m,\mathrm{EG},p,\mu}$ satisfies α-multiplied row constraints 1 and 2.

By dispersing every nonzero entry in $\mathbf{W}_{q^m,\mathrm{EG},p,\mu}$ into an α-multiplied $(q^m - 1) \times (q^m - 1)$ CPM and the single 0-entry into a $(q^m - 1) \times (q^m - 1)$ zero matrix, we obtain the following $n_2 \times n_1$ array $\mathbf{H}_{q^m,\mathrm{EG},p,\mu}$ of $q^m - 1$ α-multiplied CPMs and a single zero matrix:

$$\mathbf{H}_{q^m,\mathrm{EG},p,\mu} = \begin{bmatrix} \mathbf{A}_{0,0} & \mathbf{A}_{0,1} & \cdots & \mathbf{A}_{0,n_1-1} \\ \mathbf{A}_{1,0} & \mathbf{A}_{1,1} & \cdots & \mathbf{A}_{1,n_1-1} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{A}_{n_2-1,0} & \mathbf{A}_{n_2-1,1} & \cdots & \mathbf{A}_{n_2-1,n_1-1} \end{bmatrix}, \tag{14.38}$$

where $\mathbf{A}_{0,0} = \mathbf{O}$ (a $(q^m - 1) \times (q^m - 1)$ zero matrix). Since all the entries in $\mathbf{W}_{q^m,\mathrm{EG},p,\mu}$ are distinct, all the α-multiplied CPMs in $\mathbf{H}_{q^m,\mathrm{EG},p,\mu}$ are distinct. $\mathbf{H}_{q^m,\mathrm{EG},p,\mu}$ is an $n_2(q^m - 1) \times n_1(q^m - 1)$ matrix over GF(q^m) that satisfies the RC constraint.

For any pair $(g,\ r)$ of integers with $1 \le g \le n_2$ and $1 \le r \le n_1$, let $\mathbf{H}_{q^m,\mathrm{EG},p,\mu}(g,r)$ be a $g \times r$ subarray of $\mathbf{H}_{q^m,\mathrm{EG},p,\mu}$. Then $\mathbf{H}_{q^m,\mathrm{EG},p,\mu}(g,r)$ is a $g(q^m - 1) \times r(q^m - 1)$ matrix over GF(q^m) that also satisfies the RC-constraint. If $\mathbf{H}_{q^m,\mathrm{EG},p,\mu}(g,r)$ does not contain the single zero matrix of $\mathbf{H}_{q^m,\mathrm{EG},p,\mu}$, it has column and row weights g and r, respectively. The null space over GF(q^m) of $\mathbf{H}_{q^m,\mathrm{EG},p,\mu}(g,r)$ gives a (g,r)-regular q^m-ary QC-EG-LDPC code of length rn_1 with rate at least $(r-g)/r$, whose Tanner graph has a girth of at least 6. If $\mathbf{H}_{q^m,\mathrm{EG},p,\mu}(g,r)$ contains the single zero matrix of $\mathbf{H}_{q^m,\mathrm{EG},p,\mu}$, it has two different column weights, $g-1$ and g, and two row weights, $r-1$ and r. Then the null space of $\mathbf{H}_{q^m,\mathrm{EG},p,\mu}(g,r)$ gives a *near-regular* q^m-ary QC-EG-LDPC code. The above construction gives a class of q^m-ary QC-EG-LDPC codes. For $q = 2^s$, we obtain a subclass of 2^{ms}-ary QC-LDPC codes. In this subclass, a special case is that of $s = 1$. For this case, the geometry for code construction is EG(m,2) over GF(2), the simplest Euclidean geometry.

In the following, we use an example to illustrate the construction of q^m QC-EG-LDPC codes based on a parallel bundle of μ-flats of EG(m,q).

Example 14.9. Consider the five-dimensional Euclidean geometry EG(5,2) over GF(2) ($m = 5$ and $q = 2$). Let $\mu = 3$. For this geometry, each 3-flat consists of eight points. Each parallel bundle of 3-flats consists of four 3-flats. Take a parallel bundle $P(5, 3)$ of 3-flats for code construction. From this parallel bundle, (14.37) and (14.38), we can construct a 4×8 array $\mathbf{H}_{2^5,\mathrm{EG},p,3}$ of 31 α-multiplied 31×31 CPMs and a single zero matrix over GF(2^5). Set $g = 4$ and $r = 8$. Then $\mathbf{H}_{2^5,\mathrm{EG},p,3}(4, 8)$ is the entire array $\mathbf{H}_{2^5,\mathrm{EG},p,3}$. It is a 124×248 matrix over GF(2^5) with two different column weights, 3 (31 columns) and 4 (217 columns), and two different row weights, 7 (31 rows) and 8 (93 rows). The null space over GF(2^5) of this matrix gives a near-regular 32-ary (248,136) QC-LDPC code with rate 0.5484, whose Tanner graph has a girth of at least 6. For BPSK transmission over the binary-input AWGN channel, the bit-, symbol-, and word-error performances of this code decoded with iterative decoding using FFT-QSPA with 50 iterations are shown in Figure 14.13 which also includes the bit-, symbol-, and word-error performances of the (248,136,113) shortened RS code over GF(2^8) decoded with the HD-BM algorithm. At a BER of 10^{-6} (or a WER of 10^{-5}), the 32-ary (248,136) QC-LDPC code achieves a coding gain of 2.6 dB over the 256-ary (248,136,113) RS code. Figure 14.14 shows the word-error performances of the (248,136,113) shortened RS code over GF(2^8) decoded with the ASD-KV algorithm with interpolation-complexity coefficients equal to 4.99 and

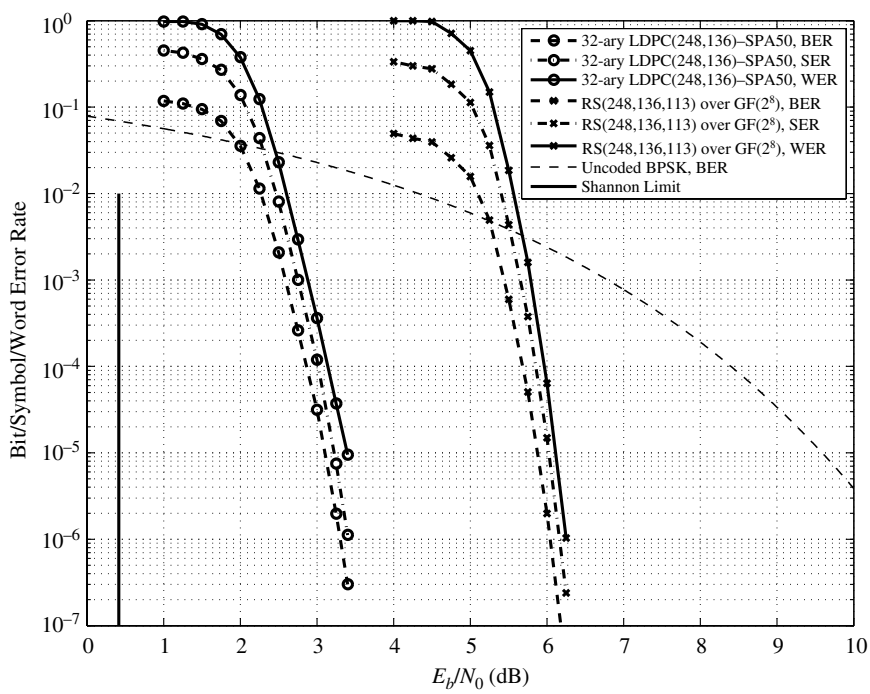


Figure 14.13 Bit-, symbol-, and word-error performances of the 32-ary (248,136) QC-EG-LDPC code given in Example 14.9.

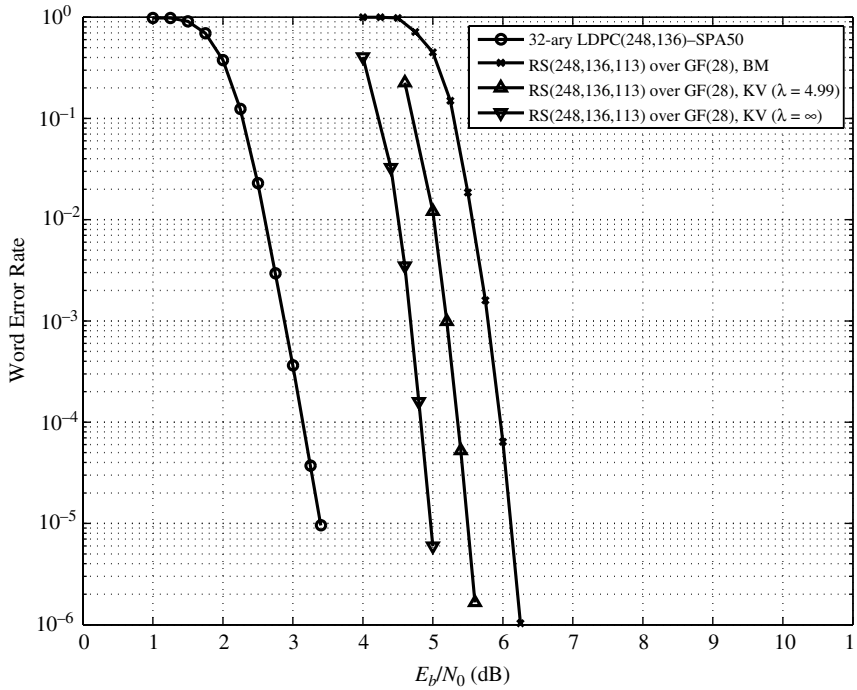


Figure 14.14 Word performances of the 32-ary (248,136) QC-EG-LDPC code decoded with the FFT-QSPA and the (248,136,113) shortened RS code over GF(2^9) decoded with the HD-BM and the ASD-KV algorithms.

∞, respectively. We see that, at a WER of 10^{-5}, the 32-ary (248,136) QC-EG-LDPC code has coding gains of 2.102 dB and 1.574 dB over the (248,136,113) shortened RS code over GF(2^8) decoded using the ASD-KV algorithm with interpolation-complexity coefficients 4.99 and ∞, respectively.

The number of computations required in order to decode the 32-ary (248,136) QC-EG-LDPC code with 50 iterations of the FFT-QSPA is on the order of 7 936 000; however, the number of computations required in order to carry out the interpolation step of the ASD-KV algorithm with interpolation-complexity coefficient 4.99 is on the order of 15 745 024. That is to say that, at a WER of 10^{-5}, the 32-ary (248,136) QC-EG-LDPC code decoded with 50 iterations of the FFT-QSPA achieves a coding gain of 2.102 dB over the (248,136,113) shortened RS code over GF(2^8) using the ASD-KV algorithm with fewer computations. Furthermore, the FFT-QSPA decoding of the (248,136) QC-LDPC code converges very fast, as shown in Figure 14.15. At a WER of 10^{-5}, even with three iterations of the FFT-QSPA, the 32-ary (248,136) QC-LDPC code provides coding gains of 1 dB and 0.5 dB over the (248,136,113) shortened RS code over GF(2^8) decoded using the ASD-KV algorithm with interpolation-complexity coefficients 4.99 and ∞, respectively. With five iterations of the FFT-QSPA, the coding gains are 1.1 dB and 1.5 dB, respectively.

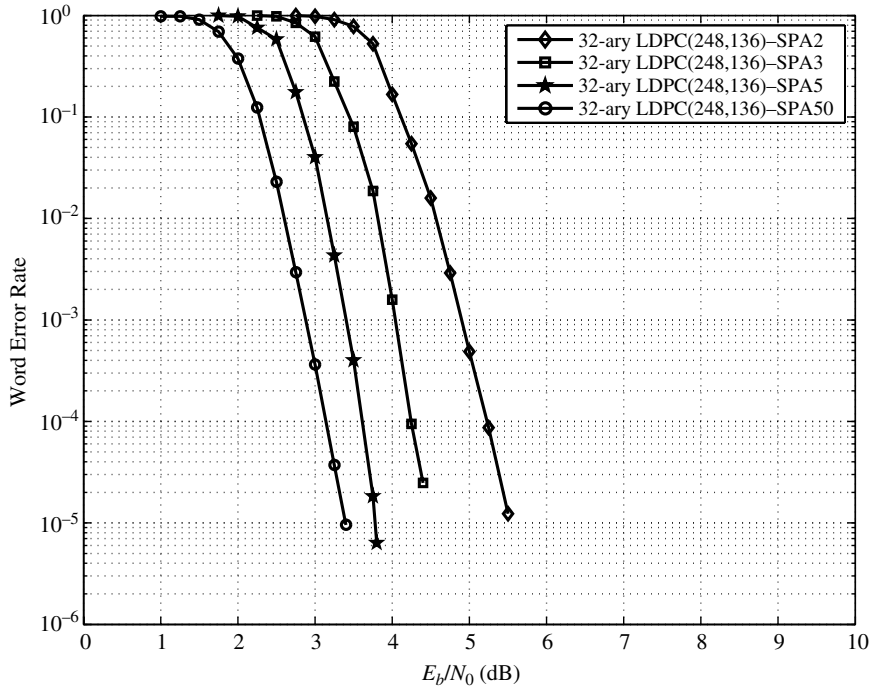


Figure 14.15 The rate of decoding convergence of the FFT-QSPA for the (248,136) QC-EG-LDPC code given in Example 14.9.

14.6 Construction of Nonbinary QC-EG-LDPC Codes Based on Intersecting Flats in Euclidean Geometries and Matrix Dispersion

In this section, we show that a matrix that satisfies α-multiplied row constraints 1 and 2 can be constructed utilizing the points of a bundle of intersecting flats of a Euclidean geometry. On dispersing this matrix, we obtain an RC-constrained array of α-multiplied CPMs. From this array, nonbinary QC-EG-LDPC codes can be constructed.

For $0 \leq \mu < m-1$, two $(\mu+1)$-flats in EG(m,q) are either disjoint or intersect on a flat with smaller dimension. The largest flat that two $(\mu+1)$-flats can intersect on is a μ-flat. The number of $(\mu+1)$-flats in EG(m,q) that intersect on a given μ-flat is $(q^{m-\mu}-1)/(q-1)$ (see (2.51)). Let EG$^*(m,q)$ be the subgeometry obtained from EG(m,q) by removing the origin and all the flats in EG(m,q) that pass through the origin. Then, for a given μ-flat $\mathcal{F}^{(\mu)}$ in EG$^*(m,q)$, there are $n_3 \triangleq q(q^{m-\mu-1}-1)/(q-1)$ $(\mu+1)$-flats in EG$^*(m,q)$ that intersect on $\mathcal{F}^{(\mu)}$. Let $\mathcal{F}_0^{(\mu+1)}, \mathcal{F}_1^{(\mu+1)}, \ldots, \mathcal{F}_{n_3-1}^{(\mu+1)}$ denote the n_3 $(\mu+1)$-flats that intersect on the given μ-flat $\mathcal{F}^{(\mu)}$. These n_3 $(\mu+1)$-flats form an *intersecting bundle* with respect to $\mathcal{F}^{(\mu)}$. Since $\mathcal{F}^{(\mu)}$ corresponds to a coset of a μ-dimensional subspace of the vector space V of all the m-tuples over GF(q), it contains q^μ points (represented by

nonzero elements of $\mathrm{GF}(q^m)$) of the following form:

$$\mathcal{F}^{(\mu)} = \left\{ \alpha^{j_0} + \sum_{k=1}^{\mu} \beta_k \alpha^{j_k} \colon \beta_k \in \mathrm{GF}(q), 1 \le k \le \mu \right\}, \tag{14.39}$$

where α is a primitive element of $\mathrm{GF}(q^m)$, and $\alpha^{j_0}, \alpha^{j_1}, \ldots, \alpha^{j_\mu}$ represent $\mu+1$ linearly independent points in $\mathrm{EG}(m,q)$. We denote the points on $\mathcal{F}^{(\mu)}$ by $\delta_0, \delta_1, \ldots, \delta_{q^\mu-1}$, where, for $0 \le j < q^\mu$, δ_j is represented by a power of α. Then, for $0 \le i < n_3$, the ith $(\mu+1)$-flat $\mathcal{F}_i^{(\mu+1)}$ which contains $\mathcal{F}^{(\mu)}$ consists of $q^{\mu+1}$ points of the following form:

$$\begin{aligned}\mathcal{F}_i^{(\mu+1)} &= \left\{ \alpha^{j_0} + \sum_{k=1}^{\mu} \beta_k \alpha^{j_k} + \beta_{\mu+1} \alpha^{j_{\mu+1,i}} \colon \beta_k, \beta_{\mu+1} \in \mathrm{GF}(q), 1 \le k \le \mu \right\} \\ &= \{\delta_j, \delta_j + \beta^0 \alpha^{j_{\mu+1,i}}, \ldots, \delta_j + \beta^{q-2} \alpha^{j_{\mu+1,i}} \colon \delta_j \in \mathcal{F}^{(\mu)}, 0 \le j < q^\mu\},\end{aligned} \tag{14.40}$$

where (1) $\alpha^{j_{\mu+1,i}}$ is linearly independent of $\alpha^{j_0}, \alpha^{j_1}, \ldots, \alpha^{j_\mu}$; (2) $\alpha^{j_{\mu+1,i}}$ is not contained in any other $(\mu+1)$-flat that contains $\mathcal{F}^{(\mu)}$; and (3) β is a primitive element of $\mathrm{GF}(q)$. From the first expression of (14.40), we see that $\mathcal{F}_i^{(\mu+1)}$ contains $\mathcal{F}^{(\mu)}$ and $q-1$ other μ-flats in $\mathrm{EG}^*(m,q)$ parallel to $\mathcal{F}^{(\mu)}$.

For $0 \le i < n_3$, on removing the points in $\mathcal{F}^{(\mu)}$ from $\mathcal{F}_i^{(\mu+1)}$, we obtain the following set of $n_4 \triangleq (q-1)q^\mu$ points:

$$\begin{aligned}\bar{\mathcal{F}}_i^{(\mu+1)} &= \mathcal{F}_i^{(\mu+1)} \backslash \mathcal{F}^{(\mu)} \\ &= \{\delta_j + \beta^0 \alpha^{j_{\mu+1,i}}, \ldots, \delta_j + \beta^{q-2} \alpha^{j_{\mu+1,i}} \colon \delta_j \in \mathcal{F}^{(\mu)}, 0 \le j < q^\mu\}.\end{aligned} \tag{14.41}$$

$\bar{\mathcal{F}}_i^{(\mu+1)}$ consists of $q-1$ parallel μ-flats in $\mathrm{EG}^*(m,q)$ and is called a $(\mu,q-1)$-frame with respect to $\mathcal{F}^{(\mu)}$. It is clear that the n_3 $(\mu,q-1)$-frames, $\bar{\mathcal{F}}_0^{(\mu+1)}, \bar{\mathcal{F}}_1^{(\mu+1)}, \ldots, \bar{\mathcal{F}}_{n_3-1}^{(\mu+1)}$, are mutually disjoint. These n_3 $(\mu,q-1)$-frames are said to form a *disjoint intersecting bundle* with respect to $\mathcal{F}^{(\mu)}$, denoted by $\mathbb{I}^{(\mu+1)}$. The bundle $\mathbb{I}^{(\mu+1)}$ contains $q^m - q^{\mu+1}$ distinct non-origin points of $\mathrm{EG}^*(m,q)$.

Using the disjoint intersecting bundle $\mathbb{I}^{(\mu+1)}$ with respect to a given μ-flat $\mathcal{F}^{(\mu)} = \{\delta_0, \delta_1, \ldots, \delta_{q^\mu-1}\}$, we form the following $n_3 \times n_4$ matrix over $\mathrm{GF}(q^m)$:

$$\mathbf{W}_{q^m,\mathrm{EG},\mathrm{int},\mu} = \begin{bmatrix} \mathbf{B}_0 & \mathbf{B}_1 & \cdots & \mathbf{B}_{q^\mu-1} \end{bmatrix}, \tag{14.42}$$

where, for $0 \le j < q^\mu$, $\mathbf{B}_j$ is an $n_3 \times (q-1)$ matrix over $\mathrm{GF}(q^m)$ defined as

$$\mathbf{B}_j = \begin{bmatrix} \delta_j + \beta^0 \alpha^{j_{\mu+1,0}} & \delta_j + \beta^1 \alpha^{j_{\mu+1,0}} & \cdots & \delta_j + \beta^{q-2} \alpha^{j_{\mu+1,0}} \\ \delta_j + \beta^0 \alpha^{j_{\mu+1,1}} & \delta_j + \beta^1 \alpha^{j_{\mu+1,1}} & \cdots & \delta_j + \beta^{q-2} \alpha^{j_{\mu+1,1}} \\ \vdots & \vdots & \ddots & \vdots \\ \delta_j + \beta^0 \alpha^{j_{\mu+1,n_3-1}} & \delta_j + \beta^1 \alpha^{j_{\mu+1,n_3-1}} & \cdots & \delta_j + \beta^{q-2} \alpha^{j_{\mu+1,n_3-1}} \end{bmatrix}. \tag{14.43}$$

The subscript "int" in $\mathbf{W}_{q^m,\mathrm{EG,int},\mu}$ stands for "intersecting." From (14.41) and (14.43), we readily see that, for $0 \le i < n_3$, the entries of the ith row of $\mathbf{W}_{q^m,\mathrm{EG,int},\mu}$ are simply the points on the ith $(\mu,q-1)$-frame $\bar{\mathcal{F}}_i^{(\mu+1)}$ in $\mathbb{I}^{(\mu+1)}$. Since the $(\mu,q-1)$-frames in $\mathbb{I}^{(\mu+1)}$ are mutually disjoint, any two rows (or two columns) of $\mathbf{W}_{q^m,\mathrm{EG,int},\mu}$ differ in all positions. Furthermore, all the entries in each row of $\mathbf{W}_{q^m,\mathrm{EG,int},\mu}$ are nonzero and distinct. From the structural properties of $\mathbb{I}^{(\mu+1)}$ and $\mathbf{W}_{q^m,\mathrm{EG,int},\mu}$, it can be easily proved that $\mathbf{W}_{q^m,\mathrm{EG,int},\mu}$ satisfies α-multiplied row constraints 1 and 2.

On dispersing each entry in $\mathbf{W}_{q^m,\mathrm{EG,int},\mu}$ into an α-multiplied $(q^m-1)\times(q^m-1)$ circulant permutation matrix over $\mathrm{GF}(q^m)$, we obtain an $n_3 \times n_4$ array $\mathbf{H}_{q^m,\mathrm{EG,int},\mu}$ of α-multiplied $(q^m-1)\times(q^m-1)$ CPMs,

$$\mathbf{H}_{q^m,\mathrm{EG,int},\mu} = \begin{bmatrix} \mathbf{A}_{0,0} & \mathbf{A}_{0,1} & \cdots & \mathbf{A}_{0,n_4-1} \\ \mathbf{A}_{1,0} & \mathbf{A}_{1,1} & \cdots & \mathbf{A}_{1,n_4-1} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{A}_{n_3-1,0} & \mathbf{A}_{n_3-1,1} & \cdots & \mathbf{A}_{n_3-1,n_4-1} \end{bmatrix}. \tag{14.44}$$

Since all the entries of $\mathbf{W}_{q^m,\mathrm{EG,int},\mu}$ are nonzero and distinct, all the α-multiplied CPMs in $\mathbf{H}_{q^m,\mathrm{EG,int},\mu}$ are distinct. The array $\mathbf{H}_{q^m,\mathrm{EG,int},\mu}$ is an $n_3(q^m-1)\times n_4(q^m-1)$ matrix over $\mathrm{GF}(q^m)$ with column and row weights n_3 and n_4, respectively. Since $\mathbf{W}_{q^m,\mathrm{EG,int},\mu}$ satisfies α-multiplied row-constraints 1 and 2, $\mathbf{W}_{q^m,\mathrm{EG,int},\mu}$, as a matrix, satisfies the RC-constraint.

