ERROR CORRECTION CODES FOR CHANNELS WITH INSERTION, DELETION AND SUBSTITUTION ERRORS

TIANBO XUE

MPhil

The Hong Kong Polytechnic University

2023

The Hong Kong Polytechnic University

Department of Electronic and Information Engineering

ERROR CORRECTION CODES FOR CHANNELS WITH INSERTION, DELETION AND SUBSTITUTION ERRORS

TIANBO XUE

A thesis submitted in partial fulfillment of

the requirements for the degree of

Master of Philosophy

November 2020

Certificate of Originality

I hereby declare that this thesis is my own work and that, to the best of my knowledge and belief, it reproduces no material previously published or written, nor material that has been accepted for the award of any other degree or diploma, except where due acknowledgment has been made in the text.

_____(Signed)

__Tianbo XUE__(Name of student)

Abstract

Error correction codes play an important role in protecting the integrity and accuracy of data transmitted in digital communication systems and data kept in storage systems. Besides substitution errors caused by noise, insertion and deletion errors may occur. Insertion and deletion errors are also categorized as synchronization errors because they affect the synchronization of a communication channel. Some recent studies have focused on coding against insertions and deletions due to some specific practical communication and storage applications. Unlike error correction codes for substitution errors, codes for insertion and deletion errors are limited. The thesis is therefore devoted to the study of error correction codes for channels impaired by insertion, deletion and substitution errors.

We first propose a new class of systematic comma-free code which can recover synchronization efficiently after insertion, deletion and substitution errors occur. The proposed code provides more choices in designing synchronization codes, and includes the classical F code as a special case. Then, we investigate codes that can correct insertion, deletion and substitution errors. We optimize the inner code of the classic Davey MacKay watermark code by proposing a soft-decision decoding based on a new metric and by enhancing the inner sparsified codebook based on better distance property and lowest-density property. We further propose a concatenated code composed of an RS outer code and a marker inner code. The proposed code not only is effective in correcting multiple insertion, deletion and substitution errors, but possesses a lower complexity decoding algorithm.

Finally, we propose codes for practical systems. We propose a probabilistic channel model with correlated insertion and deletion errors, in addition to substitution errors. The channel model captures the data dependence adapted to applications such as the write channel in bit-patterned media recording systems. We also investigate the performance of a concatenated code, consisting of an outer low-density parity-check code and an inner marker code, over this channel. DNA-based data storage systems have become a hot research topic recently. Here, we propose a systematic error correction code that can combat insertion, deletion and substitution errors. By combining this code with a proposed modulation scheme, we construct GC-balanced DNA sequences with error correction capability.

Throughout our investigations, computer simulations are used to evaluate the performance of the proposed error correction codes. Moreover, theoretical analyses are given whenever appropriate.

Publications Arising from the Thesis

Journal papers

- **T. Xue** and F. C. M. Lau, "Construction of GC-Balanced DNA With Deletion/Insertion/Mutation Error Correction for DNA Storage System," *IEEE Access*, vol. 8, pp. 140972–140980, 2020.

- **T. Xue** and F. C. M. Lau, "Generalized Systematic Comma-Free Code," *IEEE Access*, vol. 6, pp. 56800–56814, 2018.

- **T. Xue** and F. C. M. Lau, "A Concatenated LDPC-Marker Code for Channels with Correlated Insertion and Deletion Errors," submitted.

Conference papers

- **T. Xue** and F. C. M. Lau, "Concatenated Synchronization Error Correcting Code with Designed Markers," in *IEEE 5th International Conference on Computer and Communications (ICCC)*, Chengdu, China, 2019, pp. 1371–1376.

- **T. Xue** and F. C. M. Lau, "Codebook Design Optimization for Sparsified Distribution Converter in Davey-Mackay Watermark Codes over Channels with Synchronization Errors," in *25th Asia-Pacific Conference on Communications (APCC)*, Ho Chi Minh City, Vietnam, 2019, pp. 201–206.

- **T. Xue** and F. C. M. Lau, "A generalized systematic comma free code," in *23rd Asia-Pacific Conference on Communications (APCC)*, Perth, Australia, 2017, pp. 1–5.

Acknowledgments

It is a great honor to pursue and accomplish my doctorate degree at the Hong Kong Polytechnic University. It would have been impossible to complete this thesis without the help, guidance and support of many people. Therefore, I would like to take this opportunity to express my gratitude to them throughout my study at the Hong Kong Polytechnic University.

First of all, I especially would like to express my sincere and deepest gratitude to my supervisor, Prof. Francis. C. M. Lau for his patient guidance, warm encouragement, constant support and valuable advice. I am inspired by his enthusiasm in research and insights into the area of communication. I especially thank him for encouraging me to think deeply especially when I encounter difficulties in my research. I am also thankful for his generous time, listening carefully, criticizing fairly, providing directions and correction for my research papers. Without his help, I could not have accomplished my Ph.D. study.

I also thank all my colleagues in our group for numerous helpful discussions regarding the topic of channel coding. I have benefited immensely and broadened my horizon from these discussions.

I acknowledge the financial support of the Hong Kong PhD Fellowship Scheme (HKPFS) for the financial sponsorship to provide me the precious opportunity to live comfortably, attend international conferences and focus on my research during the entire period of my study.

Above all, I am very grateful to my brother Guangliang, who gave up the career in

United States and come to Hong Kong to accompany me. He always encourages me during the most difficult time. I would like to thank my parents for their unconditional love, encouragement, support and all invaluable contributions to my life. Their love, care, and support make me stronger.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

We summarize the theme and content of the thesis in this introductory chapter. Specifically, we describe the importance and necessity of channel coding in a communication system in Sect. 1.1, followed by the aspect of synchronization and error types in Sect. 1.2 and motivation in Sect. 1.3. Finally, we present the organization of the thesis in Sect. 1.4.

1.1 Channel Coding

Artificial satellites have been transmitting data back to earth from space for decades. How can data be transmitted reliably across millions of miles without being overwhelmed by noise? This is achieved by transmitting data across noisy channels with error correction codes.

We live in a data-driven modern world, with massive data being generated, transmitted, stored and processed at an amazing speed. Error free communication is possible in the ideal perfect communication channels without noise. However, noise always exists and introduces errors in practice. In a digital communication system, information may be corrupted and received incorrectly because of noisy channel. Several techniques are used to reduce errors and to minimize the effects of noise, such as re-

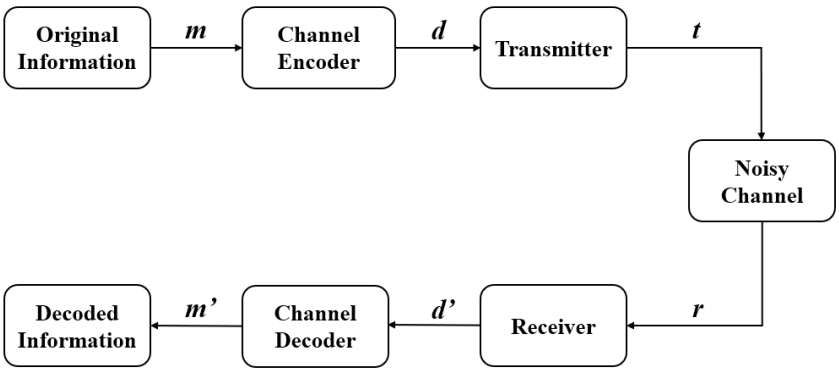


Figure 1.1: Schematic of a simplified communication system.

ducing the transmission speed, increasing the signal power, adopting automatic repeat request (ARQ). Among all techniques for protecting information, the most frequently used one is channel coding.

The block diagram of a simplified communication system is illustrated in Fig. 1.1. The communication channel is used to transmit information from the transmitter to the receiver via a physical or broadcast medium. Channel coding ensures information is transmitted efficiently and reliably to the receiver. The basic idea of channel coding is that the receiver and transmitter agree on an error correction coding scheme, in which extra bits (redundancy) are added to the original message. Errors that occurred during transmission can be detected and corrected by the receiver based on the redundant bits.

In 1948, Claude Shannon defines the notion of channel capacity, an upper bound on the rate that information can be transmitted reliably over a communication channel [2], i.e.,

$$C = B\log_2(1 + \frac{S}{N}) \tag{1.1}$$

where C denotes the channel capacity, B represents the channel bandwidth, and S/N denotes the received signal-to-noise ratio. The significance of (1.1) is that a reliable transmission over a noisy channel can be achieved with arbitrary small error probability if the information rate is below the channel capacity C. Good error correction codes usually correct the maximum number of errors while maintaining the minimum

redundancy.

Many error correction codes have been proposed since the pioneering paper by Shannon in 1948, which established the basis of classical coding theory. In late 1940s, Richard W. Hamming constructed the single error correction Hamming block code in [3] and Golay introduced another block error correction code in [4]. The Reed Muller code [5] proposed in 1954 was relatively easy to decode and was used in the field of Mars exploration in the 1970s. Bose, Ray-Chaudhuri and Hocquenghem proposed the BCH code in [6]. The convolutional code was created by Peter Eliasin in 1955 [7]. In 1960, Reed and Solomon proposed the Reed-Solomon code, which could correct multiple errors with good error performance [8]. The RS code can also correct burst errors and have been widely used in space communication, wireless communications and CD-ROMS. In 1966, Forney proposed the idea of serial concatenation of codewords [9], where long codewords with improved error correction capability can be constructed by concatenating two short codes. In 1993, turbo code was proposed in [10] [11], which made capacity-approaching codes possible. Low-density parity-check (LDPC) codes were first introduced by Gallager in his doctoral dissertation [12] and were rediscovered by Mackay and Neal in [13]. It has been proved that LDPC codes have theoretical limits approaching the channel capacity. Subsequently, LDPC codes have been widely applied and studied in the past two decades.

1.2 Aspect of Synchronization

Many factors affect the integrity and accuracy of information transmission and reception. In addition to common substitution errors, another influencing factor is insertion and deletion errors. At this stage, we consider three types of errors throughout this thesis, namely, insertion, deletion and substitution errors as shown in Fig. 1.2. We define three types of errors as follows and assume a data sequence proceeding from left to right.

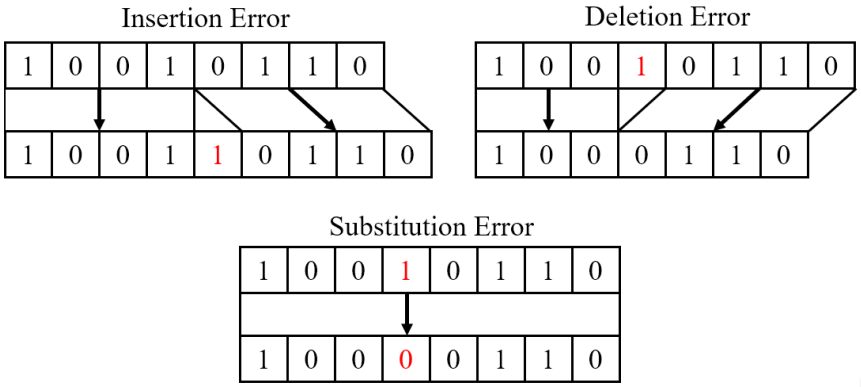


Figure 1.2: Three types of errors in a communication system.

- An insertion is defined as the detection of an un-transmitted bit at the receiver. All the subsequent bits shift one position to the right in the sequence.

- A deletion is defined a transmitted bit not detected at the receiver. All the subsequent bits after the un-detected bit shift one position to the left in the sequence.

- A substitution is defined as the replacement of a transmitted bit with a different one and the length of the sequence remains unchanged.

Synchronization is an important and integral part of a digital communication system. Synchronization errors (i.e., insertion and deletion) occur when the receiver and transmitter are not synchronized perfectly due to uncertainties in timing or time noise. As a result, random symbols may be inserted, or transmitted symbols may be deleted in the received sequence. In general, the length of the transmitted sequence is a constant parameter while the length of the received sequence is a random variable. The length of the received sequence is either lengthened or shortened depending on the predominant insertion or deletion errors. An equal number of insertion and deletion errors result in the received sequence having the same length as the transmitted sequence. This case is equivalent to having a burst of substitution errors and the boundary of the codeword is not affected.

Synchronization errors bring difficulties that other types of errors would not en-

counter. Even a single insertion or deletion causes the loss of synchronization in the subsequent sequence until the receiver gets resynchronized, which leads to a burst of substitution errors. In addition, the positions of synchronization errors are unknown to both the receiver and transmitter.

Long messages are usually divided into multiple blocks before they transmit through the channel and each block consists of several codewords. In general, there are three levels of synchronization. At the most basic level, the receiver should have frequency or phase synchronization with the carrier signal. The next level is codeword synchronization, where the interval of each codeword can be accurately aligned with the interval in the carrier. A higher level is block synchronization and it is required by most communication systems.

Accordingly, two main synchronization problems on channels with insertion and deletion need to be solved.

1. The decoder cannot recover synchronization because the boundaries of a codeword or a block cannot be determined due to potential insertion or deletion errors.

2. The original message cannot be decoded because insertion, deletion and substitution errors introduced by the channel cannot be corrected.

1.3 Motivation

Modern communication and storage systems increasingly rely on the synchronization issue. Some recent studies have focused on coding against synchronization errors due to many recent practical applications, such as bit-patterned media recording (BPMR) systems [14], image watermarking [15] [16], video digital watermarking [17], and deoxyribonucleic acid (DNA) based data storage [18–24]. Synchronization errors occur not only in physical channels, but also in data storage systems. In the case of data storage systems, the write operation can be regarded as a transmitter while the read operation can be regarded as a receiver. For example, systems that store data as DNA

nucleotides have been realized recently [18–24]. However, nucleotides used for storing data may be added, removed or substituted during the synthesis (encoding) and sequencing (decoding) processes.

On channels with feedback, the decoder can just request the transmitter to resend the message when synchronization is lost. However, on channels without feedback, the system is required to regain synchronization with the use of error correction codes. Therefore, coding for channels corrupted by insertion, deletion and substitution errors is necessary in communication and data storage systems.

The error correction codes mentioned in Sect. 1.1 are used to correct substitution errors, in which a transmitted symbol is received as a different one. The boundaries of each codeword are supposed to be known by the decoder and the synchronization issue does not need to be considered. Thus, they cannot be applied directly over channels with synchronization errors, which result in a gain or loss of transmitted data. On the other hand, there has been insufficient understanding of insertion and deletion error correction codes.

Motivated by the lack of efficient synchronization error correction coding schemes and its demand on a number of practical applications, this thesis aims to advance channel code designs for channels impaired by insertion, deletion and substitution errors.

1.4 Thesis Organization

In Chapter 2, a literature review of coding techniques to combat synchronization errors is presented. We first review three classical channel models impaired by insertion, deletion and substitution errors. Then, we review two categories of codes for such channels. One category of codes aims to recover synchronization but possesses no error correction capability while the other aims to correct synchronization errors and to recover the transmitted messages. In each of the Chapter 3 to Chapter 7, we begin with a brief introduction and an overview of the related works, followed by our contribution

and results.

In Chapter 3, we start our investigations. We focus on a family of comma-free codes, which can recover synchronization efficiently after insertion/deletion/substitution errors occur. We propose a type of generalized systematic comma-free code for channels impaired by insertion, deletion and substitution errors. We also derive the probability of false synchronization caused by a single substitution, insertion or deletion. In addition, we compare the theoretical and simulation results under different channel parameters, and analyze the factors affecting false synchronization in each case.

In Chapter 4, we study the classical Davey Mackay watermark code that can correct multiple insertion, deletion and substitution errors. We optimize the inner sparsified codebook based on both distance property and lowest density property. We also propose a hard-decision decoding based on Hamming distance and a soft-decision decoding based on a new metric for the inner symbol-level watermark decoder. Simulation results verify that the system with the improved codebook design provides better performance in terms of symbol error rate over channel with random insertion, deletion and substitution errors.

In Chapter 5, we propose an efficient concatenated RS-marker code with designed markers. The designed markers allow the inner decoder to maintain synchronization effectively at codeword boundaries while the outer RS code provides the error correction capability. Simulation results show that the proposed concatenated code is effective in correcting multiple insertion, deletion and substitution errors with the use of a low complexity and simple decoding algorithm.

Chapters 4 and 5 focus on error correction codes for channels with random insertion, deletion and substitution errors. In some practical scenarios, insertions and deletions are correlated. In Chapter 6, we propose a probabilistic channel model with correlated insertion and deletion (CID) errors, in addition to substitution errors. Furthermore, we investigate the performance of a concatenated LDPC-marker code over this channel. The concatenated code consists of an inner marker code used to maintain

synchronization and an outer low-density parity-check (LDPC) code to provide error correction capability. The forward-backward marker decoding algorithm is elaborated based on a two-dimensional state transition diagram. Simulation results show that the proposed concatenated code is effective in combating CID channel with substitution errors.

Chapter 7 constructs a GC-balanced DNA sequence with error correction capability for DNA-based storage systems. We propose a systematic single insertion/deletion/substitution error correction code and apply it to design a GC-balanced scheme for constructing DNA sequences. The proposed GC-balanced DNA sequence not only contains exactly 50% GC content, but also can correct insertion, deletion and mutation of nucleotide bases. Simulation results show that the proposed GC-balanced DNA sequences can correct base errors adequately.

Finally, the thesis concludes in chapter 8, where major contributions are summarized, and future work of the research is presented.

Chapter 2

Literature Review

2.1 Introduction

A good error detection and correction system requires two components: a precise definition of the communication channel with error characteristics and an analysis of properties of codes that permit error detection or correction for channels. In this chapter, we review literature related to error correction codes for channels with insertion, deletion and substitution errors to provide the necessary background for the rest of the thesis. Specifically, we start with three classical channel models impaired by random insertion, deletion and substitution in Sect. 2.2. Then we review literature related to synchronization recovery codes including comma-free codes and two classical systematic comma-free codes, Gilbert code and F code in Sect. 2.3. Finally, we discuss different types of synchronization error correction codes that are commonly used in Sect. 2.4.

2.2 Channel Models with Random Insertion, Deletion and Substitution Errors

A reasonable communication channel model with characteristics of insertion, deletion and substitution errors is compulsory. We review three classical channel models with random errors in this section.

2.2.1 Gallagar Channel

In [25], Gallagar has proposed a binary channel where each transmitted bit through the channel experiences one of the four scenarios: each bit is either received correctly with probability p_c, deleted with probability p_d, replaced by two uniformly distributed bits (insertion error) with probability p_i, or substituted with probability p_s. The relationship between these parameters is given by $p_c = 1 - p_i - p_d - p_s$. It is noted that the Gallagar channel does not allow insertion and deletion occur together for a transmitted bit.

2.2.2 Zigangirov Channel

Zigangirov considers a channel without substitution errors and any number of bits can be deleted or inserted during the transmission. The probability of no insertion errors is q_1 and the number of i insertions has a probability of $q_1 p_1^i$. Thus $\sum_{i=0}^{\infty} q_1 p_1^i = q_1 + q_1 p_1 + q_1 p_1^2 + \cdots = 1$ because the sum of the probabilities of all possible events is equal to one. Since $\sum_{i=0}^{\infty} q_1 p_1^i = q_1/(1 - p_1) = 1$, $p_1 = 1 - q_1$ represents the error probability that at least one bit is inserted. Denoting q_2 as the probability that a transmitted bit is not deleted, then $p_2 = 1 - q_2$ represents the probability that the transmitted bit is deleted.

2.2.3 BSID Channel

The channel model with random independent insertion, deletion and substitution errors proposed by Davey and MacKay [26] is depicted in Fig. 2.1 and the flow chart of this

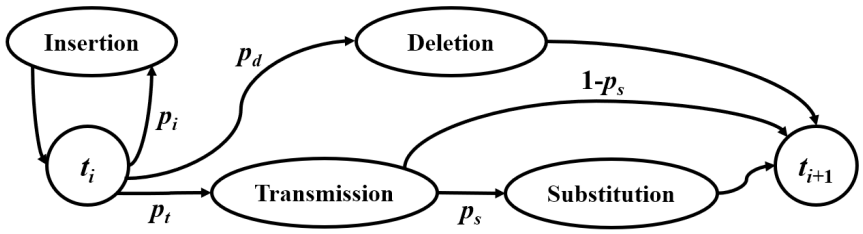


Figure 2.1: BSID channel model.

channel is shown in Fig. 2.2. For the binary case, this channel is also referred to as binary substitution/insertion/deletion (BSID) channel [27].

The BSID channel can be mathematically described by three parameters: p_d, p_i and p_s, representing a deletion error rate, an insertion error rate and a substitution error rate, respectively. It can be regarded as a binary symmetric channel (BSC) with synchronization errors. At time i, a transmitted bit t_i enters the channel and experiences (a) a random bit is inserted before t_i with probability p_i; or (b) t_i is deleted with probability p_d; or (c) t_i is transmitted with probability $p_t = 1 - p_i - p_d$. In addition, the transmitted bit may suffer from a substitution error with probability p_s. Considering a state i, it returns to the state i when an insertion occurs and moves to the next state $i + 1$ only when the current bit is either deleted or transmitted. In this channel, each transmitted bit produces a sequence of received bits with length varying between 0 and $I + 1$; a length of 0 corresponds to the deletion of the current bit and a length of $I + 1$ corresponds to a maximum of I consecutive insertions followed by a transmission.

Note that the BSID channel does not restrict the maximum number of consecutive insertions for each transmitted bit. After an insertion error has occurred, the current bit and hence the current transmission state will be considered again. The "overall" probability of insertion errors for a transmitted bit is $p'_i = p_i + p_i^2 + p_i^3 + \cdots = \frac{p_i}{1-p_i}$, and $p'_i \approx p_i$ is valid for a small value of p_i. The next bit is considered only when a deletion error or transmission occurs. The probability of the current transmitted bit being substituted is $p'_s = (1 - p_i - p_d) \times p_s$. Similar to p'_i, $p'_s \approx p_s$ when p_i and p_d are

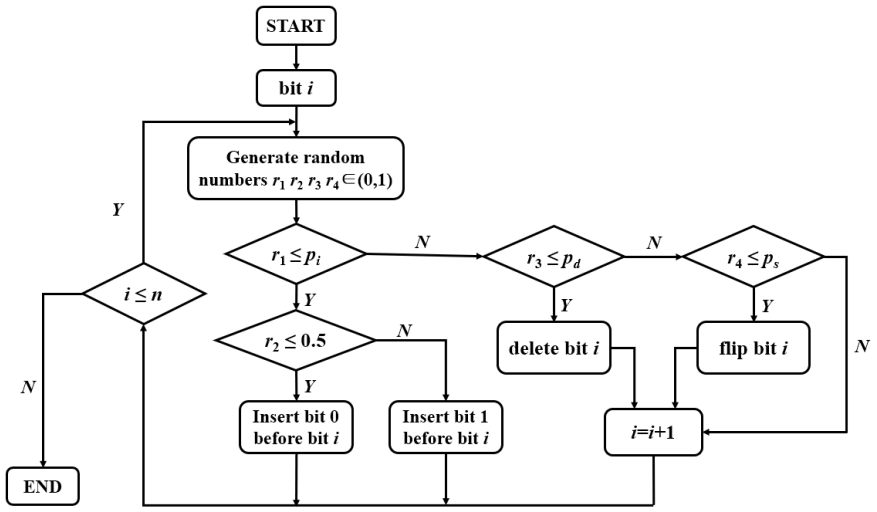


Figure 2.2: BSID channel with insertion error probability p_i, deletion error probability p_d and substitution error probability p_s. r_1,r_2,r_3,r_4 are random generated numbers and n is the total length of the transmitted sequence.

values much less than 1. The operation repeats until all bits are sent.

2.3 Coding for Recovering Synchronization after Insertion/Deletion/Substitution Errors Occur

Extensive research has been done in the field of insertion/deletion/substitution error detection and correction codes and a survey is provided in [28]. In general, there are two categories of codes. One is a set of synchronization recovery codes. The other category consists of codes that can correct insertion/deletion/substitution errors.

We summarize codes that can recover synchronization after insertion, deletion and substitution errors occur in this section. The significance of this category of codes is that they can determine the codeword boundaries efficiently and recover synchronization after insertion/deletion/substitution errors. However, they cannot correct errors or can only correct them with limited capability by combining with other error correction codes. Codewords corrupted by insertions, deletions or substitutions in this category are usually discarded.

2.3.1 Synchronization Code

Synchronization codes are constructed in such a way that the tail sequence of each codeword forms the separation between codewords. A formal definition has been given in [29] [30] and is shown below.

Definition 1 *A finite code is called synchronizable if, and only if, there exists an integer M such that the knowledge of the last M letters of any message suffices to determine a separation of codewords.*

In [29], a technique for constructing synchronization codes containing the maximum number of codewords is presented. The codes can detect errors after receiving several codewords. Other codes that are designed to recover synchronization after errors occur include prefix-synchronized codes [31], codes with bounded synchronization delay [32], synchronization with timing codes [33], and synchronization codes with the designed suffix construction [34]. More literatures on synchronization codes and their applications can be found in [35–38].

2.3.2 Comma-free Code

Stricter restrictions imposed on synchronization codes result in comma-free codes (CFC). A comparison between codes that utilize a comma and comma-free codes is presented in [39], showing that comma-free codes are more efficient in maintaining synchronization in terms of redundancy. Comma-free codes were first proposed in [40]. They form a subset of synchronization codes and need to fulfill the additional condition that the overlap between two consecutive codewords does not form another codeword. Constructions of such codes have been discussed by Gilbert [41], Golomb *et al.* [40] [42] [43], Jiggs [44] and Eastman [30].

Definition 2 *A comma-free code (CFC) is a set of codewords of length n over an alphabet such that given any two codewords* $\boldsymbol{u} = u_1 u_2 \cdots u_n$ *and* $\boldsymbol{v} = v_1 v_2 \cdots v_n$ *be-*

longing to this set, the n letter concatenation $\boldsymbol{w} = u_k \cdots u_n v_1 \cdots v_{k-1}$ $(k = 2, 3, \ldots, n)$ *cannot construct a codeword.*

Examples of comma-free codes with a codeword length of 5 are {01000, 01100, 01010, 01110, 01011, 01111} and a codeword length of 7 are {0101000, 0101100, 0101110, 0101111, 0100010, 0110010, 0111010, 0111110, 0100011, 0110011, 0111011, 0111111, 0100000, 0110000, 0111000, 0111100, 0111110, 0111111}. From a mathematical point of view, the upper bound of the maximal number of codewords in a CFC set, given a codeword length n and an alphabet size q, has been derived in [40] [44] and is given by

$$W_n(q) \leq \frac{1}{n} \sum_{d|n} \phi(d) q^{n/d}, \tag{2.1}$$

where the summation is extended over all divisors d of n, and $\phi(d)$ is the Mobius function defined as

$$\phi(d) = \begin{cases} 1 & \text{if } d = 1, \\ 0 & \text{if } d \text{ has any square factor,} \\ (-1)^r & \text{if } d = p_1, p_2, \cdots, p_r, \text{ where} \\ & p_1, p_2, \cdots, p_r \text{ are distinct primes.} \end{cases} \tag{2.2}$$

Following CFC, Golomb *et al.* first show how to construct the maximal cardinality of CFC for codeword lengths of 3, 5, 7 and 9 in [40]. They further prove that the construction can be extended to codeword length of 11, 13 and 15. In [30], Eastman has provided an approach to construct a CFC with the maximum codewords for any odd codeword length over any alphabet size. They allow the receiver to re-establish synchronization after an insertion or deletion error with a delay of at most two blocks. In [45], a method to construct a CFC with the maximum number of codewords and a specific procedure to construct CFC with variable lengths have been presented. In [46], a subclass of comma-free codes with a fixed length, called path invariant comma-free

codes, has been constructed. Even though such codes are relatively easy to encode and decode, they are not very efficient. The prefixed comma-free code, where a comma is inserted within every codeword, has been proposed [31].

2.3.3 Systematic Comma-free Code

The aforementioned codes do not provide specific positions for transmitting the information bits. A systematic comma-free code is a subclass of comma-free codes in which some fixed locations are used to maintain synchronization and the remaining locations are used to carry information. The systematic fixed-length block comma-free codes are investigated in [47]. An effective method to construct a systematic comma-free code is to fix the first few bits of a codeword to a sequence of bit 1 or bit 0. The fixed sequence is usually referred to as the primer drive and it appears at the beginning of each codeword. Two classical systematic CFCs are given in the following.

Gilbert Code

The primer drive of a Gilbert code consists of s bit zeros ("0"s) and is placed as a prefix within each codeword. Each subsequent sequence of s bits begins with bit one ("1") and the last bit is also bit one ("1"). Other positions are reserved to carry information. Gilbert code ensures that the primer drive does not appear in the body of each codeword. For example, Gilbert code with parameters $n = 16$ and $s = 4$ encodes the information sequence "10110110" into the codeword "$\mathbf{0000}1101\mathbf{1}1011\mathbf{1}101$", where the bold bits represent the fixed bits used to maintain synchronization. Obviously, even if the information bits are all zeros, the primer drive containing 4 zeros will never appear in the body of a Gilbert codeword. It is required that no primer drive can appear in the body of a Gilbert codeword and thus, the efficiency and code rate is relatively low due to this relatively strong condition.

F Code

A more efficient systematic comma-free code called "F code" is introduced in [47]. F code allows the primer drive to appear in the body of the codeword, but it ensures that no codeword is formed between the concatenated codewords.

We denote n as the codeword length and r as the smallest integer larger than or equal to $\sqrt{2(n-1)}$. We also denote s as the smallest integer larger than or equal to $\frac{r}{2}$, and $t = r - s$. An F code of length n has 0s fixed in the positions $1, 2, \ldots, s$ and 1s fixed in the positions $n, n-s, n-2s, \ldots, n-(t-1)s$ while all other positions are arbitrary and can be used to transmit information. For example, when $n = 15$, the information "101101100" is encoded into an F code "**000**10110**1**111**0**0**1**" where the digits in bold represent those fixed bits used to maintain synchronization.

The F code proposed by Clague is similar to the Gilbert code, but the condition for maintaining synchronization is relatively weak. The F code is more efficient in terms of carrying information compared with the Gilbert code because some fixed positions of a Gilbert code become arbitrary bits in an F code of the same length. The difference becomes more obvious as the codeword length increases. In fact, it has been proved that the F code is the most efficient systematic CFC that uses fixed positions to maintain synchronization [47].

2.3.4 Synchronization Code with Limited Error Correction Capability

The next logical step after developing synchronization codes and comma-free codes is to incorporate error correction capability into these codes. A family of codes that can correct at most a single substitution or synchronization error in every three consecutive codewords are constructed in [48]. By combining the comma-free code with the cyclic error correction code, a synchronization code with substitution error correction capability is constructed in [49]. Moreover, by adding a modification vector to the cyclic

error correction code, a comma-free code with error correction capability is proposed in [50].

2.4 Coding for Correcting Insertion/Deletion/Substitution Errors

We briefly summarize codes that are capable of correcting insertion/deletion/substitution errors in this section. The significance of such codes is that they can not only regain synchronization but also correct errors in the corrupted codeword.

2.4.1 Marker Codes

An approach to detecting and correcting synchronization errors is to insert periodic "markers" between codewords. In [51], a substring "001" is inserted into a burst error correction code at regular intervals to detect and correct a single insertion or deletion error. The markers are used to locate the interval where a synchronization error has occurred. A single bit is deleted or inserted within the interval to recover the codeword length and the resulting burst of substitution errors are corrected by the outer burst error correction code. Moreover, more synchronization errors can be corrected by using longer markers. Marker codes are also used to correct synchronization errors over wireless infrared channels [52]. Other codes using markers are discussed in Sect. 2.4.3.

2.4.2 Algebraic Block Codes

In [53], Varshamov and Tenegolts construct a class of binary code called VT code, which is capable of correcting one asymmetric substitution error, i.e., the substitution error probability of bit zero is higher than that of bit one, or vice versa over the Z-channel. Moreover, VT code is a class of code that can correct a single insertion, deletion or substitution error. We assume n and a are both positive integers satisfying

$0 \leq a \leq n$. The $VT(n, a)$ code contains all binary codewords of length n satisfying

$$VT(n, a) = \{(c_1, c_2, \cdots, c_n) \in \{0, 1\}^n : \sum_{i=1}^{n} i \cdot c_i \equiv a \mod (n + 1)\}. \tag{2.3}$$

Levenshtein has extended $VT(n, a)$ to the asymptotically optimal single error correction Levenshtein code $L(n, a, U)$ [54]. Levenshtein has shown that codes with length n satisfying

$$L(n, a, U) = \{(c_1, c_2, \cdots, c_n) \in \{0, 1\}^n : \sum_{i=1}^{n} i \cdot c_i \equiv a \mod U, \quad U \geq 2n\}, \tag{2.4}$$

can correct a single insertion, deletion or substitution error for any $U \geq 2n$ and $0 \leq a \leq U - 1$. In (2.4), $\boldsymbol{c} = (c_1, c_2, \cdots, c_n)$ is a Levenshtein codeword, n is the codeword length and a is the integer corresponding to the code partition. Levenshtein has also shown that the number of codewords satisfying (2.3) is asymptotically $\frac{2^n}{n}$. Levenshtein also demonstrated that the code has a minimum Hamming distance of at least 3 and is capable of correcting a single synchronization or substitution error per codeword.

In [55], Tenengolts has proposed a family of codes that can correct a deletion or two consecutive substitution errors given the boundaries of each codeword. He has extended his work to nonbinary codes that can correct a single insertion or deletion over a nonbinary alphabet in [56]. In [57], Hollman has derived the relationship between Levenshtein distance metric [54] and insertion/deletion error correction capability. Bours has proposed a code that can correct one deletion or two consecutive deletions, and has derived the bounds for codes that can correct insertions and deletions in [58].

Algebraic block codes with single error correction are surveyed in [59]. Single error correction codes, such as VT codes and Levenshtein codes cannot be generalized to correct multiple synchronization errors in a simple way. In [60], a code that can correct up to five synchronization errors has been constructed by extending the VT code. However, the code rate is extremely low and no explicit decoding algorithm is

provided.

2.4.3 Concatenated Codes

Concatenated codes use an inner code to maintain synchronization and an outer code to provide error correction capability. In [61], a concatenation of a Reed-Solomon code and a designed code using a brute force approach is used to achieve exponential error probability in a channel with random insertion and deletion errors. A sub-optimal code concatenating an LDPC code, a marker code and a Varshamov-Tenengolts (VT) code is designed in [62] to correct multiple deletion errors. The code achieves a bit error rate below 10^{-4} for channels with random deletion errors and the code rate is 0.21. Davey-Mackay (DM) watermark code is able to correct multiple substitutions, deletions and insertions based on a Hidden Markov Model (HMM) [26]. Based on the DM watermark code, a similar concatenated LDPC code with markers is shown to outperform the watermark code at low synchronization error rates [63]. In [64], the performance of the DM watermark code is further improved by using a slide decoding for a modified LDPC parity check matrix. In [65], an adaptive synchronization marker code based on the neighborhood of codewords is proposed to improve the synchronization capability of the original DM watermark code. In [66], an error correction method using fixed symbols at some bit positions between the marker codes has been proposed to enhance the synchronization and decoding performance of the watermark code.

2.4.4 Davey MacKay (DM) Watermark Code

The Davey Mackay (DM) watermark code [26] can correct multiple insertion, deletion and substitution errors over channels with random error rates. Many error correction codes with good performance are based on the DM construction. The structure of DM construction is shown in Fig. 2.3. It uses an LDPC code as the outer code and a predefined known watermark as the inner code. In this scheme, an LDPC-coded

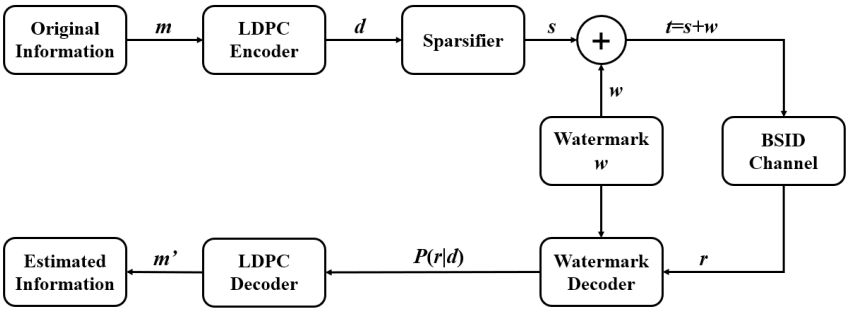


Figure 2.3: Structure of DM Construction.

sequence is mapped to sparse (low-density) sequences which are modulo-2 added to a predefined known watermark sequence. The watermark decoder compares the received sequence with the known watermark sequence to recover synchronization and returns symbol-by-symbol log-likelihood ratios (LLRs) to initialize the outer LDPC decoder. The outer LDPC decoder subsequently corrects the remaining errors from the inner watermark decoder.

The information sequence $\boldsymbol{m} = \{m_0, m_1, \ldots, m_{K_L-1}\}$ of length K_L q-ary symbols is first encoded into a sequence $\boldsymbol{d} = \{d_0, d_1, \ldots, d_{N_L-1}\}$ with a length of N_L q-ary symbols using an outer q-ary (N_L, K_L) LDPC code defined over Galois Field $GF(q)$, where $q = 2^k$. Then, each q-ary symbol d_i of $\boldsymbol{d}$ is transformed into a sparse binary sequence $s_i = \{s_{n\times i}, s_{n\times i+1}, \ldots, s_{n\times(i+1)-1}\}$ of length n, where $i = 0, 1, \ldots, N_L - 1$, using a non-linear low-weight binary code for some $n > k$. The mapping rule is to select $q = 2^k$ lowest density (Hamming weight) binary vectors of length n. Thus, the sequence $\boldsymbol{d}$ is converted to a sparse sequence $\boldsymbol{s} = \{s_0, s_1, \ldots, s_{N-1}\}$ of length N, where $N = nN_L$. The transmitted sequence $\boldsymbol{t} = \{t_0, t_1, \ldots, t_{N-1}\}$ of length N is then generated by adding modulo-2 the sparse sequence $\boldsymbol{s}$ to a binary watermark sequence $\boldsymbol{w}$ with the same length and is prepared for transmission through the channel. Besides, the watermark sequence is known to both the encoder and decoder. The rate of the concatenated code R is $\frac{k\cdot K_L}{n\cdot N_L}$. The specific encoding of the concatenated code is shown in Fig. 2.4.

A distribution converter is a code that transforms sequences of one form to another

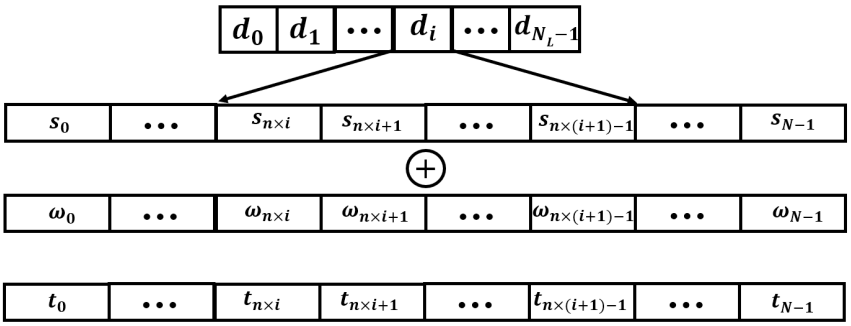


Figure 2.4: Encoding procedure of concatenated code in DM construction. Each symbol d_i is represented by the addition of watermark sequence and sparsified sequence.

[67]. A sparsified distribution converter (sparsifier) with parameters k and n maps every k bits to one of the 2^k sparse sequences of length n from a total of 2^n sequences. The sparse code causes minimal changes to the watermark sequence to ensure the decoder can track synchronization. When the sparsified sequence $\boldsymbol{s}$ is added modulo 2 to the watermark sequence, the 1s in $\boldsymbol{s}$ will flip the corresponding watermark bit. Because the percentage of 1s in the sparse code is limited, the output of the channel is mostly likely to be the watermark $\boldsymbol{w}$, and the changes caused by the sparse code are treated as substitution errors. Thus, a sparse code increases the synchronization probability of the decoder with the watermark sequence because a large bias will exist between the received sequence and the known watermark sequence when the inner decoder is not synchronized.

The transmission of the modified watermark sequence through the BSID channel can be represented as a trellis. The synchronization drift state x_i at the ith position is defined as the difference between the number of insertions and deletions that have occurred from the start transmitted bit t_1 until bit t_i is ready to be transmitted. The noisy received sequence $\boldsymbol{r}$ is decoded by operating an optimal symbol-level forward-backward (FB) algorithm on the trellis [68]. Symbol-level decoder tracks each symbol at a time by considering the codeword set. The FB algorithm is applied at the boundaries of each symbol and the aim is to output the likelihood of each possible transmit-

ted symbol $P(\boldsymbol{r}|d_i)$ with knowledge of the sparse bias level, channel parameters and the known watermark sequence based on Hidden Markov Model (HMM). Every symbol and the corresponding boundaries are identified by comparing the received sequence with the predefined watermark sequence. The likelihood of each possible transmitted symbol $P(\boldsymbol{r}|d_i)$ is given by

$$P(\boldsymbol{r}|d_i) = \sum_{x_{n\times i}, x_{n\times(i+1)}} F(n\times i, x_{n\times i}) \cdot P(\boldsymbol{r}_{n\times i+x_{n\times i}}^{n\times(i+1)+x_{n\times(i+1)}-1}, x_{n\times(i+1)}|x_{n\times i}, d_i) \cdot B(n\times(i+1), x_{n\times(i+1)}), \tag{2.5}$$

The symbol-level forward quantity $F(n \times i, x_{n\times i})$ is the probability that the first $n \times i + x_{n\times i}$ bits are in coordination with the received vector $\boldsymbol{r}$ when the state drift at $n \times i$ is y, i.e., $x_{n\times i} = y$. It is calculated by summing over all possible previous states and all possible transmitted symbols at the previous state and is given by

$$\begin{aligned} F(n \times i, y) &= P(\boldsymbol{r}_1^{n\times i+y}, x_{n\times i} = y) = P(r_1, r_2, \ldots, r_{n\times i+y}, x_{n\times i} = y) \\ &= \sum_{d_{i-1}, a} F(n \times (i-1), a) \cdot P(\boldsymbol{r}_{n\times(i-1)+a}^{n\times i+y-1}, x_{n\times i} = y|x_{n\times(i-1)} = a, d_{i-1}). \end{aligned} \tag{2.6}$$

The backward quantity $B(n\times(i+1), x_{n\times(i+1)})$ is the probability that $(r_{n\times(i+1)+x_{n\times(i+1)}}, \ldots, r_\rho)$ is received given a drift of $x_{n\times(i+1)}$ at position $n\times(i+1)$. Similarly, it is calculated based on all subsequent states and possible transmitted symbols at the subsequent state and is given by

$$B(n \times i, y) = \sum_{d_i, b} B(n \times (i+1), b) \cdot P(\boldsymbol{r}_{n\times i+y}^{n\times(i+1)+b-1}, x_{n\times(i+1)} = b|x_{n\times i} = y, d_i). \tag{2.7}$$

Even though the DM watermark code can correct multiple insertion, deletion and substitution errors, it has extremely high computational complexity [69]. The inner watermark decoder performs the forward-backward algorithm on a large trellis based

on a complicated Hidden Markov Model (HMM) in order to calculate the LLRs. The scale of the trellis is proportional to frame size and error rates from the channel. The scale becomes larger and larger as the number of synchronization errors increases. In fact, the complexity of the inner watermark code can be as large as $O(XNI)$, where X is the number of hidden states in the HMM, N is the length of hidden sequence and I is the maximum length of a burst of insertions.

2.5 Summary

This chapter provides a literature review on coding techniques for channels impaired by insertion, deletion and substitution errors. We first review three classical channel models with insertion, deletion and substitution errors. Then, we provide a survey of codes that are used to recover synchronization after insertion/deletion/substitution errors occur. We also review some existing results on codes that can correct insertion/deletion/substitution errors, including marker codes, algebraic block codes, and concatenated codes. In particular, we discuss the encoding and decoding of the classical DM watermark code in detail. In the next chapter, we start our investigation by focusing on a family of systematic comma-free codes.

Chapter 3

A Type of Generalized Systematic Comma-Free Code

Synchronization between codewords is easily lost when insertion and deletion together with substitution errors occur. Recovering synchronization after insertion/deletion/substitution errors is of great importance to the receiver which needs to decode the subsequent codewords correctly. In Sect. 2.3, we have reviewed codes that are capable of recovering synchronization after insertion/deletion/substitution errors. Moreover, comma-free codes are more efficient in terms of redundancy compared with codes with comma [39].

A systematic comma-free code is a subclass of comma-free codes and it provides specific fixed positions for transmitting data. In a systematic comma-free code, some positions are fixed and used to maintain synchronization and the remaining positions are used to transmit information. To construct a systematic comma-free code, the first few bits of a comma-free code are usually fixed to a sequence of 1s or 0s. The fixed sequence is referred to as the primer drive and it appears at the beginning of each codeword.

In fact, F code has been proved to be the most efficient systematic CFC that uses fixed places to maintain synchronization [47]. We denote n as the code length. By

definition of F code, r is the smallest integer larger than or equal to $\sqrt{2(n-1)}$, s is the smallest integer larger than or equal to $\frac{r}{2}$ and $t = r - s$. Recall that an F code of length n has 0s fixed in the positions $1, 2, \ldots, s$ and 1s fixed in the positions $n, n - s, n - 2s, \ldots, n - (t-1)s$ while all other positions are arbitrary and can be used to transmit information. Further, F code allows the primer drive to appear in the body of a codeword, but it ensures that no codeword is formed between the concatenated codewords.

In general, the concatenation of two consecutive CFCs cannot form a valid codeword. However, the decoder would give a false synchronization if a "valid" codeword appears in the concatenation of two consecutive CFC codewords due to insertion, deletion or substitution errors. Such a "valid" codeword is referred to as a false codeword.

In this chapter, we propose a new type of systematic comma-free code called the generalized $F(n, s, t)$ code which includes F code as a special case [70, 71]. The proposed code can recover synchronization after insertion/deletion/substitution errors occur. We also derive the false synchronization probability caused by a single substitution, insertion and deletion error. The rest of the chapter is organized as follows. In Sect. 3.1, we show the construction of the generalized $F(n, s, t)$ code and prove that it is a systematic comma-free code. We also show the encoding and decoding of the proposed generalized $F(n, s, t)$ code. In Sect. 3.2, we derive the probability of false synchronization caused by a single substitution, insertion or deletion error. In Sect. 3.3, the performance of the proposed code in terms of false synchronization probability are shown. Sect. 3.4 summarizes the chapter.

3.1 Generalized $F(n, s, t)$ Code

3.1.1 Proposed generalized $F(n, s, t)$ Code

For a codeword of length n, a generalized $F(n, s, t)$ code includes s fixed 0s and t fixed 1s. Moreover, the fixed 0s are at positions $1, 2, \ldots, s$ while the fixed 1s are at positions $j \cdot s + 1$ ($j = 1, 2 \ldots, t$), satisfying $s \cdot t \geq \frac{n-1}{2}$. For example, a generalized $F(n = 16, s = 3, t = 3)$ code has the format $0001xx1xx1xxxxxx$, where x denotes an information bit, which can be 0 or 1.

To prove that the generalized $F(n, s, t)$ code is a CFC, the following theorem from [47] is applied.

Theorem 3 *A fixed-place code with 0s and 1s fixed in the positions a_i ($i = 1, \ldots, s$) and b_j ($j = 1, \ldots, t$), respectively, will be synchronous if and only if the set $\{\pm(a_i - b_j)\}$ contains a complete system of nonzero residues (mod n).*

Theorem 4 *The generalized $F(n, s, t)$ is a comma-free code.*

Proof. In our proposed generalized $F(n, s, t)$ code, the positions of 0s are $a_i = i$ ($i = 1, 2, \ldots, s$) and the positions of 1s are $b_j = j \cdot s + 1$ ($j = 1, 2, \ldots, t$) where $s \cdot t \geq 0.5(n-1)$. Thus, $\{\pm(a_i - b_j)\}$ (mod n) is given by the following.

$$
\begin{array}{rr}
b_1 - a_1 = s & a_1 - b_1 = n - s \\
\ldots & \ldots \\
b_1 - a_s = 1 & a_s - b_1 = n - 1 \\
b_2 - a_1 = 2s & a_1 - b_2 = n - 2s \\
\ldots & \ldots \\
b_2 - a_s = s + 1 & a_s - b_2 = n - s - 1 \\
b_t - a_1 = ts & a_1 - b_t = n - ts \\
\ldots & \ldots \\
b_t - a_s = (t-1)s + 1 & a_s - b_t = n - (t-1)s - 1
\end{array}
$$

Note that $st+1 \geq n-st$ because the generalized $F(n,s,t)$ code requires $st \geq 0.5(n-1)$. Thus the residues shown above cover all the values between 1 and $(n-1)$ at least once. Therefore, the generalized $F(n,s,t)$ code is a systematic comma-free code.

In the classical F code having a code length n, it is required that r being the least integer no less than $\sqrt{2(n-1)}$, s being the least integer no less than $\frac{r}{2}$ and $t = r - s$. For our proposed code, however, we have proved that the condition $st \geq 0.5(n-1)$ is sufficient to make sure it is a synchronization code. As a result, more combinations of (s,t) can be selected in constructing our codes. In fact, the proposed generalized $F(n,s,t)$ code includes F code as a special case. For example, the parameters $(n = 17, s = 4, t = 2)$ and $(n = 25, s = 4, t = 4)$ can be used to construct a generalized $F(n,s,t)$ code but not a classical F code.

3.1.2 Encoder

Encoding is implemented in two steps: the binary information sequence is first divided into blocks of k information bits. Then length-s regular primers and t fixed bits are inserted periodically into each block. As a result, each block of k information bits is encoded into a generalized $F(n,s,t)$ code of n bits, where $n = k + s + t$. The encoding procedure is repeated for every block of k information bits. The content and position of the periodical primers and fixed bits in the transmitted sequence are known to and used by the decoder to maintain synchronization. The encoding sequence after the two encoding steps is ready to be transmitted through the BSID channel. The encoding procedure of a generalized $F(n,s,t)$ code is shown in Fig. 3.1.

Moreover, the code rate is defined as the ratio between the number of information bits and the codeword length, i.e., $\frac{n-s-t}{n}$. The number of fixed positions $s + t$ is minimized when s is equal to t under the condition that $2st \geq n - 1$, i.e, $s = t = \sqrt{\frac{n-1}{2}}$. Therefore, the code rate of the generalized $F(n,s,t)$ code is limited by $\frac{n-2\cdot\sqrt{\frac{n-1}{2}}}{n}$, which approaches $1 - \sqrt{\frac{2}{n}}$ when the codeword length n tends to infinity.

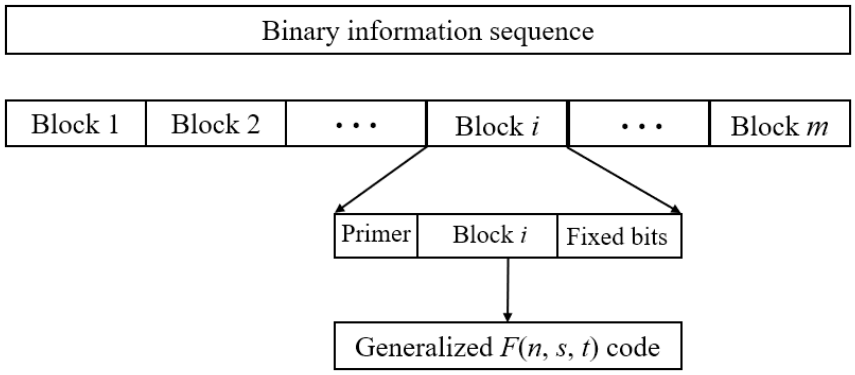


Figure 3.1: Construction of a generalized $F(n, s, t)$ code.

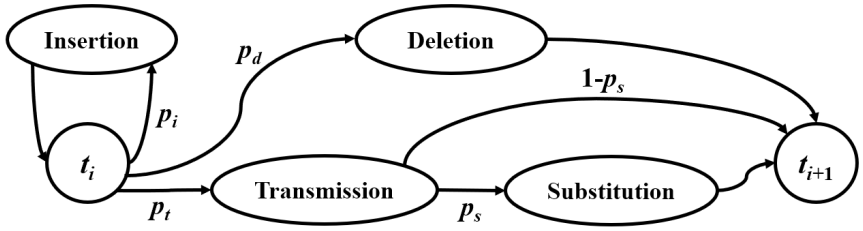


Figure 3.2: BSID channel model.

3.1.3 Channel Model

The BSID channel with random independent insertion, deletion and substitution errors is depicted in Fig. 3.2. The channel model can be mathematically described by three parameters: p_d, p_i and p_s, representing a deletion error rate, an insertion error rate and a substitution error rate, respectively. For each transmitted bit x_k, three possible events may occur: (a) x_k is deleted with probability p_d; (b) one more bit is transmitted before x_k with probability p_i; (c) x_k is transmitted with probability $p_t = 1 - p_d - p_i$. Moreover, the transmitted bit may be substituted with probability p_s. The operation repeats until all bits are sent.

3.1.4 Decoder

Due to potential insertion and deletion errors from the BSID channel, the length of the received sequence may be different from the length of the transmitted sequence. We

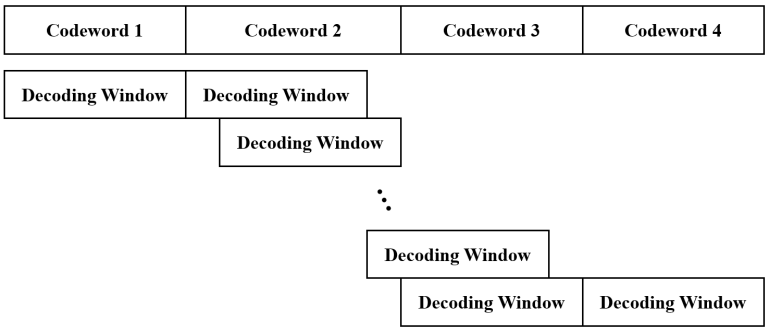


Figure 3.3: Generalized $F(n, s, t)$ decoding without a false codeword.

introduce a decoding window with size n, which is simply the codeword length in the decoder. Given a received sequence, the decoder makes full use of the information provided by the primer and fixed bits. The decoder can recover synchronization at codeword boundaries with the help of primer drive and fixed positions over the BSID channel. Furthermore, the decoder can retrieve original message based on the format of a generalized $F(n, s, t)$ code.

A codeword is immediately decoded whenever the decoding window detects the format of a $F(n, s, t)$ codeword. The decoding window then moves n positions to the right and tries to detect the next codeword. It is possible that the n bits in the decoding window cannot construct a $F(n, s, t)$ codeword due to substitution, deletion and insertion errors. Under this circumstance, the decoding window moves one position to the right at a time until another $F(n, s, t)$ is detected. The procedure repeats until the decoding window reaches the last bit.

By definition, a sequence of bits from two consecutive CFC codes cannot form a $F(n, s, t)$. However, a false codeword may appear when substitution, insertion, or deletion errors occur. Suppose an error (insertion, deletion or substitution) occurs in the i-th codeword and no errors occur in the following two codewords, the receiver can always resynchronize at the $(i + 2)$-th codeword or at the $(i + 1)$-th codeword. We use two examples for illustration.

In Fig. 3.3, suppose an insertion error occurs in Codeword 2. When the decoding

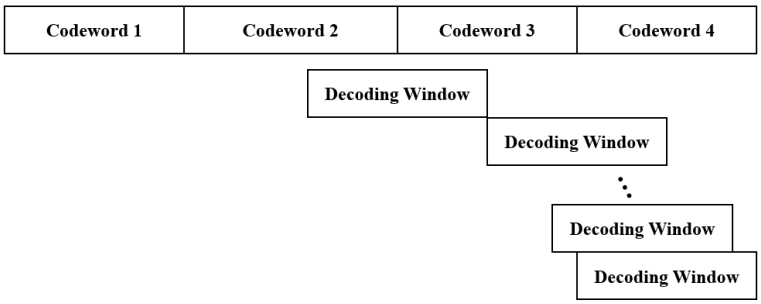


Figure 3.4: Generalized $F(n, s, t)$ decoding with a false codeword.

window is located in the first block, the content in the decoding window is Codeword 1 and hence Codeword 1 is decoded immediately. The decoding window then moves n positions to the right. However, the following n bits in the decoding window cannot form a $F(n, s, t)$ due to the insertion error and hence it moves one position to the right until the decoding window arrives at the third block, where Codeword 3 is decoded. In this case, the receiver resynchronizes at Codeword 3.

In Fig. 3.4, the tail of the corrupted Codeword 2 and the head of Codeword 3 form a false codeword due to possible errors occurring in Codeword 2. The decoding window thinks this is a "valid" codeword, so it moves n positions to the right. The content of the decoding window becomes an overlap of two $F(n, s, t)$ codewords and cannot be a codeword. After that, the decoding window moves one position to the right until it decodes Codeword 4. In this case, synchronization is re-established at Codeword 4. The presence of a false codeword excludes the decoding of the subsequent correct codeword (Codeword 3 in Fig. 3.4). Therefore, the false codeword probability is a performance metric of a systematic comma-free code over channels with insertion, deletion and substitution errors.

3.2 False Synchronization

3.2.1 Theoretical Error Rate

In this section, we calculate the theoretical error rate of a codeword with a specific error pattern. Consider a codeword of length n transmitted through the BSID channel, the error probability $Pr(N_d, N_s)$ with N_d deletions and N_s substitutions, and the error probability $Pr(N_i)$ with N_i insertions are given by (3.1) and (3.2), respectively.

$$Pr(N_d, N_s) = p_d^{N_d} \cdot (p_t \cdot p_s)^{N_s} \cdot (p_t \cdot (1 - p_s))^{n-N_d-N_s}, \tag{3.1}$$

$$Pr(N_i) = p_i^{N_i}, \tag{3.2}$$

Insertion, deletion and substitution errors are all independent and thus the probability of a codeword with the specific error pattern, containing N_i insertions, N_d deletions and N_s substitutions is given by the multiplication of (3.1) and (3.2).

$$Pr(N_i, N_d, N_s) = Pr(N_d, N_s) \cdot Pr(N_i) = p_d^{N_d} \cdot (p_t \cdot p_s)^{N_s} \cdot (p_t \cdot (1 - p_s))^{n-N_d-N_s} \cdot p_i^{N_i}. \tag{3.3}$$

Any insertion error that occurs in the boundary between two consecutive codewords is considered as an insertion error of the second codeword for the BSID channel and hence the number of combinations to insert N_i bits is $\binom{n+N_i-1}{N_i}$. In the meanwhile, we have $\binom{n}{N_d}$ combinations of deleting N_d bits from n bits and $\binom{n-N_d}{N_s}$ combinations of substituting N_s bits from $(n - N_d)$ bits. The total number of combinations with the error pattern N_i, N_d and N_s is given by

$$C(N_i, N_d, N_s) = \binom{n + N_i - 1}{N_i} \cdot \binom{n}{N_d} \cdot \binom{n - N_d}{N_s}. \tag{3.4}$$

Based on the above results, the theoretical error rate of N_i insertions, N_d deletions and N_s substitutions is given by $C(N_i, N_d, N_s) \cdot Pr(N_i, N_d, N_s)$.

3.2.2 False Codeword Probability

In this section, we derive the false codeword probability when a single substitution, insertion or deletion error occurs in a codeword. To simplify our analysis, we further assume that the codeword that follows the erroneous codeword does not contain errors. We define the following symbols.

- x: An arbitrary information bit which can be either 0 or 1
- $\mathbf{g}$: Generator of the proposed generalized $F(n, s, t)$ code. It is a vector of length n, which consists of s fixed (non-arbitrary) 0s, t fixed (non-arbitrary) 1s and $n-s-t$ arbitrary information bits x.
- $\mathbf{tt}$: Concatenation of two transmitted $F(n, s, t)$ codewords with a total vector length of $2n$.
- $\mathbf{r'r}$: Received vector when $\mathbf{tt}$ is transmitted. The vector length depends on the errors occurring.
- $\mathbf{g}(i)$: A decoding window which is the delay-i version of $\mathbf{g}$, i.e., $\mathbf{g}$ with i empty slots in front.
- $d(\mathbf{tt}, \mathbf{g}(i))$: Hamming distance only between non-arbitrary bits of $\mathbf{tt}$ and $\mathbf{g}(i)$ inside the decoding window. If $d(\mathbf{tt}, \mathbf{g}(i)) = 0$, the content in the decoding window is a valid codeword.
- $h(i)$: Number of positions where the element of $\mathbf{g}(i)$ is 0 or 1 AND the element of $\mathbf{tt}$ is x.

Single Substitution Error

When a single substitution error occurs, synchronization is lost if and only if the error occurs at one of the fixed bits. Then false codewords may arise. Thus when evaluating

s=t=3, n=16

TX	0	0	0	1	x	x	1	x	x	1	x	x	x	x	x	x	0	0	0	1	x	x	1	x	x	1	x	x	x	x	x	x	d	$h(i)$
i=1		0	0	0	1	x	x	1	x	x	1	x	x	x	x	x	x																	1	3
i=2			0	0	0	1	x	x	1	x	x	1	x	x	x	x	x	x																	1	4
i=3				0	0	0	1	x	x	1	x	x	1	x	x	x	x	x	x																	1	3
i=4					0	0	0	1	x	x	1	x	x	1	x	x	x	x	x	x															1	5
i=5						0	0	0	1	x	x	1	x	x	1	x	x	x	x	x	x														1	5
i=6							0	0	0	1	x	x	1	x	x	1	x	x	x	x	x	x													1	4
i=7								0	0	0	1	x	x	1	x	x	1	x	x	x	x	x	x												2	4
i=8									0	0	0	1	x	x	1	x	x	1	x	x	x	x	x	x											2	4
i=9										0	0	0	1	x	x	1	x	x	1	x	x	x	x	x	x										2	4
i=10											0	0	0	1	x	x	1	x	x	1	x	x	x	x	x	x									1	/
i=11												0	0	0	1	x	x	1	x	x	1	x	x	x	x	x	x								1	/

Figure 3.5: Complete shifting process of a decoding window for the generalized $F(n = 16, s = 3, t = 3)$.

the probability of a false codeword occurring due to a single substitution error, we are only interested in those i for which $d(\mathbf{tt}, \mathbf{g}(i)) = 1$. In other words, the decoding window $\mathbf{g}(i)$ and the transmitted vector $\mathbf{tt}$ differ in only one fixed position. Suppose a substitution error occurs at this position, a false codeword will occur when some specific arbitrary information bits x in $\mathbf{tt}$ match the fixed 0 or 1 bits in $\mathbf{g}(i)$. The exact matching has a probability of $(\frac{1}{2})^{h(i)}$.

We use the generalized $F(n = 16, s = 3, t = 3)$ code as an example for illustration. The complete shifting process of the decoding window is shown in Fig. 3.5. When $i = 3$, three empty slots are inserted in front of the decoding window (or the decoding window is shifted three positions to the right).

Referring to Fig. 3.6(a), the first row represents the generator; the second row represents two consecutive transmitted codewords; the third row represents the received vector which indicates the first fixed 1 in the transmitted codeword suffers from a substitution error; the last row is the shifted decoding window with $i = 3$. By comparing the second and fourth rows, we observe $h(3) = 3$. Therefore, a false codeword will occur with a probability of $(\frac{1}{2})^3$ (probability when the fifth, sixth and thirteenth transmitted bits in $\mathbf{tt}$ happen to be 0, 0 and 1 coincidentally) if the first fixed 1 is in error.

However, a point needs to be emphasized. Suppose $d(\mathbf{tt}, \mathbf{g}(i)) = 1$ for a certain i and the position where $\mathbf{tt}$ and $\mathbf{g}(i)$ differs is beyond n. For example, the position where

$$
\begin{array}{ll}
\mathbf{g}: & 0\,0\,0\,1\,x\,x\,1\,x\,x\,1\,x\,x\,x\,x\,x\,x \\
\mathbf{tt}: & 0\,0\,0\,1\,x\,x\,1\,x\,x\,1\,x\,x\,x\,x\,x\,x\,x\,0\,0\,0\,1\,x\,x\,1\,x\,x\,1\,x\,x\,x \\
\mathbf{r'r}: & 0\,0\,0\,\mathbf{0}\,x\,x\,1\,x\,x\,1\,x\,x\,x\,x\,x\,x\,0\,0\,0\,1\,x\,x\,1\,x\,x\,1\,x\,x\,x \\
\mathbf{g}(i=3): & 0\,0\,0\,1\,x\,x\,1\,x\,x\,1\,x\,x\,x\,x\,x\,x
\end{array}
$$

(a)

$$
\begin{array}{ll}
\mathbf{g}: & 0\,0\,0\,1\,x\,x\,1\,x\,x\,1\,x\,x\,x\,x\,x\,x \\
\mathbf{tt}: & 0\,0\,0\,1\,x\,x\,1\,x\,x\,1\,x\,x\,x\,x\,x\,x\,x\,0\,0\,0\,1\,x\,x\,1\,x\,x\,1\,x\,x\,x \\
\mathbf{rr'}: & 0\,0\,0\,1\,x\,x\,1\,x\,x\,1\,x\,x\,x\,x\,x\,x\,\mathbf{1}\,0\,0\,1\,x\,x\,1\,x\,x\,1\,x\,x\,x \\
\mathbf{g}(i=10): & \qquad\qquad 0\,0\,0\,1\,x\,x\,1\,x\,x\,1\,x\,x\,x\,x\,x\,x
\end{array}
$$

(b)

Figure 3.6: Decoding window for the generalized $F(n = 16, s = 3, t = 3)$ code. (a) Decoding window with $i = 3$; (b) Decoding window with $i = 10$;

the vector **tt** and **g**(10) differs is at the first bit of the second codeword as shown in Fig. 3.6(b). This means the first codeword is correct while the second codeword has a substitution error. In this case, the decoding window will shift n positions to the right after decoding the first correct codeword. The decoding window will never contain the tail of a correct codeword and the head of the subsequent corrupted codeword. Therefore, we only consider the case that a substitution error occurs in the first codeword and the second codeword is correct.