For any pair $(g,\ r)$ of integers with $1 \le g \le n_3$ and $1 \le r \le n_4$, let $\mathbf{H}_{q^m,\mathrm{EG,int},\mu}(g,r)$ be a $g \times r$ subarray of $\mathbf{H}_{q^m,\mathrm{EG,int},\mu}$. The subarray $\mathbf{H}_{q^m,\mathrm{EG,int},\mu}(g,r)$ is a $g(q^m-1)\times r(q^m-1)$ matrix over $\mathrm{GF}(q^m)$ with column and row weights g and r, respectively. The null space over $\mathrm{GF}(q^m)$ of $\mathbf{H}_{q^m,\mathrm{EG,int},\mu}(g,r)$ gives a (g,r)-regular q^m-ary QC-LDPC code of length $r(q^m-1)$ with rate at least $(r-g)/r$, whose Tanner graph has a girth of at least 6. The above construction based on the intersecting structure of flats of an Euclidean geometry over a finite field gives a family of q^m-ary QC-EG-LDPC codes.

A special case of the above construction is the case for which $\mu = 0$. For $\mu = 0$, a 0-flat is simply a non-origin point of $\mathrm{EG}^*(m,q)$. Then, the rows of $\mathbf{W}_{q^m,\mathrm{EG,int},0}$ are the points of the lines in $\mathrm{EG}^*(m,q)$ which intersect at a given non-origin point α^j in $\mathrm{EG}^*(m,q)$ with the point α^j excluded. The number of lines in $\mathrm{EG}^*(m,q)$ which intersect at a point in $\mathrm{EG}^*(m,q)$ is $n_3 \triangleq q(q^{m-1}-1)/(q-1)$. A $(0,q-1)$-frame with respect to a non-origin point in $\mathrm{EG}^*(m,q)$ has $n_4 = q-1$ points. On the basis of the n_3 $(0,q-1)$-frames with respect to a non-origin point in $\mathrm{EG}^*(m,q)$, (14.42)–(14.44), we can construct an $n_3 \times (q-1)$ array $\mathbf{H}_{q^m,\mathrm{EG,int},\mu}$ of α-multiplied $(q^m-1)\times(q^m-1)$ CPMs over $\mathrm{GF}(q^m)$. The null space of any subarray of $\mathbf{H}_{q^m,\mathrm{EG,int},0}$ gives a q^m-ary QC-LDPC code.

For $m \ge 2$, there are more rows in $\mathbf{H}_{q^m,\mathrm{EG,int},\mu}$ than there are columns. Let $\mathbf{H}^{\mathrm{T}}_{q^m,\mathrm{EG,int},\mu}$ be the transpose of $\mathbf{H}_{q^m,EG,int,\mu}$. Using the subarrays of $\mathbf{H}^T_{q^m,EG,int,\mu}$, we can construct longer codes than we can using the subarrays of $\mathbf{H}_{q^m,\mathrm{EG,int},\mu}$.

Example 14.10. Let the three-dimensional Euclidean geometry $\mathrm{EG}(3,2^2)$ over $\mathrm{GF}(2^2)$ be the code-construction geometry. The geometry contains 64 points. For any given 1-flat $\mathcal{F}^{(1)}$ (a line) in the subgeometry $\mathrm{EG}^*(3,2^2)$ of $\mathrm{EG}(3,2^2)$, there are four 2-flats intersecting on it. From these four intersecting 2-flats on $\mathcal{F}^{(1)}$, we can construct four (1,3)-frames with respect to $\mathcal{F}^{(1)}$. From these four (1,3)-frames, (14.42)–(14.44), we can construct a 4×12 array $\mathbf{H}_{2^6,\mathrm{EG},\mathrm{int},1}$ of α-multiplied 63×63 CPMs over $\mathrm{GF}(2^6)$. Choose $k = 4$ and $r = 16$. The array $\mathbf{H}_{2^6,\mathrm{EG},\mathrm{int},1}(4,16)$ is simply the entire array $\mathbf{H}_{2^6,\mathrm{EG},\mathrm{int},1}$, which is a 252×756 matrix over $\mathrm{GF}(2^6)$ with column and row weights 4 and 16, respectively. The null space over $\mathrm{GF}(2^6)$ of this matrix gives a (4,16)-regular 64-ary (756,522) QC-EG-LDPC code with rate 0.69. The bit-, symbol-, and word-error performances of this code decoded using the FFT-QSPA with 50 iterations are shown in Figure 14.16, which also includes the word-error performances of the (756,522,235) shortened RS code over $\mathrm{GF}(2^{10})$ decoded using the HD-BM algorithm and the ASD-KV algorithm with interpolation-complexity coefficients 4.99 and ∞, respectively. At a WER of 10^{-5}, the 64-ary (756,522) QC-LDPC code achieves a coding gain of 2.555 dB over the (756,522,235) shortened RS code over $\mathrm{GF}(2^{10})$ decoded with the HD-BM algorithm, while achieving coding gains of 2.15 dB and 1.8 dB over the RS code decoded using the ASD-KV algorithm with interpolation-complexity coefficients 4.99 and ∞, respectively. The iterative decoding of the (756,522) QC-LDPC code also converges fast, as shown in Figure 14.17.

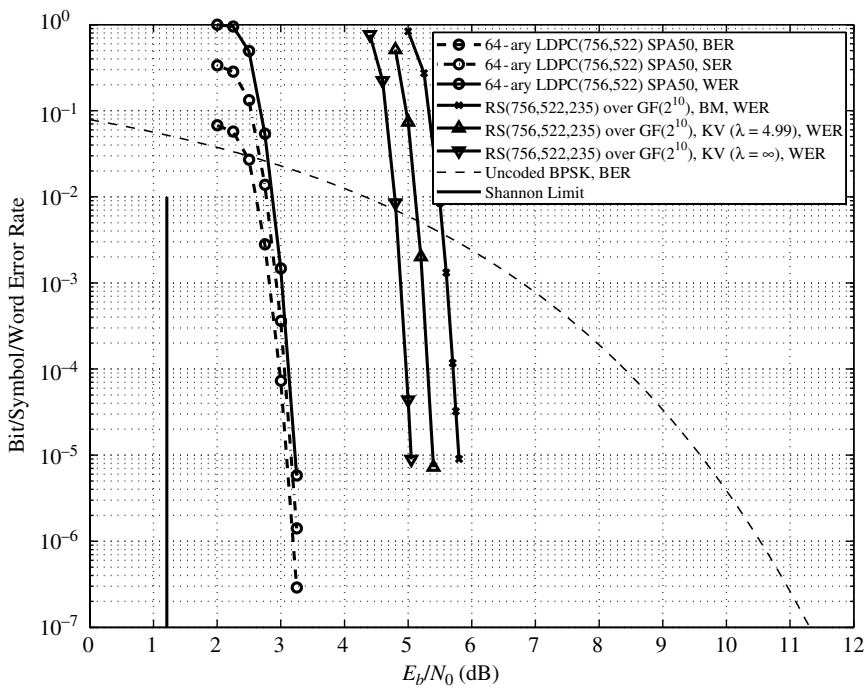


Figure 14.16 Error performances of the 64-ary (756,522) QC-EG-LDPC codes and the (756,522,235) shortened RS code over $\mathrm{GF}(2^{10})$ given in Example 14.10.

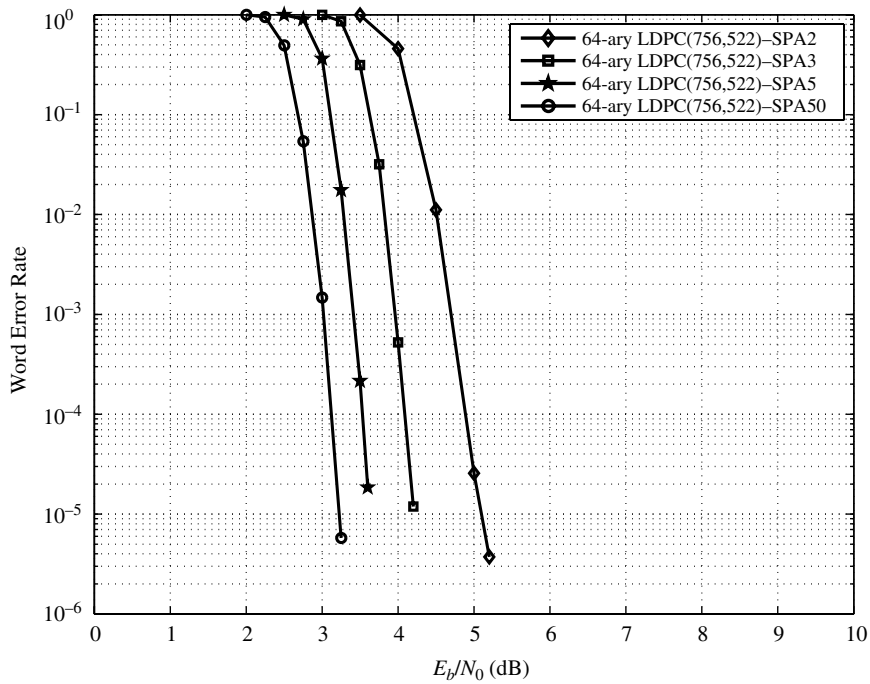


Figure 14.17 The rate of decoding convergence of the FFT-QSPA for the 64-ary (756,522) QC-EG-LDPC code given in Example 14.10

The number of computations required in order to decode the (756,522) QC-LDPC code with 50 iterations of the FFT-QSPA is on the order of 77 414 400, while the number of computations required in order to carry out the interpolation step of the ASD-KV algorithm for decoding the (756,522,235) shortened RS code over GF(2^{10}) with interpolation-complexity coefficient 4.99 is on the order of 146 313 216. So, the (756,522) QC-LDPC code decoded with 50 iterations of the FFT-QSPA achieves a coding gain of 2.15 dB over the (756,522,235) shortened RS code decoded using the ASD-KV algorithm with interpolation-complexity coefficient 4.99 with fewer computations.

14.7 Superposition–Dispersion Construction of Nonbinary QC-LDPC Codes

The technique of matrix dispersion of elements in a finite field presented in Section 11.1 and the technique of superposition presented in Section 12.7 can be combined to construct nonbinary QC-LDPC codes. We refer to this construction as the *superposition–dispersion* (SD) construction.

The SD construction of a q-ary QC-LDPC code begins with a $c \times t$ base matrix $\mathbf{B} = [b_{i,j}]$ over GF(2) that satisfies the RC constraint. Let $w_{b,c}$ and $w_{b,r}$ be the column and row weights of $\mathbf{B}$, respectively. Since $\mathbf{B}$ satisfies the RC-constraint, its associated Tanner graph is free of cycles of length 4 and hence it has a girth of at

least 6. Let $\mathbf{W} = [w_{i,j}]$ be an $m \times n$ matrix over GF(q) that satisfies α-multiplied row constraints 1 and 2, where α is a primitive element of GF(q) and $m \geqslant w_{b,c}$. Choose two positive integers k and l such that $kl \leq n$. Take an $m \times kl$ submatrix $\mathbf{M}$ from $\mathbf{W}$. Divide $\mathbf{M}$ into l submatrices, $\mathbf{M}_0, \mathbf{M}_1, \ldots, \mathbf{M}_{l-1}$, each being an $m \times k$ matrix over GF(q) and consisting of k consecutive columns of $\mathbf{M}$. For $0 \leq i < m$, the ith rows of $\mathbf{M}_0, \mathbf{M}_1, \ldots, \mathbf{M}_{l-1}$ are k-symbols sections of the ith row of $\mathbf{M}$. It is clear that all the submatrices, $\mathbf{M}$, $\mathbf{M}_0, \mathbf{M}_1, \ldots, \mathbf{M}_{l-1}$, of $\mathbf{W}$ also satisfy α-multiplied row constraints 1 and 2.

From the base matrix $\mathbf{B}$ and the submatrices, $\mathbf{M}_0, \mathbf{M}_1, \ldots, \mathbf{M}_{l-1}$, of $\mathbf{W}$, we form a $c \times kt$ matrix $\mathbf{Q}$ over GF(q) as follows: (1) replace the $w_{b,c}$ 1-entries in a column of $\mathbf{B}$ by different rows from the same submatrix $\mathbf{M}_i$ of $\mathbf{M}$; (2) replace the $w_{b,r}$ 1-entries in a row of $\mathbf{B}$ by the rows of $\mathbf{M}_0, \mathbf{M}_1, \ldots, \mathbf{M}_{l-1}$ that are sections from the same row of matrix $\mathbf{M}$; and (3) replace each 0-entry of $\mathbf{B}$ by a row of k zeros. The constraint on the replacement of 1-entries in a column of the base matrix $\mathbf{B}$ by the rows of a submatrix $\mathbf{M}_i$ of $\mathbf{M}$ is called the *column-replacement constraint*. The constraint on the replacement of 1-entries in a row of $\mathbf{B}$ by sections of a row in matrix $\mathbf{M}$ is called the *row-replacement constraint*. The RC-constraint on the rows and columns of $\mathbf{B}$, the structural properties of $\mathbf{W}$ (see Section 11.2), and the constraints on the replacement of the 1-entries in the base matrix $\mathbf{B}$ by the rows of $\mathbf{M}_0, \mathbf{M}_1, \ldots, \mathbf{M}_{l-1}$ ensure that no two rows (or two columns) of $\mathbf{Q}$ have more than one position where they both have the same nonzero symbol in GF(q), i.e., two rows (or two columns) of $\mathbf{Q}$ have at most one position where they have identical nonzero symbols in GF(q).

Next we disperse each nonzero entry of $\mathbf{Q}$ into an α-multiplied $(q-1) \times (q-1)$ CPM over GF(q) and each zero entry into a $(q-1) \times (q-1)$ zero matrix. The matrix dispersions of the entries of $\mathbf{Q}$ result in a $c \times kt$ array $\mathbf{H}_{\text{sup,disp}}(c, kt)$ of α-multiplied $(q-1) \times (q-1)$ circulant permutation and zero matrices, where "sup" and "disp" stand for superposition and dispersion, respectively. $\mathbf{H}_{\text{sup,disp}}(c, kt)$ is a $c(q-1) \times kt(q-1)$ matrix over GF(q) that satisfies the RC constraint. If $\mathbf{M}$ does not contain zero entries, then the column and row weights of $\mathbf{H}_{\text{sup,disp}}(c, kt)$ are $w_{b,c}$ and $kw_{b,r}$, respectively. The null space over GF(q) of $\mathbf{H}_{\text{sup,disp}}(c, kt)$ gives a q-ary ($w_{b,c}$,$w_{b,r}$)-regular QC-LDPC code. Any of the matrices, $\mathbf{W}^{(1)}$ to $\mathbf{W}^{(8)}$, given in Chapter 11 can be used as $\mathbf{W}$ for SD construction of q-ary QC-LDPC codes.

Example 14.11. Let $\mathbf{B}$ be the 8×8 circulant over GF(2) constructed from the type-1 binary incidence vectors of the eight lines in the two-dimensional Euclidean geometry EG(2,3) over GF(3) not passing through the origin of EG(2,3). The column and row weights of $\mathbf{B}$ are both 3. The matrix $\mathbf{B}$ is used as the base matrix for the SD construction of a q-ary QC-LDPC code. Let $\mathbf{W}^{(1)}$ be the 31×31 matrix over GF(2^5) constructed from the (31,2,30) RS code over GF(2^5) given in Example 14.6. We use $\mathbf{W}^{(1)}$ as the matrix $\mathbf{W}$ for the SD construction of a 32-ary QC-LDPC code. Choose $k = 5$ and $l = 6$. Take the first 30 columns of $\mathbf{W}^{(1)}$ to form the matrix $\mathbf{M}$ over GF(2^5). Divide $\mathbf{M}$ into six 31×5 submatrices, $\mathbf{M}_0, \mathbf{M}_1, \ldots, \mathbf{M}_5$. Using the matrices $\mathbf{B}, \mathbf{M}_0, \mathbf{M}_1, \ldots, \mathbf{M}_5$ and the

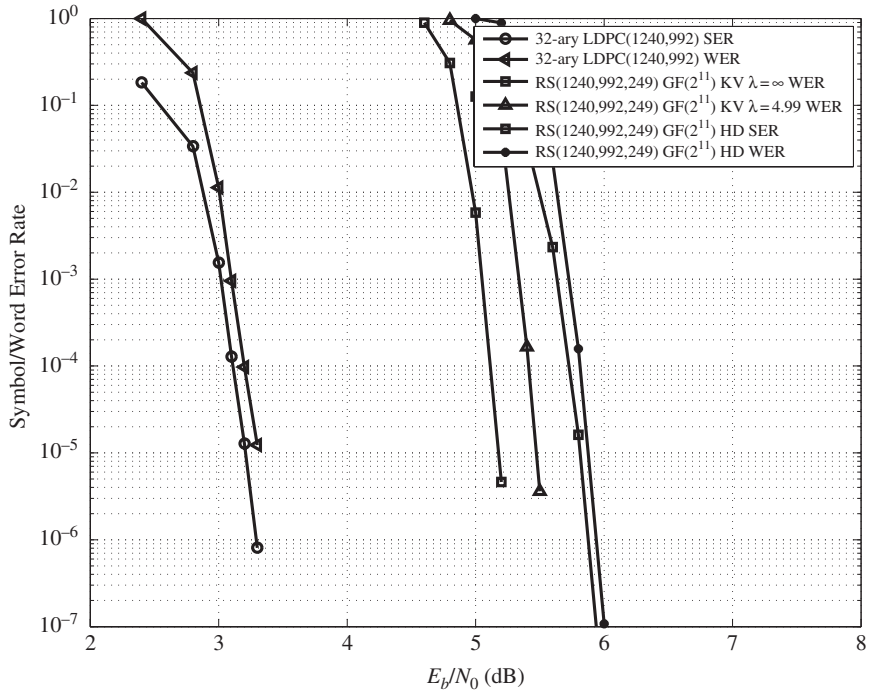


Figure 14.18 Symbol- and word-error performances of the 32-ary (1240,992) QC-LDPC code over the AWGN channel decoded with 50 iterations of the FFT-QSPA.

constraints on the replacement of the 1-entries in $\mathbf{B}$ by rows from $\mathbf{M}_0, \mathbf{M}_1, \ldots, \mathbf{M}_5$, we construct an 8×40 array $\mathbf{H}_{\text{sup,disp}}(8, 40)$ of α-multiplied 31×31 circulant permutation and zero matrices over GF(2^5). This array is a 248×1240 matrix over GF(2^5) with column and row weights 3 and 15, respectively. The null space over GF(2^5) gives a 32-ary (1240,992) QC-LDPC code over GF(2^5) with rate 0.8. The symbol- and word-error performances of this code over the binary-input AWGN channel decoded with 50 iterations of the FFT-QSPA are shown in Figure 14.18, which also includes the word performances of the (1240,992,249) shortened RS code over GF(2^{11}) decoded with the HD-BM algorithm and the ASD-KV algorithm for comparison. We see that, at a WER of 10^{-5}, the 32-ary (1240,992) QC-LDPC code achieves a coding gain of 2.5 dB over the RS code decoded with the HD-BM algorithm, while achieving coding gains of 2.15 dB and 1.85 dB over the RS code decoded using the ASD-KV algorithm with interpolation-complexity coefficients 4.99 and ∞, respectively.

The number of computations required in order to decode the 32-ary (1240,992) QC-LDPC code with 50 iterations of the FFT-QSPA is on the order of 29 760 000. However, the number of computations required in order to carry out the interpolation step of the ASD-KV algorithm with interpolation-complexity coefficient 4.99 in decoding of the (1240,992,249) shortened RS code is on the order of 393 625 600.

See also [29–36].

Problems

14.1 Let α be a primitive element of the Galois field GF(q) and $\mathbf{W}$ a matrix over GF(q) that satisfies α-multiplied row constraints 1 and 2. Show that the matrix $\mathbf{H}$ over GF(q) obtained by dispersing every nonzero entry of $\mathbf{W}$ into an α-multiplied $(q-1) \times (q-1)$ circulant permutation matrix and a zero entry into a $(q-1) \times (q-1)$ zero matrix satisfies the RC-constraint.

14.2 Find the generator polynomial of the 256-ary (255,175) cyclic EG-LDPC code given in Example 14.2.

14.3 Let the three-dimensional Euclidean geometry EG(3,2^2) over GF(2^2) be the code-construction geometry. Construct a 64-ary QC-EG-LDPC code of length 126. Determine its dimension. Assume BPSK transmission over the binary-input AWGN channel. Compute the bit-, symbol-, and word-error performances of the 64-ary QC-EG-LDPC code decoded with 50 iterations of the FFT-QSPA.

14.4 Let the two-dimensional Euclidean geometry EG(2,2^4) over GF(2^4) be the code-construction geometry. Construct a regular 255-ary regular EG-LDPC code of length 256 using six parallel bundles of lines in EG(2,2^4). Determine its dimension. Decode the constructed code with 50 iterations of the FFT-QSPA. Compute its bit-, symbol-, and block-error performances over the binary-input AWGN channel with BPSK signaling.

14.5 Construct a 15×15 array $\mathbf{H}^{(1)}_{2^4,\text{disp}}$ of α-multiplied 15×15 circulant permutation and zero matrices over GF(2^4) based on the (15,2,14) RS code over GF(2^4).

1. Determine the 16-ary QC-LDPC code given by the null space over GF(2^4) of the 15×15 array $\mathbf{H}^{(1)}_{2^4,\text{disp}}$.
2. Decode the code constructed in (a) with 50 iterations of the FFT-QSPA. Compute its bit-, symbol-, and word-error performances over the binary-input AWGN channel with BPSK signaling.

14.6 Starting from the (31,2,30) RS code over GF(2^5), construct a 31×31 array $\mathbf{H}^{(1)}_{2^5,\text{disp}}$ of α-multiplied 31×31 circulant permutation and zero matrices over GF(2^5). Take a 4×16 subarray $\mathbf{H}^{(1)}_{2^5,\text{disp}}(4,16)$ from $\mathbf{H}^{(1)}_{2^5,\text{disp}}$, avoiding zero matrices in $\mathbf{H}^{(1)}_{2^5,\text{disp}}$.

1. Determine the 32-ary QC-LDPC code given by the null space over GF(2^5) of $\mathbf{H}^{(1)}_{2^5,\text{disp}}(4,16)$.
2. Assume BPSK transmission over the binary-input AWGN channel. Decode the code constructed in (1) using the FFT-QSPA with 50 iterations. Compute its bit-, symbol-, and word-error performances. Compare the word-error performance of the code with that of a shortened RS code of the same length and rate decoded with the HD-BM algorithm.

3. Compute the word-error performances of the code constructed in (1) with 3, 5, 10, and 50 iterations of the FFT-QSPA.

14.7 Consider the 31×31 array $\mathbf{H}^{(1)}_{2^5,\text{disp}}$ constructed in Problem 14.6. Take an 8×16 subarray $\mathbf{H}^{(1)}_{2^5,\text{disp}}(8,16)$ from $\mathbf{H}^{(1)}_{2^5,\text{disp}}$. Use $\mathbf{H}^{(1)}_{2^5,\text{disp}}(8,16)$ as the base array for masking. Construct an 8×16 masking matrix $\mathbf{Z}(8,16)$ with column and row weights 3 and 6, respectively. Mask $\mathbf{H}^{(1)}_{2^5,\text{disp}}(8,16)$ with $\mathbf{Z}(8,16)$ to obtain an 8×16 masked array $\mathbf{M}^{(1)}_{2^5,\text{disp}}(8,16)$ of α-multiplied 31×31 circulant permutation and zero matrices over $\text{GF}(2^5)$.

1. Determine the 32-ary QC-LDPC code given by the null space over $\text{GF}(2^5)$ of the masked array $\mathbf{M}^{(1)}_{2^5,\text{disp}}(8,16)$.
2. Decode the code constructed in (1) with 50 iterations of the FFT-QSPA. Compute its bit-, symbol-, and word-error performances of the code over the binary-input AWGN channel with BPSK signaling.