Based on the above analysis, the last bit 1 of the decoding window $\mathbf{g}(i)$ should never reach the second codeword. Otherwise, $d(\mathbf{tt}, \mathbf{g}(i)) > 1$ or the first codeword has been decoded correctly. Hence we have $i + st + 1 \leq n$, i.e., $1 \leq i \leq n - st - 1$. Furthermore, we can readily show that

$$
h(i) = \begin{cases} i + t - 1 & 1 \leq i \leq s - 1 \\ s + \frac{i}{s} - 1 & i = s, 2s, \cdots, ts \cdot \\ s + t - 1 & \text{otherwise} \end{cases} \tag{3.5}
$$

In summary, the false codeword probability due to a single substitution error at a fixed position in a generalized $F(n, s, t)$ code is given by

***s*=*t*=3, *n*=19**

TX	**0 0 0 1 x x 1 x x 1 x x x x x x x x x 0 0 0 1 x x 1 x x 1 x x x x x x x x**	*h(i)*
Case 1	0 0 1 x x 1 x x 1 x x x x x x x x x 0 0 0 1 x x 1 x x 1 x x x x x x x x	/
Case 2	0 0 0 x x 1 x x 1 x x x x x x x x x 0 0 0 1 x x 1 x x 1 x x x x x x x x	
i=0	0 0 0 1 x x 1 x x 1 x x x x x x x x x	3
i=1	0 0 0 1 x x 1 x x 1 x x x x x x x x x	4
i=2	0 0 0 1 x x 1 x x 1 x x x x x x x x x	3
Case 3	0 0 0 1 x 1 x x 1 x x x x x x x x x 0 0 0 1 x x 1 x x 1 x x x x x x x x	
i=0	0 0 0 1 x x 1 x x 1 x x x x x x x x x	2
	0 0 0 1 x x 1 x 1 x x x x x x x x x 0 0 0 1 x x 1 x x 1 x x x x x x x x	
i=0	0 0 0 1 x x 1 x x 1 x x x x x x x x x	1
Case 4	0 0 0 1 x x x x 1 x x x x x x x x x 0 0 0 1 x x 1 x x 1 x x x x x x x x	
i=0	0 0 0 1 x x 1 x x 1 x x x x x x x x x	2
i=4	0 0 0 1 x x 1 x x 1 x x x x x x x x x	6
i=5	0 0 0 1 x x 1 x x 1 x x x x x x x x x	5
Case 5	0 0 0 1 x x 1 x x x x x x x x x x x 0 0 0 1 x x 1 x x 1 x x x x x x x x	
i=0	0 0 0 1 x x 1 x x 1 x x x x x x x x x	1
i=7	0 0 0 1 x x 1 x x 1 x x x x x x x x x	6
i=8	0 0 0 1 x x 1 x x 1 x x x x x x x x x	6
Case 6	0 0 0 1 x x 1 x x 1 x x x x x x x x 0 0 0 1 x x 1 x x 1 x x x x x x x x	
i=0	0 0 0 1 x x 1 x x 1 x x x x x x x x x	0

Figure 3.7: Complete shifting process of a decoding window for the generalized $F(n = 19, s = 3, t = 3)$.

$$P_s = P'_s \times \sum_{d(\mathbf{tt},\mathbf{g}(i))=1 \text{ and } i \le n-st-1} \left(\frac{1}{2}\right)^{h(i)}, \tag{3.6}$$

In (3.6), $P'_s = (p_t \cdot p_s) \times (p_t \cdot (1 - p_s))^{s+t-1}$ represents the probability that a substitution occurs at exactly a fixed position under the condition of $N_s = 1, N_d = 0$ and $N_i = 0$ in (3.3).

Single Deletion Error

We move on to investigating the false codeword probability P_d due to a single deletion. We define $\mathbf{r}'_z\mathbf{r}$ as the received vector, in which the zth bit ($1 \le z \le n$) of $\mathbf{tt}$ is deleted. The length of $\mathbf{r}'_z\mathbf{r}$ is $2n - 1$. Actually $\mathbf{r}'_z\mathbf{r}$ is the concatenation of a codeword with one deletion error and a correct codeword. It is obvious that the Hamming distance $d(\mathbf{g}(n-1), \mathbf{r}'_z\mathbf{r}) = 0$ because the content of the decoding window is exactly the second codeword which is correct. However, if $d(\mathbf{g}(i), \mathbf{r}'_z\mathbf{r}) = 0$ for some $i \le n - 2$ and z, we detect a false codeword with probability $(\frac{1}{2})^{h(i)}$. As in the case of substitution, the last bit 1 of the decoding window should not reach the position of the first bit 0 of the

second codeword, i.e., $i+st+1 \leq n-1$ or $i \leq n-st-2$. The complete shifting process of the decoding window for a systematic generalized $F(n = 19, s = 3, t = 3)$ code is shown in Fig. 3.7. We further divide six different scenarios to derive $h(i)$.

Case 1: A single bit among the s starting 0s is deleted

The received vector starts with $(s-1)$ 0s followed by a 1. However, the decoding window starts with s 0s. Even if all information bits are 0s, it is not possible to form a false codeword.

Case 2: The first bit 1 after the sequence of s 0s is deleted

A false codeword can be formed under the condition $0 \leq i \leq s-1$. Furthermore, it can be readily shown that

$$h(i) = \begin{cases} t+i & 0 \leq i \leq s-2 \\ s & i = s-1 \end{cases}. \tag{3.7}$$

Case 3: A single bit between jth bit 1 and $(j+1)th$ bit 1 ($1 \leq j \leq t-1$) is deleted

In this case, a false codeword may be formed only when $i = 0$ and $h(i) = t - j$.

Case 4: The jth bit 1 ($2 \leq j \leq t-1$) is deleted

Obviously, a false codeword is formed when $i = 0$ and $h(i) = t - j + 1$. Besides, a false codeword may be formed when (i) the first bit 0 of the decoding window exceeds $(j-1)$ bit 1, i.e., $1 + i \geq (j-1)s + 1 + 1$ or $i \geq (j-1)s + 1$; AND (ii) the first bit 1 of the decoding window does not exceed $(j+1)th$ bit 1, i.e., $i + s + 1 \leq (j+1)s + 1 - 1$ and $i \leq js - 1$. Furthermore, we can prove that

$$h(i) = \begin{cases} s+t & (j-1)s+1 \leq i \leq js-2 \\ s+j & i = js-1 \end{cases}. \tag{3.8}$$

Case 5: The last bit 1 is deleted

Obviously, a false codeword is formed when $i = 0$ and $h(i) = 1$. Besides, a false

codeword may be formed when the first bit 0 of the decoding window exceeds the fixed bit 1 preceding the deleted bit 1, i.e., $1 + i \geq (t-1)s + 1 + 1$ or $i \geq (t-1)s + 1$. Furthermore, we can show that

$$h(i) = \begin{cases} 1 & i = 0 \\ s + t & (t-1)s + 1 \leq i \leq n - st - 2 \end{cases}. \tag{3.9}$$

Case 6: A single bit after the last bit 1 is deleted

In this case, a false codeword may be formed only when $i = 0$ and $h(i) = 0$.

Based on all the above analysis, the false code probability due to a single deletion error is given by

$$\begin{aligned} P_d &= P'_d \times \sum_{d(\mathbf{g}(i),\mathbf{r}'_z\mathbf{r})=0} \left(\frac{1}{2}\right)^{h(i)} \\ &= P'_d \times \left\{ [\sum_{i=0}^{s-2} (\frac{1}{2})^{t+i} + (\frac{1}{2})^s] + (s-1) \cdot \sum_{j=1}^{t-1} (\frac{1}{2})^{t-j} + \sum_{j=2}^{t-1} (\frac{1}{2})^{t-j+1} + (\frac{1}{2}) + (n - ts - 1) \right\} \end{aligned} \tag{3.10}$$

In (3.10), $P'_d = p_d \cdot ((1 - p_i - p_d) \cdot (1 - p_s))^{n-1}$ represents the probability that a single bit is deleted in the first codeword under the condition of $N_d = 1, N_s = 0$ and $N_i = 0$ in (3.3).

Single Insertion Error

Finally, we investigate the false codeword probability P_i due to a single insertion error. We denote the received vector $\mathbf{r}'_q\mathbf{r}$ as a vector of length $2n + 1$, in which a random bit is inserted before the qth bit of $\mathbf{tt}$, $1 < q < n$. Actually $\mathbf{r}'_q\mathbf{r}$ is a concatenation of a codeword with one insertion error and a correct codeword. It is obvious that $d(\mathbf{g}(i), \mathbf{r}'_q\mathbf{r}) = 0$ when $i = n + 1$ because the content of the decoding window is the

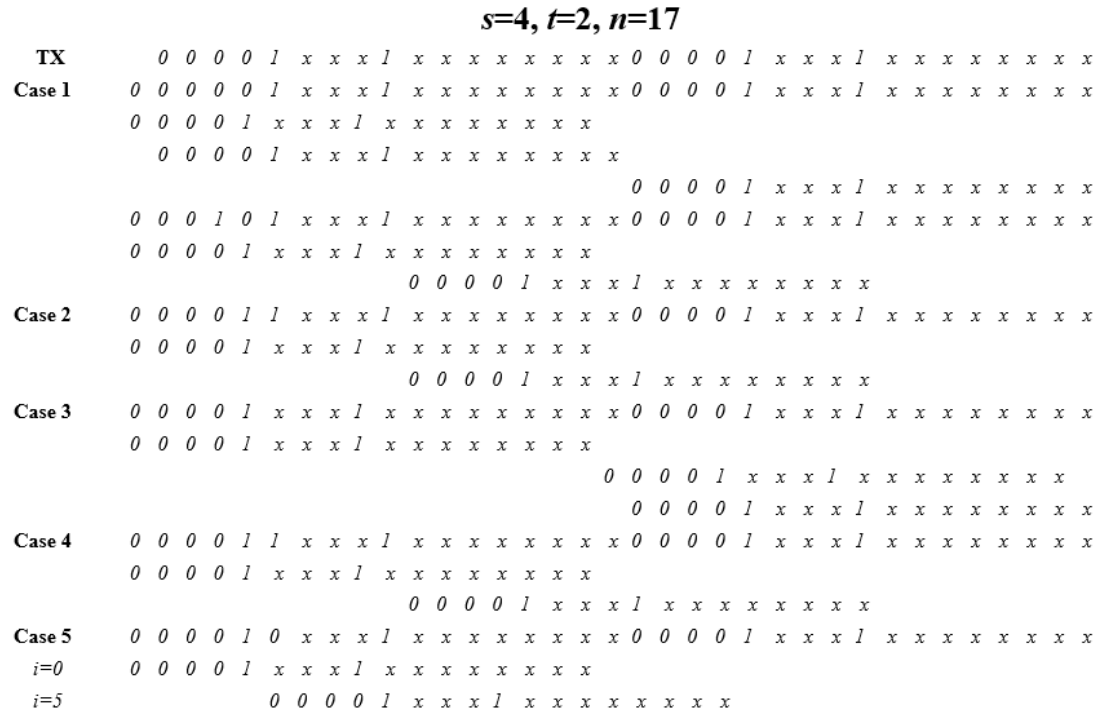


Figure 3.8: Complete shifting process of a decoding window for the generalized $F(n = 17, s = 4, t = 2)$.

second codeword. However, if $d(\mathbf{g}(i), \mathbf{r}'_q\mathbf{r}) = 0$ for some $i \leq n + 1$ and q, we can detect a false codeword with probability $(\frac{1}{2})^{h(i)}$. Same as the previous two cases, the last bit 1 of the decoding window cannot exceed the position of the first bit 0 of the second codeword, i.e., $i + st + 1 \leq n + 1$ or $i \leq n - st$. The complete shifting process of the decoding window for the generalized $F(n = 17, s = 4, t = 2)$ code is shown in Fig. 3.8.

Similarly, we further divide different scenarios to derive the corresponding $h(i)$.

Case 1: Any single bit 0/1 is inserted before any bit in the s starting zeros

Case 2: Any single bit 0/1 is inserted before the first bit 1 after the sequence of s zeros

Case 3: Any single bit 0/1 is inserted before any bits after the last bit 1

Case 4: A single bit 1 is inserted between jth bit 1 and $(j + 1)th$ bit 1 including the $(j + 1)th$ bit 1, $1 \leq j \leq t$

Case 5: A single bit 0 is inserted between jth bit 1 and $(j + 1)th$ bit 1 including the $(j + 1)th$ bit 1, $1 \leq j \leq t$

Based on the complete shifting process of a decoding window, it is impossible to form a false codeword under the first four cases[1]. A false codeword can appear with two conditions under Case 5, i.e., a bit 0 is inserted between jth bit 1 and $(j+1)th$ bit 1 including the $(j+1)$ bit 1, $1 \leq j \leq t$. The first condition is to make sure the sequence in the decoding window is not a codeword when $i = 0$ because the definition of a false codeword is the concatenation of two codewords. Only under the first condition will the decoding window shift one position to the right. The second condition is that the first bit 0 of the decoding window must arrive at the next bit of the jth bit 1 of the received codewords, i.e., $1 + i = js + 1 + 1$ or $i = js + 1$. Furthermore, we can show that $h(i) = s + j - 1$.

Most of the cases do not give rise to false codewords. Based on our analysis, the false codeword probability due to a single insertion error is given by

$$P_i = P_i' \times \sum_{j=1}^{t-1} s \left(\frac{1}{2}\right)^{s+j-1}, \tag{3.11}$$

In (3.11), $P_i' = \frac{1}{2}((1 - p_i - p_d)(1 - p_s))^n \cdot p_i$ represents the probability that a single bit 0 is inserted under the condition of $N_i = 1, N_d = 0$ and $N_s = 0$ in (3.3).

3.3 Simulation Results

The theoretical results of false synchronization for a generalized $F(n, s, t)$ code caused by a single substitution, deletion or insertion error have been derived in (3.6), (3.10) and (3.11), respectively. We carry out the simulation to verify the theoretical results. A random sequence of around 10^6 information bits is generated. The binary information sequence is first divided into blocks and each block contains k ($k = n-s-t$) information

[1]One point needs to be noticed for Case 1 where any single bit is inserted before the first bit. Even though $d(\mathbf{g}(1), \mathbf{r}_q'\mathbf{r}) = 0$ under this case, the sequence in the decoding window is the first codeword. False codeword is actually the overlap of two consecutive codewords. Based on the definition of the false codeword, this scenario cannot contribute to the false codeword probability caused by a single insertion error.

bits based on different generalized $F(n, s, t)$ codes. Then the primer drive and fixed bits are inserted in the k information bits to form a systematic generalized $F(n, s, t)$ codeword of length n. The encoding procedure is repeated for every block and the encoded sequence is transmitted though the BSID channel with different parameters p_i, p_d, and p_s. For decoding, a fixed-length decoding window of length n is introduced. The decoding window moves from the start of the received corrupted sequence until it reaches the last bit. During the whole process, we count the number of false codewords caused by a single substitution error, a single deletion error and a single insertion error, respectively.

The simulation results for the false codeword probability of the generalized $F(n, s, t)$ codes with different values of s and t are shown in Fig. 3.9, Fig. 3.10 and Fig. 3.11, respectively. The false codeword probability caused by a single substitution or insertion error are plotted using the left black ordinate axis while the false codeword probability caused by a single deletion error and the total false codeword probability are plotted using the right red ordinate axis. The simulation results represented by symbols match with those theoretical results represented by curves derived in (3.6), (3.10) and (3.11). Simulation results also show that for a generalized $F(n, s, t)$ code, deletion errors play a dominant role in false synchronization, while insertion errors have the least impact.

We further simulate the false synchronization caused by a single substitution, deletion and insertion error in a series of generalized $F(n, s, t)$ codes. The simulation results are shown in Fig. 3.12. In Fig. 3.12(a) and Fig. 3.12(b), the simulation results show that the false synchronization caused by a single substitution or deletion error increases as codeword length increases with the same values of s and t, e.g., comparing the results of $F(20, 4, 3)$, $F(23, 4, 3)$ and $F(25, 4, 3)$ in Fig. 3.12(a) and comparing the results of $F(20, 4, 3)$, $F(23, 4, 3)$ and $F(25, 4, 3)$ in Fig. 3.12(b). We also observe that in both cases, the false synchronization does not always increase with the codeword length. It is closely related to the values of s and t. This can be demonstrated by the simulation result, in which the curve of $F(25, 4, 3)$ is always lower than that of $F(17, 3, 3)$,

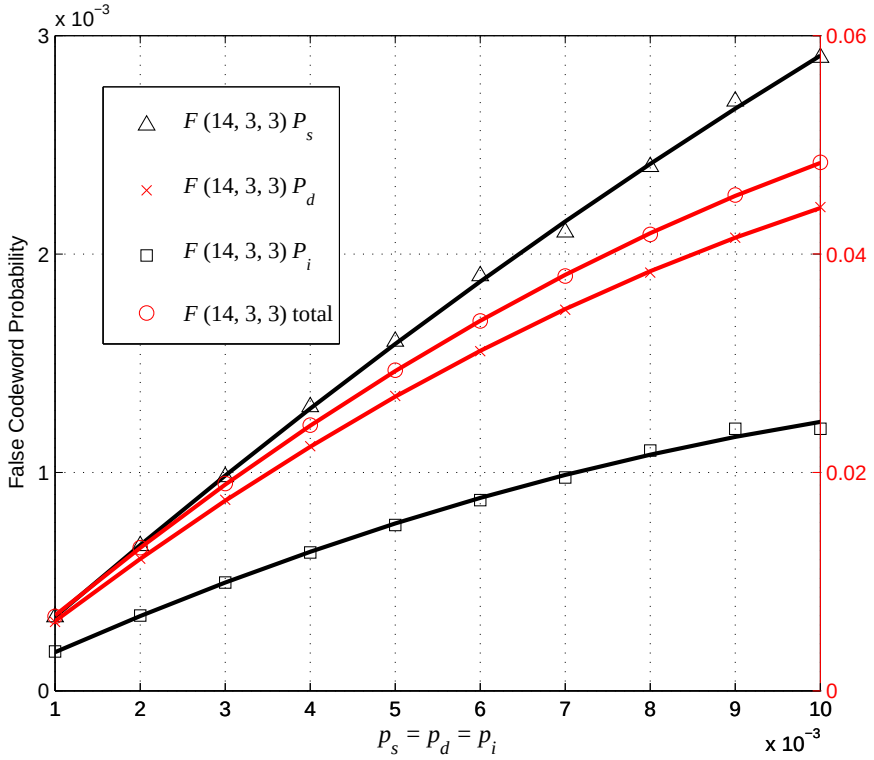
$F(19,3,3)$ and $F(17,4,2)$ in Fig. 3.12(a), and the curve of $F(27,4,4)$ is always lower than that of $F(23,4,3)$ and $F(25,4,3)$ in Fig. 3.12(b). This can also be verified by the better performance of $F(27,4,4)$ than $F(17,4,2)$ when the channel error rate is equal to or greater than 9×10^{-3} in Fig. 3.12(b).

However, different from the first two cases, the false synchronization caused by a single insertion may not always increase with codeword length even with the same combination of $\{s,t\}$. This can be verified by the better performance of $F(25,4,3)$ than $F(23,4,3)$ in Fig. 3.12(c).

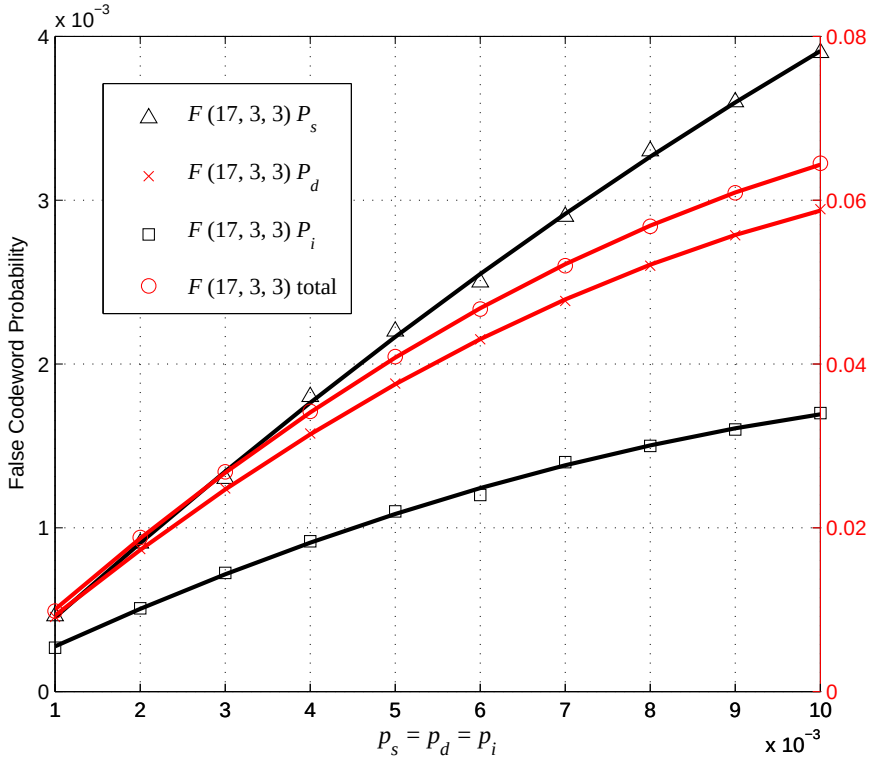
3.4 Summary

In this chapter, we have proposed a new class of systematic comma-free code, namely the generalized $F(n,s,t)$ code. We have proved that the condition $st \geq 0.5(n-1)$ is sufficient to make sure it is a synchronization code. As a result, more combinations of (s,t) can be selected in constructing our codes. We also derive the probability of false synchronization caused by a single substitution, insertion or deletion error, respectively. Simulation results show that the performance of a generalized $F(n,s,t)$ code is not only related to channel error rates but also the distribution of fixed bits and codeword length. Among all factors affecting the error performance, deletion errors have a larger effect compared with the other two types of errors. Even though generalized $F(n,s,t)$ codes with shorter length usually perform better in terms of false synchronization, codes with shorter length decrease the code rate and efficiency. Therefore, a good generalized $F(n,s,t)$ code should be chosen based on different applications and requirements, i.e., channel parameters, code rate and codeword length. This chapter focuses on the code that is used to recover synchronization without correction capability. From the next chapter, work is focused on codes that can correct insertion, deletion and substitution errors. Specifically, we will focus on the optimization of a classical concatenated code that can correct multiple insertion, deletion and substitution errors

in the next chapter.

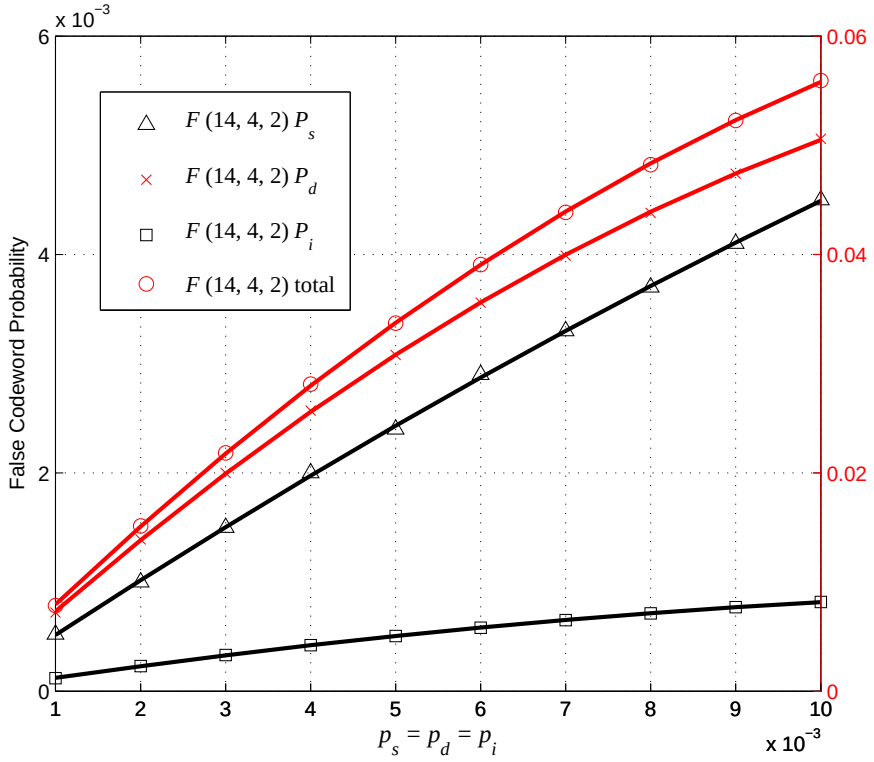


(a)

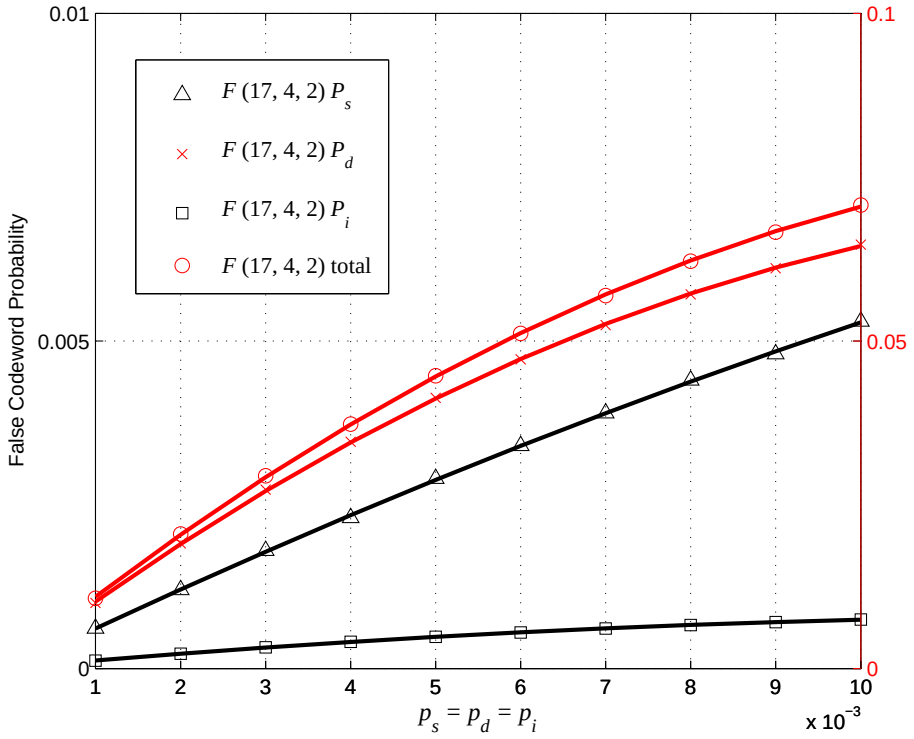


(b)

Figure 3.9: False synchronization probability in a generalized $F(n, s, t)$ code under different channel parameters. (a) $F(14, 3, 3)$; (b)$F(17, 3, 3)$. Channel error rates increase from 0.001 to 0.01. Theoretical results are plotted using the solid lines and simulated results are represented by symbols.

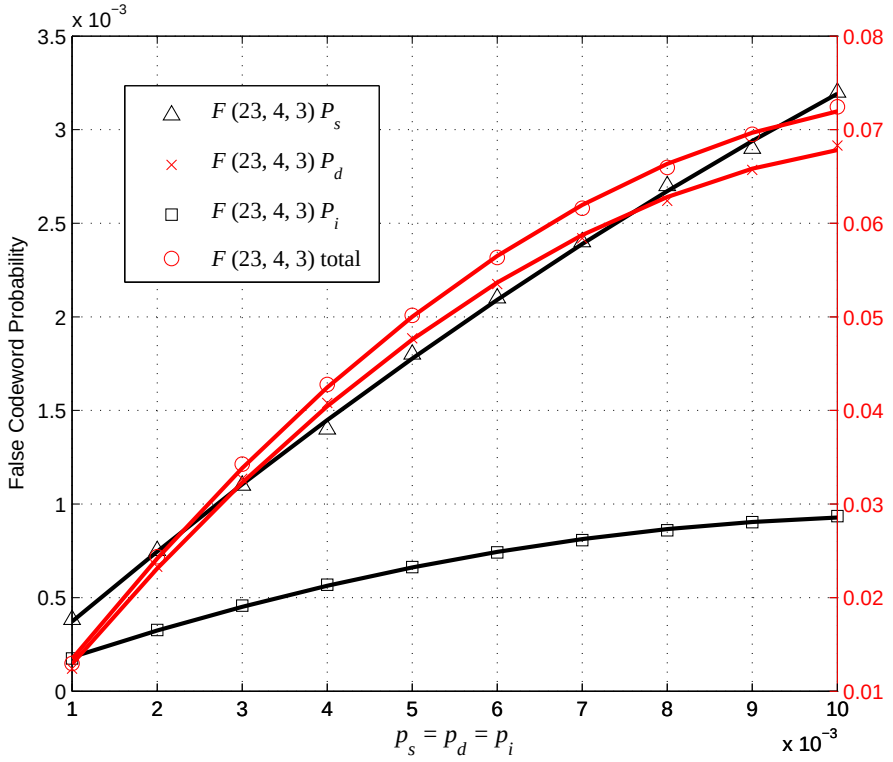


(a)

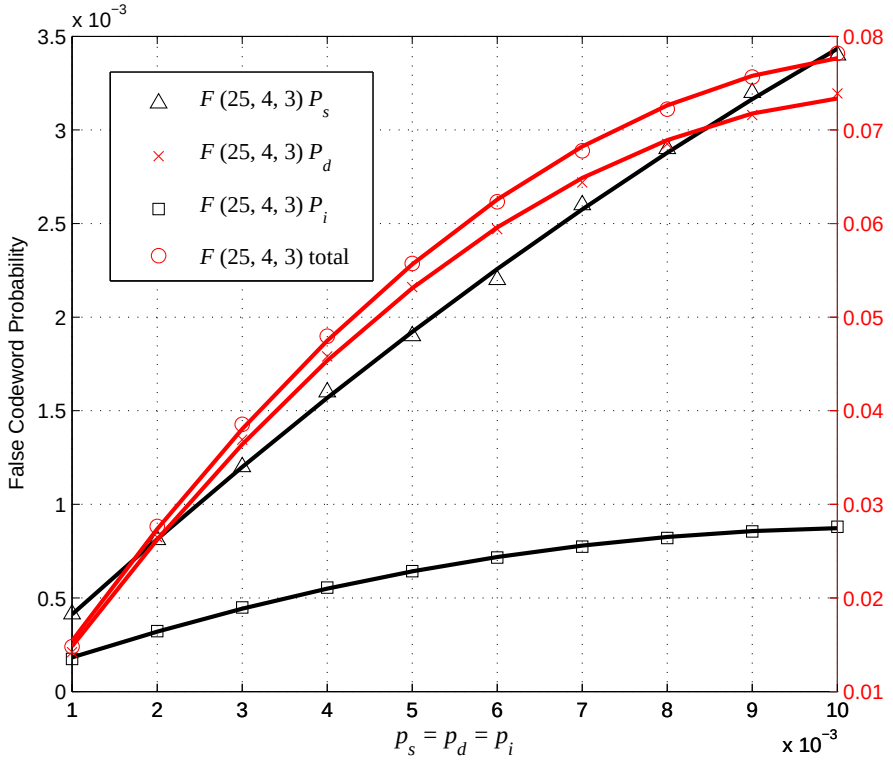


(b)

Figure 3.10: False synchronization probability in a generalized $F(n, s, t)$ code under different channel parameters. (a) $F(14, 4, 2)$; (b) $F(17, 4, 2)$. Channel error rates increase from 0.001 to 0.01. Theoretical results are plotted using the solid lines and simulated results are represented by symbols.



(a)



(b)

Figure 3.11: False synchronization probability in a generalized $F(n, s, t)$ code under different channel parameters. (a) $F(23, 4, 3)$; (b) $F(25, 4, 3)$. Channel error rates increase from 0.001 to 0.01. Theoretical results are plotted using the solid lines and simulated results are represented by symbols.

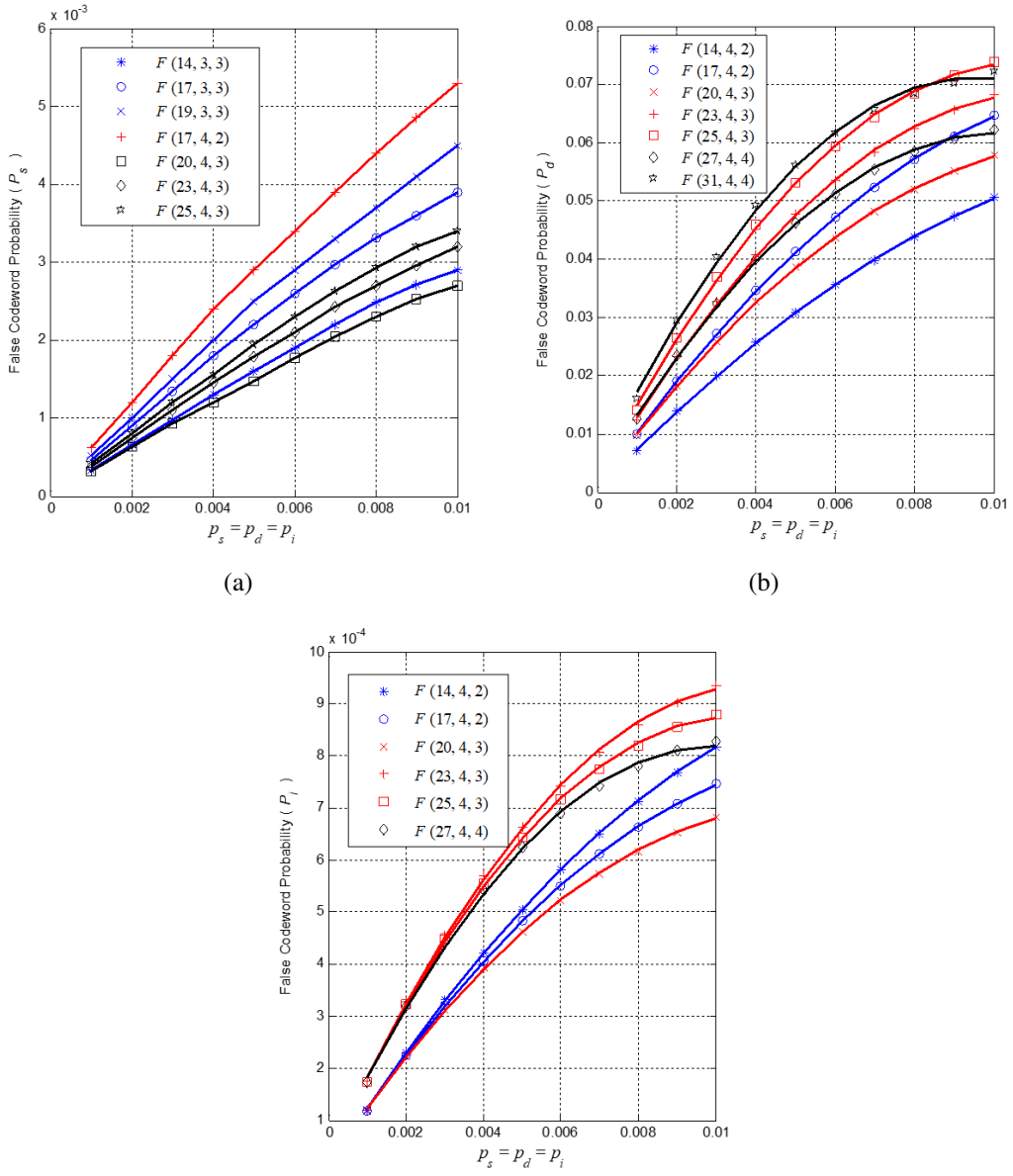


Figure 3.12: The false synchronization probability in a series of generalized $F(n, s, t)$ codes under different channel parameters caused by (a) a single substitution error at fixed positions; (b) a single deletion error; (c) a single insertion error. For all three cases, $p_s = p_d = p_i$ increase from 0.001 to 0.01. Theoretical results are plotted using the solid lines and simulated results are represented by symbols.

Chapter 4

Optimization of Sparsifier in Davey-Mackay Watermark Code

4.1 Introduction

The Davey-Mackay (DM) watermark code can correct multiple insertion, deletion and substitution errors over the BSID channel. It consists of an outer LDPC code to correct substitution errors and an inner watermark code to maintain synchronization.

4.1.1 Encoder

The structure of the DM watermark code is shown in Fig. 4.1. In this scheme, the information sequence $\boldsymbol{m}$ is first encoded into a sequence $\boldsymbol{d}$ using an outer q-ary LDPC code, where $q = 2^k$. Then, the sparsifier with parameters k and n maps each q-ary symbol (k bits) to a sparse binary sequence of length n for some $n > k$. The mapping rule is to select $q = 2^k$ lowest density (Hamming weight) binary vectors of length n. The transmitted sequence is then generated by adding modulo-2 the sparse sequence to the binary watermark sequence and is prepared for transmission through the BSID channel. Besides, the watermark sequence is known to both the encoder and decoder.

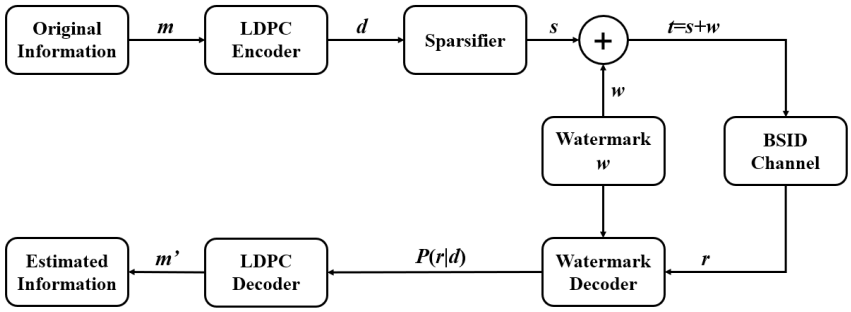


Figure 4.1: Structure of DM Construction.

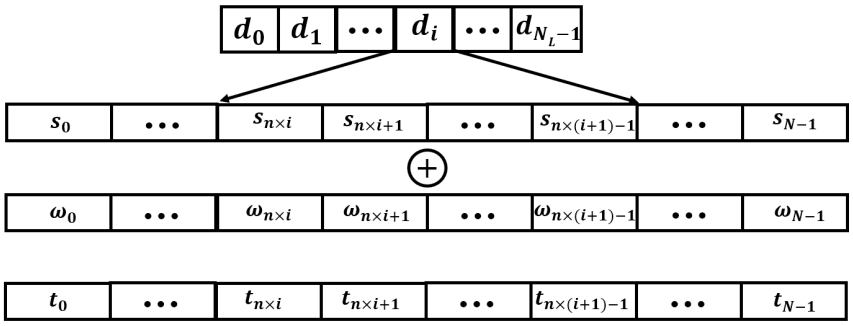


Figure 4.2: Encoding procedure of concatenated code in DM construction. Each symbol d_i is represented by the addition of watermark sequence and sparsified sequence.

The sparse code causes minimal changes to the watermark sequence to ensure the decoder can track synchronization. The specific encoding of the concatenated code is shown in Fig. 4.2.

4.1.2 Channel Model

The channel model used in this chapter is the same as the one proposed by Davey and Mackay in [26]. The BSID channel can be mathematically described by three parameters: p_d, p_i and p_s, representing a deletion error rate, an insertion error rate and a substitution error rate, respectively. At time i, a transmitted bit t_i enters the channel and experiences (a) a random bit is inserted before t_i with probability p_i; or (b) t_i is deleted with probability p_d; or (c) t_i is transmitted with probability $p_t = 1 - p_i - p_d$. In addition, the transmitted bit may suffer from a substitution error with probability p_s.

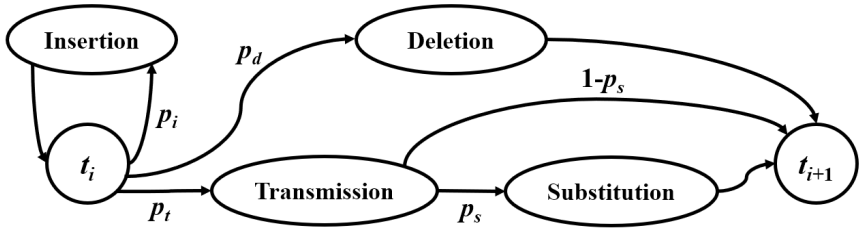


Figure 4.3: BSID channel model.

4.1.3 Decoder

The inner watermark decoder compares the received sequence with the known watermark sequence to recover synchronization and returns symbol-by-symbol log-likelihood ratios (LLRs) to initialize the outer LDPC decoder. The outer LDPC decoder subsequently corrects the remaining errors.

We have reviewed the optimal inner symbol-level forward-backward decoding algorithm on the trellis [68] in Sect. 2.4.4. The forward and backward quantities are computed for each symbol rather than each bit. The number of drift states on the boundary of each symbol depends on the difference between the number of deletions and insertions within this symbol. Using the symbols defined in Sect. 2.4.4, the likelihood of each possible transmitted symbol $P(\boldsymbol{r}|d_i)$ is given by

$$P(\boldsymbol{r}|d_i) = \sum_{x_{n\times i}, x_{n\times(i+1)}} F(n\times i, x_{n\times i}) \cdot P(\boldsymbol{r}_{n\times i+x_{n\times i}}^{n\times(i+1)+x_{n\times(i+1)}-1}, x_{n\times(i+1)}|x_{n\times i}, d_i) \cdot B(n\times(i+1), x_{n\times(i+1)}), \tag{4.1}$$

where

$$\begin{aligned} F(n\times i, y) &= P(\boldsymbol{r}_1^{n\times i+y}, x_{n\times i} = y) = P(r_1, r_2, \ldots, r_{n\times i+y}, x_{n\times i} = y) \\ &= \sum_{d_{i-1}, a} F(n\times(i-1), a) \cdot P(\boldsymbol{r}_{n\times(i-1)+a}^{n\times i+y-1}, x_{n\times i} = y|x_{n\times(i-1)} = a, d_{i-1}), \end{aligned} \tag{4.2}$$

$$B(n\times i, y) = \sum_{d_i, b} B(n\times(i+1), b) \cdot P(\boldsymbol{r}_{n\times i+y}^{n\times(i+1)+b-1}, x_{n\times(i+1)} = b|x_{n\times i} = y, d_i). \tag{4.3}$$

In this chapter, we aim to optimize the sparsifier in the DM watermark code [72].

We optimize the inner sparsified codebook based on distance property and lowest density property. We also propose a hard-decision decoding and a soft-decision decoding for the inner symbol-level watermark decoder. Simulation results show that the error performance of the inner decoder with the improved codebook is improved over the BSID channel with random insertion, deletion and substitution errors. The rest of the chapter is organized as follows. In Sect. 4.2, we propose two decoding methods for the symbol-level inner watermark decoding, including a hard-decision decoding based on Hamming distance and a soft-decision decoding based on a new metric. Sect. 4.3 shows the codebook design optimization based on both better distance property and lowest density property. Finally in Sect. 4.4, the error performance of the inner watermark code with the optimized codebook is simulated over the BSID channel. Summary is given in Sect. 4.5.

4.2 Proposed Decoding Strategies

Referring to the RHS of (4.1), the first term and the last term can be easily calculated based on (4.2) and (4.3). For the middle term $P(\boldsymbol{r}_{n\times i+x_{n\times i}}^{n\times(i+1)+x_{n\times(i+1)}-1}, x_{n\times(i+1)}|x_{n\times i}, d_i)$, it can be shown as equivalent to $P(\boldsymbol{r}_{n\times i+x_{n\times i}}^{n\times(i+1)+x_{n\times(i+1)}-1}, x_{n\times(i+1)}|x_{n\times i}, \boldsymbol{s}_{n\times i}^{n\times(i+1)-1})$ because there is a one-to-one mapping between each symbol d_i and the corresponding sparse sequence $(s_{n\times i}, \ldots, s_{n\times(i+1)-1})$ in the sparsified distribution converter. In fact, this middle term is the probability of receiving a specific sequence $\boldsymbol{r}_{n\times i+x_{n\times i}}^{n\times(i+1)+x_{n\times(i+1)}-1}$ when the sequence $(s_{n\times i}, \ldots, s_{n\times(i+1)-1})$ is transmitted given a symbol state boundary. The computation of this term can be implemented using two decoding strategies [72].

4.2.1 Hard-decision Decoding Based on Hamming distance

Given fixed $x_{n\times i}$ and $x_{n\times(i+1)}$, the codeword boundary is uniquely determined. Recall that in Sect. 2.4.4, the synchronization drift state x_i at the ith position is defined as

the difference between the number of insertions and deletions that have occurred from the start transmitted bit t_1 until bit t_i is ready to be transmitted. Thus according to the symbol drift states $x_{n\times i}$ and $x_{n\times(i+1)}$, we can prejudge whether an insertion or deletion has occurred. If $x_{n\times(i+1)} - x_{n\times i} = 1$, a single insertion error should have occurred in this symbol. Then a random bit is deleted and compared with the valid codeword set, and the valid codeword with the minimum Hamming distance is selected as the transmitted one. On the other hand, if $x_{n\times(i+1)} - x_{n\times i} = -1$, a single deletion error is assumed to occur in this symbol. Then a random bit is inserted in this corrupted codeword and compared with the valid codeword set, and the valid codeword with the minimum Hamming distance is selected as the transmitted one. If more than one codeword is available, a codeword is selected randomly. This is called hard-decision inner decoding.

4.2.2 Soft-decision Decoding Based on Accurate Transformation Probability Metric

Since as many as I bits can be inserted before each bit, many different error patterns may result in the same corrupted codeword. In other words, given the corrupted symbol and drifts, we cannot uniquely determine the actual transmitted symbol in hard-decision decoding. For example, the corrupted codeword 0000010 can be due to a deletion in any 0s before the bit 1 of 00000010, or due to a deletion in any 0s after bit 1 of 00000100. When more than one codewords exist, a random one is selected in the case of hard-decision decoding, which makes it sub-optimal. To be more accurate, we propose a soft-decision decoding based on a more accurate metric called accurate transformation probability (ATP).

The middle term in (4.1) is actually equivalent to calculating the probability $P(\boldsymbol{r}|\boldsymbol{t})$ with one symbol transmission time. The transmitted sequence $\boldsymbol{t}$ is the mod-2 addition of (i) the watermark sequence from index $n \times i + x_{n\times i}$ to index $n \times (i + 1) + x_{n\times(i+1)} - 1$ and (ii) the sparse mapping of d_i; while $\boldsymbol{r}$ is the corresponding received sequence.

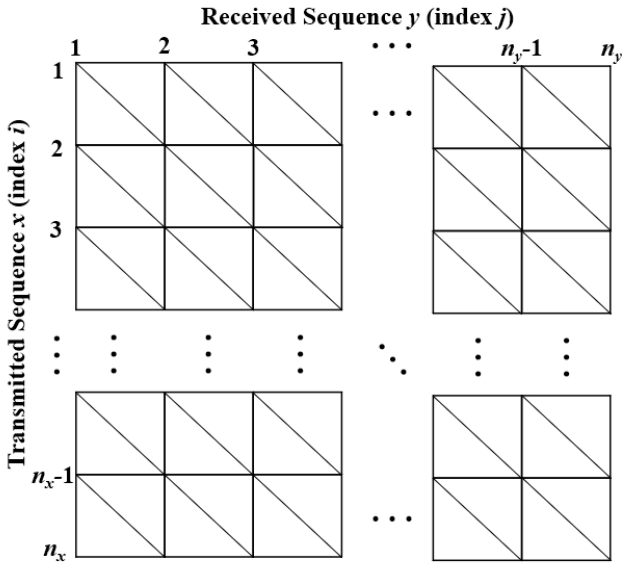


Figure 4.4: Lattice representation.

The calculation of this term can be done effectively based on the lattice structure. Fig. 4.4 shows the lattice representation: transmitted sequence $\boldsymbol{x}$ is placed along the left side and the corresponding received sequence $\boldsymbol{y}$ is placed along the top. Each error pattern is illustrated by a path between coordinates (0,0) and (n_x, n_y) on the lattice. In this representation, three kinds of shifts may occur: deletion, insertion or transmission (may be substituted). They are represented by vertical lines, horizontals and diagonals, respectively.

We define $\alpha(i, j)$ as the probability of arriving at the position (i, j) on the trellis. Then $\alpha(t_x, 0) = p_d^{t_x}$, $t_x = 0, 1, \ldots, n_x$ and $\alpha(0, t_y) = (0.5p_i)^{t_y}, t_y = 0, 1, \ldots, n_y$ are initialized. We use the forward program to calculate the middle term $P(\boldsymbol{r}|\boldsymbol{t}) = \alpha(n_x, n_y)$, i.e.,

$$\alpha(i,j) = \begin{cases} 0.5p_i\alpha(i,j-1) + p_d\alpha(i-1,j) + p_t p_s\alpha(i-1,j-1), \text{if } x_i \neq x_j, \\ 0.5p_i\alpha(i,j-1) + p_d\alpha(i-1,j) + p_t(1-p_s)\alpha(i-1,j-1), \text{if } x_i = x_j \end{cases}. \tag{4.4}$$

Recall in Fig. 2.4 that $\boldsymbol{s}$ is fixed for a given d_i. A forward pass is performed between

$x_{n\times i}$ and $x_{n\times(i+1)}$ and the result is returned to $P(\boldsymbol{r}_{n\times i+x_{n\times i}}^{n\times(i+1)+x_{n\times(i+1)-1}} x_{n\times(i+1)}|x_{n\times i}, s_{n\times i}^{n\times(i+1)-1})$. This is referred to as the soft-decision decoding.

4.2.3 Comparison Between Hard-decision and Soft-decision decoding

When more than one codeword has the same Hamming distance as the received corrupted codeword, the two decoding strategies are completely different. A codeword is randomly selected from the candidates in hard-decision decoding, while the codeword with the best ATP metric is selected for the soft-decision decoding. Therefore, the accuracy of the soft-decision decoding is much higher than that of the hard-decision decoding since only the symbol with the highest probability is returned.

4.3 Codebook Optimization

In this section, we focus on the design and performance of the sparsified code. Each symbol d_i is transformed into one of the lowest density binary vectors of length n with the sparsified transformation converter in DM construction. For sparsified code (15,4) in GF(16), the sparsified transformation converter maps each 16-ary symbol to a binary sequence of length 15. The codebook in [26] has a total of 16 codewords, including an all-zero codeword and 15 weight-1 codewords. The specific one-to-one mapping is shown in Table 4.1. However, for sparsified code (8,4) in GF(16), the DM construction does not describe the specific mapping between each q-ary symbol and the corresponding sparse codeword.

We conjecture that the performance of the inner watermark code depends not only on the sparsity, but also on the choice of the sparse representation mapping, i.e., better mutual codeword distance. The distance between sparsified codewords should be as large as possible. The performance will be improved when inner codes with larger

Table 4.1: One-to-One mapping in GF(16) for sparsified code (15,4)

symbol value d_i	correspoding codebook s_i
0	000000000000000
1	000000000000001
2	000000000000010
3	000000000000100
4	000000000001000
5	000000000010000
6	000000000100000
7	000000001000000
8	000000010000000
9	000000100000000
10	000001000000000
11	000010000000000
12	000100000000000
13	001000000000000
14	010000000000000
15	100000000000000

mutual distance are used. We propose a straight-forward and an improved sparsified mapping method for the sparsified code (8,4).

We suppose each symbol in GF(16) is mapped to a sparse codeword of length 8. The most straight-forward design is to map symbol “0” to all zero codeword, symbol “1” to symbol “7” to weight-1 codewords and map all the remaining symbols to weight-2 codewords, as shown in the first column of Table 4.2. Referring to this straight-forward mapping design, from symbol “0” there are 8 codewords at a Hamming distance of 1; from symbol “1” there are also 8 codewords at a Hamming distance of 1; and from each of the remaining symbols there are only 2 codewords at a Hamming distance of 1. A major disadvantage of the straight-forward design is that the inner decoder finds it difficult to distinguish between the sparse codewords due to their poor distance properties, especially between symbol “0” or symbol “1” and other symbols. Therefore, we improve the mapping design for sparsified code (8,4) and show it in the second column of Table 4.2. We adjust the positions of ones in the sparse sequences with weight 2 and replace the all-zero codeword with another codeword of weight 2. After the change, from each of the symbols there are no more than 2 codewords at a

Table 4.2: Straight-forward mapping and improved mapping design for sparsified code (8,4)

symbol value	straight-forward codebook	improved codebook
0	00000000	10000001
1	00000001	00000001
2	00000010	00000010
3	00000100	00000100
4	00001000	00001000
5	00010000	00010000
6	00100000	00100000
7	01000000	01000000
8	10000000	10000000
9	00000011	00000011
10	00000101	00000110
11	00001001	00001100
12	00010001	00011000
13	00100001	00110000
14	01000001	01100000
15	10000001	11000000

Hamming distance of 1 and the distance property of sparse codewords is optimized.

4.4 Simulation Results

With watermark sequences, the decoder can maintain synchronization easily at codeword boundaries when the error rates are low. Even though consecutive insertions may occur in the BSID channel, the number of consecutive insertions I is limited to 2 to reduce decoding complexity in our simulations. This assumption is reasonable especially when the error rate is low. Moreover, we focus on the symbol error rate (SER) of the inner code because it can be used as an indirect metric to measure the error performance of the concatenated code.

Table 4.3 shows the code parameters we use to evaluate the error performance. A sequence of around $N = 10^5$ bits are randomly generated and each block of k bits is mapped to a sparse binary sequence of length n. The transmitted sequence is then

Table 4.3: Parameters of Inner Watermark Codes: Length of Transmitted bits $N = n \times N_L$; Length of outer code symbols N_L; bits per symbol of an q-ary symbol k; sparsified code length n; Rate of Inner Watermark code $R_w = k/n$; Density of sparsified codes f

Code	N	N_L	k	n	R_w	density f
A	100002	16667	4	6	0.67	0.25
B	10^5	12500	4	8	0.50	0.172
C	10^5	10000	4	10	0.40	0.125

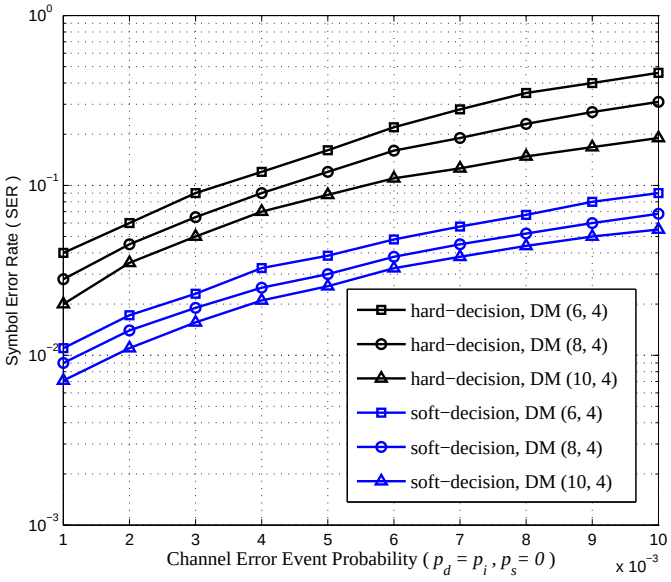


Figure 4.5: SER in different sparsified codes with soft-decision decoding and hard-decision decoding.

generated by adding modulo-2 the sparse sequence to the binary watermark sequence. The received corrupted sequence through the BSID channel is decoded with the inner symbol-level FB decoder using both hard-decision and soft-decision decoding. The decoded sequence is compared with the original input sequence and the number of erroneous decoded symbols are counted. The performance of different inner codes using hard-decision and soft-decision decoding in terms of SER are shown in Fig. 4.5. From the simulation results, we find codes with a lower code rate perform better. In other words, the performance improves as the sparsified codeword length n increases. This

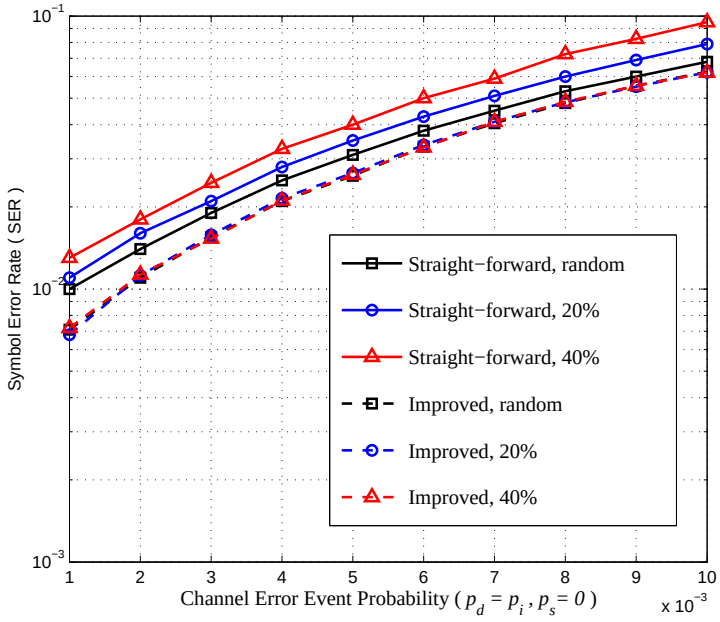


Figure 4.6: Symbol error rate in sparsified code (8,4) with straight-forward and improved mapping under different percentage of symbols 0 and 1.

is reasonable because codes with lower code rates have relatively low density, which have less impact on the watermark sequence. Furthermore, the error performance of soft-decision decoding is improved compared with hard-decision decoding in terms of SER.

Next, we compare the error performance of the sparsified code (8,4) with both straight-forward and improved mapping strategies under different channel error rates. In addition to generating random symbols from GF(16), we also generate a series of symbols, where the total number of symbols 0 and 1 account for different percentages of the total number of symbols. The simulation result for the sparsified code (8,4) with the straight-forward and improved mapping under different percentage of symbol 0 and 1 are shown in Fig. 4.6. From the simulation results, we notice that the performance with the improved mapping strategy is improved compared with that of the straight-forward one. Furthermore, we notice the percentage of symbol 0 and symbol 1 made a larger impact on the performance with the straight-forward mapping strategy. For

the straight-forward mapping strategy, SER increases with the percentage of symbols 0 and 1 and the difference becomes more obvious as channel parameters increase. For example, the number of error symbols for data generation mode at 40% increases almost 38% compared with randomly generated data under $p_i = p_d = 0.01$. However, this issue does not exist for the improved mapping strategy. The simulation result for the improved mapping strategy is that all three curves almost coincide. The symbols in error are distributed evenly. This is in consistent with the explanation described in the previous section. We confirm that the performance of the inner watermark code with the improved mapping strategy performs better, especially when certain symbols account for a high percentage.

4.5 Summary

The symbol-level watermark decoding algorithm considers the bit sequence representing a symbol instead of each bit every time. For the symbol-level decoding algorithm, we first propose a hard-decision decoding based on Hamming distance and then a more accurate soft-decision decoding based on a new metric ATP. We further propose an improved mapping strategy to optimize the inner sparsified codebook based on both better distance property and lowest density property. The simulation results show that the optimized watermark code with the improved mapping strategy has better performance. Even though the Davey Mackay watermark code can correct multiple insertion, deletion and substitution errors, the computational complexity is extremely high. In the next chapter, we will propose an efficient concatenated code to correct multiple errors with reduced decoding complexity.

Chapter 5

A Concatenated RS-Marker Code for Channels with Random Insertion, Deletion and Substitution Errors

Even though the Davey MacKay watermark code can correct multiple insertion, deletion and substitution errors, it has extremely high computational complexity [26, 69]. In this chapter, we propose a concatenated RS-marker code with designed markers for channels impaired by random insertion, deletion and substitution errors. The inner decoder can effectively maintain synchronization at codeword boundary with the help of the designed markers, while the outer Reed Solomon code is used to provide error correction capability. The frame error rate of the concatenated RS-marker code is simulated and compared with that of DM watermark code over the BSID channel with random insertion, deletion and substitution errors.

The rest of the chapter is organized as follows. In Sect. 5.1, we present the concatenated code with designed markers, including the overview of the system, inner designed marker code, decoding algorithm and complexity. In Sect. 5.2, we simulate the error performance of the proposed concatenated RS-marker code and compare it with DM watermark code over the BSID channel. Sect. 5.3 summarizes the chapter.

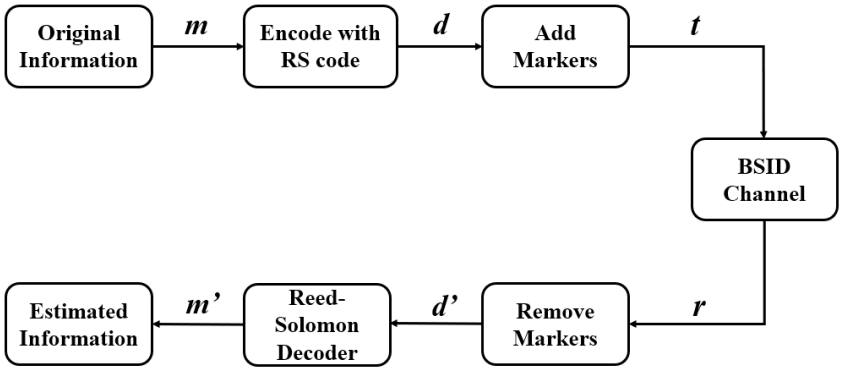


Figure 5.1: Structure of the concatenated RS-marker code.

5.1 Proposed Concatenated Marker Code

5.1.1 System Overview

This system is constructed from an inner designed marker code to maintain synchronization and an outer Reed-Solomon (RS) code to correct erasures and erroneous symbols output by the inner marker decoder [73]. The outer RS (N, K, k) code with k bits per symbol takes K information symbols $\boldsymbol{m} = (m_1, m_2, \ldots, m_K)$ and emits N symbols $\boldsymbol{d} = (d_1, d_2, \ldots, d_N)$. Subsequently, each symbol d_i of $\boldsymbol{d}$ is mapped to a binary sequence of length k and the designed markers are periodically inserted in the sequence $\boldsymbol{d}$ to form a frame F. The frame F constructed from the designed markers and N encoded symbols is subsequently prepared for transmission over the BSID channel. In addition, the content and position of the designed markers in the transmitted sequence are known by the decoder and encoder. The decoder exploits the designed markers to detect synchronization errors. The structure of the concatenated RS-marker code is shown in Fig. 5.1.

5.1.2 Error/Erasure Correction Capability of Reed-Solomon Code

The relationship between k and the maximum N is $N = 2^k - 1$ for a RS (N, K, k) code. For instance, the maximum N for a RS code with $k = 8$ is 255. An RS code

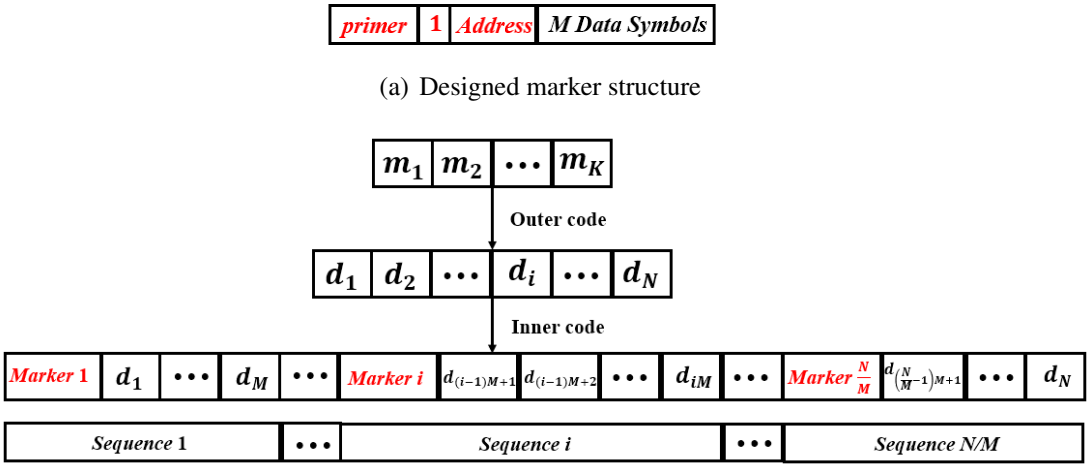
can correct up to t erroneous symbols, where $2t = N - K$. If the locations of the erroneous symbols are known, the erroneous symbols can be considered as erasures and the RS code can correct up to $2t$ erasures. When both erasures and errors with unknown locations exist, the corrupted received codeword can be decoded correctly only when $2m + n \leq 2t = N - K$, where m is the number of erroneous symbols and n is the number of erasures.