14.8 Let $\text{GF}(2^6)$ be the field for code construction and let α be a primitive element of $\text{GF}(2^6)$. Let $\mathcal{G}_1$ and $\mathcal{G}_2$ be two additive subgroups of $\text{GF}(2^6)$ spanned by the elements in $\{\alpha^0, \alpha, \alpha^2, \alpha^3\}$ and $\{\alpha^4, \alpha^5\}$, respectively. From these two groups, construct a 4×16 matrix $\mathbf{W}^{(4)}$ over $\text{GF}(2^6)$ using (11.10) that satisfies α-multiplied row constraints 1 and 2. Using $\mathbf{W}^{(4)}$ as the base matrix for dispersion, construct a 4×16 array $\mathbf{H}^{(4)}_{2^6,qc,\text{disp}}$ of 63 α-multiplied 63×63 CPMs and a single zero matrix over $\text{GF}(2^6)$. Take a 4×12 subarray $\mathbf{H}^{(4)}_{2^6,qc,\text{disp}}(4,12)$ from $\mathbf{H}^{(4)}_{2^6,qc,\text{disp}}$, avoiding the zero matrix at the top of the first column of $\mathbf{H}^{(4)}_{2^6,qc,\text{disp}}$.

1. Determine the 64-ary QC-LDPC code given by the null space over $\text{GF}(2^6)$ of $\mathbf{H}^{(4)}_{2^6,qc,\text{disp}}(4,12)$.
2. Assume BPSK transmission over the AWGN channel. Decode the code constructed in (1) with 5 and 50 iterations of the FFT-QSPA. Compute the bit-, symbol-, and word-performances of the code.

14.9 Consider the 15×15 array $\mathbf{H}^{(1)}_{2^4,\text{disp}}$ of α-multiplied 15×15 circulant permutation and zero matrices over $\text{GF}(2^4)$ constructed using the (15,2,30) RS code over $\text{GF}(2^4)$. Choose $k = 2$, $l = 6$, $s = 0$, and $t = 4$. Take a 4×8 subarray $\mathbf{H}^{(1)}_{2^4,\text{disp}}(4,8)$ (avoiding zero matrices on the main diagonal of the array $\mathbf{H}^{(1)}_{2^4,\text{disp}}$). Use $\mathbf{H}^{(1)}_{2^4,\text{disp}}(4,8)$ as the base array for dispersion as described in Sections 13.7 and 14.4.4. The 0-masked six-fold dispersion of $\mathbf{H}^{(1)}_{2^4,\text{disp}}(4,8)$ gives a 24×48 array $\mathbf{H}^{(1)}_{2^4,\text{6-f},\text{disp},0}(24,48)$ of α-multiplied 15×15 circulant permutation and zero matrices over $\text{GF}(2^4)$.

1. Determine the 16-ary QC-LDPC code given by the null space over GF(2^4) of $\mathbf{H}^{(1)}_{2^4,\text{6-f,disp},0}(24, 48)$.
2. Assume BPSK transmission over the AWGN channel. Decode the code constructed in (1) with 50 iterations of the FFT-QSPA. Compute the bit-, symbol-, and word-error performances of the code.
3. What are the erasure-burst-correction capability and efficiency of the code?

14.10 In Problem 14.9, we set now $s = 1$ and all the other parameters remain the same. Determine the QC-LDPC code given by the null space over GF(2^4) of the 1-masked six-fold-dispersed array $\mathbf{H}^{(1)}_{2^4,\text{6-f,disp},1}(24, 48)$. Compute the bit-, symbol-, and word-error performances of the code decoded with 50 iterations of the FFT-QSPA over the AWGN channel with BPSK signaling.

14.11 Assume 16-QAM transmission over the AWGN channel. Compute the symbol- and word-error performances of the code constructed in Problem 14.9 with 50 iterations of the FFT-QSPA.

References

[1] M. C. Davey and D. J. C. MacKay, "Low-density parity check codes over GF(q)," *IEEE Communications Lett.*, vol. 2, no. 6, pp. 165–167, June 1998.

[2] D. J. C. MacKay and M. C. Davey, "Evaluation of Gallager codes of short block length and high rate applications," *Proc. IMA International Conference on Mathematics and Its Applications: Codes, Systems and Graphical Models*, New York, Springer-Verlag, 2000, pp. 113–130.

[3] L. Barnault and D. Declercq, "Fast decoding algorithm for LDPC over GF(2^q)," *Proc. 2003 IEEE Information Theory Workshop*, Paris, March 31–April 4, 2003, pp. 70–73.

[4] D. Declercq and M. Fossorier, "Decoding algorithms for nonbinary LDPC codes over GF(q)," *IEEE Trans. Communications*, vol. 55, no. 4, pp. 633–643, April 2007.

[5] J. I. Boutros, A. Ghaith, and Yi Yun-Wu, "Non-binary adaptive LDPC codes for frequency selective channels: code construction and iterative decoding," *Proc. IEEE Information Theory Workshop*, Chengdu, October 2007, pp. 184–188.

[6] S. Lin, S. Song, B. Zhou, J. Kang, Y. Y. Tai, and Q. Huang, "Algebraic constructions of nonbinary quasi-cyclic LDPC codes: array masking and dispersion," *Proc. 2nd Workshop for the Center of Information Theory and its Applications*, UCSD division of Calit2 and the Jacobs School of Engineering, San Diego, CA, January 29–February 2, 2007.

[7] C. Poulliat, M. Fossorier, and D. Declercq, "Optimization of non binary LDPC codes using their binary images," *Proc. Int. Turbo Code Symp.*, Munich, April 2006.

[8] C. Poulliat, M. Fossorier, and D. Declercq, "Design of non binary LDPC codes using their binary images: algebraic properties," *Proc. IEEE Int. Symp. Information Theory*, Seattle, WA, July 2006.

[9] V. Rathi and R. Urbanke, "Density evolution, thresholds and the stability condition for nonbinary LDPC codes," *IEE Proc. Communications*, vol. 152, no. 6, pp. 1069–1074, December 2005.

[10] S. Song, L. Zeng, S. Lin, and K. Abdel-Ghaffar, "Algebraic constructions of nonbinary quasi-cyclic LDPC codes," in *Proc. IEEE Int. Symp. Information Theory*, Seattle, WA, July 9–14, 2006, pp. 83–87.

[11] A. Voicila, D. Declercq, F. Verdier, M. Fossorier, and P. Urard, "Split non-binary LDPC codes," *Proc. IEEE Int. Symp. Information Theory*, Toronto, Ontario, July 6–11, 2008.

[12] L. Zeng, L. Lan, Y. Y. Tai, and S. Lin, "Dispersed Reed–Solomon codes for iterative decoding and construction of q-ary LDPC codes," in *Proc. IEEE Global Telecommunications Conf.*, St. Louis, MO, November 28–December 2, 2005, pp. 1193–1198.

[13] L. Zeng, L. Lan, Y. Y. Tai, B. Zhou, S. Lin, and K. A. S. Abdel-Ghaffar, "Construction of nonbinary cyclic, quasi-cyclic and regular LDPC codes: a finite geometry approach," *IEEE Trans. Communications*, vol. 56, no. 3, pp. 378–387, March 2008.

[14] L. Zeng, L. Lan, Y. Y. Tai, S. Song, S. Lin, and K. Abdel-Ghaffar, "Constructions of nonbinary quasi-cyclic LDPC codes: a finite field approach," *IEEE Trans. Communications*, vol. 56, no. 4, pp. 545–554 , April 2008.

[15] B. Zhou, J. Kang, Y. Y. Tai, S. Lin, and Z. Ding, "High performance non-binary quasi-cyclic LDPC codes on Euclidean geometries," *IEEE Trans. Communications*, vol. 57, no. 4, pp. 545–554, April 2009.

[16] B. Zhou, J. Kang, S. Song, S. Lin, K. Abdel-Ghaffar, and M. Xu, "Construction of non-binary quasi-cyclic LDPC codes by arrays and array dispersions", *IEEE Trans. Communications*, vol. 57, 2009.

[17] B. Zhou, J. Kang, Y. Y. Tai, Q. Huang, and S. Lin, "High performance nonbinary quasi-cyclic LDPC codes on Euclidean geometries," *Proc. 2007 Military Communications Conference (MILCOM 2007)*, Orlando, FL, October 29–31, 2007.

[18] B. Zhou, Y. Y. Tai, L. Lan, S. Song, L. Zeng, and S. Lin, "Construction of high performance and efficiently encodable nonbinary quasi-cyclic LDPC codes," *Proc. IEEE Global Telecommunications Conf.*, San Francisco, CA, November 27–December 1, 2006.

[19] B. Zhou, L. Zhang, J. Kang, Q. Huang, Y. Y. Tai, and S. Lin, "Non-binary LDPC codes vs. Reed–Solomon codes," *Proc. 3rd Workshop for the Center of Information Theory and its Applications*, UCSD division of Calit2 and the Jacobs School of Engineering, San Diego, CA, January 27–February 1, 2008.

[20] B. Zhou, L. Zhang, Q. Huang, S. Lin, and M. Xu, "Constructions of high performance non-binary quasi-cyclic LDPC codes," *Proc. IEEE Information Theory Workshop (ITW)*, pp. 71–75, Porto, May 5–9, 2008.

[21] B. Zhou, L. Zhang, J. Y. Kang, Q. Huang, S. Lin and K. Abdel-Ghaffar, "Array dispersions of matrices and constructions of quasi-cyclic LDPC codes over non-binary fields," *Proc. IEEE Int. Symp. on Information Theory*, Toronto, Ontario, July 6–11, 2008.

[22] H. Wymeersch, H. Steendam, and M. Moeneclaey, "Log-domain decoding of LDPC codes over GF(q)," in *Proc. IEEE Int. Conf. on Communications*, Paris, June 2004, pp. 772–776.

[23] E. R. Berlekamp, *Algebraic Coding Theory*, New York, McGraw-Hill, 1968; and revised edn., Laguna Hill, CA, Aegean Park Press, 1984.

[24] J. L. Massey, "Shift-register synthesis and BCH decoding," *IEEE Trans. Information Theory*, vol. 15, no. 1, pp. 122–127, January 1969.

[25] R. Kötter and A. Vardy, "Algebraic soft-decision decoding of Reed–Solomon codes," *IEEE Trans. Information Theory*, vol. 49, no. 11, pp. 2809–2825, November 2003.

[26] W. J. Gross, F. R. Kschischang, R. Kötter, and P. G. Gulak, "Applications of algebraic soft-decision decoding of Reed–Solomon codes," *IEEE Trans. Communications*, vol. 54, no. 7, pp. 1224–1234, July 2006.

[27] M. El-Khamy and R. McEliece, "Iterative algebraic soft-decision list decoding of RS codes," *IEEE J. Selected Areas Communications*, vol. 24, no. 3, pp. 481–490, March 2006.

[28] S. Lin and D.J. Costello, Jr., *Error Control Coding: Fundamentals and Applications*, 2nd edn., Upper Saddle River, NJ, Prentice-Hall, 2004.

[29] A. Bennatan and D. Burshtein, "Design and analysis of nonbinary LDPC codes for arbitrary discrete-memoryless channels," *IEEE Trans. Information Theory*, vol. 52, no. 2, pp. 549–583, February 2006.

[30] D. J. C. MacKay, "Good error-correcting codes based on very sparse matrices," *IEEE Trans. Information Theory*, vol. 45, no. 2, pp. 399–432, March 1999.

[31] A. Goupil, M. Colas, G. Gelle, and D. Declercq, "On BP decoding of LDPC codes over groups," *Proc. Int. Symp. Turbo Codes*, Munich, April 2006, CD-ROM.

[32] A. Goupil, M. Colas, G. Gelle, and D. Declercq, "FFT-based BP decoding of general LDPC codes over Abelian groups," *IEEE Trans. Communications*, vol. 55, no. 4, pp. 644–649, April 2007.

[33] Y. Kou, S. Lin, and M. Fossorier, "Low-density parity-check codes based on finite geometries: a rediscovery and new results," *IEEE Trans. Information Theory*, vol. 47, no. 7, pp. 2711–2736, November 2001.

[34] Z. Li, L. Chen, L. Zeng, S. Lin, and W. Fong, "Efficient encoding of quasi-cyclic low-density parity-check codes," *IEEE Trans. Communications*, vol. 54, no. 1, pp. 71–81, January 2006.

[35] X. Li and M.R. Soleymani, "A proof of the Hadamard transform decoding of the belief propagation decoding for LDPCC over GF(q)," *Proc. Fall Vehicle Technol. Conf.*, vol. 4, September 2004, pp. 2518–2519.

[36] H. Song and J. R. Cruz, "Reduced-complexity decoding of Q-ary LDPC codes for magnetic recording," *IEEE Trans. Magnetics*, vol. 39, no. 2, pp. 1081–1087, March 2003.

15 LDPC Code Applications and Advanced Topics

In this chapter we cover some of the more common applications of LDPC codes. The breadth and depth of work reported in the literature on the application of LDPC codes to data transmission and storage would fill many volumes. Thus, this chapter has the modest goal of introducing selected applications. These applications include coded modulation, turbo equalization and code design for intersymbol-interference channels, estimation of the level LDPC error floors, and decoder design for low error floors. We also provide in this chapter a brief introduction to LDPC convolutional codes. Because of their importance, we also include a section on so-called digital fountain codes, which have application to file transfer over networks and closely resemble LDPC codes.

15.1 LDPC-Coded Modulation

Trellis-coded modulation [1], multilevel coding [2, 3], and bit-interleaved coded modulation (BICM) [4, 5], are all well-known bandwidth- and power-efficient coded-modulation schemes. More recently, turbo and LDPC codes have been combined with higher-order modulation schemes. For example, LDPC codes have been designed for multilevel coding in [6] and for MIMO channels in [7]. Near-capacity performance has been demonstrated in these higher-order signaling systems. The system under study is shown in Figure 15.1. We do not consider the multilevel coding format because it is shown in [8] and elsewhere that the single-code format of Figure 15.1 is superior for LDPC codes. Although this is a BICM-like system, because an LDPC encoder is employed, a bit interleaver is not necessary here. An interleaver would be equivalent to permuting the columns in the code's parity-check matrix. The M-ary modulator takes $m = \log_2 M$ code bits at a time and maps them to a complex symbol in accordance with the constellation-labeling scheme. The channel adds a complex white-Gaussian-noise sample, $n = n_I + jn_Q$, to each such symbol, where n_I and n_Q are independent Gaussian random variables with zero mean and variance $N_0/2$.

The demodulator is a maximum *a posteriori* (MAP) *symbol-to-bit metric calculator* [4, 5] and its (soft) output can be derived as follows. Let x_i denote the ith code bit ($i = 0, 1, \ldots, m-1$) of the m-bit signal label $x = (x_0, x_1, \ldots, x_{m-1})$ and let χ_b^i be the subset of signals whose labels have the value $b \in \{0, 1\}$ at the

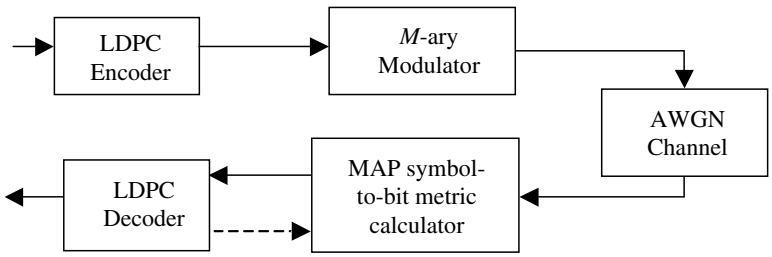


Figure 15.1 A block diagram of an LDPC-coded modulation system.

ith bit position. The output of the metric calculator is the log-likelihood ratio (LLR) of the coded bit x_i given received complex channel sample y, which is

$$\begin{aligned} L(x_i|y) &= \ln\left(\frac{\displaystyle\sum_{s\in\chi_1^i}\exp\left(\frac{-\|y-s\|^2}{2\sigma^2}\right)}{\displaystyle\sum_{s\in\chi_0^i}\exp\left(\frac{-\|y-s\|^2}{2\sigma^2}\right)}\right) \\ &= \max_{s\in\chi_1^i}{}^*\left(\frac{-\|y-s\|^2}{2\sigma^2}\right) - \max_{s\in\chi_0^i}{}^*\left(\frac{-\|y-s\|^2}{2\sigma^2}\right) \\ &= \max_{s\in\chi_1^i}{}^*\left(\frac{y_I s_I + y_Q s_Q}{\sigma^2}\right) - \max_{s\in\chi_0^i}{}^*\left(\frac{y_I s_I + y_Q s_Q}{\sigma^2}\right), \end{aligned} \tag{15.1}$$

where, as defined in earlier chapters,

$$\max^*(x,y) = \max(x,y) + \log(1+\exp(|x-y|)).$$

Also, $y = y_I + jy_Q$ and $s = s_I + js_Q$. The symbol-to-bit metric $L(x_i|y)$ is passed to the LDPC decoder, which employs soft iterative decoding. It is also possible to use iterative decoding/demodulation at the receiver. The dashed arrow in Figure 15.1 represents possible iterations between the LDPC decoder and the symbol-to-bit metric calculator. When Gray labeling is used, there is no advantage to utilizing such feedback because the metric calculator has a horizontal EXIT curve in this case [7].

Gray labeling also yields the greatest capacity. We can examine the performance of Gray versus natural signal point labeling via capacity computations as follows. Assume that the M-ary modulation is modeled by a two-dimensional equiprobable-signal set $A = \{a_i\}_{i=0}^{M-1}$ and each signal corresponds to a binary label of length m. The capacity of M-ary coded modulation is then given [5] by

$$C = M - \frac{1}{2}\sum_{k=0}^{m-1}\sum_{b=0}^{1} E\left\{\log_2\left(\frac{\sum_{i=0}^{M-1} p(y|a_i)}{\sum_{a_j\in A_b^k} p(y|a_j)}\right)\right\}, \tag{15.2}$$

where y represents the received signal and A_b^k represents the subset of signals in A whose labels equal b in the kth bit, $k \in \{0, 1, \ldots, m-1\}$ (bit 0 is the rightmost

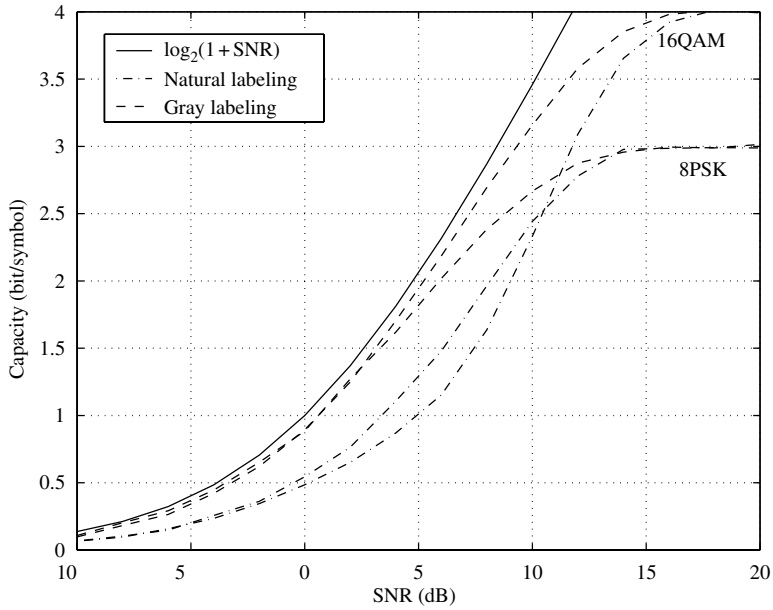


Figure 15.2 The capacity of coded-modulation for 16QAM and 8PSK on the AWGN channel.

bit). In Figure 15.2, the capacities of Gray- and natural-labeled coded modulation for 8PSK and 16QAM are plotted against E_s/N_0, where E_s is the average signal energy. We note that Gray labeling provides a greater capacity than does natural labeling. It is interesting to note that BICM with natural labeling and iterative decoding outperforms BICM with Gray labeling and iterative decoding [9], but this is not the case for LDPC-coded modulation.

15.1.1 Design Based on EXIT Charts

We now describe an LDPC code-design technique from [10] for coded modulation based on EXIT charts. We focus on 8PSK with Gray labeling. Extensions to other modulation formats are straightforward. Consider the Gray-labeled 8PSK signal constellation in Figure 15.3, with labels (000, 001, 011, 010, 110, 111, 101, 100) for the signal points $(s_0, s_1, s_2, s_3, s_4, s_5, s_6, s_7)$, respectively. The regions indicated by the dashed lines and the signal point indices $(0, 1, 2, 3, 4, 5, 6, 7)$ correspond to the decision regions R_i for the signals $(s_0, s_1, s_2, s_3, s_4, s_5, s_6, s_7)$, respectively.

Considering now bit x_0, we take symbols s_0 and s_1 as representatives for analysis. We first draw a line through the origin and the symbol s_0 to partition the signal-space into two half-planes. The neighboring symbols $\{s_1, s_2, s_3\}$ and $\{s_7, s_6, s_5\}$ for s_0 in the two half-planes have x_0 values of $\{1, 1, 0\}$ and $\{0, 1, 1\}$, respectively. The symbol s_4, which is also intersected by the partition line, has an x_0 value of 0. We now draw a line through the origin and symbol s_1 to again

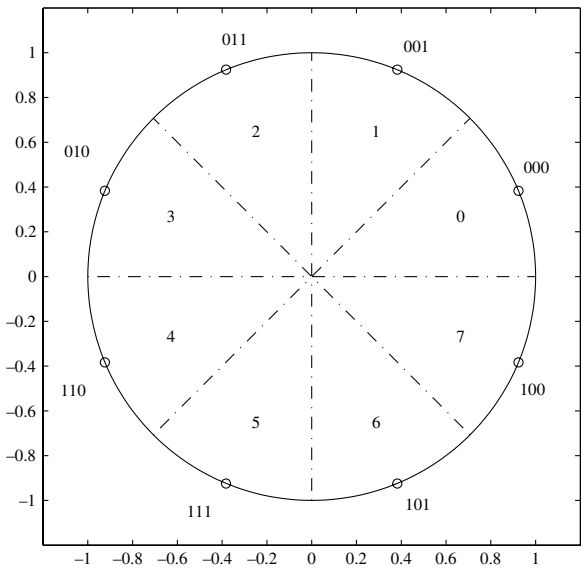


Figure 15.3 The 8PSK constellation with Gray labeling.

partition the signal-space into two half-planes. The neighbors of s_1 in the two half-planes, $\{s_2, s_3, s_4\}$ and $\{s_0, s_7, s_6\}$, have x_0 equal to $\{1, 0, 0\}$ and $\{0, 0, 1\}$, respectively. The other symbol on the partition line, s_5, has x_0 equal to 1. Thus, the x_0-distribution for the neighbors of s_0 is the complement of that for the neighbors of s_1. It is easy to see that the x_0-distributions of the neighbors of s_0,s_2,s_4,s_6 are the same and similarly for the neighbors of s_1,s_3,s_5,s_7. Further, the x_0-distributions of these two sets are complements of each other.

Empirical results indicate that $L(x_0|y = s_i + n)$, $i = 0, 1, \ldots, 7$, have identical (near-)Gaussian distributions up to the signs of their means. The discussion above gives an indication of why this is true. Because they have Gaussian distributions, we can obtain closed-form pdf expressions for $L(x_0|y)$ for each of the cases $L(x_0|y = s_i + n)$, $i = 0, 1, \ldots, 7$.

However, for $L(x_1|y)$ it is difficult to derive closed-form pdfs because $L(x_1|y = s_i + n)$, $i = 0, \ldots, 7$, do not have the same distributions. For example, the x_1-distributions for the half-plane neighbors of s_0 are $\{1, 1, 0\}$ and $\{1, 0, 0\}$, and the x_1-distributions for the half-plane neighbors of s_1 are $\{1, 1, 1\}$ and $\{0, 0, 0\}$. Similarly, $L(x_2|y)$, $L(x_2|y = s_i + n)$, $i = 0, \ldots, 7$, do not have the same distributions. However, due to the symmetry of Gray labeling, for x_1 we can partition the set of symbols into the following two subsets which will lead us to closed-form pdfs:

$$S_1^{\mathrm{n}} = \{s_1, s_2, s_5, s_6\},$$
$$S_1^{\mathrm{f}} = \{s_0, s_3, s_4, s_7\}.$$

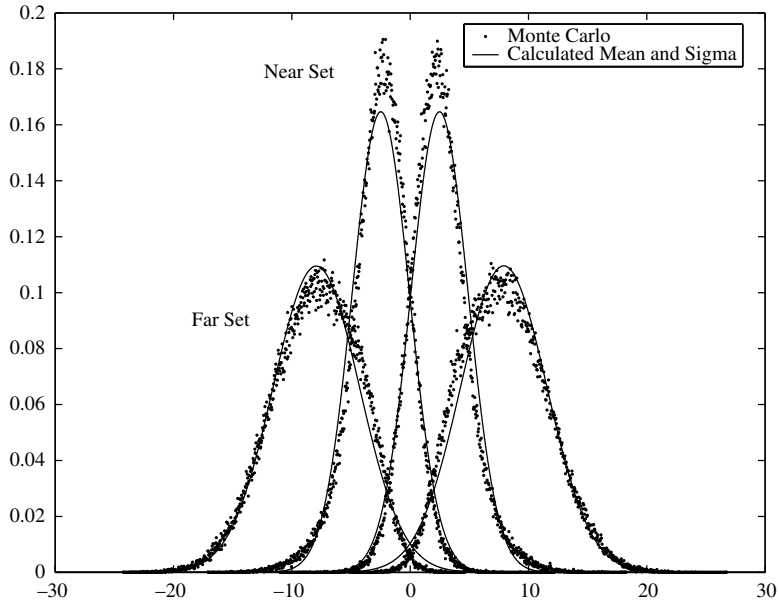


Figure 15.4 LLR distributions for the near and far symbol subsets.

The superscript n indicates that the symbols in the set are "near" the $(x_1 = 1)$-versus-$(x_1 = 0)$ decision boundary and the superscript f refers to symbols "far" from the boundary. Similarly, for x_2, we partition the symbols into the sets

$$S_2^{\mathrm{n}} = \{s_0, s_3, s_4, s_7\},$$
$$S_2^{\mathrm{f}} = \{s_1, s_2, s_5, s_6\}.$$

Since distributions of x_1 (x_2) of the symbols in the subsets with the same superscript are the same (or complements), it can be expected that $L(x_j|y = s_i + n, s_i \in S_j^{\mathrm{f}})$, $i = 0, \ldots, 7$, $j = 1, 2$, have identical Gaussian distributions up to different signs for the means. A similar comment holds for the near-symbols case, $L(x_j|y = s_i + n, s_i \in S_j^{\mathrm{n}})$.

The pdfs (scaled histograms) obtained via Monte Carlo simulations for $L(x_j|y = s_i + n, s_i \in S_j^{\mathrm{f}})$ and $L(x_j|y = s_i + n, s_i \in S_j^{\mathrm{n}})$ with $E_s/N_0 = 3\mathrm{dB}$ are plotted in Figure 15.4 together with the pdfs of Gaussian random variables of the same means and variances. We observe close agreement between the Gaussian pdfs and the simulated data. Such a result provides strong support for the employment of a Gaussian-approximation algorithm in the design of LDPC-coded modulation systems. The means $(\mu_0, \mu_{\mathrm{f}}, \mu_{\mathrm{n}})$ and standard deviations $(\sigma_0, \sigma_{\mathrm{f}}, \sigma_{\mathrm{n}})$ for $L(x_0|y)$, $L(x_j|y = s_i + n, s_i \in S_j^{\mathrm{n}})$ and $L(x_j|y = s_i + n, s_i \in S_j^{\mathrm{f}})$ can easily be estimated. Furthermore, as validated by simulations, the consistency condition $2\mu \approx \sigma^2$ (defined in Chapter 9) holds for those means and variances.

To incorporate the Gaussian approximation just discussed, we define (as in Chapter 9)

$$J(\sigma) = 1 - \int \frac{e^{-((x-\sigma^2/2)^2/2\sigma^2)}}{\sqrt{2\pi}\sigma} \log_2(1+e^{-x})dx, \tag{15.3}$$

which is the mutual information I between a binary random variable v and $y = v + n$, where n is a $\mathcal{N}(\sigma^2/2, \sigma^2)$ Gaussian random variable. Then $J^{-1}(I)$ returns the standard deviation σ of the Gaussian random variable y corresponding to the mutual information I. For our system, 1/3 of the code bits are x_0, 1/3 of the code bits are x_1 (or x_2) among the symbols in S^{f}, and 1/3 of the code bits are x_1 (or x_2) among the symbols in S^{n}. Given this and the EXIT-chart results from Chapter 9, the VN EXIT function $I_{E,V}$ for an irregular LDPC code with VN distribution $\{\lambda_d\}$ is given by

$$I_{E,V}(I_{E,C}) = \sum_{\mu} \sum_{d} \frac{1}{3} \lambda_d J\left(\sqrt{(d-1)[J^{-1}(I_{E,C})]^2 + 2\mu}\right), \tag{15.4}$$

where the first summation is over all $\mu \in \{\mu_0, \mu_{\mathrm{f}}, \mu_{\mathrm{n}}\}$ and where $I_{E,C}$ is the CN EXIT function.

The usual way to obtain the EXIT function for the CNs is to assume that a duality property holds (see Chapter 9). Following [10], we present an alternative route. Consider the situation in which the decoder VN and CN processors are driven by *consistent* Gaussian inputs acting as extrinsic information. Thus, a check-node message will be distributed as $\mathcal{N}\left(\mu^{(\mathrm{c})}, 2\mu^{(\mathrm{c})}\right)$ and have mutual information

$$I_{E,C} = J\left(\sqrt{2\mu^{(\mathrm{c})}}\right) \tag{15.5}$$

where, from Chapter 9,

$$\mu^{(\mathrm{c})} = \sum_{\delta} \rho_\delta \Phi^{-1}\left(1 - \left[1 - \sum_{d=1}^{d_v} \lambda_d \Phi\left(\mu^{(v)}\right)\right]^{\delta-1}\right). \tag{15.6}$$

Here $\{\rho_\delta\}$ represents the CN degree distribution and

$$\mu^{(\mathrm{v})} = \tfrac{1}{2}\left[J^{-1}(I_{E,V})\right]^2, \tag{15.7}$$

as follows from $I_{E,V} = J\left(\sqrt{2\mu^{(\mathrm{v})}}\right)$. Also, in this expression, the function $\Phi(\mu)$ is defined as

$$\Phi(\mu) \triangleq 1 - \frac{1}{\sqrt{4\pi\mu}} \int_{-\infty}^{\infty} \tanh(\tau/2) \exp\left[-(\tau-\mu)^2/(4\mu)\right] d\tau. \tag{15.8}$$

Substitution of (15.5) into (15.4) gives,

$$I_{E,V} = \sum_{\mu} \sum_{d} \frac{1}{3} \lambda_d J\left(\sqrt{(d-1)\left[2\mu^{(c)}\right] + 2\mu}\right). \tag{15.9}$$

Equations (15.5)–(15.9) are the ones used to produce an EXIT chart for the coded-8PSK under discussion.

The EXIT-chart technique is used as a tool to find optimal distributions $\{\lambda_d\}$ and $\{\rho_\delta\}$. Optimality here is in the sense of minimizing the mean-squared distance between the $I_{E,V}$ and $I_{E,C}$ transfer curves. This may be formulated as minimizing the mean-squared error with error defined as

$$e = I_{E,V}(I_A, \mathrm{SNR}) - I_{E,C}^{-1}(I_A),$$

subject to the constraint $e > 0$ [11]. In this expression, I_A represents the *a priori* information, or the abscissa value in an EXIT chart. Taking the mean-squared error as the cost function, differential evolution [12] is used to optimize the degree distributions. At the same time, the EXIT charts yield the decoding thresholds for intermediate degree distributions.

An 8PSK coded-modulation scheme with a throughput of 2 bits/use was designed using the above EXIT-chart technique. The check-node degree was fixed to 14 (so that $\rho(X) = X^{13}$) and a range of variable-node degrees was allowed, with a maximum degree of 30. The optimum rate-2/3 code we have designed has variable-node distribution

$$\begin{aligned}\lambda(X) = {} & 0.187\,785X + 0.194\,142X^2 + 0.037\,685X^3 + 0.195\,438X^6 + 0.014\,284X^7 \\ & + 0.004\,84X^{14} + 0.195\,438X^{15} + 0.014\,284X^{28} + 0.126\,85X^{29},\end{aligned}$$

and has a decoding threshold of SNR 3.2 dB on the Gray-labeled 8PSK AWGN channel. Compared with the capacity of 2.85 dB in Figure 15.2, there is a gap of 0.35 dB. The EXIT chart of this code at $E_b/N_0 = 3.21$ dB is plotted in Figure 15.5. Note that the two curves match very well. An open tunnel still exists between the two curves to permit iterative decoding convergence. A ($24K$,$16K$) code ($K = 1024$) based on the above parameters has been constructed and the simulation result is shown in Figure 15.6. Note that this coded-modulation scheme operates about 0.6 dB from the capacity limit at a bit error rate of 10^{-6}.

For moderate-length codes, some additional design measures might be helpful. As is well known, different variable-node degrees imply different reliabilities after decoding: the larger the degree, the higher the average reliability. For M-ary modulation, we transmit m bits, $(x_{m-1}, \ldots, x_1, x_0)$, in different levels (or "bit planes"), where x_0 is in the least-significant-bit (LSB) level and x_{m-1} is in the most-significant-bit (MSB) level. Bits transmitted at different levels are protected differently. The LSB level has the weakest protection and the MSB level has the strongest protection. On the basis of this knowledge, a bit-reliability-mapping strategy was proposed in [13]. In the bit-reliability-mapping strategy, the less

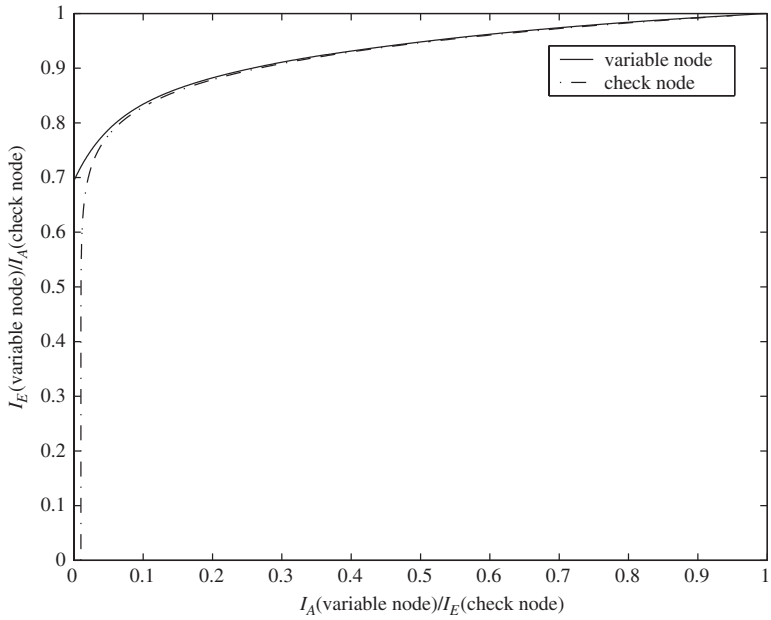


Figure 15.5 The EXIT chart for a rate-2/3 LDPC-coded 8PSK ensemble.

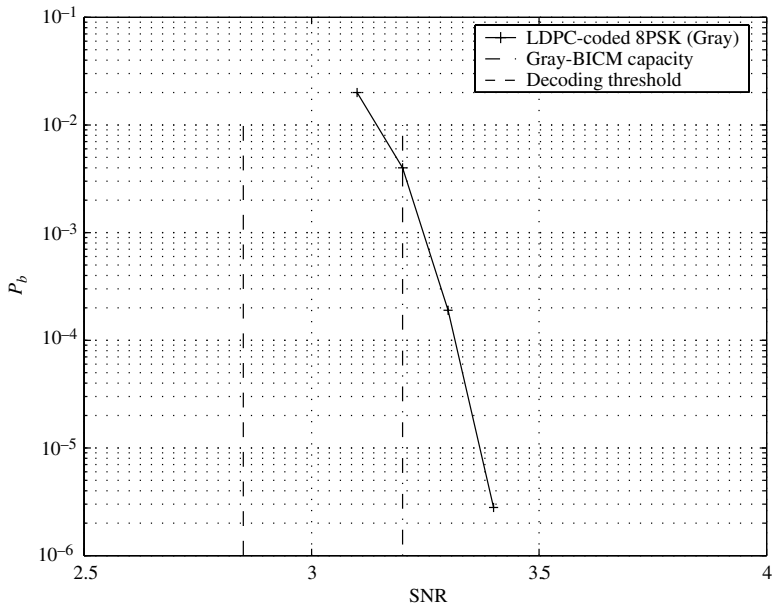


Figure 15.6 The performance of $(24K,16K)$ LDPC-coded 8PSK. The information rate is 2 bits/channel symbol ($K = 1024$).

reliable LDPC code bits are mapped to the lower-level modulation bits and the more reliable bits to the higher-level bits. Doing so can result in an improvement of 0.2 dB for moderate-length codes. For additional information on LDPC-coded modulation, see [7, 14].

15.2 Turbo Equalization and LDPC Code Design for ISI Channels

15.2.1 Turbo Equalization

The theory and practice of equalization for intersymbol-interference (ISI) channels has a long history (see [15, 16]). Early work utilized the so-called zero-forcing (ZF) equalizer which forces ISI to zero without regard to noise enhancement. The ZF equalizer was followed by the minimum-mean-squared-error (MMSE) equalizer, which accounts for the presence of noise. Then came their adaptive versions, driven by the least-mean-square (LMS) algorithm. Also early on was the invention of the decision-feedback equalizer (DFE), both ZF and MMSE versions. Shortly thereafter it was recognized that there were certain performance and implementation advantages to fractionally spaced equalizers.

However, none of these techniques are optimal in the sense of minimizing the probability of error. Forney's seminal paper [17] taught the community that the optimal "equalizer," in the sense of minimizing the probability of sequence errors, is the maximum-likelihood sequence detector (MLSD), which could be implemented by the Viterbi algorithm. (Strictly speaking, with the MLSD, the channel is not being equalized; rather, the equalizer is replaced by a sequence detector.) Following Forney's paper was the now-famous BCJR paper [18] which introduced the maximum *a posteriori* (MAP) bit-wise detection algorithm, an algorithm that is optimum in the sense of minimizing the bit error probability.

The BCJR algorithm languished for almost 20 years until the advent of turbo codes in 1993, which shortly thereafter inspired the invention called *turbo equalization* [19]. Prior to the invention of turbo equalization, for complexity reasons, there was limited interaction between the equalizer and the decoder in receiver designs (one exception is [20]). In turbo equalization, the ISI channel is treated as an inner code in a serial-concatenated system. Thus, just as turbo decoding refers to two (or more) soft-output decoders working iteratively to obtain near-optimum decisions on the transmitted bits, turbo equalization refers to a soft-output *detector* working iteratively with a soft-output *decoder* to obtain near-optimum decisions on the transmitted bits. In contrast with the "Viterbi equalizer," which conventionally produces hard outputs, the "BCJR equalizer" naturally has available soft outputs in the form of the LLRs computed by the BCJR algorithm. The BCJR algorithm for an ISI channel is much like that for a convolutional code (discussed in Chapter 4). One difference is that the trellis for the ISI channel has only a single channel output for each trellis branch, whereas a rate-$k/(k+1)$

convolutional code has $k+1$ bit labels per trellis branch. Another difference is that the ISI output is nonbinary.

An alternative to single-carrier signaling over a frequency-selective (i.e., ISI) channel combined with turbo equalization is multi-carrier signaling with no equalization. Multi-carrier signaling is called *OFDM signaling* in practice (orthogonal frequency-division multiplexing). The idea is that, whereas a wideband single-carrier signal would experience non-uniform attenuation as a function of frequency, multiple narrowband carriers occupying the same overall frequency span as the wideband signal would each (approximately) experience uniform (flat) attenuation across their individual (narrow) frequency bands. Thus, there would be no need for equalization for each of the individual channels. Further, each carrier would convey one or more code bits per modulation symbol, with each codeword typically spanning one or more *OFDM symbols* (an OFDM symbol is a block of carriers with individual modulation formats). Much like the coded-modulation situation of the previous section, the OFDM receiver can then provide (bit-wise) likelihood information to the channel decoder. Because of the power of today's digital signal-processing chips, OFDM has become ubiquitous in wideband signaling on frequency-selective channels.

In view of the foregoing, it appears that OFDM has rendered turbo equalization obsolete. However, there exist several "communication channels" for which the OFDM solution is not possible. Most prominent among these are magnetic and optical data storage for which no carriers exist, that is, only baseband signaling is utilized. Further, because ever-higher data rates and storage densities are sought, the bits in modern storage devices undergo intersymbol interference, the bandwidth limitation being a function of the physical characteristics of the storage channels. For the past two decades, the state-of-the-art receiver in magnetic disk and tape drives involved "shaping" (equalizing) the channel to some convenient ISI format (called *partial response* (PR)). Then a Viterbi detector matched to the partial response would provide its bit decisions to the Reed–Solomon decoder.

More recently, magnetic-disk-drive manufacturers have moved toward LDPC codes on PR-equalized channels with turbo equalization. A highly simplified block diagram of the magnetic recording "channel'' that has been equalized to some partial-response target $f(D)$ is presented in Figure 15.7. For inductive (differentiating) playback heads, the partial-response polynomial $f(D)$ is typically of the form $(1-D)(1+D)^n$. More recently, (non-differentiating) magneto-resistive heads have been employed, for which $f(D)$ of the form $(1+D)^n$ is typical.

The details of the turbo-equalization receiver of Figure 15.7 are presented in Figure 15.8. Observe that the LDPC-coded PR channel of Figure 15.7 can be deemed as a serial concatenation of the code and the PR channel, with the channel acting as a rate-1 inner code. Thus, the receiver of Figure 15.8 has the same form as the iterative decoder for serial-concatenated codes as presented in Chapter 7. Observe in Figure 15.8 the presence of the subtractors whose role is to ensure that only extrinsic information is sent from the detector to the decoder,

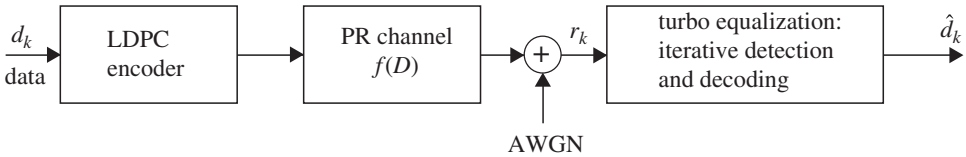


Figure 15.7 A block diagram of an LDPC-coded partial-response channel with turbo equalization.

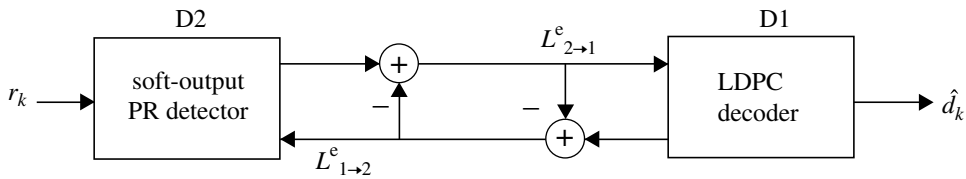


Figure 15.8 Details of the turbo-equalization receiver that appears in Figure 15.7.

and vice versa. This is consistent with the turbo principle presented in Chapter 5. The extrinsic information $L^{e}_{1\to2}$ from the decoder to the detector is scaled by $u_k/2$ and then added to the metric, $-0.5\|r_k - s_k\|^2/\sigma^2$, computed from the channel sample r_k. (s_k is the noiseless channel value, $u_K E\{\pm1\}$ is the channel input corresponding to S_K, σ^2 is the variance of the AWGN sample.) The extrinsic information $L^{e}_{2\to1}$ from the detector to the decoder is used directly as the iterative LDPC decoder input.

Note that interleavers and de-interleavers on the transmit and receive sides are unnecessary because interleaving amounts to a re-ordering of the columns of the **H** matrix of an LDPC code. Thus, turbo equalization is essentially identical to the iterative decoding of serial-concatenated codes except that, for the case of turbo equalization with LDPC codes, no interleaving is necessary. The original papers on coding for PR channels [21–25] utilized convolutional codes and turbo codes, for which interleavers are necessary.

Beside the absence of interleavers, another difference from the serial-concatenated code setup of Chapter 7 is that here the outer code, the LDPC code, is an iteratively decodable code. Thus, the turbo-equalization schedule can be modified so that, after each detector iteration, the LDPC decoder iterates $I > 1$ times. Such a modification generally leads to a performance gain, although performance does not increase with increasing I: an optimal I appears to exist.

Example 15.1. Consider rate-0.9 LDPC coding on the $1 + D$ partial-response (PR1) channel with AWGN, a BCJR PR1 detector, and an SPA LDPC decoder. While the

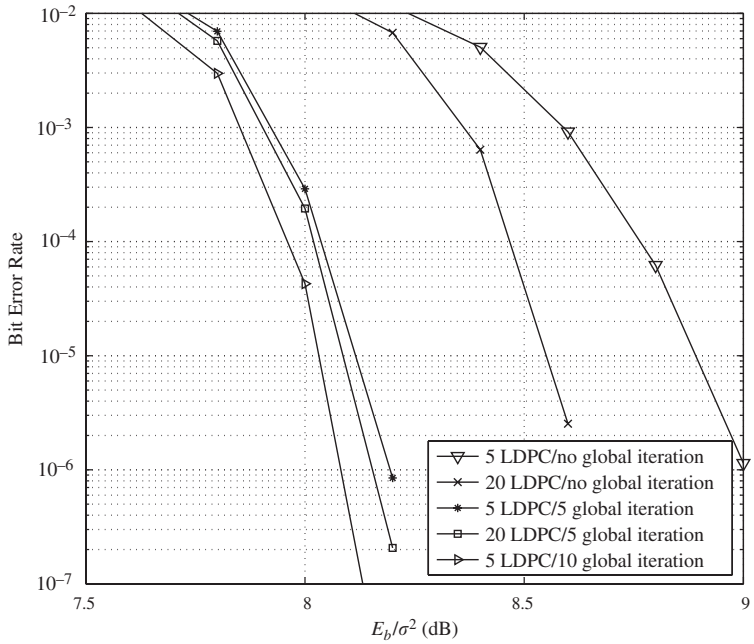


Figure 15.9 The performance of the rate-0.9 (18176,16384) IRA code on the PR1/AWGN channel with turbo equalization.

disk-drive sector size (the minimum file size) has been 512 bytes (4096 bits) for many years, state-of-the-art disk drives have a sector size equal to 4096 bytes (32 768 bits). As a result, a rate-0.9 (36352,32768) LDPC code might be appropriate. However, we instead examine a rate-0.9 (18176,16384) LDPC code for its lower complexity. This code is an IRA code with 16 384 weight-5 columns in the systematic portion of its $\mathbf{H}$ matrix.

Its performance is plotted in Figure 15.9 against the SNR measure E_b/σ^2 for various scenarios. In the ratio E_b/σ^2, E_b is the average information bit energy and σ^2 is the variance of each AWGN sample. The two curves on the right correspond to the situation when there is no feedback from the LDPC decoder to the BCJR detector. It can be seen that the loss due to there being no feedback is on the order of 0.5 dB. The three curves on the left correspond to the following situations: (1) $I = 5$ decoder iterations after each detector iteration, with a total of $I_g = 5$ detector (or *global*) iterations; (2) $I = 20$ and $I_g = 5$; and (3) $I = 5$ and $I_g = 10$. So we see that, in this case, I and I_g can be traded off to balance performance and latency. Observe that case (2) corresponds to $I \cdot I_g = 100$ overall LDPC decoder iterations, whereas case (3) corresponds to $I \cdot I_g = 50$ overall decoder iterations, but the latter case gives superior performance.

We remark that a rate-0.9 (1518,1366) Reed–Solomon code with 12-bit code symbols achieves BER $= 10^{-6}$ at $E_b/\sigma^2 = 10.1$ dB, a gap of about 2 dB from the $(I, I_g) = (5, 10)$ case. Also, the i.i.d. uniform capacity limit at rate 0.9 for the PR1/AWGN channel is $E_b/\sigma^2 = 6.9$ dB, a gap of about 1.2 dB from the $(I, I_g) = (5, 10)$ case at BER $= 10^{-6}$.