5.1.3 Inner Designed Marker Code

We consider a RS (N, K, k) code and we divide the N coded symbols into blocks of M symbols. We also assume the number of blocks N/M is an integer. A designed marker is inserted at the beginning of each block and it consists of two components: a primer drive and an address index. The primer drive is used to identify the start of the marker and is set as a sequence of $k + 1$ 0s. The address index is to indicate the order of the symbol blocks and consists of a minimum of $\lceil \log_2 (N/M) \rceil$ bits. The first block has an address of $N/M - 1$, the second one $N/M - 2$, and so on, and the last block has an address of 0. We also use a 1 as a delimiter to separate the primer and the address index. The structure of the designed marker and the encoding procedure of the concatenated RS-marker code is shown in Fig. 5.2(a) and Fig. 5.2(b), respectively. The code rate of the concatenated code is $R = \frac{K}{N} \cdot \frac{M \cdot k}{M \cdot k + p + a + 1}$, where p is the primer length and a is the address index length.

5.1.4 Decoding Algorithm

We assume a BSID channel. The inner marker decoder is assumed to process one received frame at a time, and each frame contains the coded symbols of one RS codeword. The decoder reads the frame and searches for valid primers and record the address index that follows. The recorded addresses are compared with the sequence $(N/M - 1, N/M - 2, \ldots, 0)$ to obtain the maximum monotonic decreasing convention



(a) Designed marker structure

(b) encoding procedure

Figure 5.2: Encoding procedure of concatenated RS-marker code

subsequence (MMDCS) set A, the size of which is denoted by $|A|$. For each valid address not in the set A, the inner decoder returns M erased symbols corresponding to the missing address.

Since a RS (N, K, k) code can correct $N - K$ erasures, the RS decoder requires to receive at least K correct symbols in order to correctly decode the codeword. The inner decoder subsequently compares the value of $|A|$ with K/M. If the number of addresses, i.e., $|A|$, is smaller than K/M, this frame cannot be decoded correctly. Otherwise, if $A \geq K/M$, we send the $|A|M$ to the RS decoder for decoding. The integrated decoding algorithm is shown in Fig. 5.3.

5.1.5 Decoding Complexity of the Inner Marker Code

The inner decoder reads a received frame and attempts to find all valid primers and their associated address indices. When a valid primer and its associated address index have been identified, the decoder needs to perform a counting procedure in order to confirm the validity of the primers. The steps are linear with N and the complexity is $O(N)$. The next step is to find the MMDCS between the address index sequence A and the sequence $(N/M - 1, N/M - 2, \ldots, 0)$, and its complexity is $O\left(\frac{N}{M} \log_{10} \frac{N}{M}\right)$ with the

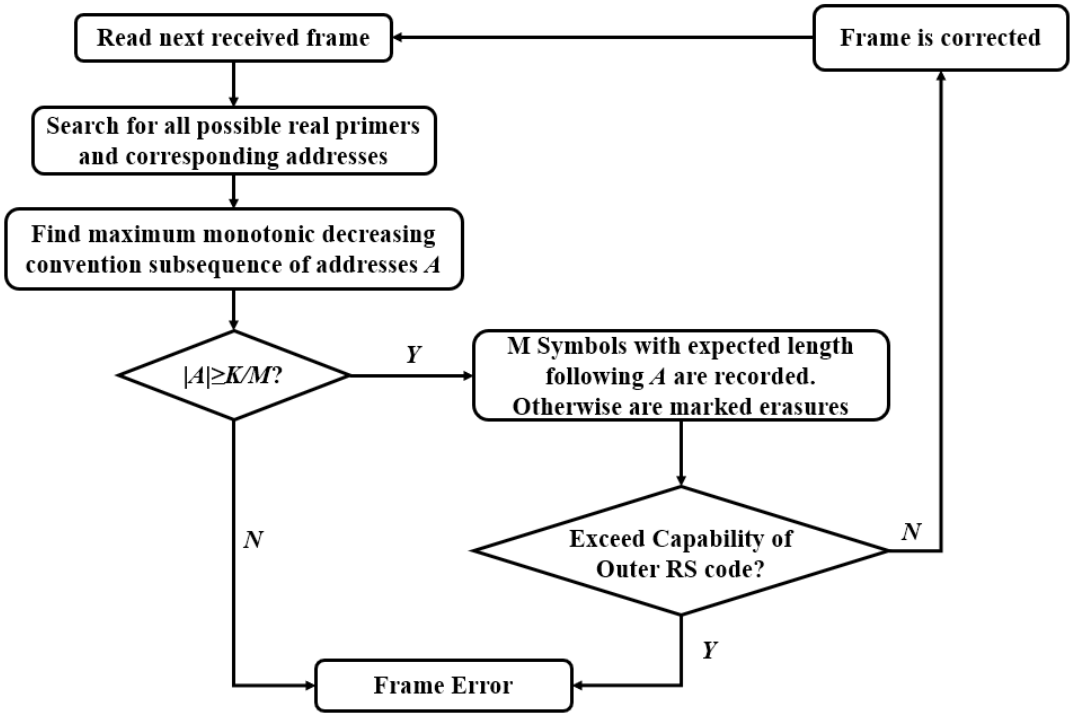


Figure 5.3: Flow chart of decoding algorithm of the concatenated RS-marker code.

use of dynamic programming [74]. Therefore, the overall complexity is $O\left(\frac{N}{M}\log_{10}\frac{N}{M}\right)$, which is much lower than that of the watermark code illustrated in Sect. 2.4.4.

5.2 Simulation Results

We use the RS (255,223,8) code with $k = 8$ bits per symbol as the outer code. This code can correct up to 16 (= $(N-K)/2$) erroneous symbols or 32 (= $N-K$) erasures in each codeword (frame). Since the error correction capability of the outer RS code is known, we only need to simulate the error performance of the inner marker code and use it to derive the overall error performance of the concatenated system.

To simulate the inner code, we generate 255 8-bit symbols (total 2040 bits) randomly. These symbols are divided into blocks of M symbols and then encoded into a frame according to Sect. 5.1.3. $M = 3$ and $M = 4$ are used our study. A total of 10^5 frames are generated and sent through the BSID channel in the simulation. Sub-

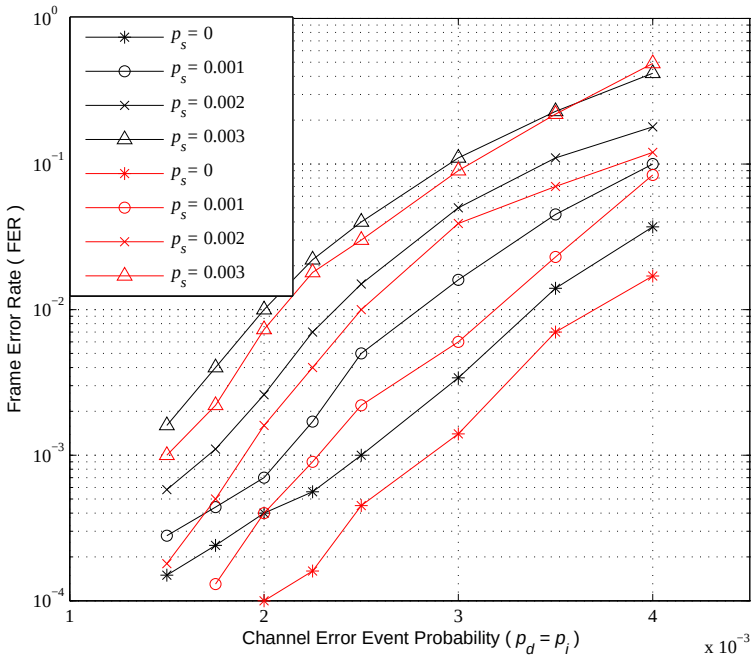


Figure 5.4: Frame error rate of the concatenated RS-marker code with $M = 3$ at rate 0.51 and the watermark code at rate 0.71 under different channel parameters. Red and black curves represent marker code with $M = 3$ and watermark code, respectively.

sequently, the decoding is performed accordingly to the flow chart in Fig. 5.3. The number of erasures and erroneous symbols are counted for each frame. The frame can be decoded correctly if errors are within the correction capability of the RS code, otherwise the frame is erroneous.

The error performance of the inner code with $M = 3$ and $M = 4$ concatenated with RS (255,223,8) outer code in terms of FER is shown in Fig. 5.4 and Fig. 5.5, respectively. In addition, the performance is also compared with that of the DM watermark code at code rate 0.71. From the simulation results, we find marker code with $M = 3$ at rate 0.51 performs much better compared with the watermark code at rate 0.71. The performance of the marker code with $M = 4$ at rate 0.58 performs as well as, even better than the watermark code at low substitution error rates p_s. However, the error performance becomes worse than that of the watermark code as the substitution error rate increases. We have also simulated the results for the cases $M = 2$ and $M = 5$.

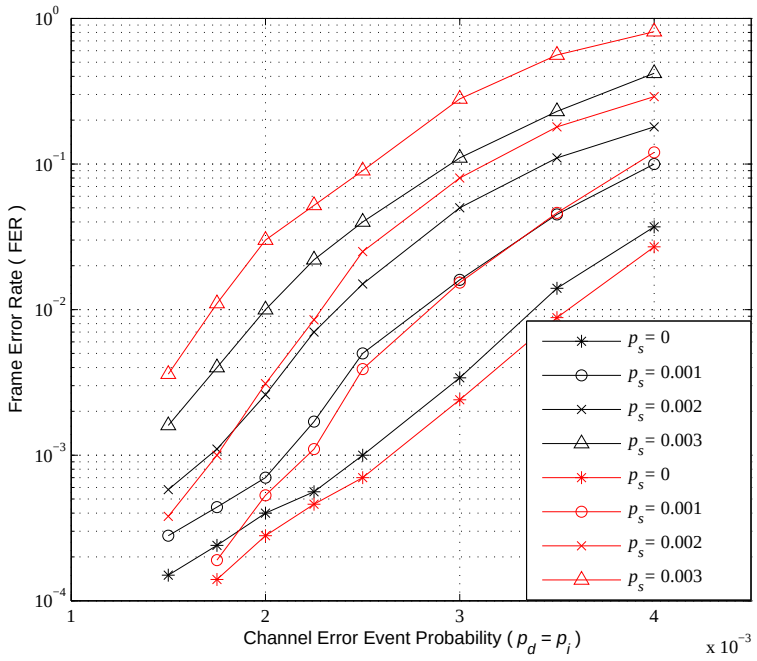


Figure 5.5: Frame error rate of the concatenated RS-marker code with $M = 4$ at rate 0.58 and the watermark code at rate 0.71 under different channel parameters. Red and black curves represent marker code with $M = 4$ and watermark code, respectively.

Although the marker code with $M = 2$ performs much better than watermark code, its code rate is very low. On the other hand, the error performance of the marker code with $M = 5$ is very poor.

Remark: In the watermark code in [26], soft messages, i.e., log-likelihood ratios, are passed from the inner decoder to the outer LDPC decoder, which then makes use of iterative decoding to decode the codeword. In our proposed concatenated RS-marker code, the inner code only needs to pass hard messages, i.e., recovered symbols, to the outer RS decoder, which does not require iterations to perform decoding. Thus, our outer decoder is also much simpler compared with the outer LDPC decoder in [26].

5.3 Summary

In this chapter, we design a concatenated RS-marker code with the capability to correct multiple random insertion, deletion and substitution errors. Simulation results show that the performance of our code at code rate 0.51 is much better than that of the watermark code at code rate 0.71. Even though our code rate is lower than that of the watermark code, the complexities of both the inner decoder and the outer decoder are much simpler. Thus, our system achieves a great reduction in decoding complexity, but at the cost of reducing the code rate. Chapter 4 and Chapter 5 focus on error correction codes developed for channels with random independent insertion, deletion and substitution errors. However, correlated insertion and deletion errors exist in some practical applications. In the next chapter, we will study error correction codes for channels with correlated errors.

Chapter 6

A Concatenated LDPC-Marker Code for Channels with Correlated Insertion and Deletion Errors

In Chapter 4 and Chapter 5, we have investigated error correction codes used in channels where insertion and deletion errors occur randomly. Very few works focus on channels with correlated synchronization errors. In fact, correlated synchronization errors exist in some practical applications, such as bit-patterned media recording (BPMR) systems. BPMR writes each data symbol into a single domain magnetic bit island [75]. Ideally, the writing head is located at the center of each bit island. Each bit island is written successfully when at least half of the bit island period overlaps with the write signal clock cycle. However, timing jitter, such as write clock stability, head vibration and disk-speed variation can result in the loss of synchronization between the intended write signal and the predefined bit island positions during the writing process [76]. As a result, a shorter write signal clock cycle causes a deletion error while a longer one results in an insertion error, as shown in Fig. 6.1 and Fig. 6.2, respectively.

Moreover, deletions and insertions often occur in pairs (i.e., a deletion is followed by an insertion in a latter position, and vice-versa) in BPMR [77]. Particularly, ex-

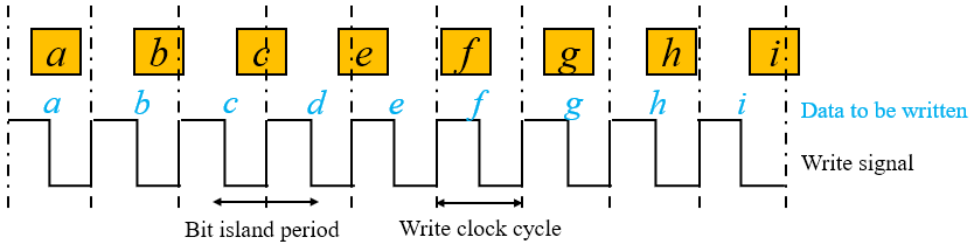


Figure 6.1: The write signal clock cycle is shorter than the bit island period. A deletion occurs and the bit *d* is deleted.

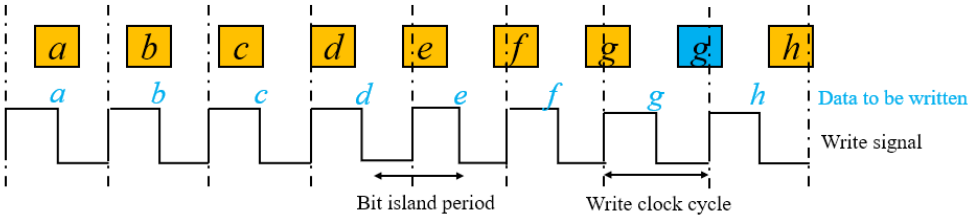


Figure 6.2: The write signal clock cycle is longer than the bit island period. An insertion error occurs and one more bit *g* is inserted.

periments are conducted to provide written-in error statistics and experimental data is presented in [78], where synchronization is first lost due to an insertion and then resynchronize following a deletion, and vice-versa. It is further noted that the subsequent synchronization error is more likely to occur at the position closer to the preceding one and the error probability decreases with the distance. Thus, insertions and deletions are correlated in the writing process of BPMR. Moreover, the writing head sometimes is not strong enough to write a bit and thus random substitution errors occur during the writing process.

Several channel models adapted to the writing process of BPMR have been proposed. More specifically, Hu *et al.* have proposed a mathematical channel with written-in errors from a signal processing perspective [79]. They have emphasized the importance of a channel model incorporating insertion, deletion and substitution errors. However, they only consider substitution errors to reduce complexity. Dependence of written-in errors is also considered in [80], but the only case studied is a bit erroneously changed to the preceding value when binary data is written on the medium.

In [1], a channel model with correlated insertion and deletion errors adapted to BPMR systems is presented. However, the proposed channel is not accurate enough to mimic the impairments of BPMR. While it forbids the occurrence of consecutive insertion errors or consecutive deletion errors, it does not consider insertion and deletion errors occurring in pairs.

In order to deal with the correlated paired synchronization errors and present a more accurate channel model for the BPMR systems, we propose a probabilistic channel model with correlated insertion and deletion errors. The channel model captures, for example, the data dependence adapted to the write channel in BPMR more accurately. Furthermore, we concatenate an LDPC code with a marker code to combat correlated insertion and deletion errors, in addition to substitution errors. We also elaborate a marker decoding algorithm based on a two-dimensional state transition diagram.

The rest of the chapter is organized as follows. Sect. 6.1 introduces the proposed channel model with correlated errors, the proposed concatenated LDPC-marker code, and the encoding and decoding steps. Sect. 6.2 describes the details of the decoding algorithm based on a two-dimensional transition diagram. Sect. 6.3 shows the simulated error performance of the concatenated LDPC code over the proposed channel. Sect. 6.4 gives a summary of the chapter.

6.1 Concatenated LDPC-Marker Code

6.1.1 Proposed CID Channel with Correlated Insertion and Deletion Errors

In this section, we propose a probabilistic channel with correlated insertion and deletion (CID) errors, i.e., an insertion and a deletion occur in pairs while substitution errors occur randomly within a sequence. Such a channel is referred to as a correlated insertion/deletion (CID) channel [81].

Given a transmitted sequence $\boldsymbol{x}_1^t = (x_1, x_2, \ldots, x_t) \in \{0,1\}^t$, we assume synchronization errors occur to $x_{a_1}, x_{b_1}, x_{a_2}, x_{b_2}, x_{a_3}, x_{b_3}, \ldots$ where $a_1 < b_1 < a_2 < b_2 < a_3 < b_3 < \cdots$. We also call (x_{a_n}, x_{b_n}) $(n = 1, 2, \ldots)$ a synchronization error bit pair.

The characteristics of the proposed probabilistic channel model are as follows.

1. The synchronization error probability of x_{a_n} is determined by the BSID channel.

2. We assume x_{a_n} and x_{b_n} suffer from different synchronization errors. In other words, if x_{a_n} suffers from an insertion error, x_{b_n} will suffer from a deletion error; and vice versa.

3. The synchronization error probability of x_{b_n} is determined by the synchronization error type of x_{a_n} and the bit separation l between x_{a_n} and x_{b_n}. The synchronization error probability of x_{b_n} decreases as the separation l increases.

We consider the synchronization error bit pair (x_{a_n}, x_{b_n}). After a synchronization error has occurred to x_{a_n}, we denote the probability of the synchronization error occurring at x_{a_n+l} by $\Pr(x_{b_n} = x_{a_n+l})$. Since $\Pr(x_{b_n} = x_{a_n+l})$ is a decreasing function of l, we assume that $\Pr(x_{b_n} = x_{a_n+l})$ $(l = 1, 2, \ldots)$ forms a geometric progression with a common ratio of $r < 1$, i.e.,

$$\Pr(x_{b_n} = x_{a_n+l}) = Ar^{l-1}; l = 1, 2, \ldots \tag{6.1}$$

where A is a constant. After a synchronization error has occurred to x_{a_n}, we use p_d', p_i' and p_t' to represent the new deletion probability, new insertion probability and new

transmission probability, respectively, for x_{a_n+l}. Moreover, they are given by

$$p'_d = \begin{cases} Ar^{l-1} & \text{if an insertion error has occurred to } x_{a_n} \\ 0 & \text{if a deletion error has occurred to } x_{a_n} \end{cases} \tag{6.2}$$

$$p'_i = \begin{cases} 0 & \text{if an insertion error has occurred to } x_{a_n} \\ Ar^{l-1} & \text{if a deletion error has occurred to } x_{a_n} \end{cases} \tag{6.3}$$

$$p'_t = 1 - p'_i - p'_d = 1 - Ar^{l-1}. \tag{6.4}$$

The probabilities will be reset to p_d, p_i and p_t once a synchronization error occurs to x_{b_n}. We further assume that random substitution errors always exist with an error probability p_s. The above operation repeats until all transmitted bits are sent.

6.1.2 Encoder

The presented scheme consists of an outer LPDC code and an inner marker code, as shown in Fig. 6.3. The outer code is used to correct errors while the role of the inner marker code is to maintain synchronization. Encoding is implemented in two steps: the message is first encoded into an outer LDPC code and then regular markers [51] with length N_m are inserted periodically every N_c LDPC code bits. For an LDPC code with length N, the regularly inserted markers divide the LDPC code into N/N_c segments (assuming N/N_c is an integer). Each of the segment consists of $N_c + N_m$ bits where the first N_c bits are the LDPC code bits and the last N_m bits are the marker bits. The content and position of the periodical markers in the transmitted sequence are known to and used by the receiver to maintain synchronization. The encoded sequence after the two encoding steps is ready to be transmitted through the channel. The code rate of the concatenated code is $R = R_C \cdot R_M$, where R_C and $R_M = \frac{N_c}{N_c+N_m}$ are the code rates of the LDPC code and the marker code, respectively.

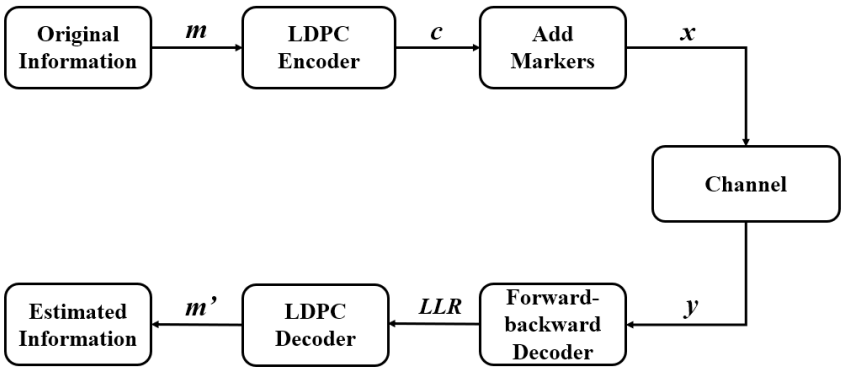


Figure 6.3: Flow chart of the encoding and decoding of the concatenated LDPC-marker code.

6.1.3 Decoder

Inner Marker Decoding

We denote $\boldsymbol{x}_1^t = (x_1, x_2, \cdots, x_t)$, $x_i \in \{0, 1\}$ for $i = 1, 2, \ldots, t$ and $\boldsymbol{y}_1^r = (y_1, y_2, \ldots, y_r)$, $y_i \in \{0, 1\}$ for $i = 1, 2, \ldots, r$ as the transmitted and the received sequences, respectively. At the receiver, the corrupted sequence $\boldsymbol{y}_1^r$ is first passed into the inner marker decoder. Given the received vector $\boldsymbol{y}_1^r$, the inner marker decoder makes full use of the information provided by the markers to compute the likelihood $p(\boldsymbol{y}_1^r|x_k)$ for $x_k \in \{0, 1\}$ and $k = 1, 2, \ldots, t$. To derive the likelihood $p(\boldsymbol{y}_1^r|x_k)$, the forward-backward algorithm [82] is used, as will be explained in Sect. 6.2.

The log-likelihood ratio (LLR) value $L(x_k \mid \boldsymbol{y}_1^r)$ is computed by the ratio between the conditional probabilities $\Pr(x_k = 0 \mid \boldsymbol{y}_1^r)$ and $\Pr(x_k = 1 \mid \boldsymbol{y}_1^r)$, i.e.,

$$L(x_k \mid \boldsymbol{y}_1^r) = \ln \frac{\Pr(x_k = 0 \mid \boldsymbol{y}_1^r)}{\Pr(x_k = 1 \mid \boldsymbol{y}_1^r)}. \tag{6.5}$$

Applying Bayes' rule to (6.5), we obtain

$$\begin{aligned} L(x_k \mid \boldsymbol{y}_1^r) &= \ln \frac{p(\boldsymbol{y}_1^r \mid x_k = 0) \cdot \Pr(x_k = 0)/p(\boldsymbol{y}_1^r)}{p(\boldsymbol{y}_1^r \mid x_k = 1) \cdot \Pr(x_k = 1)/p(\boldsymbol{y}_1^r)} \\ &= \ln \frac{p(\boldsymbol{y}_1^r \mid x_k = 0)}{p(\boldsymbol{y}_1^r \mid x_k = 1)} + \ln \frac{\Pr(x_k = 0)}{\Pr(x_k = 1)} \end{aligned} \tag{6.6}$$

where $\ln \frac{\Pr(x_k=0)}{\Pr(x_k=1)}$ denotes the *a priori* LLRs and $p(\boldsymbol{y}_1^r)$ denotes the probability density function of the received vector. Note that the *a priori* LLRs (i.e., $\ln \frac{\Pr(x_k=0)}{\Pr(x_k=1)}$) equal zero for LDPC code bits because 0 and 1 in LDPC codewords occur with equal probabilities. Let $\mathcal{L}$ be the set of positions where the outer LDPC code bits are located in the transmitted sequence. Substituting $\ln \frac{\Pr(x_k=0)}{\Pr(x_k=1)} = 0$ into (6.6), the LLR $L(x_k \mid \boldsymbol{y}_1^r)$ at bit position $k \in \mathcal{L}$ can be computed using

$$L(x_k \mid \boldsymbol{y}_1^r) = \ln \frac{p(\boldsymbol{y}_1^r \mid x_k = 0)}{p(\boldsymbol{y}_1^r \mid x_k = 1)} \quad k \in \mathcal{L} \tag{6.7}$$

which are then sent to the outer LDPC decoder for further decoding.

Outer LPDC Decoding

We apply the sum-product decoding algorithm (also called belief propagation algorithm) [83] to decode the original information by iteratively passing updated LLR messages between variable nodes and check nodes of the LDPC code. At the end of each iteration, the updated *a posteriori* LLR for each transmitted bit is calculated and an estimated codeword is obtained by making hard decisions based on the LLRs. The LDPC decoder subsequently checks whether the estimated codeword satisfies all check equations. The codeword is successfully decoded if all the check equations are satisfied; otherwise the iteration continues. If the maximum number of iterations is reached and not all check equations are satisfied, the decoding fails.

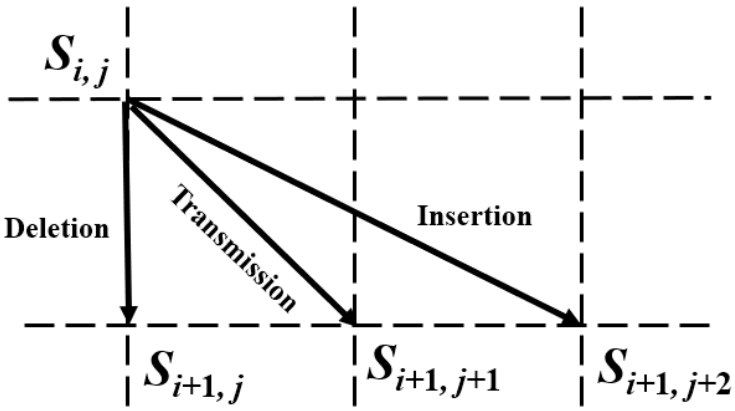


Figure 6.4: State transition paths for insertion, deletion and transmission when x_{i+1} is transmitted on a two-dimensional grid. (1) An insertion error corresponds to a state transition path across two cells from $S_{i,j}$ to $S_{i+1,j+2}$; (2) A deletion error corresponds to a state transition path from $S_{i,j}$ to $S_{i+1,j}$; (3) A transmission corresponds to a state transition path from $S_{i,j}$ to $S_{i+1,j+1}$. Notice that substitution errors are reflected in the figure. i increases downwards and j increases rightwards.

6.2 Forward-Backward Algorithm Based on the State Transition Diagram

6.2.1 State Transition Diagram

We define $S_{0,0}$ as the initial state when the transmission begins; and $S_{t,r}$ as the end state when t bits have been sent and r bits have been received. We also define an intermediate state $S_{i,j}$ when i bits ($i = 1, 2, \ldots, t$) have been sent and j bits ($j = 0, 1, \ldots, r$) have been received. State $S_{i,j}$ may transit to $S_{i+1,j}$, $S_{i+1,j+1}$ and $S_{i+1,j+2}$ if there is a deletion error, no deletion/insertion error, and an insertion error, respectively, for the transmitted bit x_{i+1}. The transitions are illustrated in Fig. 6.4. When the state transition paths corresponding to individual transmitted bits are connected together, a complete transmission path like the one shown in Fig. 6.5 is formed. For our CID channel, the complete state transmission path is bounded between the lower boundary ($i - j = 1$) and the upper boundary ($j - i = 1$).

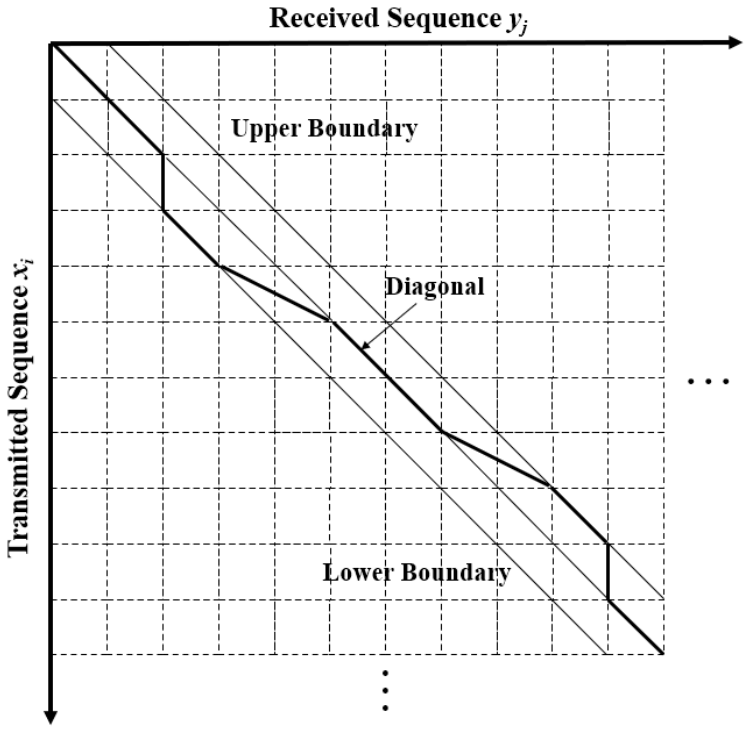


Figure 6.5: A complete transmission path over the channel representing a one-to-one mapping between the received and transmitted sequence on a two-dimensional grid diagram. The horizontal axis and vertical axis represent the received sequence and the transmitted sequence, respectively. The three thin lines represent the lower boundary, diagonal and upper boundary, respectively. The diagonal represents the transmission path under ideal condition without insertions or deletions while the bold line is the actual transmission path. The actual transmission path over CID channel must be between the lower and upper boundaries.

6.2.2 Forward-backward Decoding

In this section, we describe the decoding algorithm that the inner (marker) decoder used to compute the LLR for each transmitted bit over the CID channel. A transmitted bit x_i is either an LDPC code bit or a marker bit. For an LDPC code bit x_i, $\Pr(x_i = 0) = \Pr(x_i = 1) = 0.5$ because no information about an LDPC code bit x_i is provided to the decoder and the receiver does not know the value of x_i. However, when x_i is a bit from the marker, $\Pr(x_i)$ is a determined value, i.e., (a) $\Pr(x_i = 0) = 0$ and $\Pr(x_i = 1) = 1$; or (b) $\Pr(x_i = 0) = 1$ and $\Pr(x_i = 1) = 0$. The reason is that the exact value and position of a marker bit x_i are accurately known to the receiver.

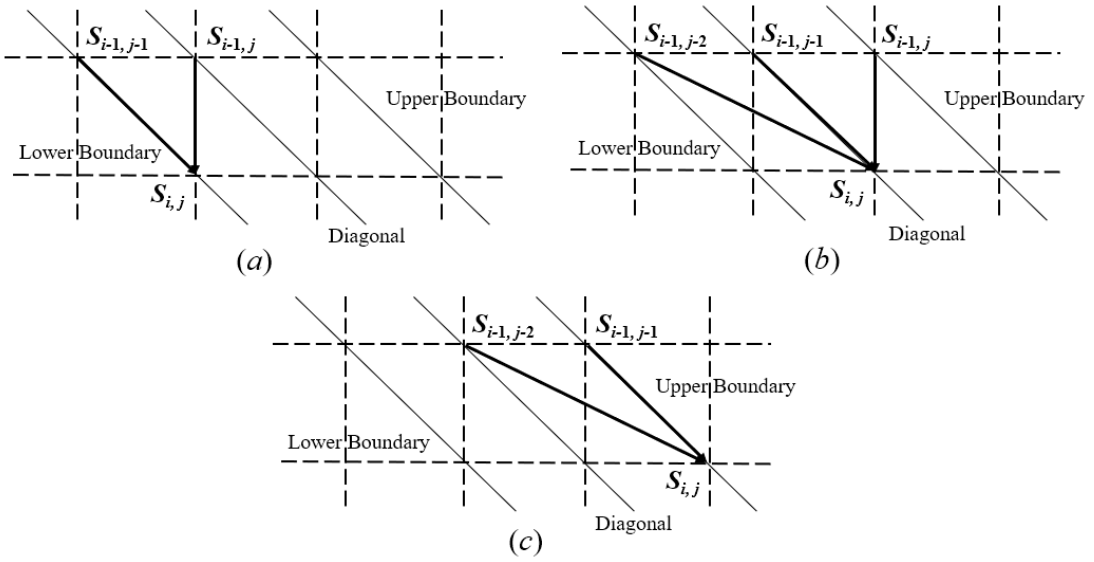


Figure 6.6: Possible state transitions when x_k is transmitted on condition that $S_{i,j}$ occurs. (a) $S_{i,j}$ is located on the lower boundary ($i - j = 1$); (b) $S_{i,j}$ is located on the diagonal ($i - j = 0$); (c) $S_{i,j}$ is located on the upper boundary ($j - i = 1$).