The discussion in the example above mentions the use of a BCJR detector as the soft-input/soft-output (SISO) detector for the channel. Other alternatives are possible. The first alternative considered in the literature was the soft-output Viterbi algorithm [20], which later saw improvements in [26, 27]. Other soft-output equalization approaches include decision-aided equalization [28], linear MMSE equalization [29, 30], soft-feedback equalization [31], and those in [32]. Also, a detector/decoder schedule that is an alternative to the one described in the example is proposed in [33], where improved performance is demonstrated.

15.2.2 LDPC Code Design for ISI Channels

The example above used an IRA code that was designed for good performance on the AWGN channel, particularly in the floor region. If one were interested in operating closer to the capacity limit of a PR or ISI channel, then a (long) LDPC code that had optimal degree distributions would have to be designed. These degree distributions could be obtained using the density-evolution approach discussed in [34, 35]. However, because this approach is numerically very complex, an approach based on EXIT charts [36] is recommended since it requires much lower complexity. The computation of an EXIT chart for an LDPC-coded ISI channel is nearly identical to that for a serial-concatenated code. As motivated by the area property of EXIT charts (see Section 9.8), given the ISI channel and its EXIT curve, the LDPC code's degree distributions should be chosen so that the shape of its EXIT curve matches that of the ISI channel. That is, as discussed in Chapter 9, the LDPC code-design problem essentially becomes a curve-fitting problem. We now present the EXIT-chart approach used in [37, 38], which was adapted from [36]. We do not present the optimization (curve-fitting) technique, however. Such techniques may be found in [36, 39].

Let the ISI channel be modeled by the polynomial $f(D) = f_0 + f_1 D + \cdots + f_L D^L$ so that the channel output is given by

$$r_k = x_k * f_k + n_k,$$

where $x_k \in \{\pm 1\}$ are the channel inputs and $n_k \sim \mathcal{N}(0, N_0/2)$ are the AWGN samples. For convenience, we write E_1 for the extrinsic information at the LDPC decoder input ($L^{\mathrm{e}}_{2\to 1}$ above) and E_2 for the extrinsic information at the detector input ($L^{\mathrm{e}}_{1\to 2}$ above). Further, the assumption is that the LDPC code is very long and its decoder is allowed to converge (via a large number of iterations) before it transmits its extrinsic information to the soft-in/soft-out channel detector.

The EXIT curve for the SISO detector for the ISI channel is obtained via Monte Carlo simulations as follows. The detector receives the simulated channel output samples r_k as in standard simulations. In addition, it receives extrinsic information $E_{2,k}$ from the decoder, which, using the consistent Gaussian assumption, is modeled as Gaussian with mean $\pm\sigma_2^2/2$ and variance σ_2^2. The sign of the mean depends on the true sign of the bit of interest, so the mean of the extrinsic information corresponding to x_k is $x_k\sigma_2^2/2$. Then, with the two inputs, r_k and

$E_{2,k}$, the detector is simulated to empirically obtain the conditional pdfs for the extrinsic information outputs $E_{1,k}$, denoted by $p(E_1|x)$. (Note that $E_{1,k}$ is a stationary process.) From this, we may compute the mutual information corresponding to $\{E_{1,k}\}$ according to

$$I_{E1} = \frac{1}{2} \sum_{x \in \{\pm 1\}} p(E_1|x) \int_{-\infty}^{\infty} \log_2 \left(\frac{2p(E_1|x)}{p(E_1|x=+1) + p(E_1|x=-1)} \right) dE_1.$$

Observe that, in addition to being a function of the noise variance, I_{E1} is a function of the extrinsic information variance σ_2^2. However, the EXIT curve of interest plots I_{E1} against I_{E2}, the mutual information corresponding to E_2. This is easily accommodated via the relationship $I_{E2} = J(\sigma_2)$, where $J(\cdot)$ is defined in (15.3).

To obtain the I_{E2} versus I_{E1} EXIT curve for the LDPC decoder, we first model its only input, E_1, as a consistent Gaussian r.v. with mean $\sigma_1^2/2$ and variance σ_1^2, where σ_1 is computed from $\sigma_1 = J^{-1}(I_{E1})$. I_{E2} may then be computed in a fashion analogous to the I_{E1} computation, but matters can be simplified by using the Gaussian-approximation density-evolution results of [39] and Chapter 9. In this case, for a given set of degree distributions $\{\lambda_d\}$ and $\{\rho_\delta\}$, we may write

$$I_{E2} = \sum_d \lambda_d J\left(\sqrt{d \cdot 2\mu^{(c)}}\right),$$

where $\mu^{(c)}$ is computed recursively from (see Section 9.4.2)

$$\mu_\ell^{(c)} = \sum_{\delta=1}^{d_c} \rho_\delta \Phi^{-1}\left(1 - \left[1 - \sum_{d=1}^{d_v} \lambda_d \Phi\left(\sigma_1^2/2 + (d-1)\mu_{\ell-1}^{(c)}\right)\right]^{\delta-1}\right), \tag{15.10}$$

where

$$\Phi(\mu) \triangleq 1 - \frac{1}{\sqrt{4\pi\mu}} \int_{-\infty}^{\infty} \tanh(\tau/2) \exp\left[-(\tau-\mu)^2/(4\mu)\right] d\tau.$$

The recursion (15.10) is initialized with $\sigma_1 = J^{-1}(I_{E1})$ and $\mu_0^{(c)} = 0$.

The previous two paragraphs detail the computation of the I_{E1} versus I_{E2} EXIT curve for the detector and the I_{E2} versus I_{E1} EXIT curve for the LDPC decoder. For a given SNR and set of degree distributions, the latter curve (for the *outer* LDPC code) is plotted on transposed axes together with the former curve, which is plotted in the standard way. If the curves intersect, the SNR must be increased; if they do not intersect, the SNR must be decreased; the SNR at which they just touch is the decoding threshold. After each determination of a decoding threshold, an outer optimization algorithm chooses the next set of degree distributions until an optimum threshold has been attained. However, the threshold can be determined quickly and automatically without actually plotting the two EXIT curves, thus allowing the programmer to act as the outer optimizer.

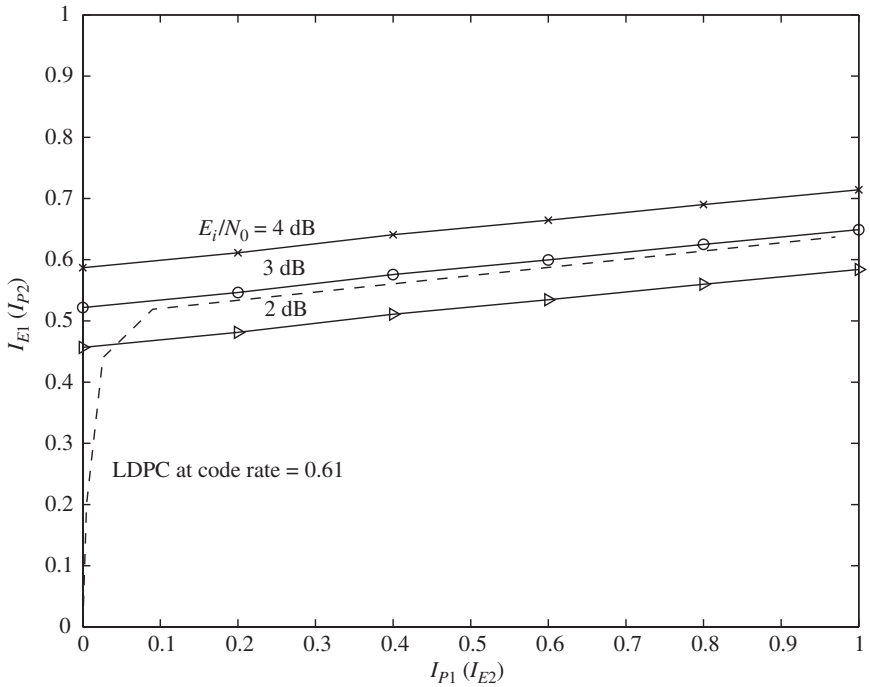


Figure 15.10 An example EXIT chart for Lorentzian channel density $S_c = 1/3$. Dashed line: EXIT curve for optimized LDPC code degree distributions at a code rate of 0.61. Solid lines: EXIT curve for the channel BCJR detector.

Example 15.2. We consider designing an LDPC code for a magnetic-recording channel modeled by the step response

$$s(t) = \sqrt{\frac{4E_i}{\pi\, pw_{50}}}\left(\frac{1}{1 + (2t/(pw_{50}))^2}\right),$$

where pw_{50} is the width of this "isolated" pulse measured at half of its height and E_i is its energy. We assume AWGN with two-sided power-spectral density $N_0/2$ and define the SNR to be E_i/N_0. The recording density on this channel model is summarized in the ratio $S_c = pw_{50}/T$, where T is the bit duration of the bipolar waveform being recorded. In [37, 38] it was determined that the discrete-time equivalent model for $S_c = 1/3$ is $f(D) = 1 - 0.878D - 0.0695D^2 - 0.0134D^3 - 0.0044D^4$. The EXIT chart for this channel for a code rate of $R = 0.61$ and for selected SNR values is displayed in Figure 15.10. The subscript "P" in the axis labels refer to *a priori* information: the extrinsic output of one soft-output processor is used as *a priori* information in the counterpart processor. It is observed in Figure 15.10 that the decoding threshold is $(E_i/N_0)^* \simeq 3\,\text{dB}$. The i.i.d. uniform information rate for this setup is 0.64 [37, 38], so the decoding threshold is very close to the limit.

15.3 Estimation of LDPC Error Floors

The error-rate-floor phenomenon for LDPC codes is perhaps the most significant impediment to their use in many communication and storage systems. This phenomenon can be characterized as an abrupt decrease in the slope of a code's performance curve from the moderate-SNR waterfall region to the high-SNR floor region. Solving the error-floor problem has been a critical issue during the past decade. Investigating the error floors of practical LDPC codes is very difficult because software simulations at very low error rates often take months to run. In this section, we discuss the primary cause of error floors and present a semi-analytic technique for estimating the level of the floor of a given LDPC code with iterative decoding. The technique is due to Richardson [40] and the presentation in this section is adapted from [41]. The standard sum–product-algorithm decoder (with the flooding schedule) is assumed throughout this section. The subsequent section presents three classes of decoder enhancements that vastly reduce the level of the floor.

15.3.1 The Error-Floor Phenomenon and Trapping Sets

The error-floor phenomenon was first noticed by MacKay and Postol in [42], who pointed out a "weakness" of the (2640,1320) Margulis code, namely, the floor. This floor was attributed to the existence of near-codewords: a (w,v) *near-codeword* is a weight-w, length-n binary word resulting in v unsatisfied checks. Following [42], Richardson [40] introduced the notion of *trapping sets* to describe configurations in the Tanner graphs of codes that cause failures of specific decoding schemes: a (w,v) *trapping set* is a set of w variable nodes (VNs) that induces a subgraph with v odd-degree check nodes (CNs) (i.e., v unsatisfied checks) and an arbitrary number of even-degree CNs. Note that the support of a (w,v) near-codeword is exactly a (w,v) trapping set. These researchers discovered that the performance of LDPC codes with message-passing decoding in the low-error-rate region is determined by the multiplicity and structure of the low-weight near-codewords, or trapping sets, rather than the minimum distance of the code. This is because iterative decoders are frequently led by certain noise patterns into trapping-set situations from which they are unable to escape.

An *elementary* (w,v) *trapping set* is a trapping set for which all check nodes in the induced subgraph have either degree 1 or degree 2 [56]. From computer simulations reported in the literature, it was observed that most trapping sets that exert a strong influence on the error-floor level are of the elementary form. Furthermore, although CNs of odd degree larger than 1 are possible, they seem to be very unlikely within the small trapping sets that dominate the floor. This fact will be used to design low-error-floor decoders in the next section.

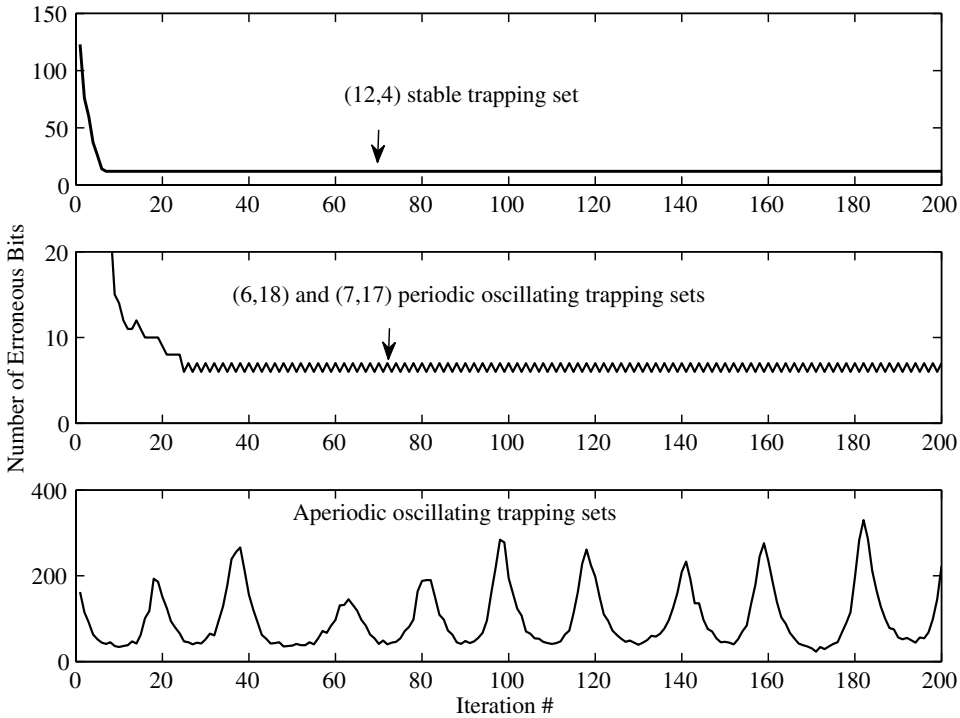


Figure 15.11 The Margulis-code trapping-set behavior as a function of the number of iterations.

Trapping sets can be classified into one of three classes according to their behavior as follows.

1. A *stable trapping set* (also called a *fixed-point trapping set*) is a trapping set responsible for all erroneous decisions at the end of the decoding process, and from which the decoder cannot escape with additional iterations.
2. In a *periodically oscillating trapping-set* situation, the decoder periodically cycles through a number of different error patterns as decoding iterations continue.
3. An *aperiodic oscillating trapping set* contains error patterns from which erroneous messages propagate to a number of external variables and exhibit a random pattern.

The behavior of each class of trapping set is demonstrated in Figure 15.11 for selected Margulis-code trapping sets. Stable trapping sets can be described as absorbing states, which are the most harmful ones to the error-floor performance because they are more likely to cause decoding failure in the high-SNR region. Oscillating trapping-set behavior is attributed to the dynamics of the message exchange in which a few variable nodes (VNs) propagate incorrect messages through their neighboring unsatisfied checks. As a result, these make some of their other neighboring VNs admit incorrect values, which are propagated further to

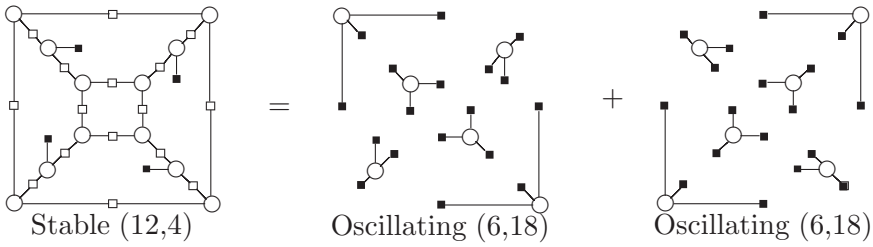


Figure 15.12 An example of oscillating periodic trapping sets of the Margulis code. ◯ = VNs. □ = mis-satisfied CNs. ■ = unsatisfied CNs.

additional VNs. As the number of incorrect VNs increases, so does the number of neighboring checks. Eventually there will be a sufficient number of unsatisfied checks to support the correct values so that the number of erroneous VNs starts to decrease until a certain point at which the incorrect messages of a small number of VNs start to propagate again. Simulations have indicated that some stable trapping sets may exhibit oscillating behavior. For example, as depicted in Figure 15.12, the fixed-point (12,4) trapping sets of the Margulis code can show up as two oscillating (6,18) trapping sets with period 2. Their variable nodes are disjoint and the union of the two (6,18) trapping sets form the (12,4) trapping set. Such periodic trapping sets are also important since they can be viewed as alternative appearances of stable trapping sets.

As we will see in the next subsection, in general the estimation of the floor level due to trapping sets requires full knowledge of the trapping sets that dominate the floor. This is something of a chicken/egg problem insofar as we want to estimate the floor level because it takes a long time to simulate in the floor region, but simulations are required in order to find the trapping sets necessary to estimate the floor level. Further, trapping sets depend not only on the graphical structure of the code, but also on the channel, the decoding algorithm, and decoder implementation details (e.g., the number of quantization levels). Still, there are techniques available to accelerate the search for trapping sets.

The existing methods in the literature for finding trapping sets can be divided into two categories. One uses graph searching and heuristics, as in [40,43–45], whereby trapping sets can be identified using software or FPGA hardware simulations. The other category employs an importance-sampling procedure that biases the noise in such a way that simulation times are vastly reduced [47, 48]. Standard Monte Carlo simulation is in principle the most straightforward and thorough approach, but it is too slow in practice, so trapping sets of interest generally cannot be found in a reasonable amount of time in this way. But, since trapping-set-induced subgraphs typically consist of certain combinations of the shortest cycles of a code, some graph-search algorithms are usually used together with simulations for such enumeration tasks. This is especially true for the quasi-cyclic LDPC (QC-LDPC) codes which lead to isomorphic trapping sets (due to the

presence of circulants), thus simplifying the graph-search algorithms. For example, in [44, 45] several dominant trapping sets were first observed through simulations, and then the code's Tanner graph was searched for these known configurations and from this the multiplicities of these trapping sets were obtained.

In 2002, an error-impulse method was proposed in [49] for computing a minimum distance based on the ability of the soft-input decoder to overcome error-impulse input patterns. Hu *et al.* [50] proposed a similar algorithm to tackle the minimum-distance problem, which searches codewords locally around the all-zeros codeword perturbed by noise, anticipating that, when the decoder errs, it will chose a codeword that is the minimum distance from the all-zeros codeword. These methods can be generalized to the case of a trapping-set search. For example, the trapping-set-search algorithm in [47] combines a graph-search technique with importance-sampling simulations. However, the effectiveness of this technique for high-rate practical-length codes that have a much larger number of shortest cycles associated with each bit is diminished because the set of error impulses is so large that the iterative decoder is overwhelmed and converges to random, long error events. Another method using a similar principle was proposed by Cole *et al.* in [48]. It is a multi-bit, multilevel impulse method constituting an exhaustive search procedure that tests a large number of impulses at each VN. The discussion of these algorithms is beyond our scope (see [41, 45] and the references therein). We now discuss the floor-estimation method.

15.3.2 Error-Floor Estimation

In [40], Richardson proposed a semi-analytic method to predict the performance of LDPC codes in the floor region. In his method, it is assumed that a nearly complete list $\mathcal{T}$ of the dominant trapping-set candidates is found first, which can be achieved by using the methods described in the references cited in the previous two paragraphs. The frame-error rate can be expressed as

$$\mathrm{FER} = \Pr\left\{ \bigcup_{\text{all } T} \xi_T \right\} = \sum_{\text{all } T} \Pr\{\xi_T\} \geq \sum_{T \in \mathcal{T}} \Pr\{\xi_T\}, \tag{15.11}$$

where ξ_T denotes the set of decoder inputs that cause a failure of a particular trapping set T, i.e., all VNs in T are in error and no VNs outside T are in error. Note that $\bigcup_T \xi_T$ represents all decoder inputs that cause a decoding failure, and the second equality of (15.11) is due to the fact that the decoder input space is partitioned into disjoint regions. In the very-high-SNR region, most of the error events are due to trapping-set error events. For example, for the Margulis code, two trapping-set classes contribute to 98% of the error-floor performance. Therefore, as will be demonstrated, the lower bound is tight using these two trapping sets and can be used to predict the error-floor level. Here, a *trapping-set class* is defined as a set of isomorphic trapping sets that contribute equally to the error-floor performance.

The contribution of each trapping-set class to the FER is evaluated by using an importance-sampling method [47, 48, 51] that biases the received channel samples (decoder inputs) in a manner that produces error events much more frequently, thus facilitating error-rate measurement. The biased measurement $\Pr\{\xi_T|\mathcal{B}\}$ is converted to the unbiased measurement $\Pr\{\xi_T\}$ according to

$$\begin{aligned}\Pr\{\xi_T\} &= E_{\mathcal{B}}[\Pr\{\xi_T|\mathcal{B}\}] \\ &= \int_{-\infty}^{\infty} \Pr\{\xi_T|\mathcal{B}=b\} f(b) db, \end{aligned} \tag{15.12}$$

where $\mathcal{B}$ is the random variable with pdf $f(b)$ that controls noise biasing, $E_{\mathcal{B}}$ represents the expectation over $\mathcal{B}$, and the conditional error rate $\Pr\{\xi_T|\mathcal{B}=b\}$ is obtained by simulations. Since $\Pr\{\xi_T|\mathcal{B}=b\} \gg \Pr\{\xi_T\}$, the conditional error rate is more easily measured, thus giving rise to a vast increase in simulation speed.

The above now begs the following question: How does one bias the noise to attain this vast increase in simulation speed? We present this approach for a length-n LDPC code on the binary-input AWGN channel. Suppose the channel noise sequence is denoted by $z_1, z_2, \ldots, z_n$, which are i.i.d. $\mathcal{N}(0, \sigma^2)$ Gaussian random variables, and the channel inputs are equiprobable binary (± 1) sequences. Without loss of generality, we can assume that the all-zeros codeword is transmitted. Then the scaled channel outputs are given by $y_i = (2/\sigma^2)(1 + z_i)$, $i = 1, \ldots, n$. Consider a trapping set T with w variable nodes having bit indices $I_T = (i_1, i_2, \ldots, i_w)$, with the complementary set $\bar{T}$ having indices denoted by $I_{\bar{T}}$. Then partition the n-tuple noise vector into two subvectors: trapping-set-related noise, $\mathbf{z}_T = (z_i, i \in I_T)$, and trapping-set-unrelated noise, $\mathbf{z}_{\bar{T}} = (z_i, i \notin I_T)$. Therefore, the probability of a failure due to this trapping set can be written as

$$\begin{aligned}\Pr\{\xi_T\} &= \iint_{-\infty}^{\infty} \Pr\{\xi_T|z_1 \cdots z_n\}\Pr\{z_1 \cdots z_n\} dz_1 \cdots dz_n \\ &\approx \sum_{\mathbf{z}_T} \sum_{\mathbf{z}_{\bar{T}}} \Pr\{\xi_T|\mathbf{z}_T, \mathbf{z}_{\bar{T}}\}\Pr\{\mathbf{z}_T\}\Pr\{\mathbf{z}_{\bar{T}}\} \\ &= \int \sum_{\mathbf{z}_{\bar{T}}} \Pr\{\mathbf{z}_{\bar{T}}|\mathcal{S}\} \sum_{\mathbf{z}_T} \Pr\{\xi_T|\mathbf{z}_T, \mathbf{z}_{\bar{T}}, \mathcal{S}\}\Pr\{\mathbf{z}_T|\mathcal{S}\} p(\mathcal{S}) d\mathcal{S} \\ &= \sum_{\mathbf{z}_{\bar{T}}} \Pr\{\mathbf{z}_{\bar{T}}\} \int \Pr\{\xi_T|\mathbf{z}_{\bar{T}}, \mathcal{S}\} p(\mathcal{S}) d\mathcal{S}, \end{aligned} \tag{15.13}$$

where $\mathcal{S}$ is the random variable that controls the biasing particularly on $\mathbf{z}_T$. This approach is used because of the observation through extensive simulations that the failure rate on one particular trapping set is strongly dependent on $\mathbf{z}_T$, and therefore only the noise samples related to the trapping sets are biased to introduce more frequent trapping-set failures. Hence, $\Pr\{\mathbf{z}_{\bar{T}}|\mathcal{S}\} = \Pr\{\mathbf{z}_{\bar{T}}\}$ and $\Pr\{\xi_T|\mathbf{z}_T, \mathbf{z}_{\bar{T}}, \mathcal{S}\} = \Pr\{\xi_T|\mathbf{z}_{\bar{T}}, \mathcal{S}\}$, from which the fourth line of (15.13) follows.

We need now to select a single, easy-to-control random variable $\mathcal{S}$ to bias the vector $\mathbf{z}_T$ effectively.

For the trapping-set-related samples, it is desirable to have a biased noise sample z' that translates a channel output $y = (2/\sigma^2)(1+z)$ to $y' = (2/\sigma^2)(1+z') = (2/\sigma^2)(-1+z'')$, leading to a likely bit error. This suggests the biased noise samples should be in the direction of $(-2,-2,\ldots,-2)$ for the subvector $\mathbf{z}_T$. Now, consider changing from the standard basis $(\alpha_1,\ldots,\alpha_w)$ to a new orthonormal basis $(\gamma_1,\ldots,\gamma_w)$ for $\mathbf{z}_T \in \mathbb{R}^w$, where we fix the first basis vector to $\gamma_1 = (-2,\ldots,-2)/\sqrt{4w} = (-1,\ldots,-1)/\sqrt{w}$, the normalized trapping-set direction vector. The purpose here is to impose various levels of biasing in the direction that is sensitive to trapping-set-induced error events through adjustment of the coordinate γ_1, and then monitor the decoding behavior. The rest of the $w-1$ new basis vectors can be calculated using the Gram–Schmidt process to orthogonalize the set of vectors $\{\gamma_1,\alpha_2,\alpha_3,\ldots,\alpha_w\}$. Thus, we express the subvector $\mathbf{z}_T = (z_{i_1},\ldots,z_{i_w}) = z_{i_1}\alpha_1 + z_{i_2}\alpha_2 + \cdots + z_{i_w}\alpha_w$ as $b_1\gamma_1 + b_2\gamma_2 + \cdots + b_w\gamma_w$, where $b_1,\ldots,b_w$ are obtained by projecting $\mathbf{z}_T$ onto the new basis. Further, the first coordinate b_1 is replaced by a parameter b for controlling different levels of noise biasing, whose values are drawn according to the random variable $\mathcal{S}$. Then we have $\mathbf{z}'_T = b\gamma_1 + b_2\gamma_2 + \cdots + b_w\gamma_w = (b/\sqrt{w})(-1,\ldots,-1) + b_2\gamma_2 + \cdots + b_w\gamma_w$. Since $\mathbf{z}_T$ is a Gaussian vector with zero mean and covariance matrix $\sigma^2\mathbf{I}_{w\times w}$, $\mathcal{S} = b$ has a Gaussian distribution

$$p(b) = \frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{b^2}{2\sigma^2}\right). \tag{15.14}$$

Thus, from (15.13),

$$\begin{aligned}\Pr\{\xi_T\} &= \frac{1}{\sqrt{2\pi}\sigma}\int_{-\infty}^{\infty}\sum_{\mathbf{z}_{\bar{T}}}\Pr\{\mathbf{z}_{\bar{T}}\}\Pr\{\xi_T|\mathbf{z}_{\bar{T}},\mathcal{S}=b\}\exp\left(-\frac{b^2}{2\sigma^2}\right)db \\ &= \frac{1}{\sqrt{2\pi}\sigma}\int_{-\infty}^{\infty}\Pr\{\xi_T|\mathcal{S}=b\}\exp\left(-\frac{b^2}{2\sigma^2}\right)db. \end{aligned}\tag{15.15}$$

To summarize, during the Monte Carlo simulations used to estimate $\Pr\{\xi_T|\mathcal{S}=b\}$, $\mathbf{z}'_T$ (which depends on b) is used in place of $\mathbf{z}_T$, and the components of $\mathbf{z}_{\bar{T}}$ are normal, $\mathcal{N}(0,\sigma^2)$. Note that, when $b = \sqrt{4w}$, $b\gamma_1 = (-2,-2,\ldots,-2)$, which biases the received signal all the way across the boundary to the trapping-set-induced signal point. Therefore, $b \in \left(0,\sqrt{4w}\right]$ is the interval of interest when simulating the conditional rate curve.

Figure 15.13 summarizes the error-floor evaluation at SNR 2.8 dB of a (12,4) trapping set (solid lines) and a (14,4) trapping set (dashed lines) in the graph of the Margulis code. The conditional error-rate ($\Pr\{\xi_T|\mathcal{S}=b\}$) curves were simulated with various levels of noise biasing b. The distribution of $\mathcal{S}=b$ is Gaussian, $\mathcal{N}(0,\sigma^2)$, and the product of the conditional error rate and the Gaussian pdf is integrated to obtain the contributed error rate $\Pr\{\xi_T\}$ of the trapping

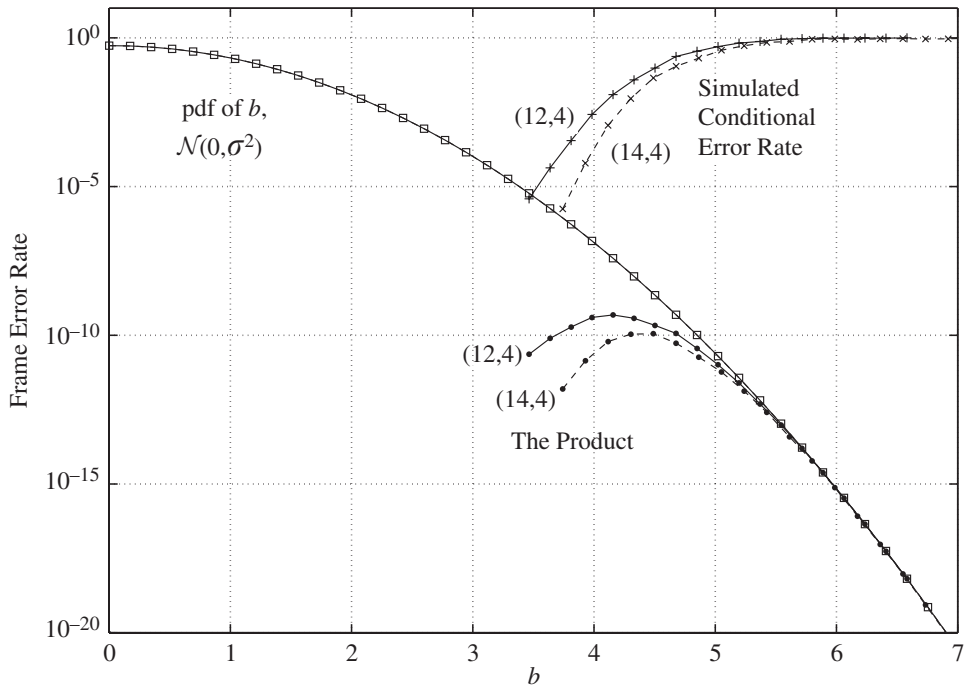


Figure 15.13 Trapping-set evaluations for the Margulis code: (12,4) and (14,4) trapping sets.

set under study. Finally, the overall FER at 2.8 dB is the summation over all trapping-set classes of the contributed rates multiplied by the multiplicity of each trapping-set class, which is 1320 both for the (12,4) and for the (14,4) classes for this code.

The FER prediction curve is shown in Figure 15.14 together with the Monte Carlo simulation curves, where the two predicted data points are indicated by filled black circles. In fact, the curve can be extrapolated from one data point [40] by making σ vary locally such that the simulated conditional error rate changes relatively little in comparison with the changes of the biasing variable pdf. Observe that the prediction curve closely matches the Monte Carlo-simulation curve.

15.4 LDPC Decoder Design for Low Error Floors

The error-rate-floor phenomenon for LDPC codes had remained an unresolved problem for some time. As discussed in Chapter 5, this phenomenon can be characterized as an abrupt decrease in the slope of a code's performance curve on going from the moderate-SNR waterfall region to the high-SNR floor region. Since many systems, such as data-storage devices and optical communication systems,

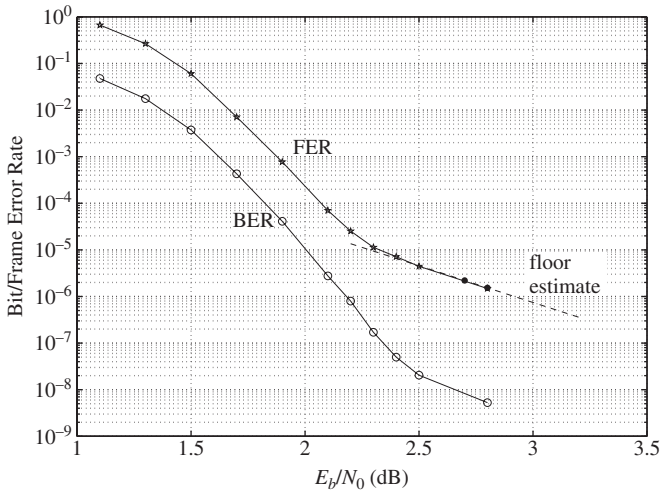


Figure 15.14 Error-floor predictions for the Margulis code on the AWGN channel and the EPR4 channel.

require extremely low error rates, solving the error-floor problem has been a critical issue. Many attempts have been made to solve the floor problem by improving LDPC code design. In this section, we discuss decoder-based strategies for lowering error floors, closely following [41, 52]. Three decoder designs will be discussed: a bi-mode decoder, a bit-pinning decoder together with an outer algebraic code, and a generalized-LDPC (G-LDPC) decoder. These decoding techniques will be presented in the order just given, which is the order of increasing complexity.

As discussed in Chapter 5, the error floors of LDPC codes under message-passing decoding are usually due to low-weight *near-codewords* [42], or *trapping sets* [40], rather than low-weight codewords. A (w,v) trapping set is a set of w VNs that induces a subgraph with v odd-degree CNs and an arbitrary number of even-degree CNs. Errors in each of the w VNs create v parity-check failures and tend to lead to situations from which the iterative decoder cannot escape. Throughout this chapter, we shall often use "trapping set" when we are referring to the induced subgraph.

Iterative decoders are susceptible to trapping-set problems because they work locally in a distributed-processing fashion to (so one hopes) arrive at a globally optimum decision. Of course, iterative decoders are vulnerable to cycles in the graph on which the decoder is based, for cycles often lead the iterative decoder away from the maximum-likelihood (ML) codeword. Further, trapping sets are unions of several cycles. In [53], Koetter and Vontobel derived a framework for the finite-length analysis of iterative decoders of LDPC codes by introducing the concept of graph-cover decoding. It was shown that the iterative decoder cannot

determine whether it is acting on the Tanner graph of the code itself or on some finite cover of the graph. The decoder decodes to the "pseudo-signal" that has the highest correlation with the channel output, and the set of pseudo-codewords (related to trapping sets) from all finite graph covers competes with the transmitted codeword for being the "best" solution. In other words, the ML decision space is a subspace of the iterative decoder decision space.

We shall not delve into such theory in this chapter. Rather, we shall explore specific codes and the trapping sets which dominate the floors of their iterative decoding performance curves. The most dominant trapping sets can be determined from computer simulations in the floor region (see [41, 45]). Moreover, for most practical codes, because the trapping-set-induced subgraphs associated with the error floor are relatively small, and because their cardinalities are not too large, it is feasible to discover, enumerate, and evaluate all of the dominant trapping sets. Hence, once the trapping-set information for an LDPC code has been obtained by simulation and by graph-search techniques, one may explicitly target the known trapping sets with novel, custom-tailored iterative decoder designs. These low-floor decoders, described below, lower the floor by orders of magnitude.

We consider the binary-input additive-white-Gaussian-noise (AWGN) channel and the sum–product-algorithm (SPA) decoder with floating-point precision. We chose two LDPC codes to demonstrate the effectiveness of these decoders: (1) the rate-0.5 (2640,1320) Margulis code, which is notorious for its trapping-set-induced floors [40, 42]; and (2) a short quasi-cyclic rate-0.3 (640,192) code that we devised for this research.

The rest of this section is organized as follows. We first introduce the two LDPC codes under study and their dominant trapping sets. Then a *bi-mode decoder* that recovers trapping sets by use of a post-processing erasure decoding algorithm is discussed. Next, a *bit-pinning decoder* that utilizes one or more outer algebraic codes is presented. Finally, a generalized-LDPC decoder that exploits the concept of generalized LDPC Tanner graphs is presented.

15.4.1 Codes under Study

Both of the LDPC codes we consider have the following structure: the parity-check matrix $\mathbf{H}$ can be conveniently arranged into an $M \times N$ array of $Q \times Q$ permutation matrices and $Q \times Q$ zero matrices. This structure simplifies the analysis of the code because the Tanner graph possesses an automorphism of order Q, and thus simplifies the search for all of the trapping sets of a code. We will refer to the Q variable nodes associated with a column of permutation matrices as a *VN group* and the Q constraint nodes associated with a row of permutation matrices as a *CN group*. A VN of such a code can be represented by an integer pair (v_p, v_o), where $v_p \in \{0, 1, \ldots, N-1\}$ is the index of the VN group and $v_o \in \{0, 1, \ldots, Q-1\}$ is the offset within the VN group. Similarly, a

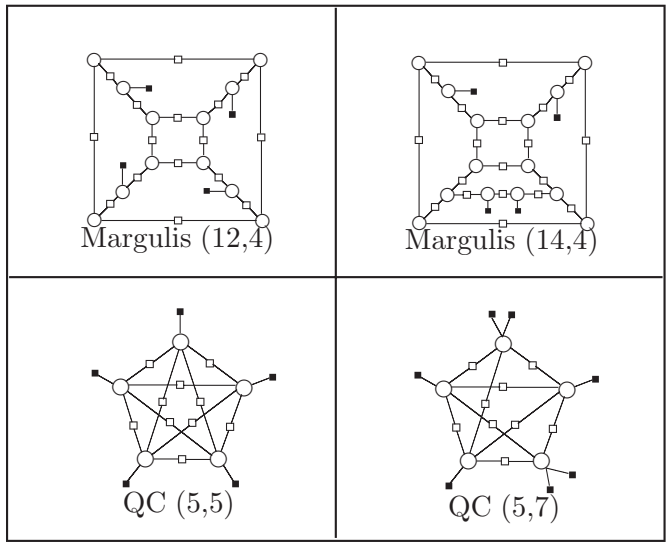


Figure 15.15 Trapping sets of the Margulis and QC codes decoded by the SPA decoder. ◯ = VNs. □ = mis-satisfied CNs. ■ = unsatisfied CNs.

CN can be represented using the pair (c_p, c_o), where $c_p \in \{0, 1, \ldots, M-1\}$ and $c_o \in \{0, 1, \ldots, Q-1\}$.

15.4.1.1 *The Short QC Code*

The **H** matrix for the rate-0.3 (640,192) QC code is a submatrix of the **H** matrix for the rate-0.9 (4550,4096) quasi-cyclic IRA code presented in Chapter 5 when discussing trapping sets and error floors. The latter code was not used because simulations for it are much slower, making floor measurement a time-consuming process. Thus, we have extracted from it a shorter code whose floor is due to the same dominant trapping sets. As for the parent code, the parity-check matrix for this (640,192) code has a circulant size $Q = 64$ and column weight $w_c = 5$. It is a 7×10 array of 64×64 circulant permutation matrices. We observed in our simulations two trapping-set classes with 64 isomorphic trapping sets in each class, with a representative from each class shown in Figure 15.15. The (5,5) trapping set is the dominant one, since it contributes to over 90% of the error-floor level.

We remark that it is very common for structured LDPC codes to have one or two dominant trapping-set classes because their structured graphs lead to isomorphic classes of trapping sets. Unstructured LDPC codes are more likely to have a more diverse collection of trapping sets, but these codes are rarely of interest in practice. Still, the techniques in this section can be straightforwardly extended to unstructured codes as well.

15.4.1.2 *The Margulis Code*

The Margulis construction of a regular (3,6) Gallager code has been well studied [42, 54, 55]. The parity-check matrix for the rate-1/2 (2640,1320) Margulis code we consider can be expressed as a 120×240 array of 11×11 permutation matrices. The "weakness" of this algebraically constructed code is a relatively high error floor due to trapping sets, as discovered by MacKay and Postol in [42] using computer simulations. The source of the error floor is 1320 isomorphic (12,4) trapping sets and 1320 isomorphic (14,4) trapping sets, both types of which are depicted in Figure 15.15. As can be observed in Figure 15.15, a (14,4) trapping set has a structure similar to a (12,4) trapping set, and in fact each (14,4) trapping set contains a unique (12,4) trapping set as a subgraph. Hence, there is a one-to-one correspondence between the (12,4) and (14,4) trapping sets. For both trapping-set classes, half of the bit errors are systematic errors and half are parity errors.

This code is extremely difficult to deal with in terms of decoder design (as we will show later) because its trapping sets are highly entangled. For instance, every VN of the code belongs to six different (12,4) trapping sets. It is for this reason that the Margulis code is a good candidate for the study of floor-lowering techniques.

15.4.2 The Bi-Mode Decoder

In the error-floor region, there are three types of error events: (1) unstable error events, which dynamically change from iteration to iteration and for which w and v are typically large; (2) stable trapping sets, for which w and v are typically small and thus are the main cause of the error floor; and (3) oscillating trapping sets, which periodically vary with the number of decoder iterations and are sometimes subsets of stable trapping sets. The targets of the bi-mode decoder are the dominant stable trapping sets and some of the dominant oscillating trapping sets. In the first mode, SPA decoding is performed with a sufficient number of iterations for the decoder to converge to a codeword or to reach one of the three error-event situations just listed. If an error event is reached, the second "post-processing'' mode is activated only when the syndrome weight of the error event falls into the set of syndrome weights of the target trapping sets. The key role of the second mode is to produce, using syndrome information, an erasure set that contains all of the VNs of the trapping set reached by the decoder, thus resulting in a pure binary erasure channel (BEC) with only correct bits and erasures. Iterative erasure decoding based on the LDPC code's graph can then resolve all of the erasures, including the bits that were originally part of the trapping-set error event.

It was observed that, in the error-floor region, after running extensive Monte Carlo simulations, most of the error patterns correspond to the so-called *elementary trapping sets* [56] whose induced subgraphs have only degree-1 and

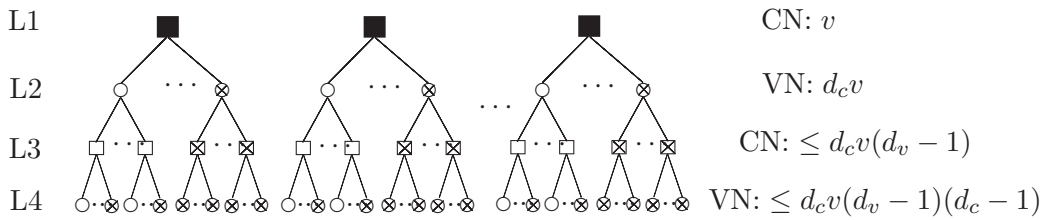


Figure 15.16 Syndrome-erasure flagging trees. ■ = unsatisfied CN. ◯ = VN in trapping set. ⊗ = VN outside trapping set. □ = mis-satisfied CN. ⊠ = CN outside trapping set.

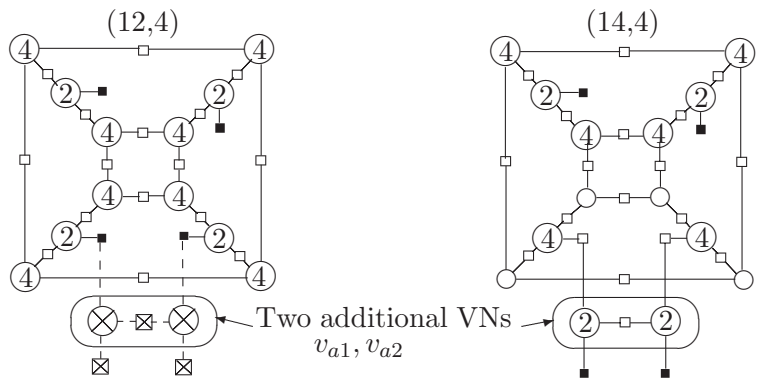


Figure 15.17 Margulis-code erasure flagging with four-level trees (the numbers on the VNs are tree-level numbers). Flipping the two additional VNs causes the switching between the two trapping sets.

degree-2 CNs. That is, for an elementary trapping set, the unsatisfied CNs are usually connected to the trapping set exactly once (i.e., there is one bit error per unsatisfied CN). This is the case for the Margulis and QC codes we consider. However, the applicability of the bi-mode decoding algorithm is not limited to codes with elementary trapping sets.

Assume now that the decoder converges to a trapping set and suppose an unsatisfied CN has degree d_c. The goal is to find which one of its d_c neighbors is in error. Experiments have shown that the LLR magnitude of the bit in error is not necessarily the smallest among the d_c VNs connected to the unsatisfied CN. Thus we flag as erasures all of the d_c neighbors of the unsatisfied CN and then perform iterative erasure decoding in the neighborhood of that unsatisfied CN.

The set of erasures is generated by erasing the VNs of v trees whose roots are the v unsatisfied CNs, as in Figure 15.16. The depth of the trees depends on the trapping-set structures of the code being decoded. Consider a regular LDPC code with row weight d_c and column weight d_v. For a (w,v) trapping set, the number of nodes in each level of the trees is listed in Figure 15.17. For a trapping

set with $w \leq v$, i.e., every VN is connected to at least one unsatisfied CN, trees with two levels can cover the whole trapping set. Otherwise, when $w > v$, larger trees are necessary. As the trees grow, the number of erasures grows exponentially and the whole trapping set will eventually be covered. However, the growth must be limited, because the presence of too many erasures might lead to the formation of stopping sets and overwhelm the decoder. As will be explained below for the Margulis code, for some codes, additional steps are necessary in order to produce a smaller erasure set covering the whole trapping set. We now present the bi-mode decoding solutions for the Margulis and QC codes.

We remark that the bi-mode decoder has the following advantages relative to other floor-lowering techniques.

1. Beside stable trapping sets and oscillating trapping sets, this technique can also handle some unstable error events with $w \leq v$.
2. No outer codes are employed, so the gain is achieved without any loss in code rate.
3. The erasure decoding post-processing has very low computational complexity, which is equivalent to solving linear equations with binary unknowns, and thus involves only a collection of binary XORs.

15.4.2.1 *Short-QC-Code Solution*

For both trapping-set classes of the short QC code, every VN is associated with at least one unsatisfied CN ($w \leq v$). Hence, one level of VNs per tree is enough to include a whole trapping set in the erasure set. Whenever the SPA decoder gets trapped in an error event with syndrome weight 5 or 7, the decoding enters the second mode. Thirty-four erasures are flagged for any (5,5) trapping set and 54 are flagged for any (5,7) trapping set, all of which can be recovered with one erasure decoding iteration.

Because the flooring phenomenon is more pronounced for frame error rate (FER) than it is for bit error rate, in this section we present FER results for the various decoders presented. The FER performance of this code with bi-mode decoder is presented in Figure 15.18, with 100 block errors collected, except for the last two data points for the bi-mode curves, which correspond to 20 block errors. No floor is seen below FER $\sim 10^{-6}$, so the floor has been lowered by at least two orders of magnitude relative to the SPA decoder. For compactness and for comparison, Figure 15.18 includes the FER curves of the QC code with the SPA decoder and the other decoders to be discussed. It also includes the Margulis code FER curves for the various decoders to be discussed.