We define $T(x_i, y_j)$ as the transmission probability that x_i is transmitted (with/without insertion) and the corresponding received bit is y_j, i.e.,

$$T(x_i, y_j) = \Pr(x_i) \cdot C(x_i, y_j), \tag{6.8}$$

where $C(x_i, y_j)$ denotes the probability that the transmitted bit x_i and the corresponding received bit y_j have or do not have the same value. If $x_i = y_j$, x_i has been transmitted without any substitution error and thus $C(x_i, y_j) = 1 - p_s$; otherwise, x_i is substituted and $C(x_i, y_j) = p_s$. Thus, we have

$$C(x_i, y_j) = \begin{cases} 1 - p_s & \text{if } x_i = y_j \\ p_s & \text{if } x_i \neq y_j \end{cases}. \tag{6.9}$$

Forward Algorithm

The forward quantity $\alpha_{i,j}$ is the joint probability that i bits are transmitted and the j bits $\boldsymbol{y}_1^j = (y_1, y_2, \ldots, y_j)$ are received. In other words, it is the probability that $\boldsymbol{y}_1^j$ has been

received when the transmission state transits from $S_{0,0}$ to $S_{i,j}$. The forward quantity $\alpha_{i,j}$ is given by

$$\alpha_{i,j} = \Pr(\boldsymbol{y}_1^j, S_{i,j}). \tag{6.10}$$

Suppose x_i is transmitted over the CID channel. According to the relationship between i and j and the location of the previous state, three scenarios are to be considered in order to calculate $\alpha_{i,j}$.

Case 1: $S_{i,j}$ is located on the lower boundary when $i - j = 1$ and it can be reached from $S_{i-1,j-1}$ and $S_{i-1,j}$, as shown in Fig. 6.6(a).

x_i either suffers from a deletion error or is transmitted without any synchronization errors. (a) $S_{i-1,j}$ is located on the diagonal. Hence either no synchronization errors have occurred or synchronization errors have occurred in pairs before x_i is transmitted. The transition probability (deletion error) from $S_{i-1,j}$ to $S_{i,j}$ is thus p_d. (b) $S_{i-1,j-1}$ is located on the lower boundary and a deletion has already occurred before x_i is transmitted. The transition probability (transmission without synchronization errors) from $S_{i-1,j-1}$ to $S_{i,j}$ is thus p'_t.

Thus, when $i - j = 1$, $\alpha_{i,j}$ is given by

$$\alpha_{i,j} = p_d \alpha_{i-1,j} + p'_t \alpha_{i-1,j-1} T(x_i, y_j). \tag{6.11}$$

Case 2: $S_{i,j}$ is located on the diagonal when $i - j = 0$ and it can be reached from $S_{i-1,j-2}$, $S_{i-1,j-1}$ and $S_{i-1,j}$, as shown in Fig. 6.6(b).

x_i may transmit without synchronization errors or may suffer from a deletion or an insertion error. (a) The transition probability (deletion error) from $S_{i-1,j}$ to $S_{i,j}$ is p'_d since $S_{i-1,j}$ is located on the upper boundary and an insertion error has occurred before x_i is transmitted. (b) The transition probability (transmission without synchronization errors) from $S_{i-1,j-1}$ to $S_{i,j}$ is p_t since $S_{i-1,j-1}$ is located on the diagonal. (c) The transition probability (insertion error) from $S_{i-1,j-2}$ to $S_{i,j}$ is $0.5p'_i$ since $S_{i-1,j-2}$ is located

on the lower boundary and a deletion error has occurred before x_i is transmitted. An additional factor of 0.5 is necessary because the inserted bit y_{j-1} can be 0 or 1 with equal probability.

Thus, when $i - j = 0$, $\alpha_{i,j}$ is given by

$$\begin{aligned}\alpha_{i,j} &= p'_d\alpha_{i-1,j} + p_t\alpha_{i-1,j-1}T(x_i, y_j) + p'_i\alpha_{i-1,j-2}T(x_i, y_j)\Pr(\text{inserted bit} = y_{j-1}) \\ &= p'_d\alpha_{i-1,j} + p_t\alpha_{i-1,j-1}T(x_i, y_j) + \frac{p'_i}{2}\alpha_{i-1,j-2}T(x_i, y_j).\end{aligned} \tag{6.12}$$

Case 3: $S_{i,j}$ is located on the upper boundary when $j - i = 1$ and it can be reached from either $S_{i-1,j-2}$ or $S_{i-1,j-1}$, as shown in Fig. 6.6(c).

x_i either suffers from an insertion error or transmits without synchronization errors. (a) The transition probability (transmission without synchronization errors) from $S_{i-1,j-1}$ to $S_{i,j}$ is p'_t since $S_{i-1,j-1}$ is located on the upper boundary and an insertion error has occurred. (b) The transition probability (insertion error) from $S_{i-1,j-2}$ to $S_{i,j}$ is $0.5p_i$ since $S_{i-1,j-2}$ is located on the diagonal and the inserted bit y_{j-1} can be 0 or 1 with equal probability.

Thus, when $j - i = 1$, $\alpha_{i,j}$ is given by

$$\begin{aligned}\alpha_{i,j} &= p'_t\alpha_{i-1,j-1}T(x_i, y_j) + p_i\alpha_{i-1,j-2}T(x_i, y_j)\Pr(\text{inserted bit} = y_{j-1}) \\ &= p'_t\alpha_{i-1,j-1}T(x_i, y_j) + \frac{p_i}{2}\alpha_{i-1,j-2}T(x_i, y_j).\end{aligned} \tag{6.13}$$

In summary, given the initial conditions $\alpha_{0,0} = 1$ and $\alpha_{0,1} = 0$, all other values of $\alpha_{i,j}$ within the boundaries in Fig. 6.5 can be calculated iteratively.

Backward Algorithm

Similarly, the backward quantity $\beta_{i,j}$ denotes the probability that the remaining $r - j$ received bits are $\boldsymbol{y}_{j+1}^r = (y_{j+1}, \ldots, y_r)$ given the transmission state $S_{i,j}$ has occurred.

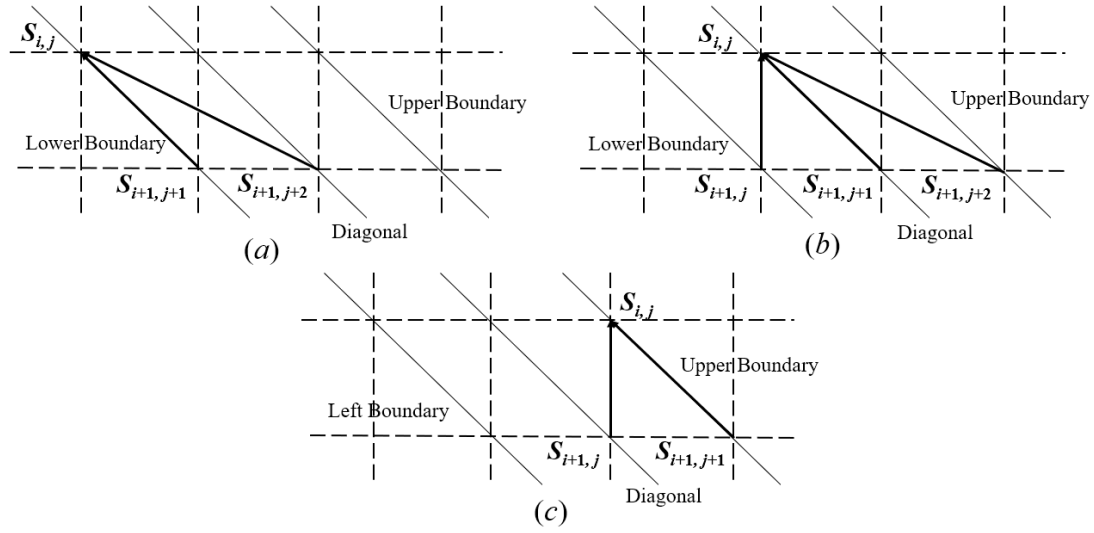


Figure 6.7: Possible state transitions when x_{i+1} is transmitted on condition that $S_{i,j}$ has occurred. (a) $S_{i,j}$ is located on the lower boundary ($i - j = 1$); (b) $S_{i,j}$ is located on the diagonal ($j - i = 0$); (c) $S_{i,j}$ is located on the upper boundary ($j - i = 1$).

The backward quantity $\beta_{i,j}$ is therefore denoted by

$$\beta_{i,j} = \Pr(\boldsymbol{y}_{j+1}^{r} | S_{i,j}). \tag{6.14}$$

Suppose x_{i+1} is transmitted through CID channel. According to the relationship between i and j, three scenarios are to be considered in order to calculate $\beta_{i,j}$. Moreover, the channel error types and accompanying synchronization error probabilities of each transition over the CID channel depends on the location of $S_{i,j}$.

Case 1: $S_{i,j}$ is located on the lower boundary when $i - j = 1$ and it can transit to $S_{i+1,j+1}$ or $S_{i+1,j+2}$, as shown in Fig. 6.7(a).

A deletion has already occurred when $S_{i,j}$ is located on the lower boundary ($i - j = 1$). From this state, only transmission without synchronization errors and insertion error are possible. (a) The transition probability (transmission without synchronization errors) from $S_{i,j}$ to $S_{i+1,j+1}$ is p_t'. (b) The transition probability (insertion error) from $S_{i,j}$ to $S_{i+1,j+2}$ is $0.5p_i'$ because the inserted bit y_{j+1} can be 0 or 1 with equal probability.

Thus, when $i - j = 1$, $\beta_{i,j}$ is given by

$$\begin{aligned}\beta_{i,j} &= p'_t\beta_{i+1,j+1}T(x_{i+1}, y_{j+1}) + p'_i\beta_{i+1,j+2}T(x_{i+1}, y_{j+2})\Pr(\text{inserted bit} = y_{j+1}) \\ &= p'_t\beta_{i+1,j+1}T(x_{i+1}, y_{j+1}) + \frac{p'_i}{2}\beta_{i+1,j+2}T(x_{i+1}, y_{j+2}).\end{aligned} \tag{6.15}$$

Case 2: $S_{i,j}$ is located on the diagonal when $i - j = 0$ and it can transit to $S_{i+1,j}$, $S_{i+1,j+1}$ or $S_{i+1,j+2}$, as shown in Fig. 6.7(b).

Either no synchronization errors have occurred or synchronization errors have occurred in pairs when $S_{i,j}$ is located on the diagonal ($j - i = 0$). From this state, transmission, insertion and deletion may occur and the error probability is determined by p_d, p_i, p_t from the channel. (a) The transition probability (insertion error) from $S_{i,j}$ to $S_{i+1,j+2}$ is $0.5p_i$ since the inserted bit y_{j+1} can be 0 or 1 with equal probability. (b) The transition probability (deletion error) from $S_{i,j}$ to $S_{i+1,j}$ is p_d. (c) The transition probability (transmission without synchronization errors) from $S_{i,j}$ to $S_{i+1,j+1}$ is p_t.

Thus, when $i - j = 0$, $\beta_{i,j}$ is given by

$$\begin{aligned}\beta_{i,j} &= p_d\beta_{i+1,j} + p_t\beta_{i+1,j+1}T(x_{i+1}, y_{j+1}) + p_i\beta_{i+1,j+2}T(x_{i+1}, y_{j+2})\Pr(\text{inserted bit} = y_{j+1}) \\ &= p_d\beta_{i+1,j} + p_t\beta_{i+1,j+1}T(x_{i+1}, y_{j+1}) + \frac{p_i}{2}\beta_{i+1,j+2}T(x_{i+1}, y_{j+2}).\end{aligned} \tag{6.16}$$

Case 3: $S_{i,j}$ is located on the upper boundary when $j - i = 1$ and it can transit to $S_{i+1,j}$ or $S_{i+1,j+1}$, as shown in Fig. 6.7(c).

An insertion has already occurred when $S_{i,j}$ is located on the upper boundary ($j-i = 1$). From this state, only transmission without synchronization errors and deletion error can occur. The transition probability (transmission without synchronization errors) from $S_{i,j}$ to $S_{i+1,j+1}$ and the transition probability (deletion error) from $S_{i,j}$ to $S_{i+1,j}$ are p'_t and p'_d, respectively. Hence when $j - i = 1$, $\beta_{i,j}$ is given by

$$\beta_{i,j} = p'_d\beta_{i+1,j} + p'_t\beta_{i+1,j+1}T(x_{i+1}, y_{j+1}). \tag{6.17}$$

In summary, given the initial conditions $\beta_{t,r} = 1$ and $\beta_{t,r-1} = 0$, all other values of $\beta_{i,j}$ within the boundaries in Fig. 6.5 can be calculated iteratively.

Likelihood

The likelihood expressions for $p(\boldsymbol{y}_1^r|x_i)$ are first derived in terms of $\alpha_{i,j}$ and $\beta_{i,j}$ under three cases, i.e., x_i is deleted; x_i is transmitted without deletion or insertion error; and x_i suffers from an insertion error. They are derived as follows.

Case 1: Only two scenarios are possible when x_i is deleted.

$$p(\boldsymbol{y}_1^r|x_i \text{ is deleted}) = p_d\alpha_{i-1,j}\beta_{i,j}|_{j=i-1} + p'_d\alpha_{i-1,j}\beta_{i,j}|_{j=i}. \tag{6.18}$$

Case 2: Three scenarios are possible when x_i is transmitted without synchronization errors.

$$\begin{aligned} &p(\boldsymbol{y}_1^r|x_i \text{ is transmitted}) \\ &\quad = p'_t\alpha_{i-1,j-1}\beta_{i,j} \cdot C(x_i, y_j)|_{j=i-1} \\ &\qquad + p_t\alpha_{i-1,j-1}\beta_{i,j}C(x_i, y_j)|_{j=i} \\ &\qquad + p'_t\alpha_{i-1,j-1}\beta_{i,j}C(x_i, y_j)|_{j=i+1}. \end{aligned} \tag{6.19}$$

Case 3: Only two scenarios are possible when x_i is transmitted with an insertion error.

$$p(\boldsymbol{y}_1^r|x_i \text{ is inserted}) = \frac{p'_i}{2}\alpha_{i-1,j-2}\beta_{i,j}C(x_i, y_j)|_{j=i} + \frac{p_i}{2}\alpha_{i-1,j-2}\beta_{i,j}C(x_i, y_j)|_{j=i+1}. \tag{6.20}$$

Because all the cases are independent, the overall $p(\boldsymbol{y}_1^r|x_i)$ is the sum of (6.18), (6.19) and (6.20), where $x_i \in \{0, 1\}$. The LLRs are subsequently computed using (6.7) and are used as soft inputs to the LDPC decoder.

6.3 Simulation Results

In this section, simulations are conducted to investigate the error performance of the concatenated LDPC-marker code over the CID channel. First, messages are encoded by the LDPC code and then by the marker code. Then the encoded sequence is transmitted through the CID channel. Finally, the received corrupted sequence is decoded with the decoding algorithm mentioned in Sect. 6.2. Both the bit error rate (BER) and block error rate (BLER) are used as metrics to evaluate the performance of the concatenated code. For all simulations, we consider $p_i = p_d$ and set $A = r = 0.5$ in (6.1). The $(4521, 3552)$ LDPC code [84] with rate $R_C = 0.79$ is used as the outer code.

In the first set of simulations, regular markers '10' of length $N_m = 2$ are inserted at the beginning of each LDPC codeword, and also periodically every $N_c = 18$ LDPC code bits. In other words, each marker is followed by 18 LDPC code bits, except the last marker which is followed only by 3 code bits. Therefore, the total length N of the transmitted sequence is 5023 bits per codeword and the overall code rate of the concatenated LPDC-marker code is $R = 3552/5023 = 0.71$. The BLER and BER performance of the concatenated code are shown in Fig. 6.8 and Fig. 6.9, respectively, under different channel insertion, deletion and substitution error rates. For example, when $p_i = p_d = 3 \times 10^{-3}$ and $p_s = 0.01$, BLER and BER equals to 8×10^{-3} and 1.7×10^{-4}, respectively. As expected, both BLER and BER increase as p_i and/or p_d and/or p_s increases.

Next, we investigate the effect of the marker length N_m on the concatenated code. We fix $N_c = 18$ and use $N_m = 2, 3, 4$. The markers used are '10', '101' and '1010' while the corresponding overall code rates are 0.71, 0.67 and 0.64, respectively. Fig. 6.10 plots the BLER curves for concatenated codes with different marker length N_m under different insertion, deletion and substitution error rates. The simulation result indicates that the error performance of the concatenated code can be improved as the marker length N_m increases.

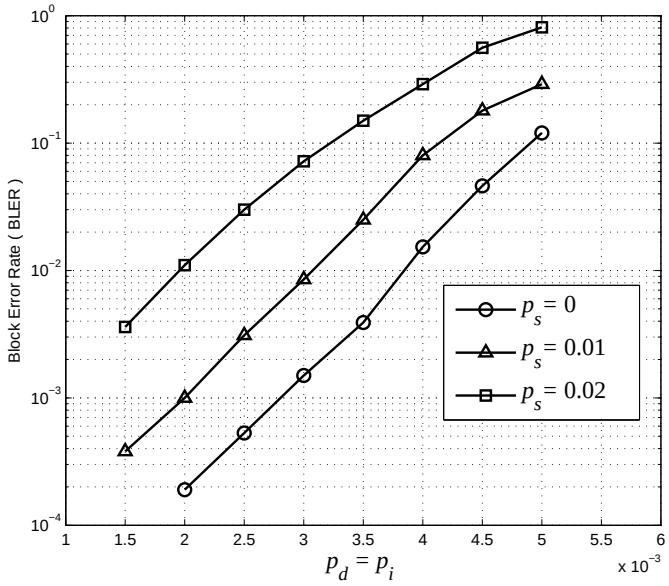


Figure 6.8: BLER of the concatenated LDPC-marker code over the CID channel. N_c = 18 and N_m = 2.

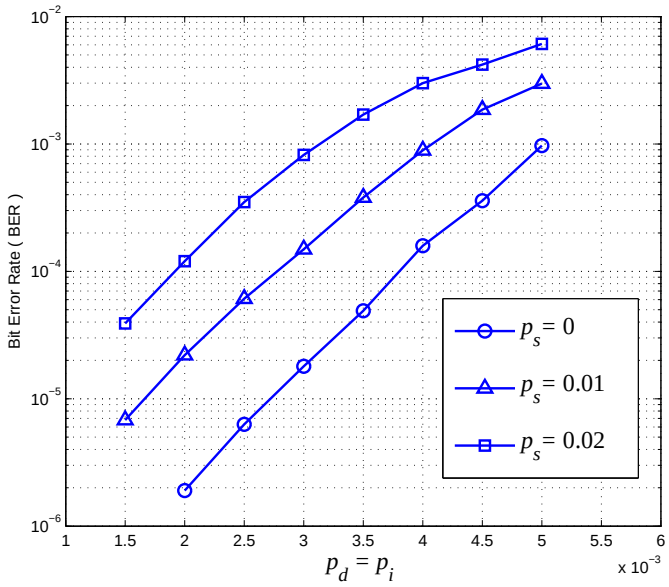


Figure 6.9: BER of the concatenated LDPC-marker code over the CID channel. N_c = 18 and N_m = 2.

We also study the effect of marker interval N_c on the error performance of the concatenated code. We fix $N_m = 2$ and use $N_c = 18, 24, 30$. The corresponding overall code rates are $0.71, 0.73$ and 0.74, respectively. Fig. 6.11 plots BLER curves. It can

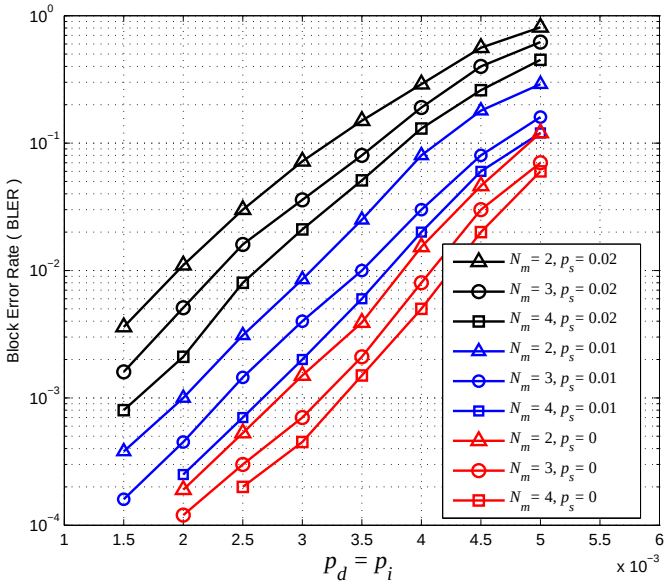


Figure 6.10: BLER of the concatenated LDPC-marker code over the CID channel. $N_c = 18$ and $N_m = 2, 3, 4$.

be observed that the error performance improves as the marker interval N_c decreases. In other words, inserting more markers enhances the error performance of the concatenated LDPC-marker code. The above results indicate that a lower marker code rate (increases N_m and/or reduces N_c) leads to a better synchronization capability and hence error performance. It is reasonable because more marker bits can provide more information for the receiver to locate the inserted and deleted bits more precisely.

Finally, we compare the error performance of the proposed concatenated LDPC-marker code with the result in [1]. The channel model used in [1] is fully described by a finite-state machine, i.e., the current event (insertion, deletion or transmission) is totally dependent on the previous event. The detailed channel model in [1] is described as follows.

1. When the previous event is transmission, the probability of insertion and deletion for the current event is p_i and p_d, respectively. The probability of a consecutive transmission event is thus $p_t = 1 - p_i - p_d$.

2. When the previous event is insertion, the probability of insertion and deletion for

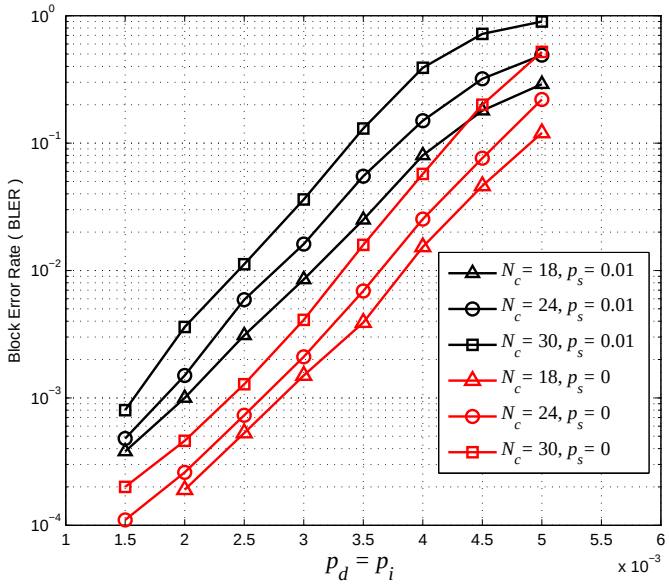


Figure 6.11: BLER of the concatenated LDPC-marker code over the CID channel. $N_m = 2$ and $N_c = 18, 24, 30$.

the current event is 0 and p_{id}, respectively. The probability of transmission event is $p_t = 1 - p_{id}$. The occurrence of consecutive insertion event is forbidden.

3. When the previous event is deletion, the probability of insertion and deletion for the current event is p_{di} and 0, respectively. The probability of transmission event is $p_t = 1 - p_{di}$. Again, the occurrence of consecutive deletion event is also forbidden.

Although the occurrence of consecutive insertion/deletion errors is forbidden in the channel model described in [1], the insertion event and the deletion event are not guaranteed to appear in pairs. For example, the event: insertion→transmission→insertion is valid in that channel model. Moreover, the synchronization error probabilities are fixed to $p_{id} = p_{di} = 0.5$, which is not consistent with the fact that the subsequent synchronization error is more likely to occur at a closer position of the preceding one.

In order to make a fair comparison with [1], we set $A = 0.5$ and $r = 1$ in (6.1). Thus, $p'_d = p'_i = 0.5$ are consistent with $p_{id} = p_{di} = 0.5$ in [1] for subsequent bits after a synchronization error. Furthermore, the inner marker code with $N_m = 2$ and $N_c = 18$

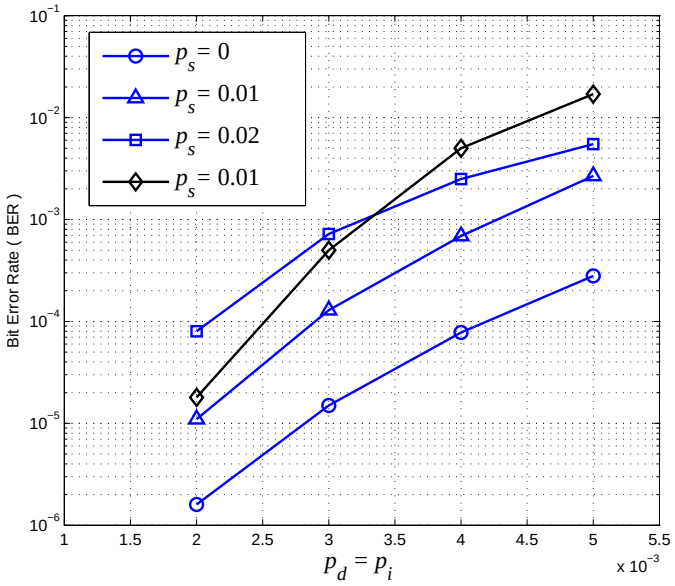


Figure 6.12: Comparision performance under different channel parameters. $p_d = p_i$ increase from 2×10^{-3} to 5×10^{-3} and p_s increases from 0 to 0.02. The blue curves represent the error performance of the proposed code with different substitution error rates and the black curve represents the error performance of the best code in [1] when the substitution error rate is 0.01.

is adopted so that both codes have the same code rate 0.71 for a fair comparison. The BLER curves of the proposed concatenated code over the CID channel and the BLER curve with the best performance in [1] under the same channel parameters are compared. The simulation result in Fig. 6.12 shows that the proposed code can perform much better than the best code in [1].

We also compare the error performance of the proposed concatenated code in Fig. 6.9 ($r = 0.5$) and Fig. 6.12 ($r = 1$). It can be observed that the error performance of the concatenated code is improved when r is increased from 0.5 to 1. The explanation is as follows. We consider the synchronization error bit pair (x_{a_n}, x_{b_n}).

When r is set to 1, (6.2) to (6.4) become

$$p'_d = \begin{cases} A & \text{if an insertion error has occurred to } x_{a_n} \\ 0 & \text{if a deletion error has occurred to } x_{a_n} \end{cases} \tag{6.21}$$

$$p'_i = \begin{cases} 0 & \text{if an insertion error has occurred to } x_{a_n} \\ A & \text{if a deletion error has occurred to } x_{a_n} \end{cases} \tag{6.22}$$

$$p'_t = 1 - p'_i - p'_d = 1 - A. \tag{6.23}$$

The probability that x_{b_n} occurs is no longer decreasing with its distance with x_{a_n}. Thus x_{b_n} is more likely to occur compared with the case $r = 0.5$. Once x_{b_n} has occurred, the received sequence is re-synchronized and the subsequent LLRs sent to the LDPC code becomes more accurate.

6.4 Summary

In this chapter, we have proposed a probabilistic channel with correlated synchronization errors. We have further concatenated an outer LDPC code with an inner marker code for such a channel. Explicit decoding algorithm is described in detail based on the transition path on a two-dimensional transition diagram. Simulation results have shown that the decoding algorithm is effective in correcting substitution and correlated synchronization errors. Moreover, the error performance is improved when more markers are inserted in the transmitted sequence. In the next chapter, we will propose an error correction scheme for another type of data storage system — DNA-based data storage systems. Specifically, we will present a technique for constructing GC-balanced DNA sequences with error correction capability.

Chapter 7

GC-balanced DNA Sequences with Error Correction Capability for DNA Storage Systems

7.1 Introduction

Data explosion due to extensive use of social networking and cloud mass storage produces an immense amount of data. However, a significant amount of data is in the form of storage. Therefore, it has imposed new challenges to strive for new medium to store the infrequently used data. Even though magnetic tapes, optical disks, hard drives, cloud data storages and other data storage systems have made great progresses in increasing the storage capacity, they are subject to many restrictions. They are prone to decay and need to be maintained regularly. In addition, they are not friendly to the environment because they consume energy and release a large amount of heat.

The possibility of storing messages in synthetic deoxyribonucleic acid (DNA) has been first demonstrated in [18]. It has been proposed that DNA could be a good storage medium for storing digital data for a long period due to its ultra-high density and outstanding durability [85]. DNA is a molecule that uses four nucleotide bases to store

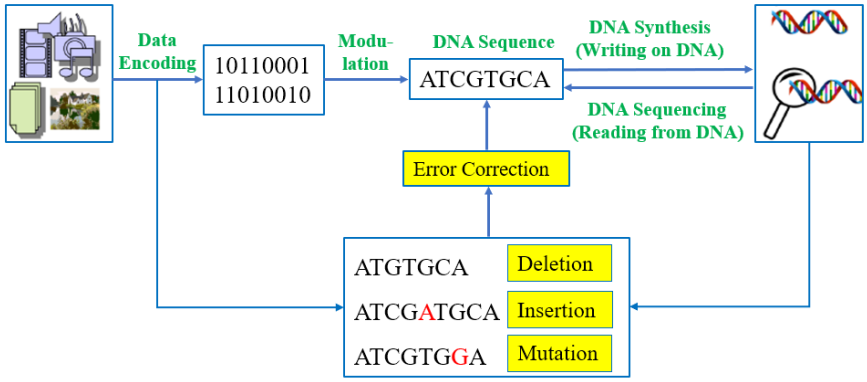


Figure 7.1: Digital data is encoded and modulated to DNA sequence. The corresponding DNA sequence is synthesized and stored in the DNA library. Original data is retrieved by the sequencing, demodulation and decoding processes.

genetic information, i.e., A (Adenine), C (Cytosine), G (Guanine), and T (Thymine). The potential benefits of DNA storage systems are as follows: (a) the storage density of DNA can be as high as 10^{18} bytes/mm^3, which is several orders of magnitude higher than traditional storage medium; and (b) DNA is stable, highly resistant to damage and can be stored for thousands of years.

The block diagram of a typical DNA-based data storage architecture is depicted in Fig. 7.1. Media such as text, images, music or videos are firstly converted into binary sequences with some standard algorithms. Modulation, defined as the mapping of a binary sequence to a DNA sequence, is then performed. Each binary sequence is therefore modulated to a DNA sequence consisting of nucleotide bases over an alphabet set {A, C, G, T}. One simple modulation strategy is to map 0 randomly to A or C and to map 1 randomly to G or T. This type of modulation has been adopted by Church *et al.* [20]. The DNA sequence is subsequently synthesized and stored in a DNA storage library. The original digital data can be retrieved by the sequencing, demodulation and decoding processes

However, synthesizing and sequencing DNA sequences are far from perfect [86]. Modern DNA synthesizers can generate DNA sequences of length up to 250 nucleotides with an acceptable error rate and the error rate increases significantly as the DNA

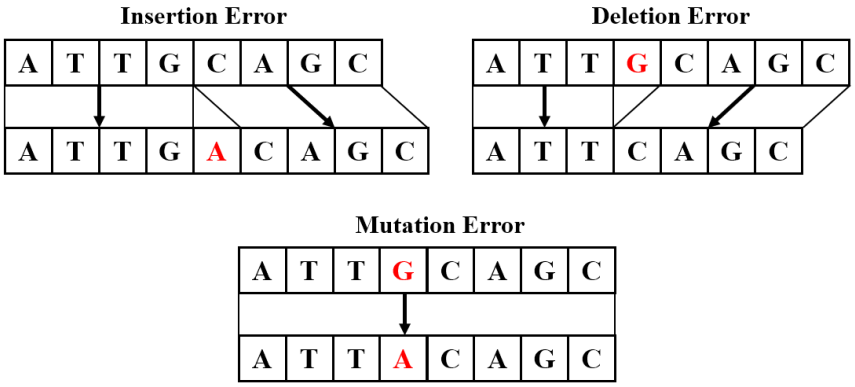


Figure 7.2: Three types of errors in synthesis and sequencing of DNA sequence. Insertion and deletion errors are collectively referred to as synchronization errors.

length becomes longer [87] [88]. Besides, DNA sequencing is not perfect and it introduces an average error rate of about 1% per nucleotide [89]. Most errors occurring in DNA synthesis and sequencing processes are categorized into insertion, deletion and mutation[1] as shown in Fig. 7.2. Moreover, insertion errors and deletion errors are collectively referred to as synchronization errors. One of the main challenges in realizing DNA storage systems is to design appropriate mechanisms to combat errors.