15.4.2.2 *Margulis-Code Solution*

It has been seen in simulations that the Margulis code with an SPA decoder is sometimes trapped in two oscillating (6,18) trapping sets or two oscillating (7,21) trapping sets, both with period 2. The union of the (6,18) pair is a (12,4) trapping set, and that of the (7,21) pair is a (14,4) trapping set. In the decoder

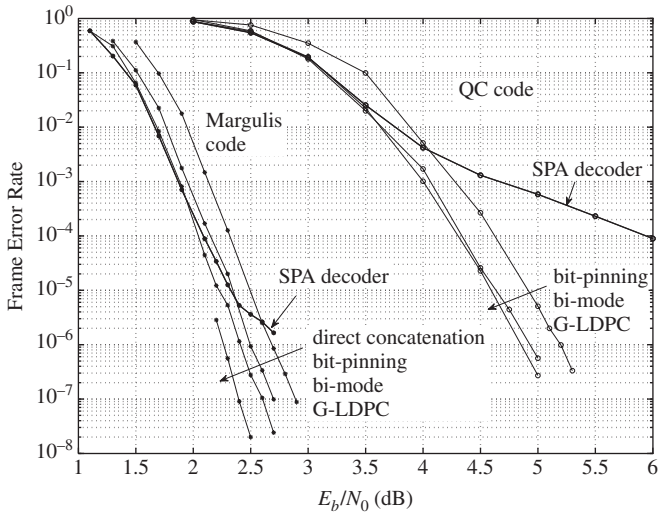


Figure 15.18 FER performances of various low-floor decoders presented in this section for the Margulis code and the QC code.

solution for this code, the target trapping sets include the stable trapping sets as well as the two oscillating configurations. It is obvious that the oscillating trapping sets and some moderate-length unstable error events with $w \leq v$ can be flagged with two-level trees and recovered successfully. As depicted in Figure 15.17, any (12,4) trapping set can be flagged and recovered using trees with four levels; however, four-level trees cover only ten VNs of any (14,4) trapping set. If six-level trees are used, the erasure decoder is overwhelmed by the occurrence of too many erasures.

To solve this problem, an auxiliary step is necessary (Algorithm 15.1). As discussed in Section 15.4.1.2, any (14,4) trapping set contains a (12,4) subset plus two additional VNs. From the syndrome weight itself, the decoder cannot distinguish which of these two trapping-set classes it is dealing with. The algorithm described below which resolves this issue is based on the following observation of the structure of the trees generated from the (12,4) and (14,4) trapping sets. As seen in Figure 15.17, for either trapping set, there exist exactly two VNs in the second level that share a common level-3 CN. These two VNs are what distinguish the (12,4) and (14,4) trapping sets (examine Figure 15.17). Toggling the values of these VNs will switch a (12,4) trapping set to a (14,4) trapping set, and vice versa.

Note that c_a, v_{a_1}, and v_{a_2} in the algorithm below are unique for any (12,4) or (14,4) trapping set. The difference is as follows: if a (14,4) trapping set is reached in the first decoding mode, v_{a_1} and v_{a_2} are incorrect; otherwise, when

Algorithm 15.1 An Auxiliary Algorithm for (12,4) and (14,4) Trapping Sets

1. Take the four unsatisfied CNs and create four trees, each of four levels, with these CNs as roots.
2. Erase all the VNs in the second level, and mark them with flag "A."
3. Erase all the VNs in the fourth level, and mark them with flag "B."
4. For each CN in level-3, count how many of its neighboring VNs have flag "A."Once a CN c_a, with two neighboring VNs (v_{a_1} and v_{a_2}) that are flagged with "A,"has been found, stop the search and go to Step 5.
5. Flip the bit values of v_{a_1} and v_{a_2}, and re-calculate the syndrome, producing a new set of four unsatisfied CNs.
6. Repeat Steps 1–3 with the new syndrome information and then go to Step 7.
7. Including all the VNs with A-type erasure flags all the VNs with and B-type erasure flags, perform iterative erasure decoding in that neighborhood until all of the erasures have been resolved.

a (12,4) trapping set is reached, the two VNs have correct bit values. Note also that, when repeating Steps 1–3, the decoder adds to the erasures already flagged in the first round of Steps 1–3. Flipping the two bits of a (14,4) trapping set essentially corrects two bit errors and turns it into a (12,4) trapping set. However, since both trapping-set classes have the two unique VNs identified in Step 4, the decoder still cannot distinguish which trapping set it is operating on. Thus, if a (12,4) trapping set is reached, v_{a_1} and v_{a_2} are flipped anyway, resulting in a (14,4) trapping set. For this case, repeating Steps 1–3 simply flags as erasures more VNs that are not in error, but does not overwhelm the erasure decoder. By using this algorithm, 352 erasures will be produced for any (12,4) or (14,4) trapping set, which can be recovered by the LDPC code successfully within three or four iterations.

The FER curves for this bi-mode decoder with the Margulis code are presented in Figure 15.18. For the SPA curves in the waterfall region, 100 LDPC codeword errors were collected; in the floor region, 50 codeword errors were collected, except for the last point at 2.7 dB, which corresponds to 20 codeword errors. For the bi-mode curves, the numbers are similar, except that the last three data points contain at least ten block errors. No floor is observed down to FER $\sim 10^{-8}$ for the bi-mode decoder.

15.4.2.3 *Bi-Mode-Decoder Extension*

With certain modifications custom-tailored to tackle different trapping sets, this simple yet effective bi-mode technique can apparently be applied to many LDPC codes. One scenario that requires modification is when the erased bits contain a stopping set, which usually happens if the trapping set has a large number of check violations and/or the code's graph has large check-node degrees. One

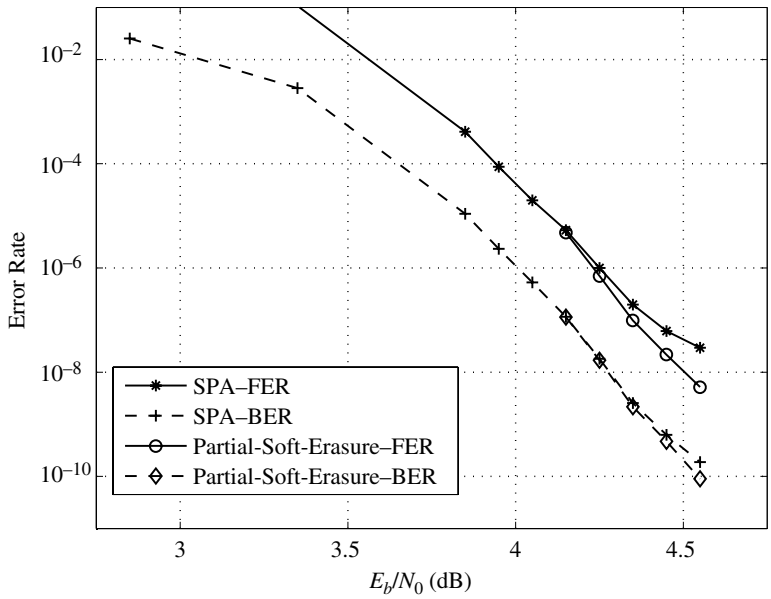


Figure 15.19 The performance of the partial-soft-erase algorithm with the (2048,1723) LDPC code adopted in the IEEE 802.3an 10GBASE-T standard. This code possesses an isomorphic class of (8,8) trapping sets.

solution is to use the *soft-erasing* method, which retains the soft information (LLRs) of the bits outside of the erasure set, while erasing the erasure bits by setting each of their LLRs to zero, and then performing the conventional iterative decoding. Another variation of the bi-mode technique is the *partial-soft-erasing* technique, which erases (by setting LLRs to zeros) the neighboring bits of one of the unsatisfied checks and then performs normal decoding; if the decoder fails to converge, the decoder is reset with erasures based on another unsatisfied check and this procedure is repeated until the decoder converges or all unsatisfied checks have been used.

The authors of [57] show that the (2048,1723) LDPC code adopted in the IEEE 802.3an 10GBASE-T standard possesses an isomorphic class of (8,8) trapping sets. Using the bi-mode decoder described earlier leads to a stopping-set failure. However, these (8,8) trapping sets are easily resolved using the partial-soft-erasing bi-mode decoder, which requires on average about 20 additional decoding iterations to correct the entire trapping set. The resulting performance curve is shown in Figure 15.19.

15.4.3 Concatenation and Bit-Pinning

It is well known that the error floors of LDPC codes are usually a consequence of frequently occurring block errors with a small number of bit errors in each

block error. A natural solution for lowering the floor in this case is to concatenate the LDPC code with a high-rate outer algebraic code to clean up the residual systematic errors, at the expense of a code-rate loss. The serial concatenation of an outer code with the LDPC code has in fact been incorporated into the digital video broadcasting (DVB-S2) standard [58]. However, there exists a more effective concatenation technique, which reduces the code-rate loss by exploiting the trapping-set knowledge of the LDPC code of interest.

The technique combines outer code concatenation with the *bit-pinning* technique [44, 45]. In bit-pinning, the encoder fixes (pins) to zero one or more bits of each trapping set of interest and these bits are deleted prior to transmission. The decoder then sets the magnitude of the *log-likelihood ratios* (LLRs) for these pinned bits to the maximum possible value. Simulations have shown that this procedure offers a substantial improvement in the floor region [44]. However, this solution is not as effective for some codes, such as the Margulis code, which have a large number of overlapped trapping sets due to the substantial code-rate loss introduced. The code-rate loss can be substantially reduced for this code or any other code by the combination of concatenation and bit-pinning.

In one solution to this problem, one or more high-rate outer codes can be concatenated with the LDPC code, whereby the bit assignments in outer codes are arranged such that, whenever a trapping-set error event occurs, at least one (systematic) bit is corrected by an outer algebraic decoder. The goal is to ensure that the short, low-complexity, high-rate outer codes target the trapping sets of interest. The multiple decoders are coordinated as follows. First, a sufficient number of LDPC decoder iterations is performed to guarantee that either all errors are corrected or a stable trapping set is reached. Then the outer hard-decision decoder(s) correct(s) some of the residual errors and feed(s) back the signs of the corrected bits to the LDPC decoder, which pins the absolute values of LLRs corresponding to these bits to the maximum possible value. Then the LDPC decoder continues to iterate. With probability near unity, the LDPC decoder will correct both systematic and parity errors caused by a trapping set within a few additional iterations.

15.4.3.1 *Short-QC-Code Solution*

As mentioned earlier, the (640,192) QC code has two trapping-set configurations. We can concatenate that code with two $t = 1$ binary BCH (64,58) codes. The bits of one BCH code belong to 64 different (5,5) trapping sets, and those of the other belong to 64 different (5,7) trapping sets. Thus, whenever a trapping set is in error, one of its five bits is correctable by one BCH code. Once corrected, this information is fed back to the LDPC decoder, which pins its LLR magnitude to the maximum possible (with the appropriate sign). The overall code rate is reduced from 0.3 to 0.2813 (0.28 dB rate loss). The simulation results are presented in Figure 15.18, where it is seen that no floor exists down to FER $\sim 10^{-7}$. Twenty LDPC codeword errors were collected in the high-SNR region.

15.4.3.2 *Margulis-Code Solution*

Since each of the dominant trapping-set classes of the Margulis code corresponds to either six or seven systematic bits, a single t-error-correcting BCH code with $t = 7$ is an obvious solution. In this case, no pinning is necessary. We choose the $t = 7$ (1320,1243) BCH code with roots in GF(2^{11}), which is shortened from a primitive (2047,1970) BCH code. The code rate of the overall system is reduced from 0.5 to 0.47, corresponding to a rate loss of 0.26 dB.

The code-rate loss can be reduced by considering multiple BCH codes of higher rates, which exploit the fact that the trapping sets of this code are highly overlapped. Four BCH codes with $t = 1$ are sufficient when allowing feedback from the BCH decoders to the LDPC decoder. The BCH-code bit assignments to the Margulis-code bits are shown below. For convenience, the bit assignments are listed by VN group indices v_p since all 11 bits in the VN group will be part of the BCH codeword.

1. BCH(132,124): 24, 74, 88, 113, 81, 80, 87, 162, 34, 53, 137, 60.
2. BCH(77,70): 148, 70, 231, 149, 103, 131, 30.
3. BCH(55,49): 193, 239, 160, 208, 33.
4. BCH(33,28): 94, 116, 203.

The overall code rate is 0.49 (a 0.086 dB loss). The FER curves of the single-BCH-code (direct concatenation, no pinning) and four-BCH-code (with pinning) solutions are shown in Figure 15.18. All curves have at least 20 frame-error occurrences in the range of $E_b/N_0 \geq 2.5$ dB. We observe that both solutions lower the floor beyond the reach of our simulations, and that the code-rate loss for the four-BCH-code solution is about 0.2 dB less than that of the single-BCH-code solution, as indicated above.

15.4.4 Generalized-LDPC Decoder

In this section, we present an SPA decoder that, loosely speaking, is designed by transforming the LDPC code into a generalized LDPC (G-LDPC) code. As described in Chapters 5 and 6, a G-LDPC code, like an LDPC code, is a code that can be described by a sparse bipartite graph with variable nodes and constraint nodes. However, for G-LDPC codes the constraints may be more general than single parity-check (SPC) constraints. For example, a constraint node can represent an arbitrary (n',k') binary linear code.

To see how the G-LDPC philosophy arises in the context of floor-lowering decoders, we remind the reader of the discussion in the introduction regarding locally optimum versus globally optimum decoders. Each constraint-node decoder in an LDPC SPA decoder is locally optimum. It is possible in principle to group all of the SPC constraints into one combined global constraint and design a decoder for that graph, but that would be an ML decoder (for example), which has unacceptable complexity. The strategy here is to instead take one step toward that ideal and cleverly combine only a few SPC constraints at a time. Specifically, we

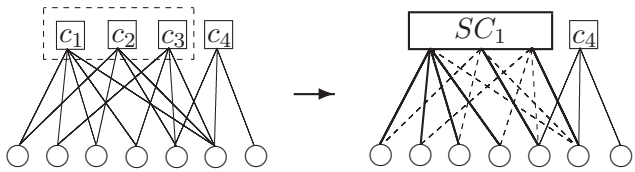


Figure 15.20 An example of combining three SPC constraint processors into the GCP SC_1: the redundant edges (dashed lines) on the right are deleted after grouping. Conventional iterative decoding updates messages based on the graph on the left, employing SPA constraint processors for each SPC separately. The G-LDPC decoder updates messages based on the generalized graph on the right, using the BCJR algorithm, for example, to calculate extrinsic information to be sent from SC_1 to neighboring VNs. Note that we allow SPC constraints to coexist in the generalized graph.

combine check-node processors corresponding to unsatisfied checks in the problematic trapping sets and call the combination a *generalized-constraint processor* (GCP). Observe that an advantage of doing so is that cycles and other deleterious graphical structures (from the perspective of an iterative decoder) may be removed. See Figure 15.20 for an illustration of this. The notion of combining check nodes into a super node to remove cycles first appeared in [46]. The approach here more directly targets problematic trapping sets rather than cycles.

The constituent decoder for the GCP can be any soft-input/soft-output decoder. We use a locally optimal decoder that employs the BCJR algorithm designed to the "BCJR trellis" [18, 59] for the linear code represented by the GCP. We allow SPCs and GCPs to coexist in the generalized Tanner graph, i.e., some of the SPCs are not combined to form a GCP. The G-LDPC decoder, essentially the SPA with GCPs (denoted by SPA-GCP), passes soft information iteratively between VN processors and GCPs in the same manner as the SPA decoder.

Below we present G-LDPC decoders (equivalently, methods for grouping SPCs into GCPs) both for the Margulis code and for the QC code. The approaches are based on the knowledge of the dominant trapping sets. The goal of the SPC-grouping methods is to eliminate the dominant trapping sets, thus lowering the error floor. In selected cases, we predict the FER performance in the floor region using the method in [40] and Section 15.3.

15.4.4.1 *Short-QC-Code Solution*

Take any (5,5) trapping set; its five VNs belong to five VN groups with indices $[0, 1, 2, 3, 4]$; the five unsatisfied CNs belong to four CN groups with indices $[0, 1, 2, 3]$. Thus, we take any two (5,5) trapping sets and group the three unsatisfied CNs which belong to disjoint CN groups. The 64 (5,5) trapping sets can be combined to give $32 \cdot 3 = 96$ GCPs. The FER performance curves with this decoder are shown in Figure 15.18, and no trapping sets are observed down to FER $\sim 10^{-7}$. The curves were drawn with at least 30 block-error occurrences.

15.4.4.2 *Margulis-Code Solution*

It was observed through simulations of many LDPC codes that, in the floor region, a frame-error event usually contains a single trapping set. Instead of combining unsatisfied CNs within a trapping set, the G-LDPC decoder takes trapping sets in pairs and combines an unsatisfied CN of one trapping set with that of the other. When the decoder is trapped in one trapping set, reliable information from its companion trapping set will be passed along the GCP, allowing recovery from the trapping-set error event.

The G-LDPC decoder solution for the Margulis code contains 660 GCPs, each consisting of two SPCs. This number came from the 1320 (12,4) trapping sets (660 pairs). Thus there are no SPC CNs in the generalized graph in this case. Owing to the overlap of trapping sets (for example, every CN is one of the unsatisfied CNs in four different trapping sets), all the trapping sets form a linked overlapped network. The generalization [41] of the importance-sampling method in Section 15.3 was applied to this decoder and no trapping sets were observed. The effectiveness of this simple G-LDPC decoder is also confirmed by Monte Carlo simulations (Figure 15.18), which show that there is no floor down to FER $\sim 10^{-8}$. We collected 50 error events at 2.2 dB and 2.3 dB, 10 error events at 2.4 dB, and 5 error events at 2.5 dB.

15.4.5 Remarks

These floor-lowering decoder techniques can be extended to partial-response channels that model magnetic-storage channels. This has successfully been carried out in [41, 60]. Also, a possible replacement for the complex BCJR processors used in the G-LDPC decoder is the SOVA algorithm (see Chapter 7 or [61]) or a soft-output Chase-like processor (see Chapter 7 or [62]). Further work on evaluating the performance of these techniques with fewer LDPC decoder iterations, on and how to determine the minimum number of trapping-set bits to pin so that the iterative decoder can correct the residual errors, is recommended.

15.5 LDPC Convolutional Codes

LDPC convolutional codes are the stream-oriented counterpart to the LDPC block codes discussed throughout this book. They first appeared in the coding literature in 1999 [63], although the basic idea appeared earlier in a 1981 U.S. patent application [64]. Relative to block LDPC block codes, LDPC convolutional codes are still in their infancy at the time of writing. In this section, we present a brief introduction to these codes. Further details may be found in the references.

We consider a rate $R = b/c$ binary LDPC convolutional code, which may be time-varying in general. We denote the binary encoder input sequence for

discrete-time index $t > 0$ by

$$\mathbf{u}_{[0,t]} = [\mathbf{u}_0, \mathbf{u}_1, \ldots, \mathbf{u}_t],$$

where $\mathbf{u}_i = \left(u_i^{(1)}, u_i^{(2)}, \ldots, u_i^{(b)}\right)$ for $0 \le i \le t$. This sequence is encoded into the code sequence

$$\mathbf{v}_{[0,t]} = [\mathbf{v}_0, \mathbf{v}_1, \ldots, \mathbf{v}_t],$$

where $\mathbf{v}_i = \left(v_i^{(1)}, v_i^{(2)}, \ldots, v_i^{(c)}\right)$. The encoder is generally systematic, in which case $\mathbf{v}_i$ has the form $\mathbf{v}_i = \left[\mathbf{v}_i^{(0)}, \mathbf{v}_i^{(1)}\right]$, where $\mathbf{v}_i^{(0)} = \mathbf{u}_i$ and $\mathbf{v}_i^{(1)}$ is a parity word of length $c - b$.

Each code sequence $\mathbf{v}_{[0,\infty]}$ satisfies the parity-check equations

$$\mathbf{v}_{[0,\infty]}\mathbf{H}_{[0,\infty]}^{\mathrm{T}} = \mathbf{0},$$

where, for $t \le t'$,

$$\mathbf{H}_{[t,t']} = \begin{bmatrix} \mathbf{H}_0(t) & & & \\ \vdots & \mathbf{H}_0(t+1) & & \\ \mathbf{H}_\mu(t+\mu) & \vdots & \ddots & \\ & \mathbf{H}_\mu(t+\mu+1) & \ddots & \mathbf{H}_0(t') \\ & & \ddots & \vdots \\ & & & \mathbf{H}_\mu(t'+\mu) \end{bmatrix} \tag{15.16}$$

is a submatrix of the parity-check matrix $\mathbf{H}_{[0,\infty]}$ (also called the *syndrome former*). The parameter μ is the memory of the code. The submatrices $\mathbf{H}_i(t)$ have dimension $(c - b) \times c$ and satisfy the following properties.

1. $\mathbf{H}_i(t) = \mathbf{0}$, $i < 0$ and $i > \mu$, for all t.
2. $\mathbf{H}_\mu(t) \ne \mathbf{0}$ for at least one value of t.
3. $\mathbf{H}_0(t)$ has full rank for all t.

To ensure low-complexity encoding and (iterative) decoding, i.e., to ensure that $\mathbf{H}_{[t,t']}$ corresponds to an LDPC convolutional code, the weight of each of its columns must be much less than $(c - b)(\mu + 1)$. Further, the code is (r,g)-regular if $\mathbf{H}_{[t,t']}$ has maximum row weight r and column weight g. The LDPC convolutional code is periodic if $\mathbf{H}_i(t) = \mathbf{H}_i(t + T)$ for all i and for some $T > 1$. It is time-invariant if $\mathbf{H}_i(t) = \mathbf{H}_i(t')$ for all i and for all $t \ne t'$.

LDPC convolutional codes can be constructed randomly (e.g., via computer) or they may be derived from LDPC block codes. An example of how an LDPC convolutional code may be derived from an LDPC block code is presented in Figure 15.21. On the left is the $\mathbf{H}$ matrix for a rate-1/2 (10,5) LDPC block code with row weight $r = 4$ and column weight $g = 2$. If that matrix is partitioned as indicated in the left of Figure 15.21 and then the bottom partition is placed above the top

$$
\begin{array}{cccccccccc}
1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\
0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\
0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1
\end{array}
\qquad
\begin{array}{cccccccccc}
1 & 1 & & & & & & & & \\
1 & 0 & 0 & 0 & & & & & & \\
0 & 1 & 0 & 0 & 1 & 0 & & & & \\
0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & & \\
0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\
\hline
 & & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
 & & & & 1 & 1 & 1 & 0 & 0 & 0 \\
 & & & & & & 0 & 1 & 1 & 0 \\
 & & & & & & & & 0 & 1
\end{array}
$$

Figure 15.21 An illustration of how an LDPC convolutional-code parity-check matrix (right) can be derived from that of an LDPC block code (left).

partition as indicated on the right of the figure, the result is a diagonal band of submatrices. This diagonal band can be concatenated with itself indefinitely to create a parity-check matrix in the form of (15.16), corresponding to a rate-1/2 LDPC convolutional code with period 5, row weight $r = 4$, and column weight $g = 2$.

Encoders for an LDPC convolutional code may be derived from (15.16). The so-called syndrome-former realization follows from the fact that (15.16) may be rewritten as

$$\mathbf{v}_t \mathbf{H}_0^{\mathrm{T}}(t) + \mathbf{v}_{t-1}\mathbf{H}_1^{\mathrm{T}}(t) + \cdots + \mathbf{v}_{t-\mu}\mathbf{H}_\mu^{\mathrm{T}}(t) = \mathbf{0}. \tag{15.17}$$

Because the submatrices $\mathbf{H}_0^{\mathrm{T}}(t)$ are full rank for all t, this equation can be used to determine $\mathbf{v}_t = \left[\mathbf{v}_t^{(0)}, \mathbf{v}_t^{(1)}\right]$. Since $\mathbf{v}_t^{(0)} = \mathbf{u}_t$ and the previous blocks $\mathbf{v}_{t-1}, \mathbf{v}_{t-2}, \ldots, \mathbf{v}_{t-\mu}$ are known, the parity word $\mathbf{v}_t^{(1)}$ can be determined from (15.17). Further details about encoding, including a partial syndrome-former realization, may be found in [65].

The iterative decoder for LDPC convolutional codes uses belief propagation much like that for an LDPC block code. In accordance with the infinite band structure of the parity-check matrix, the Tanner graph for an LDPC convolutional code is of infinite length and its node connections are localized within one constraint length, defined as $(\mu + 1)c$. The decoder architecture can involve a sliding-window processor or multiple pipelined processors. Large storage requirements and a long initial decoding delay are two drawbacks of LDPC convolutional-code decoders. Proposals to mitigate these shortcomings are presented in [65].