Several DNA data storage systems have been experimented over the past years. In [20], Church *et al.* have adopted a one-bit-to-one-base modulation strategy and have encoded messages in a way to avoid sequences that are difficult to write or read, such as repeats, extreme G/C content, or secondary structure. Multiple copies of each DNA sequence are then synthesized and stored in the DNA library. Since errors rarely happen in the same place during the sequencing process, errors in a DNA sequence can potentially be corrected by comparing multiple copies of the same DNA sequence. Yet, the strategy suffers from a relatively high error rate because it has not deployed any mechanism to protect the integrity of the DNA sequences. In [21], Goldman *et al.* have further implemented a single parity-check code and have overlapped DNA block contents for a four-fold redundancy. The method is not efficient because the

[1]Mutation errors occurring in DNA data storage systems are equivalent to substitution errors in conventional data storage and digital communication systems. In this chapter, mutation errors and substitution errors may be used interchangeably.

same data are repeatedly stored in four DNA fragments. Neither of the above two architectures has implemented error correction scheme beyond a single parity check code. Furthermore, the two systems do not directly deal with insertion or deletion of nucleotide bases. Due to the relatively low accuracy of DNA synthesis and sequencing, the lack of error correction scheme has become a crucial issue.

Recent research works are focusing on the error correction codes adapted to DNA-based storage systems. In [90], the construction of the constraint code for DNA storage systems is described, taking into account the maximum repetition length and the balance between AT and GC pairs. An improved algorithm is proposed in [91] to store information more efficiently considering Hamming distance and run-length constraints. The architecture in [92] has implemented an efficient coding technique to limit the length of repeated occurrences of the same base that are difficult to sequence. Information is encoded by two independent Reed-Solomon codes in a cascade manner in [23]. The work in [24] has applied a hybrid coding error protection scheme to ensure the integrity of the stored data during DNA sequencing.

DNA sequences capable of correcting both synchronization and mutation errors can store date efficiently and enhance the reliability of a DNA storage system. In addition, DNA sequences with 50% GC content are less susceptible to errors and more stable than those with higher or lower GC content. Extreme high or low GC content may result in more errors during DNA sequencing and synthesis [93]. The effect of GC constraint on the DNA sequence has been considered in [94], where a GC constraint is imposed on part of the DNA sequence. The design achieves a GC content close to but not exactly 50%. As far as we know, no effective DNA sequences with GC constraint and capable of correcting mutation, deletion or insertion errors have been proposed for DNA storage systems. For the above reasons, we focus on the construction of GC-balanced DNA sequence capable of correcting all three types of errors and with a GC content occupying exactly 50%.

This chapter presents the construction of a GC-balanced DNA sequence with error

correction capability. Specifically, we first propose a systematic single error correction code. The proposed code and the designed GC-balanced modulation scheme are combined to produce GC-balanced DNA sequences with the capability of correcting insertion, deletion and mutation errors. The rest of the chapter is organized as follows. Sect. 7.2 starts with the encoding and decoding of the proposed systematic single error correction code. Sect. 7.3 shows the modulation, construction and decoding of the designed GC-balanced DNA sequence with a single base error correction. Simulation results of the proposed code and the designed DNA sequence are shown in Sect. 7.4. Summary remarks are given in Sect. 7.5.

7.2 Proposed Systematic Error Correction Code

We propose a systematic error correction code based on the Levenshtein code. Being a subset of the Levenshtein code, the proposed code is not only a systematic code, but also possesses the capability of correcting a single insertion, deletion or substitution error [95].

7.2.1 Proposed Systematic Code

The proposed systematic code, denoted by $X(n, a, U)$, has a length of n and is capable of correcting a single insertion, deletion or substitution error. The message bits $m_1, m_2, \ldots, m_k$ of length k is encoded to the proposed codeword $x_1, x_2, \ldots, x_n$ of length n by adding r parity check bits $p_1, p_2, \ldots, p_r$, where $r = n-k$. The parity check bits are located at the 2^i-th positions ($i = 0, 1, \ldots$) as well as the last position of the codeword while the message bits are located at the remaining positions. The locations of the parity bits and message bits in the proposed systematic code is illustrated in Fig. 7.3. Note that we require the last code bit to be a parity bit and the second last code bit to

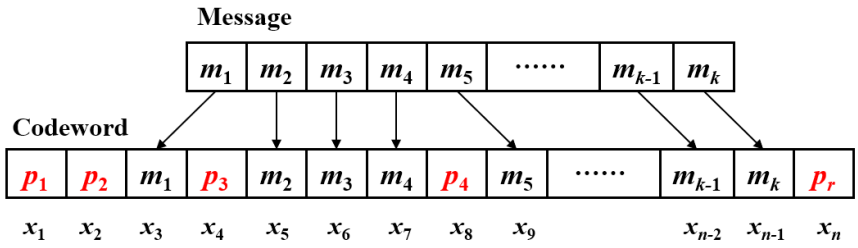


Figure 7.3: The locations of the parity bits and message bits in the proposed systematic code. The proposed codeword $x_1, x_2, \ldots, x_n$ is constructed from the message sequence $m_1, m_2, \ldots, m_k$ and the parity check bits $p_1, p_2, \ldots, p_r$.

be a message bit. For a given k, the code length n can be computed by

$$n = \min \tilde{n} \quad \text{s.t.} \quad k = \tilde{n} - \lceil \log_2 \tilde{n} \rceil - 1 \tag{7.1}$$

where $\lceil \tilde{n} \rceil$ represents the minimum integer equal to or greater than $\tilde{n}$. For example, when $k = 4$, we have $\tilde{n} = 8, 9$ and hence $n = 8$. When $k = 5$, we have $\tilde{n} = 10$ and hence $n = 10$.

Referring to Fig. 7.3, the code bits can be expressed in terms of the message bits and parity check bits according to

$$x_i = \begin{cases} p_{1+\log_2 i} & i = 2^0, 2^1, \ldots, 2^{r-2}, \\ p_r & i = n, \\ m_{i-\lceil \log_2 i \rceil} & i \neq 2^0, 2^1, \ldots, 2^{r-2} \text{ and } i \neq n. \end{cases} \tag{7.2}$$

Moreover, we require the code satisfying

$$\begin{aligned} X(n, a, U) = \{&(x_1, x_2, \cdots, x_n) \in \{0,1\}^n : \\ &\sum_{i=1}^{n} i \cdot x_i \equiv a \mod U, \ \ 2n \leq U \leq n + 2^{n-k-1}\} \end{aligned} \tag{7.3}$$

where $0 \leq a \leq U - 1$. The equation governing the code bits can be further re-written

as

$$\sum_{\substack{i=1 \\ i\neq 2^0,2^1,\ldots,2^{r-2}}}^{n-1} i\cdot m_{i-\lceil \log_2 i\rceil} + \sum_{j=0}^{r-2} 2^j \cdot p_{j+1} + n\cdot p_r \equiv a \pmod{U},$$
$$2n \leq U \leq n + 2^{n-k-1}. \tag{7.4}$$

Theorem 5 *There exists at least one U that satisfies $2n \leq U \leq n + 2^{n-k-1}$.*

Proof. Since all n, k and U are positive integers, we only need to prove $2n \leq n + 2^{n-k-1}$.

$$\begin{aligned} & 2n \leq n + 2^{n-k-1} \\ \Leftrightarrow \quad & n + 2^{n-k-1} - 2n \geq 0 \\ \Leftrightarrow \quad & 2^{n-k-1} - n \geq 0 \\ \Leftrightarrow \quad & 2^{\lceil \log_2 n\rceil} - n \geq 0 \\ \Leftrightarrow \quad & \lceil \log_2 n\rceil - \log_2 n \geq 0 \Leftrightarrow \text{true} \end{aligned} \tag{7.5}$$

Theorem 6 *For a given specific message of length k and a fixed value U with $2n \leq U \leq n+2^{n-k-1}$, there exists at least one set of parity-check bits $\{p_1, p_2, \ldots, p_r\}$ satisfying* (7.4).

Proof. Let

$$\alpha = \sum_{\substack{i=1 \\ i\neq 2^0,2^1,\ldots,2^{r-2}}}^{n-1} i\cdot m_{i-\lceil \log_2 i\rceil} \ , \tag{7.6}$$

$$\beta = \sum_{j=0}^{r-2} 2^j \cdot p_{j+1} \ , \tag{7.7}$$

$$\gamma = n\cdot p_r \ . \tag{7.8}$$

Since the parity bits $p_1, p_2, \ldots, p_r$ are either 0 or 1, it can be readily proved that

$$\beta \in \{0, 1, \ldots, 2^{r-1} - 1\} = \{0, 1, \ldots, 2^{n-k-1} - 1\} \tag{7.9}$$

$$\gamma \in \{0, n\}. \tag{7.10}$$

In Theorem 5, it has been proved that $2^{n-k-1} - n \geq 0$. Therefore, we have

$$\begin{aligned} &\beta + \gamma \in \{0, 1, \ldots, 2^{n-k-1} - 1\} \cup \{n, n+1, \ldots, n + 2^{n-k-1} - 1\} \\ \Rightarrow \quad &\beta + \gamma \in \{0, 1, \ldots, 2^{n-k-1} - 1, 2^{n-k-1}, 2^{n-k-1} + 1, \ldots, 2^{n-k-1} + n - 1\}. \end{aligned} \tag{7.11}$$

In other words, there always exists at least one set of $\{p_1, p_2, \ldots, p_r\}$ such that $\beta+\gamma = a'$ where $0 \leq a' \leq 2^{n-k-1} + n - 1$. There also exists at least one set of $\{p_1, p_2, \ldots, p_r\}$ such that $\beta + \gamma = a' \mod U$ where $U \leq 2^{n-k-1} + n$.

For any given α, we define $b \equiv \alpha \mod U$ where $0 \leq b \leq U - 1$. Moreover, for a fixed a where $0 \leq a \leq U - 1$, we can select a set of $\{p_1, p_2, \ldots, p_r\}$ such that

$$\beta + \gamma = a' = \begin{cases} a - b & \text{if } a \geq b \\ U - (b - a) & \text{if } a < b \end{cases} \tag{7.12}$$

with $U \leq 2^{n-k-1} + n$. Hence we can guarantee

$$\alpha + \beta + \gamma \equiv a \mod U. \tag{7.13}$$

Combining the above results with Theorem 5 completes the proof.

Comparing (7.3) and (2.4) clearly shows that our proposed systematic code is a subset of the Levenshtein code. The size of a Levenshtein codebook is maximized when $a = 0$ [96] and hence, the case $a = 0$ will be considered in the following. Note that our proposed code is also applicable to the cases where $a \neq 0$.

7.2.2 Encoding of the Systematic Code

Supposing $a = 0$, we can rewrite (7.12) as

$$\beta + \gamma = a' = \begin{cases} 0 & \text{if } b = 0 \\ U - b & \text{if } b \neq 0. \end{cases} \tag{7.14}$$

For each value of $b \in \{0, 1, \ldots, U-1\}$, we pre-evaluate the set of parity check bits $\{p_1, p_2, \ldots, p_r\}$ such that (7.14) is satisfied. Then, given a message $(m_1, m_2, \ldots, m_k)$, we can compute α based on (7.6) and hence $b \equiv \alpha \mod U$. Finally, the corresponding parity check bits can be chosen.

Example: To illustrate the encoding procedures, we consider the message 0010100 1101001 of length $k = 14$ bits, with the leftmost bit $m_1 = 0$ and the rightmost bit $m_{14} = 1$. Solving (7.1) shows a codeword length of $n = 20$ bits. We also arbitrarily choose $U = 40$ among all values satisfying $2n = 40 \leq U \leq 52 = 2^{n-k-1} + n$. Using (7.6), we have $\alpha = 74$ and $b = 34$. From the pre-evaluated parity check sets, we conclude $p_1 = p_4 = p_5 = p_6 = 0$ and $p_2 = p_3 = 1$, i.e., the set of parity check bits satisfies (7.4). Since the parity check bits are located at the $1st$, $2nd$, $4th$, $8th$, $16th$ and the last (i.e., $20th$) positions while the messages bits are located at the remaining positions, the codeword is $\mathbf{0}1\mathbf{0}1\mathbf{0}0\mathbf{1}00\mathbf{1}10\mathbf{1}00\mathbf{0}1\mathbf{0}$, in which the bits in bold are the parity check bits.

7.2.3 Decoding of the Systematic Code

Since our proposed systematic code is a subset of the Levenshtein code, the decoding algorithm of our code is the same as that of the Levenshtein code. Below we briefly review the algorithm, which is depicted in Fig. 7.4.

Let $\boldsymbol{t} = (x_1, x_2, \cdots, x_n)$, $x_i \in \{0, 1\}$ for $i = 1, 2, \ldots, n$ and $\boldsymbol{y} = (x'_1, x'_2, \cdots, x'_l)$, $x'_i \in \{0, 1\}$ for $i = 1, 2, \ldots, l$ be the transmitted and received sequences, respectively. The

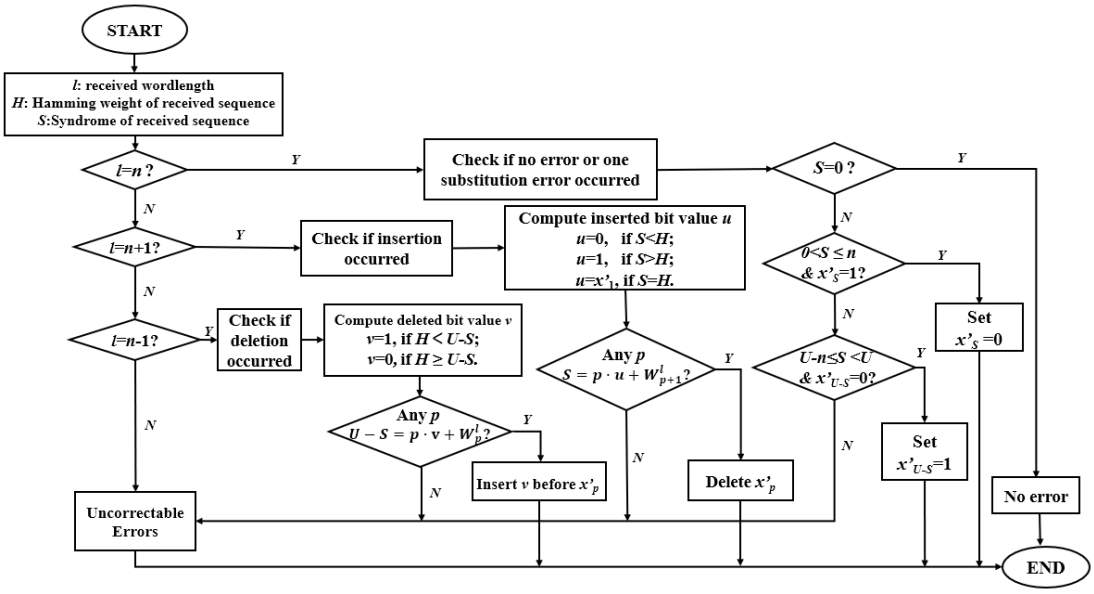


Figure 7.4: Flow chart for the decoding of the proposed systematic code.

length of the transmitted sequence n is a constant parameter while the length of the received sequence l is a random variable. We denote H as the Hamming weight of the received sequence. Moreover, the syndrome of the received sequence is computed using

$$S = \sum_{i=1}^{l} x'_i \cdot i \pmod{U}. \tag{7.15}$$

Assuming there is at most one single error within one received codeword, the decoding algorithms for different error types are divided into three cases, i.e., $l = n$, $l = n+1$ and $l = n-1$. If l does not belong to either of these values, the errors cannot be corrected.

Case 1: $l = n$

1. If $S = 0$, there is no error in the received codeword.

2. If both $0 < S \leq n$ and $x'_S = 1$ are satisfied, the S-th bit was substituted and we flip the value of bit x'_S to 0.

3. If both $S \geq U - n$ and $x'_{U-S} = 0$ are satisfied, the $(U - S)$-th bit was substituted and we flip the value of bit x'_{U-S} to 1.

4. Otherwise, the error types cannot be corrected. For example, there is more than one substitution error within the codeword.

Case 2: $l = n + 1$

A single insertion error is assumed to occur within the codeword. The value and position of the inserted bit can be determined as follows. The bit value u that has been inserted is determined by comparing S and H, i.e.,

$$u = \begin{cases} 0 & \text{if } S < H \\ 1 & \text{if } S > H \\ x'_1 & \text{if } S = H. \end{cases} \tag{7.16}$$

Moreover, the position (denoted by p) that the insertion error occurs can be determined by solving

$$S = p \cdot u + W^l_{p+1} \tag{7.17}$$

where W^l_{p+1} is the Hamming weight of sequence $(x'_{p+1}, x'_{p+2}, \cdots, x'_l)$.

If a specific p is found, the corrupted codeword with an insertion error can be corrected by deleting the p-th bit with value u. Otherwise, the corrupted codeword can be detected but not corrected. For example, the codeword may suffer from two insertions and one deletion even though the received codeword length is $n + 1$.

Case 3: $l = n - 1$

A single deletion error is assumed to occur in the codeword. The bit value v and position p of the deleted bit can be determined, respectively, by

$$v = \begin{cases} 1 & \text{if } H < (U - S) \mod U \\ 0 & \text{if } H \geq (U - S) \mod U \end{cases} \tag{7.18}$$

and

$$(U - S) \mod U = p \cdot v + W^l_p \tag{7.19}$$

In (7.19), W_p^l is the Hamming weight of sequence $(x'_p, x'_{p+1}, \cdots, x'_l)$. If a specific p is found, the corrupted codeword with a deletion error can be corrected by inserting a bit with value v exactly right before x'_p. Otherwise, the corrupted codeword can be detected but not corrected. For example, two deletions and one insertion may occur simultaneously in the codeword while the codeword length remains $n-1$.

In summary, the proposed systematic code can correct a single insertion, deletion or substitution error and can detect multiple errors.

7.3 GC-balanced DNA Sequences with Error Correction Capability

In this section, we design a DNA sequence with GC content occupying exactly 50% and the proposed GC-balanced DNA sequence can correct a base mutation, insertion and deletion error [95]. We introduce our strategy in three aspects: modulation, construction and decoding.

7.3.1 Modulation Scheme for GC-balanced DNA

The proposed modulation maps a binary sequence $\boldsymbol{x} = (x_1, x_2, \cdots, x_{2n})$ to a DNA sequence $\mathcal{N}$ with GC content occupying exactly 50%. We define $\boldsymbol{x}_{odd} = (x_1, x_3, \cdots, x_{2n-1})$ and $\boldsymbol{x}_{even} = (x_2, x_4, \cdots, x_{2n})$, which consist of all the elements at, respectively, odd positions and even positions of $\boldsymbol{x}$. With four different nucleotide bases {A, C, T, G}, DNA can maximally encode two bits per base, which gives four different possibilities. We adopt the modulation strategy shown in Table 7.1, where the bit pairs 10, 00, 11, 01 are modulated to the bases A, C, T, G, respectively.

Theorem 7 *Assume n is an even number. If the Hamming weight of* $\boldsymbol{x}_{odd} = (x_1, x_3, \cdots, x_{2n-1})$ *is exactly* $n/2$ *and the modulation shown in Table 7.1 is adopted, a binary sequence*

Table 7.1: Modulation of two consecutive bits $x_{2i-1}x_{2i}$ to DNA nucleotide bases. Bit pairs 10, 00, 11, 01 are modulated to the bases A, C, T, G, respectively.

nucleotide base	A	C	T	G
$x_{2q-1}x_{2q}$ $(q = 1, 2, \ldots)$	10	00	11	01

$\boldsymbol{x}$ of length $2n$ bits will be modulated to a unique DNA sequence $\mathcal{N}$ of length n bases with GC content occupying exactly 50%.

Proof. If the Hamming weight of $\boldsymbol{x}_{odd}$ is exactly $n/2$, $n/2$ of the nucleotide bases will be C or G (for bit value $x_{2q-1} = 0$) and $n/2$ of the nucleotide bases will be A or T (for bit value $x_{2q-1} = 1$). Hence the resultant DNA sequence will contain exactly 50% content either C or G.

Note the following:

- A single base insertion or deletion occurring in $\mathcal{N}$ results in both $\boldsymbol{x}_{odd}$ and $\boldsymbol{x}_{even}$ having a single insertion or deletion error at the same position.

- A single base mutation occurring in $\mathcal{N}$ results in three possible scenarios: (a) both $\boldsymbol{x}_{odd}$ and $\boldsymbol{x}_{even}$ have a single substitution error at the same position; or (b) $\boldsymbol{x}_{odd}$ is correct but $\boldsymbol{x}_{even}$ has a single substitution error; or (c) $\boldsymbol{x}_{odd}$ has a single substitution error but $\boldsymbol{x}_{even}$ is correct.

7.3.2 Construction of GC-balanced DNA

A GC-balanced DNA sequence capable of correcting base synchronization or mutation errors is constructed by combining the proposed systematic code and the aforementioned modulation scheme. The structure of the resultant DNA sequence is shown in Fig. 7.5.

A binary sequence to be stored in DNA is first divided into several blocks with equal length. We require the length of each block be represented by $N_{length} = 2n -$

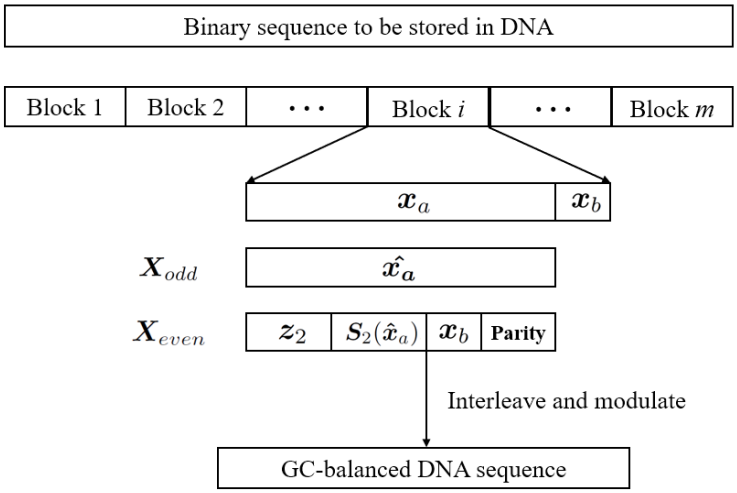


Figure 7.5: Construction of a DNA sequence from a block of data.

$3\lceil \log_2 n \rceil - 2$ for some even number n. The block is further divided into two sub-blocks $\boldsymbol{x}_a = (x_{a_1}, x_{a_2}, \cdots, x_{a_n})$ with length n and $\boldsymbol{x}_b$ with length $n - 3\lceil \log_2 n \rceil - 2$. Starting from the last bit, we flip the bits in $\boldsymbol{x}_a$ one-by-one. Assuming that after flipping the last z bits, the resultant sub-block $\hat{\boldsymbol{x}}_a = (x_{a_1}, x_{a_2}, \cdots, x_{a_{n-z}}, \bar{x}_{a_{n-z+1}}, \cdots, \bar{x}_{a_{n-1}}, \bar{x}_{a_n})$ has a Hamming weight of exactly $n/2$. Here, $\bar{x}$ denotes the complement of x. We further denote $\hat{\boldsymbol{x}}_a$ as $\boldsymbol{X}_{odd}$.

The syndrome $S(\hat{\boldsymbol{x}}_a)$ of $\hat{\boldsymbol{x}}_a$ is calculated using

$$S(\hat{\boldsymbol{x}}_a) = \sum_{i=1}^{i=n} \hat{x}_{a_i} \cdot i \,(\mathrm{mod}\, 2n). \tag{7.20}$$

We denote $\boldsymbol{S}_2(\hat{\boldsymbol{x}}_a)$, with length $\lceil \log_2 n \rceil + 1$, as the binary representation of the syndrome $S(\hat{\boldsymbol{x}_a})$; and $\boldsymbol{z}_2$, with length $\lceil \log_2 n \rceil$, as the binary representation of the flipping number z. The sequence $(\boldsymbol{z}_2 \;\; \boldsymbol{S}_2(\hat{\boldsymbol{x}}_a) \;\; \boldsymbol{x}_b)$, with length $k = n - \lceil \log_2 n \rceil - 1$, is subsequently encoded by the proposed code in Sect. 7.2.2 to form a codeword of length n. We further denote this code as $\boldsymbol{X}_{even}$.

By pairing the bits in $\boldsymbol{X}_{odd}$ and $\boldsymbol{X}_{even}$ and using the modulation scheme in Table 7.1, we can construct the corresponding DNA sequence $\mathcal{N}$. Since the Hamming weight of $\boldsymbol{X}_{odd}$ is exactly $n/2$, the modulated sequence $\mathcal{N}$ is a GC-balanced DNA with GC

content occupying exactly 50% (see Theorem 7).

In summary, the proposed construction method can encode the sub-blocks $\boldsymbol{x}_a$ and $\boldsymbol{x}_b$ with a total length of $2n - 3\lceil \log_2 n \rceil - 2$ to a GC-balanced DNA sequence with n bases. Thus, the coding efficiency is given by $\frac{2n-3\lceil \log_2 n \rceil - 2}{n}$.

7.3.3 Decoding of GC-balanced DNA

The constructed DNA sequence may suffer from insertion, deletion or mutation errors during synthesis, storage and sequencing. We first demodulate the potentially erroneous DNA sequence $\widetilde{\mathcal{N}}$ to a binary sequence $\widetilde{\boldsymbol{X}}$ based on Table 7.1. Two subsequences, denoted by $\widetilde{\boldsymbol{X}}_{odd}$ and $\widetilde{\boldsymbol{X}}_{even}$, are extracted from $\widetilde{\boldsymbol{X}}$ using the bits in the odd-numbered locations and even-numbered locations, respectively.

We assume that only one base error has occurred. Since our proposed systematic code can correct one synchronization or mutation error, $\boldsymbol{X}_{even}$ can be recovered correctly from $\widetilde{\boldsymbol{X}}_{even}$ with the error location determined. In other words, the location in $\widetilde{\boldsymbol{X}}_{odd}$ where an error occurs is also known. Moreover, our systematic code allows us to easily separate $\boldsymbol{z}_2$, $\boldsymbol{S}_2(\hat{\boldsymbol{x}}_a)$ and $\boldsymbol{x}_b$ from $\boldsymbol{X}_{even}$. If an insertion error has occurred in $\widetilde{\boldsymbol{X}}_{even}$, the corresponding bit can be removed from $\widetilde{\boldsymbol{X}}_{odd}$ and $\boldsymbol{X}_{odd}$ is recovered. If a deletion or mutation error has occurred in $\widetilde{\boldsymbol{X}}_{even}$, the corresponding bit can be reinserted or corrected in $\widetilde{\boldsymbol{X}}_{odd}$ based on (7.20) and the recovered $\boldsymbol{S}_2(\hat{\boldsymbol{x}}_a)$. After $\boldsymbol{X}_{odd}$ has been correctly recovered, the last z bits will be flipped to recover the original $\boldsymbol{x}_a$.

7.3.4 Example

Encoding

We use an example to illustrate the proposed encoding and decoding algorithms of the GC-balanced DNA sequence. Suppose the length of $\boldsymbol{x}_a$ is $n = 20$ and hence, the length of $\boldsymbol{x}_b$ is determined as $n - 3 \cdot \lceil \log_2^n \rceil - 2 = 3$. Assume the data block, consisting of the 23 bits, to be stored as a DNA sequence is 01011011001110101111001. Therefore,

$\boldsymbol{x}_a = 01011011001110101111$ and $\boldsymbol{x}_b = 001$. For the sub-block $\boldsymbol{x}_a$, we can flip the last $z = 5$ bits and results in a corresponding sequence $\hat{\boldsymbol{x}}_a = 01011011001110110000 \equiv \boldsymbol{X}_{odd}$ with Hamming weight exactly $n/2 = 10$.

The syndrome of $\hat{\boldsymbol{x}}_a$, computed using (7.20), is given by $S(\hat{\boldsymbol{x}}_a) = 2 + 4 + 5 + 7 + 8 + 11 + 12 + 13 + 15 + 16 = 13 \,(\text{mod } 40)$. Hence $\boldsymbol{S}_2(\hat{\boldsymbol{x}}_a) = 001101$, $\boldsymbol{z}_2 = 00101$ and $\boldsymbol{x}_b = 001$. The sequence $(\boldsymbol{z}_2 \ \boldsymbol{S}_2(\hat{\boldsymbol{x}}_a) \ \boldsymbol{x}_b) = (00101001101001)$ of length $n - \lceil \log_2 n \rceil - 1 = 14$ is constructed. It is subsequently encoded by the proposed systematic code to form $\boldsymbol{X}_{even} \equiv (\mathbf{0}1\mathbf{0}1\mathbf{0}1\mathbf{0}0\mathbf{0}1\mathbf{0}0\mathbf{1}1\mathbf{0}1\mathbf{0}\mathbf{0}\mathbf{0}\mathbf{1}\mathbf{0})$ of length $n = 20$. The parity check bits $\mathbf{011000}$ are located at the $1st$, $2nd$, $4th$, $8th$, $16th$ and $20th$ position and all the remaining locations are reserved for $(\boldsymbol{z}_2 \ \boldsymbol{S}_2(\hat{\boldsymbol{x}}_a) \ \boldsymbol{x}_b)$. $\boldsymbol{X}_{odd} = 01011011001110110000$ and $\boldsymbol{X}_{even} = 01010100100110100010$ are interleaved to construct the new sequence $\boldsymbol{X} = 00110011100110100100101111001110000000100$. Finally, according to Table 7.1, the binary sequence $\boldsymbol{X}$ is modulated to a GC-balanced DNA sequence $\mathcal{N} =$ CTCTAGAAGCATTCTACCGC with GC bases occupying exactly 50%. $\mathcal{N}$ will be synthesized and stored in the DNA storage library.

Decoding

Assume the $8th$ nucleotide base is somehow missing (deleted) during DNA sequencing. The extracted DNA sequence $\widetilde{\mathcal{N}} =$ CTCTAGA$\underline{\text{A}}$GCATTCTACCGC, in which the underlined base A is deleted. The demodulated binary sequence $\widetilde{\boldsymbol{X}} = 00110011100110$ 01001011110011100000100 is subsequently deinterleaved to form $\widetilde{\boldsymbol{X}}_{odd} = 01011010$ 01110110000 and $\widetilde{\boldsymbol{X}}_{even} = 0101010100110100010$. Both $\widetilde{\boldsymbol{X}}_{odd}$ and $\widetilde{\boldsymbol{X}}_{even}$ are one bit shorter than $\boldsymbol{X}_{odd}$ and $\boldsymbol{X}_{even}$.

$\widetilde{\boldsymbol{X}}_{even}$ is first decoded according to the decoding algorithm shown in Fig. 7.4. The deleted bit value v is 0 since $U - S = 40 - (2 + 4 + 6 + 8 + 11 + 12 + 14 + 18) = -35 \ (mod \ 40) = 5 < H = 8$ according to (7.18). The position $p = 8$ of the deleted bit can be determined by (7.19). Therefore, we insert bit 0 before the $8th$ bit in the

erroneous $\widetilde{\boldsymbol{X}}_{even}$ to recover $\boldsymbol{X}_{even}$.

We subsequently remove the parity bits to obtain the sequence $(\boldsymbol{z}_2 \ \boldsymbol{S}_2(\hat{\boldsymbol{x}}_a) \ \boldsymbol{x}_b)$. $\boldsymbol{x}_b$ can be readily extracted. Realizing that a bit should be re-inserted before the $8th$ bit in the erroneous $\widetilde{\boldsymbol{X}}_{odd}$ and using $\boldsymbol{S}_2(\hat{\boldsymbol{x}}_a)$ and (7.20), the value of the $8th$ bit can be computed (= 1) and $\widetilde{\boldsymbol{X}}_{odd}$ with a single deletion error can be decoded correctly. Converting $\boldsymbol{z}_2$ to z and flipping the last z bits of $\boldsymbol{X}_{odd}$ recovers $\boldsymbol{x}_a$. Finally, we successfully extract the original information stored in the DNA, i.e., the concatenation of $\boldsymbol{x}_a$ = 01011011001110101111 and $\boldsymbol{x}_b$ = 001.

7.4 Simulation Results

7.4.1 Performance of the Proposed Systematic Code

Simulations are conducted to evaluate the error performance of the proposed systematic code under different values of U satisfying (7.4). A random sequence of $k \times 10^5$ message bits is first generated and divided into $N = 10^5$ blocks, each of which contains k message bits. Then each block of k message bits is encoded into a codeword of n bits by our proposed systematic code in Sect. 7.2. The encoding procedure is repeated for every block. Each encoded block is separately transmitted through a "channel" with bit insertion error probability p_i, deletion error probability p_d and substitution error probability p_s.

We perform the decoding procedure based on the flow chart in Fig. 7.4 and count the total number of decoded blocks in error. The block error rate (BLER) is used as a metric to assess the performance of the proposed code. The values of U satisfying (7.4) are randomly chosen and the performance for different sets of (n, k, U) in terms of BLER is shown in Fig. 7.6 and Fig. 7.7.

In Fig. 7.6, only one type of error, namely substitution error, is assumed in the channel. The results show that for the same k and n, the BLERs are the same for

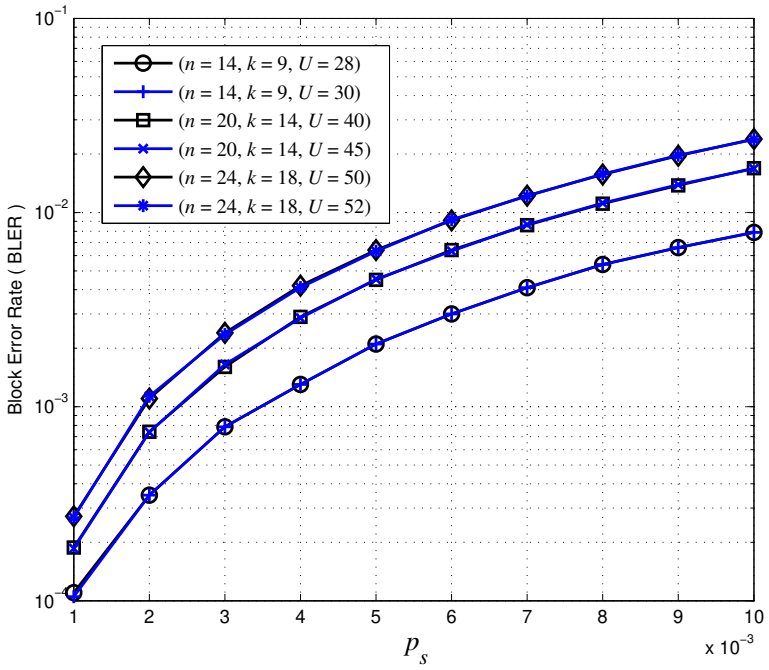


Figure 7.6: Performance of the proposed code with different parameters (n, k, U). p_s increases from 10^{-3} to 10^{-2} while $p_d = p_i = 0$.

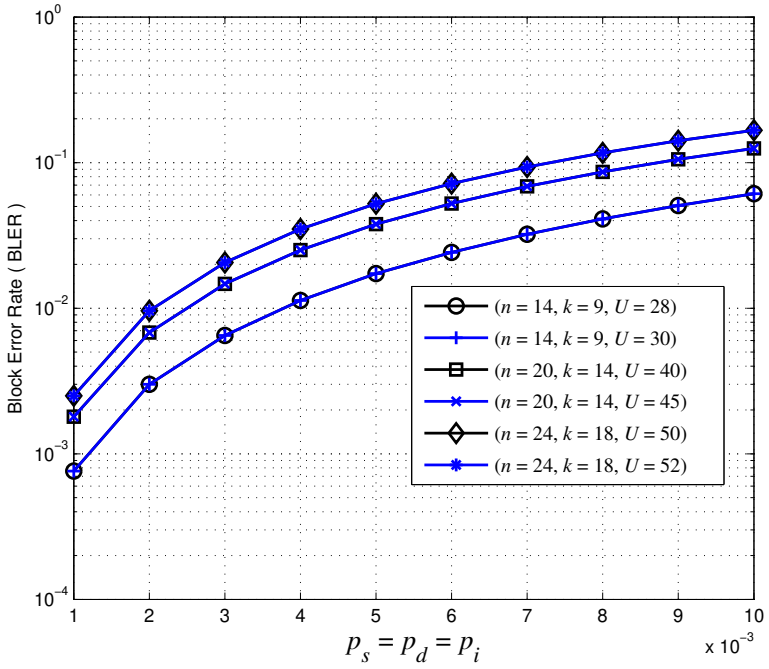


Figure 7.7: Performance of the proposed code with different parameters (n, k, U). $p_s = p_d = p_i$ increase from 10^{-3} to 10^{-2}.

different values of U. In other words, the performance of the proposed systematic code is not affected by the value of U, as long as U satisfies (7.4). We also notice that the BLER increases as the codeword length n increases. It is because each code bit suffers from the same error probability. The longer the codelength, the probability of more than one bit errors occurring in the codeword increases. Since the systematic code can correct only one error, the BLER increases with the codelength. Note that even though codes with shorter lengths perform better in terms of BLER, the information transmission efficiency (i.e., k/n) is lower. We have further simulated the situations where only insertions or only deletions exist in the channel. The BLERs are exactly the same as the curves shown in Fig. 7.6, i.e., when only substitution errors occur. In other words, when only one type of error occurs, the BLER performance is the same regardless of the error type (substitution, insertion or deletion). It is reasonable because the proposed systematic code can correct substitution, insertion or deletion errors equally well.

In Fig. 7.7, we plot the BLER curves of the proposed code when all three types of errors occur at the same time and $p_s = p_d = p_i$. Compared with Fig. 7.6, the BLERs in Fig. 7.7 are higher for the same set of (n, k, U) simply because errors are occurring with a higher chance. As in Fig. 7.6, Fig. 7.7 also indicates that (a) the BLER performance for the same set of (n, k) is independent of U, and (b) the BLER increases with the codelength.

7.4.2 Performance of the Proposed GC-balanced DNA Sequences

Each block of 23 message bits is encoded into a DNA sequence of 20 bases, as illustrated in the example shown in Sect. 7.3.4. Simulations are conducted to evaluate the the error performance of the proposed DNA sequence design. A total of 10^5 DNA blocks are generated for the simulation. We consider the base error rate instead of bit error rate during the sequencing of DNA blocks. Assume each base has an error rate of

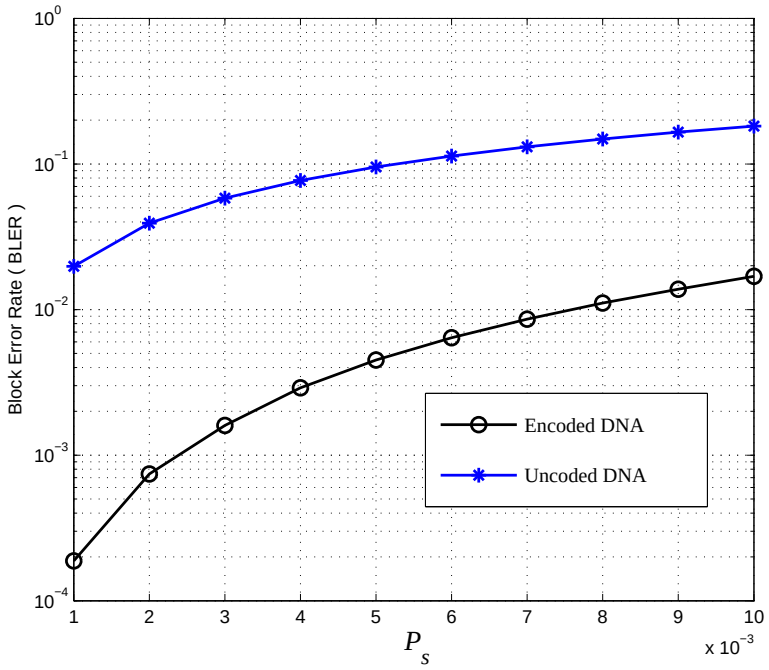


Figure 7.8: Performance of encoded and uncoded DNA sequences. Black curve represents the performance of the encoded DNA sequences and the blue curve represents the performance of random uncoded DNA sequences. Mutation error probability P_s increases from 10^{-3} to 10^{-2} while $P_d = P_i = 0$.

P_s, P_d, and P_i, representing base mutation error rate, base deletion error rate and base insertion error rate, respectively. The demodulated DNA sequences are decoded with our proposed method.

We first assume only mutations exist within the DNA sequences. The block error rates of the sequences under different mutation error rates are shown in Fig. 7.8. The results show that the performance of the encoded DNA sequences in terms of BLER improves a lot compared with the uncoded DNA. For example, the BLER improves from 2×10^{-2} to 2×10^{-4} when $P_s = 10^{-3}$. Simulations are further carried out separately when there are only insertions or only deletions in the DNA sequences. The simulation results are found to be identical to those when only mutations occur (i.e., Fig. 7.8). Hence the designed DNA sequences can combat mutation, insertion and deletion equally well.

In Fig. 7.9, we show the simulation results when all three types of errors exist in

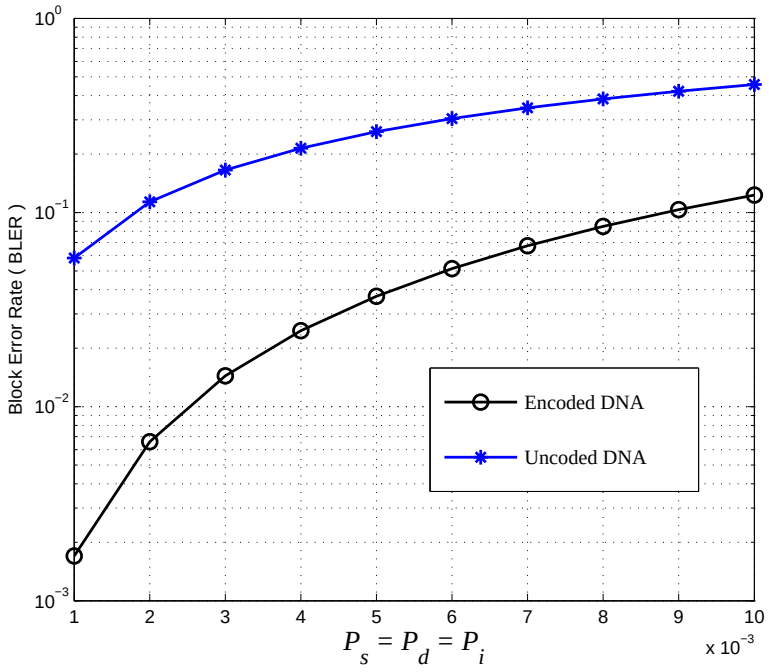


Figure 7.9: Performance of encoded and uncoded DNA sequences. Black curve represents the performance of the encoded DNA sequences and the blue curve represents the performance of random uncoded DNA sequences. $P_s = P_d = P_i$ increase from 10^{-3} to 10^{-2}.

the DNA sequences. Again, the encoded DNA sequences outperform the uncoded ones substantially. Thus the proposed DNA sequences can correct base errors adequately and can be readily used in practice.

7.5 Summary

In this chapter, we have proposed a systematic code which is able to correct a single insertion/deletion/substitution and to detect multiple errors. Combining the proposed code and the designed modulation scheme, GC-balanced DNA sequences with a single base-error-correction capability are constructed. On one hand, GC-balanced DNA sequence have the maximum endurance to errors during its sequencing and synthesis. On the other hand, it can correct both insertion/deletion and mutation of the nucleotide bases. Adopting the coding and modulation strategies can avoid producing redundant

repetition copies and making more economical use of DNA sequences for data storage. The decoding procedure for the sequences is described and can readily be used in practice. Simulation results show that the proposed GC-balanced DNA sequences can correct base errors adequately.

Chapter 8

Conclusions and Suggestions for Future Work

In this chapter, we summarize the main contributions of the thesis and discuss some potential directions for future research.

8.1 Main Contributions of the Thesis

In the past two decades, we have experienced an exponential growth of data being generated, transmitted, processed and stored. Error correction codes play an important role in ensuring that data is transmitted to the destination reliably and efficiently through the channel; and that data is stored and retrieved accurately. Besides substitution errors, insertion and deletion errors exist in many practical applications and represent an important aspect of channel impairments in communication systems and data storage systems. This thesis investigates and contributes towards error correction codes for channels impaired by insertion, deletion and substitution errors.

In Chapter 3, we construct a new class of systematic comma-free code to recover synchronization after insertion, deletion and substitution errors occur. The proposed code includes the classical F code as a special case and it gives more choices com-

pared with the F code. We also derive the false synchronization probability caused by different types of errors and analyze the factors affecting false synchronizations.

In Chapter 4, the inner code of the classical concatenated Davey MacKay watermark code is optimized. For the symbol-level decoding algorithm, we first propose a hard-decision decoding based on Hamming distance and then a more accurate soft-decision decoding based on a new metric called ATP. We further propose an improved mapping strategy to optimize the inner sparsified codebook based on both better distance property and lowest density property. The simulation results show that the optimized watermark code with the improved mapping strategy improves the error performance compared with the watermark code.

In Chapter 5, we design a concatenated RS-marker code with the capability of correcting insertion, deletion and substitution errors. With the designed markers, the inner marker decoder can recover synchronization at codeword boundaries. Simulation results show that the performance of our code is better than that of the watermark code at low error rates. Even though our code rate is lower than that of the watermark code, the decoding complexity is much simpler, which saves a large number of computations and reduces decoding time.

In Chapter 6, we propose a probabilistic channel model with correlated insertion and deletion (CID) errors, in addition to substitution errors. The correlated insertion/deletion channel model captures the data dependence adapted to applications such as the write channel in bit-patterned media recording (BPMR) systems. Furthermore, we investigate the performance of a concatenated code over this channel. The concatenated code consists of an inner marker code used to maintain synchronization and an outer low-density parity-check (LDPC) code to provide error correction capability. The forward-backward marker decoding algorithm is elaborated based on a two-dimensional state transition diagram. Simulation results show that the proposed concatenated code is effective in combating CID channel with substitution errors.

Finally in Chapter 7, we construct a GC-balanced DNA sequence with error correc-

tion capability for a DNA-based data storage system. We propose a systematic single insertion/deletion/substitution error correction code and combine it with the proposed modulation scheme to construct GC-balanced DNA sequences. On one hand, such DNA sequences have the maximum endurance to errors during its practical sequencing and synthesis. On the other hand, it can correct a single insertion/deletion/mutation base error. The proposed coding scheme can readily be used in practice. It avoids producing redundant repetition copies of the data and improves the efficiency of DNA data storage systems.

8.2 Suggestions for Future Work

Many problems remain open after our investigations. In the following, we propose some possible future research directions.

In Chapter 5, the performance of the proposed concatenated code is jointly determined by both the outer RS code and the proposed inner marker code. The factors affecting the overall performance are the outer code length N, information length K, length of the primer p, coded symbol group size M, and channel parameters. For the outer RS (N, K, k) code, K can be any value smaller than $2^k - 1$. However, the value of K affects both the error correction capability and the code rate. For the inner marker code, M affects the block size, and hence the number of blocks in a frame and the code rate. The performance tradeoff among these different parameters can be further investigated in the future.

In Chapter 6, we have assumed binary data transmission over the CID channel with correlated insertion and deletion errors. One potential research direction is to extend the investigation to M-ary symbols. In Chapter 7, we construct a GC-balanced DNA sequence with a single base error correction capability by combining a single insertion/deletion/substitution error correction code with a new modulation scheme. Future study may include designing GC-balanced DNA sequences with multiple base

error correction capabilities.

Bibliography

[1] Y. Suzuki and H. Kaneko, "Correlated insertion/deletion error correction coding for bit-patterned media," in *2017 IEEE International Conference on Consumer Electronics - Taiwan (ICCE-TW)*, 2017, pp. 7–8.

[2] C. E. Shannon, "A mathematical theory of communication," *The Bell System Technical Journal*, vol. 27, no. 3, pp. 379–423, 1948.

[3] R. W. Hamming, "Error detecting and error correcting codes," *The Bell System Technical Journal*, vol. 29, no. 2, pp. 147–160, 1950.

[4] M. J. Golay, "Notes on digital coding," *Proc. IEEE*, vol. 37, p. 657, 1949.

[5] I. Reed, "A class of multiple-error-correcting codes and the decoding scheme," *Transactions of the IRE Professional Group on Information Theory*, vol. 4, no. 4, pp. 38–49, 1954.

[6] A. Hocquenghem, "Codes correcteurs derreurs," *Chiffres*, vol. 2, no. 2, pp. 147–156, 1959.

[7] P. Elias, "Predictive coding–I," *IRE Transactions on Information Theory*, vol. 1, no. 1, pp. 16–24, 1955.

[8] I. S. Reed and G. Solomon, "Polynomial codes over certain finite fields," *Journal of the Society for Industrial and Applied Mathematics*, vol. 8, no. 2, pp. 300–304, 1960.

[9] D. Forney, "Concatenated codes," *Scholarpedia*, vol. 4, no. 2, p. 8374, 2009.

[10] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near shannon limit error-correcting coding and decoding: Turbo-codes. 1," in *Proceedings of ICC'93-IEEE International Conference on Communications*, vol. 2, 1993, pp. 1064–1070.

[11] C. Berrou and A. Glavieux, "Near optimum error correcting coding and decoding: turbo-codes," *IEEE Transactions on Communications*, vol. 44, no. 10, pp. 1261–1271, 1996.

[12] R. Gallager, "Low-density parity-check codes," *IRE Transactions on Information Theory*, vol. 8, no. 1, pp. 21–28, 1962.

[13] D. J. C. MacKay, "Good error-correcting codes based on very sparse matrices," *IEEE Transactions on Information Theory*, vol. 45, no. 2, pp. 399–431, 1999.

[14] G. Han, Y. L. Guan, K. Cai, K. S. Chan, and L. Kong, "Coding and detection for channels with written-in errors and inter-symbol interference," *IEEE Transactions on Magnetics*, vol. 50, no. 10, pp. 1–6, 2014.

[15] V. Licks and R. Jordan, "Geometric attacks on image watermarking systems," *IEEE MultiMedia*, vol. 12, no. 3, pp. 68–78, 2005.

[16] Q. Wang, T. Wen, H. Yang, and X. Wang, "Synchronisation correction-based colour image watermarking using proximal classifier with consistency and multivariate generalised gaussian model," *Electronics Letters*, vol. 55, no. 24, pp. 1305–1307, 2019.

[17] X. Zheng, Y. Zhao, N. Li, G. Liu, and W. Zhou, "Research of synchronization robustness in video digital watermarking," in *2008 ISECS International Colloquium on Computing, Communication, Control, and Management*, vol. 1, 2008, pp. 276–279.

[18] C. T. Clelland, V. Risca, and C. Bancroft, "Hiding messages in DNA microdots," *Nature*, vol. 399, no. 6736, pp. 533–534, 1999.

[19] C. Bancroft, T. Bowler, B. Bloom, and C. T. Clelland, "Long-term storage of information in DNA," *Science*, vol. 293, no. 5536, pp. 1763–1763, 2001.

[20] G. M. Church, Y. Gao, and S. Kosuri, "Next-generation digital information storage in DNA," *Science*, vol. 337, no. 6102, pp. 1628–1628, 2012.

[21] N. Goldman, P. Bertone, S. Chen, C. Dessimoz, E. M. LeProust, B. Sipos, and E. Birney, "Towards practical, high-capacity, low-maintenance information storage in synthesized DNA," *Nature*, vol. 494, no. 7435, pp. 77–80, 2013.

[22] A. Lenz, P. H. Siegel, A. Wachter-Zeh, and E. Yaakobi, "Coding over sets for DNA storage," *IEEE Transactions on Information Theory*, vol. 66, no. 4, pp. 2331–2351, 2020.

[23] R. N. Grass, R. Heckel, M. Puddu, D. Paunescu, and W. J. Stark, "Robust chemical preservation of digital information on DNA in silica with error-correcting codes," *Angewandte Chemie International Edition*, vol. 54, no. 8, pp. 2552–2555, 2015.

[24] L. Deng, Y. Wang, M. Noor-A-Rahim, Y. L. Guan, Z. Shi, E. Gunawan, and C. L. Poh, "Optimized code design for constrained DNA data storage with asymmetric errors," *IEEE Access*, vol. 7, pp. 84 107–84 121, 2019.

[25] R. G. Gallager, "Sequential decoding for binary channels with noise and synchronization errors. lincoln lab," *Group Report, October*, 1961.

[26] M. C. Davey and D. J. MacKay, "Reliable communication over channels with insertions, deletions, and substitutions," *IEEE Transactions on Information Theory*, vol. 47, no. 2, pp. 687–698, 2001.

[27] J. A. Briffa and H. G. Schaathun, "Improvement of the Davey-MacKay construction," in *2008 International Symposium on Information Theory and Its Applications*, 2008, pp. 1–4.

[28] H. Mercier, V. K. Bhargava, and V. Tarokh, "A survey of error-correcting codes for channels with symbol synchronization errors," *IEEE Communications Surveys Tutorials*, vol. 12, no. 1, pp. 87–96, 2010.

[29] W. Eastman and S. Even, "On synchronizable and PSK-synchronizable block codes," *IEEE Transactions on Information Theory*, vol. 10, no. 4, pp. 351–356, 1964.

[30] W. Eastman, "On the construction of comma-free codes," *IEEE Transactions on Information Theory*, vol. 11, no. 2, pp. 263–267, 1965.

[31] C. Ramamoorthy and D. Tufts, "Reinforced prefixed comma-free codes," *IEEE Transactions on Information Theory*, vol. 13, no. 3, pp. 366–371, 1967.

[32] S. W. Golomb and B. Gordon, "Codes with bounded synchronization delay," *Information and Control*, vol. 8, no. 4, pp. 355–372, 1965.

[33] N. Kashyap and D. L. Neuhoff, "Data synchronization with timing," *IEEE Transactions on Information Theory*, vol. 47, no. 4, pp. 1444–1460, 2001.

[34] R. Scholtz, "Codes with synchronization capability," *IEEE Transactions on Information Theory*, vol. 12, no. 2, pp. 135–142, 1966.

[35] J. J. Stiffler, *Theory of synchronous communications*. Prentice Hall, 1971.

[36] L. Cummings, "Aspects of synchronizable coding," *Journal of Combinatorial Mathematics and Combinatorial Computing*, vol. 1, pp. 67–84, 1987.

[37] H. Morita, A. J. van Wijngaarden, and A. H. Vinck, "On the construction of maximal prefix-synchronized codes," *IEEE Transactions on Information Theory*, vol. 42, no. 6, pp. 2158–2166, 1996.

[38] S. Konstantinidis, S. Perron, and L. A. Wilcox-O'Hearn, "On a simple method for detecting synchronization errors in coded messages," *IEEE Transactions on Information Theory*, vol. 49, no. 5, pp. 1355–1363, 2003.

[39] R. Scholtz, "Frame synchronization techniques," *IEEE Transactions on Communications*, vol. 28, no. 8, pp. 1204–1213, 1980.

[40] S. W. Golomb, B. Gordon, and L. R. Welch, "Comma-free codes," *Canadian Journal of Mathematics*, vol. 10, pp. 202–209, 1958.

[41] E. Gilbert, "Synchronization of binary messages," *IRE Transactions on Information Theory*, vol. 6, no. 4, pp. 470–477, 1960.

[42] S. W. Golomb, L. R. Welch, and M. Delbrück, *Construction and properties of comma-free codes*. i kommission hos Ejnar Munksgaard, 1958.

[43] S. Golomb, "Efficient coding for the desoxyribonucleic channel," in *Proceedings of Symposia in Applied Mathematics*, vol. 14, 1962, pp. 87–100.

[44] B. Jiggs, "Regent results in Comma-Free Codes," *Canadian Journal of Mathematics*, vol. 15, pp. 178–187, 1963.

[45] R. Scholtz, "Maximal and variable word-length comma-free codes," *IEEE Transactions on Information Theory*, vol. 15, no. 2, pp. 300–306, 1969.

[46] W. Kendall and I. Reed, "Path-invariant comma-free codes," *IRE Transactions on Information Theory*, vol. 8, no. 6, pp. 350–355, 1962.

[47] D. J. Clague, "New classes of synchronous codes," *IEEE Transactions on Electronic Computers*, no. 3, pp. 290–298, 1967.

[48] L. Calabi and W. Hartnett, "A family of codes for the correction of substitution and synchronization errors," *IEEE Transactions on Information Theory*, vol. 15, no. 1, pp. 102–106, 1969.

[49] J. Stiffler, "Comma-free error-correcting codes," *IEEE Transactions on Information Theory*, vol. 11, no. 1, pp. 107–112, 1965.

[50] J. Levy, "Self-synchronizing codes derived from binary cyclic codes," *IEEE Transactions on Information Theory*, vol. 12, no. 3, pp. 286–290, 1966.

[51] F. Sellers, "Bit loss and gain correction code," *IRE Transactions on Information Theory*, vol. 8, no. 1, pp. 35–38, 1962.

[52] U. Sethakaset and T. A. Gulliver, "Marker codes to correct insertion/deletion errors in differential pulse-position modulation for wireless infrared communications," in *PACRIM. 2005 IEEE Pacific Rim Conference on Communications, Computers and signal Processing*, 2005, pp. 181–184.

[53] R. Varshamov and T. GM, "Correction code for single asymmetrical errors," *Automation and Remote Control*, vol. 26, no. 2, pp. 288–292, 1965.

[54] V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions, and reversals," in *Soviet Physics Doklady*, vol. 10, no. 8, 1966, pp. 707–710.

[55] G. Tenengol'ts, "A class of codes correcting bit loss and errors in the preceding symbol," *Avtomatika i Telemekhanika*, no. 5, pp. 174–179, 1976.

[56] G. Tenengolts, "Nonbinary codes, correcting single deletion or insertion (corresp.)," *IEEE Transactions on Information Theory*, vol. 30, no. 5, pp. 766–769, 1984.

[57] T. Hatcher, "On a family of error correcting and synchronizable codes," in *Foundations of Coding Theory*. Springer, 1974, pp. 173–192.

[58] P. Bours, "Bounds for codes that are capable of correcting insertions and deletions," Internal Report, Tech. Rep., 1991.

[59] N. J. Sloane, "On single-deletion-correcting codes," *Codes and designs*, vol. 10, pp. 273–291, 2000.

[60] A. S. Helberg and H. C. Ferreira, "On multiple insertion/deletion correcting codes," *IEEE Transactions on Information Theory*, vol. 48, no. 1, pp. 305–308, 2002.

[61] L. J. Schulman and D. Zuckerman, "Asymptotically good codes correcting insertions, deletions, and transpositions," *IEEE Transactions on Information Theory*, vol. 45, no. 7, pp. 2552–2557, 1999.

[62] Johnny Chen, M. Mitzenmacher, Chaki Ng, and N. Varnica, "Concatenated codes for deletion channels," in *IEEE International Symposium on Information Theory, 2003. Proceedings.*, 2003, pp. 218–218.

[63] E. A. Ratzer, "Marker codes for channels with insertions and deletions," in *Annales des télécommunications*, vol. 60. Springer, 2005, pp. 29–44.

[64] T. Shigehiro, H. Yabe, and K. Iwamura, "An improvement of insertion/deletion/substitution error correction capabilities of LDPC codes using slide decoding," in *2012 International Symposium on Information Theory and its Applications*, 2012, pp. 216–220.

[65] M. Inoue and H. Kaneko, "Adaptive synchronization marker for insertion/deletion/substitution error correction," in *2012 IEEE International Symposium on Information Theory Proceedings*, 2012, pp. 508–512.

[66] R. Shibata, G. Hosoya, and H. Yashima, "Fixed-symbols-based synchronization for insertion/deletion/substitution channels," in *2016 International Symposium on Information Theory and Its Applications (ISITA)*, 2016, pp. 686–690.

[67] S. Aviran, P. H. Siegel, and J. K. Wolf, "Optimal parsing trees for run-length coding of biased data," *IEEE Transactions on Information Theory*, vol. 54, no. 2, pp. 841–849, 2008.

[68] J. A. Briffa, H. G. Schaathun, and S. Wesemeyer, "An improved decoding algorithm for the Davey-MacKay construction," in *2010 IEEE International Conference on Communications*, 2010, pp. 1–5.

[69] Xiaopeng Jiao and M. A. Armand, "Interleaved LDPC codes, reduced-complexity inner decoder and an iterative decoder for the Davey-MacKay construction," in *2011 IEEE International Symposium on Information Theory Proceedings*, 2011, pp. 742–746.

[70] T. Xue and F. C. M. Lau, "A generalized systematic comma free code," in *2017 23rd Asia-Pacific Conference on Communications (APCC)*, 2017, pp. 1–5.

[71] ——, "Generalized Systematic Comma-Free Code," *IEEE Access*, vol. 6, pp. 56 800–56 814, 2018.

[72] ——, "Codebook Design Optimization for Sparsified Distribution Converter in Davey-Mackay Watermark Codes over Channels with Synchronization Errors," in *2019 25th Asia-Pacific Conference on Communications (APCC)*, 2019, pp. 201–206.

[73] ——, "Concatenated Synchronization Error Correcting Code with Designed Markers," in *2019 IEEE 5th International Conference on Computer and Communications (ICCC)*, 2019, pp. 1371–1376.

[74] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*. MIT press, 2009.

[75] W. Chang and J. Cruz, "Inter-track interference mitigation for bit-patterned magnetic recording," *IEEE Transactions on Magnetics*, vol. 46, no. 11, pp. 3899–3908, 2010.

[76] Y. Tang, K. Moon, and H. J. Lee, "Write synchronization in bit-patterned media," *IEEE Transactions on Magnetics*, vol. 45, no. 2, pp. 822–827, 2009.

[77] A. R. Iyengar, P. H. Siegel, and J. K. Wolf, "Write channel model for bit-patterned media recording," *IEEE Transactions on Magnetics*, vol. 47, no. 1, pp. 35–45, 2010.

[78] S. Zhang, K. Cai, M. Lin-Yu, J. Zhang, Z. Qin, K. K. Teo, W. E. Wong, and E. T. Ong, "Timing and written-in errors characterization for bit patterned media," *IEEE Transactions on Magnetics*, vol. 47, no. 10, pp. 2555–2558, 2011.

[79] J. Hu, T. M. Duman, E. M. Kurtas, and M. F. Erden, "Bit-patterned media with written-in errors: Modeling, detection, and theoretical limits," *IEEE transactions on magnetics*, vol. 43, no. 8, pp. 3517–3524, 2007.

[80] A. Mazumdar, A. Barg, and N. Kashyap, "Coding for high-density recording on a 1-D granular magnetic medium," *IEEE Transactions on Information Theory*, vol. 57, no. 11, pp. 7403–7417, 2011.

[81] T. Xue and F. C. M. Lau, "A Concatenated LDPC-Marker Code for Channels with Correlated Insertion and Deletion Errors," submitted.

[82] R. Durbin, S. R. Eddy, A. Krogh, and G. Mitchison, *Biological sequence analysis: probabilistic models of proteins and nucleic acids*. Cambridge university press, 1998.

[83] R. Gallager, "Low-density parity-check codes," *IRE Transactions on Information Theory*, vol. 8, no. 1, pp. 21–28, 1962.

[84] D. J. C. MacKay, "Good error-correcting codes based on very sparse matrices," *IEEE Transactions on Information Theory*, vol. 45, no. 2, pp. 399–431, 1999.

[85] S. Kosuri and G. M. Church, "Large-scale de novo DNA synthesis: technologies and applications," *Nature methods*, vol. 11, no. 5, p. 499, 2014.

[86] S. H. T. Yazdi, H. M. Kiah, E. Garcia-Ruiz, J. Ma, H. Zhao, and O. Milenkovic, "DNA-based storage: Trends and methods," *IEEE Transactions on Molecular, Biological and Multi-Scale Communications*, vol. 1, no. 3, pp. 230–248, 2015.

[87] S. Ma, N. Tang, and J. Tian, "DNA synthesis, assembly and applications in synthetic biology," *Current Opinion in Chemical Biology*, vol. 16, no. 3-4, pp. 260–267, 2012.

[88] S. Ma, I. Saaem, and J. Tian, "Error correction in gene synthesis technology," *Trends in Biotechnology*, vol. 30, no. 3, pp. 147–154, 2012.

[89] L. Organick, S. D. Ang, Y.-J. Chen, R. Lopez, S. Yekhanin, K. Makarychev, M. Z. Racz, G. Kamath, P. Gopalan, B. Nguyen *et al.*, "Random access in large-scale DNA data storage," *Nature Biotechnology*, vol. 36, no. 3, p. 242, 2018.

[90] K. A. S. Immink and K. Cai, "Properties and constructions of constrained codes for DNA-based data storage," *IEEE Access*, vol. 8, pp. 49 523–49 531, 2020.

[91] B. Cao, S. Zhao, X. Li, and B. Wang, "K-means multi-verse optimizer (KMVO) algorithm to construct DNA storage codes," *IEEE Access*, vol. 8, pp. 29 547–29 556, 2020.

[92] K. A. S. Immink and K. Cai, "Design of capacity-approaching constrained codes for DNA-based storage systems," *IEEE Communications Letters*, vol. 22, no. 2, pp. 224–227, 2017.

[93] M. G. Ross, C. Russ, M. Costello, A. Hollinger, N. J. Lennon, R. Hegarty, C. Nusbaum, and D. B. Jaffe, "Characterizing and measuring bias in sequence data," *Genome Biology*, vol. 14, no. 5, p. R51, 2013.

[94] S. H. T. Yazdi, Y. Yuan, J. Ma, H. Zhao, and O. Milenkovic, "A rewritable, random-access DNA-based storage system," *Scientific Reports*, vol. 5, p. 14138, 2015.

[95] T. Xue and F. C. M. Lau, "Construction of GC-Balanced DNA With Deletion/Insertion/Mutation Error Correction for DNA Storage System," *IEEE Access*, vol. 8, pp. 140 972–140 980, 2020.

[96] R. R. Varshamov, "An arithmetic function applicable to coding theory," in *Soviet Physics Doklady*, vol. 10, 1965, pp. 185–187.