15.6 Fountain Codes

Fountain codes were invented for the packet-based-file multicast application in the late 1990s [66, 67]. The motivation for their use derives from the fact that automatic-repeat-request (ARQ) schemes can be inefficient relative to forward-

error-control (FEC) schemes, particularly when there are many intended receivers, many of which might request retransmissions simultaneously, or when there is a long round-trip delay. These issues motivated the invention of fountain codes. Since their invention, they have found other applications. For example, in the 3GPP multimedia broadcast/multicast service standard, the turbo code at the physical layer, originally designed for voice, provides a sufficiently low bit error rate for voice ($< 10^{-4}$), but a block error rate on the order of 0.01. This is not acceptable for video transmission, for example. Thus, a fountain code (raptor code, discussed below) has been added at a higher network layer.

Fountain codes are not a subclass of LDPC codes, per se, but most fountain codes are graphically described codes like LDPC codes. (An exception is a Reed–Solomon code that can be used as a fountain code.) Fountain codes encode blocks of packets rather than blocks of bits, with each packet being treated as a code symbol. Fountain codes are so called because in the file-distribution application an encoded data file is broadcast repeatedly in a so-called data-carousel fashion, and the entire codeword can be recovered with near-unity probability after receiving any $(1 + \varepsilon)k$ packets from the codeword (here, k is the data-block length in packets; $\varepsilon = 0.05$ is typical). This is reminiscent of a circular water fountain from which it is possible to fully fill your glass from anywhere around its perimeter.

Here, we give only an introduction to fountain codes. Their design and other details may be found in the references. Competing LDPC code techniques may also be found in the literature [68, 69]. See also Chapter 13 on erasure codes.

15.6.1 Tornado Codes

Tornado codes [66, 67] are not structured LDPC codes in the sense of the other LDPC codes presented in this section, but they are described by a graph that facilitates encoding. A tornado-code encoder can be thought of as a cascade of $s + 1$ rate-$(1/(1 + \beta))$ LDPC encoders followed by a single rate-$(1 - \beta)$ conventional erasure-correcting code, such as a large Reed–Solomon code. Tornado codes are generally very long codes, and the usual graphical representation is presented in Figure 15.22. The leftmost column of nodes in the figure represents the k packets to be encoded. The next column of nodes represents the βk parity packets for the code at level 0. Note that this graphical representation differs from that of standard Tanner graphs in that each parity node represents a packet whose value is the component-wise modulo-2 sum of the neighboring packets to its left. The column of nodes to the right of the level-0 parity nodes represents the $\beta^2 k$ parity packets computed for the level-1 code having as input the βk parity packets of level-0. Finally, the input to the conventional erasure code is the $\beta^{s+1}k$ parity packets from the level-s LDPC code, which then produces $\beta^{s+1}k(\beta/(1 - \beta))$ parity

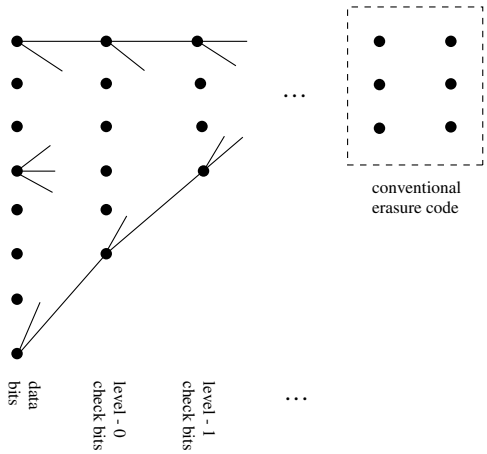


Figure 15.22 A tornado-code graph.

packets. The total number of tornado-code parity packets is

$$m = \sum_{i=1}^{s+1} \beta^i k + \beta^{s+1} k \left(\frac{\beta}{1-\beta} \right) = k \left(\frac{\beta}{1-\beta} \right)$$

so that the overall code rate is $1 - m/(k+m) = 1 - \beta$.

Whereas encoding proceeds from left to right in the graph of Figure 15.22, decoding proceeds from right to left. Thus, the conventional erasure decoder decodes first and the level-0 decoder decodes last. The conventional erasure decoder is typically a Reed–Solomon code, and any Reed–Solomon erasure decoding algorithm will do at this first decoding stage. Then decoding moves leftward to the sparse-graph stage at level s. The iterative decoder at this stage continually looks for check equations with a single erasure (i.e., a single unknown) and solves for the unknowns until all erasures have been filled in. Then the decoder moves leftward to the next sparse-graph stage, applying the same erasure decoding algorithm, and so on, until level 0 has been reached and all of the k original packets have been recovered.

15.6.2 Luby Transform Codes

One disadvantage of tornado codes is that they are of fixed rate. In the ARQ scenario for which they are intended, this is a drawback because, when the decoder fails to fully recover the data, the only recourse is to retransmit the entire codeword. By contrast, in hybrid ARQ/FEC schemes, so-called *rate-compatible codes* that allow the simple incremental transmission of additional parity rather than the retransmission of an entire codeword are used. It was in this spirit that Luby Transform (LT) codes were invented [70]. Such codes are *rateless codes* in the

sense that, starting from a block of k data packets, an arbitrary number of parity packets may be created on demand. (Other rateless codes that appeared at around the same time, but received less notice, are called *online codes* [71].)

The encoding process for LT codes is as follows.

1. Randomly choose the degree d of the next parity packet from some degree distribution. (The appropriate degree distribution may be found in the references.)
2. Choose d data packets uniformly at random.
3. The parity packet is the componentwise exclusive-or of the d chosen data packets, which are called the *neighbors* of the parity packet. If another parity packet is required, go to Step 1.

Observe that this encoding can proceed indefinitely and that it is a non-systematic encoding. Thus, what we are calling parity packets here are not strictly parity packets and might be called *code packets*. As for the decoder, it clearly requires knowledge of the chosen degree and chosen neighbor for each parity packet before the above iterative erasure-filling algorithm can be applied. Several possibilities exist, the suitability of each depending on the application. For example, keeping in mind that the packets can be very large so that some overhead will affect throughput efficiency negligibly, the degree/neighbor information for a given parity packet can be sent together with that packet. As another example, the encoder and decoder can employ the same degree/neighbor algorithm so that the same degree/neighbor information can be computed for a given packet both at the encoder and at the decoder. Such algorithms may be found in the references, as can discussions on the optimal degree distribution.

15.6.3 Raptor Codes

Raptor codes [72] constitute an extension of LT codes and solve the following deficiency of LT codes. As mentioned above, the goal of fountain codes is to be able to reliably recover the original k data packets from $(1+\varepsilon)k$ received code packets, where ε is small. This is possible for LT codes, but reliability is achieved at the expense of LT encoder/decoder complexity. Briefly, raptor codes solve this problem by adding an outer erasure code to a less-reliable, lower-complexity LT code. Thus, this outer code encodes k data packets into a block of k' outer code packets, which are then encoded by an LT code in the fashion described above. Although the less-complex LT inner code is not as reliable as a stand-alone LT code, the overall raptor code is reliable due to the presence of the outer code which can resolve any erasures missed by the LT decoder. The outer code can be an appropriately designed LDPC code. It is also possible to design systematic raptor codes. A nice description of the 3GPP raptor code and a proposed improvement (in performance and encoder/decoder complexity) appears in [73].

Problems

15.1 Put in all of the details in the derivation of (15.1), starting with the definition of $L(x_i|y)$.

15.2 Reproduce Figure 15.2.

15.3 Via Monte Carlo techniques, reproduce the empirical pdfs shown in Figure 15.4.

15.4 Reproduce the EXIT chart shown in Figure 15.5. The degree distributions are given in the discussion of that figure.

15.5 Write a simulation program for the Viterbi decoder on the PR1/AWGN channel and plot the bit error rate P_b versus E_b/σ^2, where E_b is the average energy per information bit and σ^2 is the variance of each AWGN sample.

15.6 Repeat the previous problem for the BCJR detector. (If you did the previous problem, compare the performances of the two detectors.) Also, examine the empirical pdfs of the conditional LLRs $L(x_i = +1|\mathbf{y})$ and $L(x_i = -1|\mathbf{y})$. Here $\mathbf{y}$ is the received channel sequence (of sufficient length). At what SNR values (SNR $= E_b/\sigma^2$ as in the previous problem) are the pdfs approximately Gaussian (if any). If they are approximately Gaussian, are they consistent?

15.7 (Project) Design a 0.9(18176,16384) LDPC code for the PR1/AWGN channel using the EXIT-chart design technique outlined in Section 15.2.2 that outperforms the IRA code in Figure 15.9.

15.8 (Project) Simulate the Margulis code with various iterative decoders to determine whether or not the trapping sets are decoder-dependent. Include in your set of decoders the SPA decoder, the min-sum decoder, the min-sum with correction-factor decoder, and the min-sum with offset decoder.

15.9 (Project) Reproduce Figure 15.13 and from this produce the error-floor estimate seen in Figure 15.14 for the binary-input AWGN channel.

15.10 (Project) Reproduce the simulation curves for the bi-mode decoding of the Margulis code in Figure 15.18.

15.11 Use the (7,4) Hamming code with the $\mathbf{H}$ matrix below as a fountain code in which the packets are 4 bits long. Find the missing packets in the received codeword $[0101, \bar{e}, 0000, \bar{e}, 1100, 1011, 0111]$, where $\bar{e}$ represents an erased packet, i.e., a packet lost in transmission.

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

References

[1] G. Ungerboeck, "Channel coding with multilevel/phase signals," *IEEE Trans. Information Theory*, vol. 28, no. 1, pp. 55–67, January 1982.

[2] H. Imai and S. Hirakawa, "A new multilevel coding method using error correcting codes," *IEEE Trans. Information Theory*, vol. 23, no. 5, pp. 371–377, May 1977.

[3] U. Wachsmann, R. F. H. Fischer, and J. B. Huber, "Multilevel codes, theoretical concept and practical design rules," *IEEE Trans. Information Theory*, vol. 45, no. 7, pp. 1361–1391, July 1999.

[4] E. Zehavi, "8-PSK trellis codes for Rayleigh channel," *IEEE Trans. Communications,* vol. 40, no. 5, pp. 873–884, May 1992.

[5] G. Caire, G. Taricco, and E. Biglieri, "Bit-interleaved coded modulation," *IEEE Trans. Information Theory*, vol. 44, no. 5, pp. 927–946, May 1998.

[6] J. Hou, P. Siegel, L. Milstein, and H. Pfister, "Multilevel coding with low-density parity-check component codes," *IEEE 2001 GlobeCom Conf.*, 2001, pp. 1016–1020.

[7] S. ten Brink, G. Kramer, and A. Ashikhmin, "Design of low-density parity check codes for modulation and detection," *IEEE Trans. Communications*, vol. 50, no. 6, pp. 670–678, June 2002.

[8] J. Hou, P. Siegel, L. Milstein, and H. Pfister, "Capacity-approaching bandwidth-efficient coded modulation schemes based on low-density parity-check codes", *IEEE Trans. Information Theory*, vol. 49, no. 9, pp. 2141–2155, September 2003.

[9] A. Chindapol, X. Li, and J. Ritcey, "Bit-interleaved coded modulation with iterative decoding and 8PSK signaling," *IEEE Trans. Communications*, vol. 50, no. 8, pp. 1250–1257, August 2002.

[10] Y. Li and W. E. Ryan, "Design of LDPC-coded modulation schemes," *Proc. 3rd Int. Symp. on Turbo Codes and Related Topics*, Brest, September 2003, pp. 551–554.

[11] M. Tuchler and J. Hagenauer, "EXIT charts of irregular codes," *Proc. 2002 CISS*, Princeton University, March 2002, pp. 748–753.

[12] K. Price and R. Storn, http://www.icsi.berkeley.edu/~storn/code.html.

[13] Y. Li and W. E. Ryan, "Bit-reliability mapping in LDPC-coded modulation systems," *IEEE Communications Lett.*, vol. 9, no. 1, pp. 1–3, January 2005.

[14] R. Narayanaswami, "Coded modulation with low density parity check codes," M.S. Thesis, ECE Dept., Texas A&M University, 2001.

[15] J. G. Proakis and M. Salehi, *Digital Communications*, 5th edn., New York, McGraw-Hill, 2008.

[16] J. R. Barry, E. A. Lee, and D. G. Messerschmitt, *Digital Communication*, 3rd edn., Dordrecht, Kluwer, 2003.

[17] G. D. Forney, "Maximum-likelihood sequence estimation of digital sequences in the presence of intersymbol interference," *IEEE Trans. Information Theory*, pp. 363–378, May 1972.

[18] L. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate," *IEEE Trans. Information Theory*, vol. 20, no. 3, pp. 284–287, March 1974.

[19] C. Douillard, M. Jezequel, C. Berrou, A. Picart, P. Didier, and A. Glaviex, "Iterative correction of intersymbol interference: turbo equalization," *European Trans. Telecomm.*, vol. 6, no. 5, pp. 507–511, September–October 1995.

[20] J. Hagenauer and P. Hoeher, "A Viterbi algorithm with soft-decision outputs and its applications," *1989 IEEE Global Telecommunications Conf.*, November 1989, pp. 1680–1686.

[21] W. E. Ryan, "Performance of high rate turbo codes on a PR4-equalized magnetic recording channel," *1998 IEEE Int. Conf. on Communications*, pp. 947–951, June 1998.

[22] W. Ryan, L. McPheters, and S. McLaughlin, "Combined turbo coding and turbo equalization for PR4-equalized Lorentzian channels," *1998 Conf. on Information Sciences and Systems*, Princeton University.

[23] T. Souvignier, A. Friedmann, M. Oberg, P. Siegel, R. Swanson, and J. Wolf, "Turbo decoding for PR4: parallel versus serial concatenation," *1999 IEEE Int. Conf. on Communications*, June 1999, pp. 1638–1642.

[24] W. Ryan, "Concatenated codes for class IV partial response channels," *IEEE Trans. Communications*, vol. 49, no. 3, pp. 445–454, March 2001.

[25] A. Ghrayeb and W. Ryan, "Concatenated code system design for storage channels," *IEEE J. Selected Areas in Communications*, vol. 19, no. 4, pp. 709–718, April 2001.

[26] C. X. Huang, and A. Ghrayeb, "A simple remedy for the exaggerated extrinsic information produced by the SOVA algorithm," *IEEE Trans. Wireless Communications*, vol. 5, no. 5, pp. 996–1002, May 2006.

[27] A. Ghrayeb, and C. X. Huang, "Improvements in SOVA-based decoding for turbo-coded storage channels," *IEEE Trans. Magnetics*, vol. 41, no. 12, pp. 4435–4442, December 2005.

[28] Z.-N. Wu, and J. M. Cioffi, "Turbo decision aided equalization for magnetic recording channels," *1999 IEEE GlobeCom Conf.*, December 1999, pp. 733–738.

[29] M. Tuechler, R. Koetter, and A. Singer, "Turbo equalization: principles and new results," *IEEE Trans. Communications*, vol. 50, no. 5, pp. 754–767, May 2002.

[30] R. Koetter, A. Singer, and M. Tuechler, "Turbo equalization," *IEEE Signal Processing Mag.*, pp. 67–80, January 2004.

[31] R. Lopes and J. R. Barry, "The soft-feedback equalizer for turbo equalization of highly dispersive channels," *IEEE Trans. Communications*, vol. 54, no. 5, pp. 783–788, May 2006.

[32] M. Nissila and S. Pasupathy, "Soft-input soft-output equalizers for turbo receivers: a statistical physics perspective," *IEEE Trans. Communications*, vol. 55, no. 5, pp. 1300–1307, July 2007.

[33] J. A. Anguita, M. Chertkov, M. Neifeld, and B. Vasic, "Bethe free energy approach to LDPC decoding on memory channels," http://arXiv.org/abs/0904.0747, 2007.

[34] A. Kavcic, X. Ma, and M. Mitzenmacher, "Binary intersymbol interference channels: Gallager codes, density evolution, and code performance bounds," *IEEE Trans. Information Theory*, vol. 49, no. 7, pp. 1636–1652, July 2003.

[35] N. Varnica and A. Kavcic, "Optimized low-density parity-check codes for partial response channels," *IEEE Communications Lett.*, vol. 7, no. 4, pp. 168–170, April 2003.

[36] K. R. Narayanan, D. N. Doan, and R. Tamma, "Design and analysis of LDPC codes for turbo equalization with optimal and suboptimal soft output equalizers," *Proc. Annual Allerton Conf. on Communications, Control, and Computing*, Monticello, IL, October 2002, pp. 737–746.

[37] F. Wang, *Optimal Code Rates for the Lorentzian Channel and Irregular Block Turbo Codes*, Master's Thesis, ECE Dept., University of Arizona, November 2003.

[38] W. E. Ryan, F. Wang, R. Wood, and Y. Li, "Optimal code rates for the Lorentzian channel: Shannon codes and LDPC codes," *IEEE Trans. Magnetics*, vol. 40, no. 11, pp. 3559–3565, November 2004.

[39] S.-Y. Chung, T. Richardson, and R. Urbanke, "Analysis of sum-product decoding of LDPC codes using a Gaussian approximation," *IEEE Trans. Information Theory*, vol. 47, no. 2, pp. 657–670, February 2001.

[40] T. Richardson, "Error floors of LDPC codes," in *Proc. 41st Allerton Conf. on Communications, Control, and Computing*, Allerton House, Monticello, IL, October 2003.

[41] Y. Han, *LDPC Coding for Magnetic Storage: Low-Floor Decoding Algorithms, System Design, and Performance Analysis*, Ph.D. Dissertation, ECE Dept., University of Arizona, August 2008.

[42] D. MacKay and M. Postol, "Weaknesses of Margulis and Ramanujan–Margulis low-density parity-check codes," in *Proc. MFCSIT*, Galway, 2002.

[43] Z. Zhang, L. Dolecek, B. Nikolic, V. Anantharam, and M. Wainwright, "Investigation of error floors of structured low-density parity-check codes by hardware emulation," in *Proceedings of 2006 IEEE GlobeCom*, San Francisco, CA, November 2006.

[44] Y. Zhang and W. E. Ryan, "Toward low LDPC-code floors: a case study," *IEEE Trans. Communications*, vol. 57, no. 5, pp. May 2009.

[45] Y. Zhang, *Design of Low-Floor Quasi-Cyclic IRA Codes and Their FPGA Decoders,* Ph.D. Dissertation, ECE Dept., University of Arizona, May 2007.

[46] F. Kschischang, B. Frey, and H.-A. Loeliger, "Factor graphs and the sum–product algorithm," *IEEE Trans. Information Theory*, vol. 47, no. 2, pp. 498–519, February 2001.

[47] E. Cavus, C. L. Haymes, and B. Baneshrad, "An IS simulation technique for very low BER performance evaluation of LDPC codes," *IEEE International Conf. on Communications*, vol. 3, June 2006, pp. 1095–1100.

[48] C. A. Cole, S. G. Wilson, E. K. Hall and T. R. Giallorenzi, "A general method for finding low error rates of LDPC codes, " submitted to *IEEE Trans. Inform. Theory*, June 2006.

[49] C. Berrou and S. Vaton, "Computing the minimum distances of linear codes by the error impulse method," *IEEE International Symposium on Information Theory*, Lausanne, June 30–July 5, 2002.

[50] X. Y. Hu, M. P. C. Fossorier, and E. Eleftheriou, "On the computation of the minimum distance of low-density parity-check codes," in *Proceedings of 2004 IEEE Int. Conf. on Communications*, 20–24 June 2004, vol. 2, pp. 767–771.

[51] B. Xia and W. E. Ryan, "On importance sampling for linear block codes," *Proc. 2003 IEEE Int. Conf. Communications*, May 2003, vol. 4, pp. 2904–2908.

[52] Y. Han and W. E. Ryan, "Low-floor decoders for LDPC codes," *IEEE Trans. Communications*, vol. 57, no. 5, May 2009.

[53] R. Koetter and P. Vontobel, "Graph-covers and iterative decoding of finite length codes," *Int. Symp. on Turbo Codes*, Brest, 2003, pp. 75–82.

[54] G. A. Margulis, "Explicit constructions of graphs without short cycles and low-density codes," *Combinatorica*, vol. 2, no. 1, pp. 71–78, 1982.

[55] J. Rosenthal and P. O. Vontobel, "Constructions of LDPC codes using Ramanujan graphs and ideas from Margulis," in *Proceedings of the 38th Annual Allerton Conference on Communication, Control, and Computing*, 2000, pp. 248–257.

[56] O. Milenkovic, E. Soljanin, and P. Whiting, "Asymptotic spectra of trapping sets in regular and irregular LDPC code ensembles," *IEEE Trans. Information Theory*, vol. 53, no. 1, pp. 39–55, January 2007.

[57] L. Dolecek, Z. Zhang, M. Wainwright, V. Anantharam, and B. Nikolić, "Evaluation of the low frame error rate performance of LDPC codes using importance sampling," *2007 IEEE Information Theory Workshop*, September 2–6, 2007, 00. 202–207.

[58] *Digital Video Broadcasting (DVB) Second Generation Farming Structure, Channel Coding and Modulation Systems for Broadcasting, Interactive Services, News Gathering and Other Broadband Satellite Applications.* DRAFT EN 302 307 DVBS2-74r15, European Telecommunications Standards Institute, 2003.

[59] R. McEliece, "On the BCJR trellis for linear block codes," *IEEE Trans. Information Theory*, vol. 42, no. 7, pp. 1072–1092, July 1996.

[60] Y. Han and W. E. Ryan, "Pinning techniques for low-floor detection/decoding of LDPC-coded partial response channels," *2008 Symposium on Turbo Codes and Related Topics*, Lausanne, 2008, pp. 49–54.

[61] J. Hagenauer and P. Hoeher, "A Viterbi algorithm with soft-decision outputs and its applications," *Proc. of IEEE GlobeCom*, November 1989, vol. 3, pp. 1680–1686.

[62] R. Pyndiah, "Near optimum decoding of product codes: block turbo codes," *IEEE Trans. Communications*, vol. 46, no. 8, pp. 1003–1010, August 1998.

[63] A. Jimenez-Felstrom and K. Sh. Zigangirov, "Time-varying periodic convolutional codes with low-density parity-check matrix," *IEEE Trans. Information Theory*, vol. 45, no. 9, pp. 513–526, September 1999.

[64] R. M Tanner, "Error-correcting coding system," U.S. patent 4,295,218, October 1981.

[65] A. Pusane, A. Jimenez-Feltstrom, A. Sridharan, A. Lentmaier, M. Zigangirov, and D. Costello, "Implementation aspects of LDPC convolutional codes," *IEEE Trans. Communications*, pp. 1060–1069, July 2008.

[66] J. Byers, M. Luby, M. Mitzenmacher, and A. Rege, "A digital fountain approach to reliable distribution of bulk data," *ACM SIGCOMM Computer Communication Review*, vol. 28, no. 4, pp. 56–67, October 1998.

[67] M. Luby, M. Mitzenmacher, A. Shokrollahi, and D. Spielman, "Efficient erasure correcting codes," *IEEE Trans. Information Theory,* vol. 47, no. 2, pp. 569–584, February 2001.

[68] E. Paolini, G. Liva, M. Varrella, B. Matuz, and M. Chiani, "Low-complexity LDPC codes with near-optimum performance over the BEC," *Proc. 4th Advanced Satellite Mobile Systems Conference*, Bologna, August 2008.

[69] E. Paolini, G. Liva, B. Matuz, and M. Chiani, "Generalized IRA erasure correcting codes for hybrid iterative/maximum likelihood decoding," *IEEE Communications Lett.*, vol. 12, no. 6, pp. 450–452, June 2008.

[70] M. Luby, "LT codes," *Proc. 43rd Ann. Symp. on Foundations in Computer Science*, Vancouver, BC, pp. 271–280, November 2002.

[71] P. Maymoiunkov, *Online Codes*, New York University Technical Report, 2002.

[72] A. Shokrollahi, "Raptor codes," *IEEE Trans. Information Theory*, vol. 52, no. 6, pp. 2551–2567, June 2006.

[73] S. Abu-Surra, F. Khan, and C. Zhang, "Raptor code for multimedia broadcast/multicast service," *Samsung Tech. Conf.*, 2008.

Index