DOCTOR OF PHILOSOPHY

RESEARCH ON CODE DESIGN FOR CORRECTING SYNCHRONIZATION ERRORS

The Graduate School
of the University of Ulsan

Department of Electrical, Electronic
and Computer Engineering

THI-HUONG KHUAT

Research on Code Design for Correcting Synchronization Errors

Supervisor: Sunghwan Kim

A Dissertation

Submitted to
the Graduate School of the University of Ulsan
In partial Fulfillment of the Requirements
for the Degree of

DOCTOR OF PHILOSOPHY
(Electrical Engineering)

by

Thi-Huong Khuat

Department of Electrical, Electronic
and Computer Engineering
University of Ulsan, Korea

August 2024

Research on Code Design for Correcting Synchronization Errors

This certifies that the dissertation of Thi-Huong Khuat is approved:

Sungoh Kwon [illegible] (signature)

Prof. Sungoh Kwon, *Committee chair*

Hee-Youl [illegible] [illegible] (signature)

Prof. Hee-Youl Kwak, *Committee member*

Min-Ho Jang [illegible] (signature)

Prof. Min-Ho Jang, *Committee member*

Yongjune Kim Y. Kim (signature)

Prof. Yongjune Kim, *Committee member*

Sunghwan Kim [illegible] (signature)

Prof. Sunghwan Kim, *Committee member*

Department of Electrical, Electronic and Computer Engineering

University of Ulsan, Korea

August 2024

VITA

Thi-Huong Khuat received the B.E. degree (2017) from Le Quy Don Technical University, Ha Noi, Viet Nam. In March 2019, she began to pursue the combined M.S. and Ph.D. degree at the Department of Electrical, Electronic, and Computer Engineering, University of Ulsan, South Korea, under the supervision of Professor Sunghwan Kim. Her research interests encompass 5G communication, error correction codes, and storage systems.

ACKNOWLEDGMENTS

This thesis would not have been possible without the unquestionable support of many people. I would like to express my gratitude to everyone who contributed to this thesis.

First and foremost, I would like to express my sincere gratitude to my supervisor, Prof. Sunghwan Kim for his patient guidance, support, and encouragement. During my studies, he used his profound theoretical knowledge and extensive experience to continuously help me refine my ideas and provide valuable feedback on my paper writing. Furthermore, I am indebted to him for his kindness, understanding in both academic and personal aspects.

Next, I am grateful to my labmates and friends in Coding and Information Theory (CIT) lab and at University of Ulsan for their friendship, enthusiastic help, and cheerfulness during my stay in Ulsan, Korea.

Additionally, I am deeply grateful to University of Ulsan (UoU) for offering me an exceptional research atmosphere and financial support (through BK21+ programs). Moreover, UoU gave me the chance to conduct useful courses with excellent instructors that will give me a way to think and work properly for the rest of my life.

Finally, I would like to express my gratitude to my parents, my husband, and my friends for their love and encouragement, which enables me to keep moving forward even under pressure and difficulties.

ABSTRACT

Research on Code Design for Correcting Synchronization Errors

by

Thi-Huong Khuat

Supervisor: Professor Sunghwan Kim

Submitted in Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy (Electrical Engineering)

August 2024

Synchronization errors represent a significant challenge in various communication and storage systems, undermining the reliability and efficiency of data transmission and retrieval processes. These errors occur when the timing or alignment of data becomes distorted during transmission or storage, leading to misinterpretation or loss of information. Addressing synchronization errors is crucial for ensuring data integrity, minimizing data loss, and maximizing system performance. To mitigate the effects of synchronization errors and fortify the resilience of information systems, error correction codes are known as a highly effective approach. Hence, designing synchronization error correction codes is necessary and critical in data transmission and retrieval processes.

In information theory, synchronization errors can be manifested in various forms, including deletions, insertions, substitutions, and transpositions of symbols. Consequently, designing synchronization error correction codes means constructing codes that can address deletion, insertion, substitution, and transposition errors. A large body of this thesis is the development of such codes using the number theoretic approach. Specifically, this research delves into the exploration of two novel code designs, each tailored with specific constraints aimed at safeguarding sequences from synchronization errors.

The first class of codes involves a binary code aimed at simultaneously rectifying two types of errors while employing techniques such as syndrome and accumulation values. Historically, prior research concentrated on rectifying singular type of errors, such as either a deletion or insertion error. The first proposed code is engineered to counteract the presence of both a deletion and an insertion error within a codeword. To broaden the spectrum of error correction, the second class of codes introduces a nonbinary code designed to rectify different errors, including a single deletion, insertion, substitution, or adjacent transposition errors. This thesis provides insights into the construction of code designs tailored to accommodate various error scenarios based on the specified errors, thereby showcasing the error correction capabilities of these proposed code designs through mathematical analysis. Alongside code designs, this thesis also presents decoding procedures to recover codewords from errors.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Introduction

The digital age has ushered in a myriad of technological advancements, from artificial intelligence and blockchain to 5G networks and cloud computing. One of the significant advantages of this era is the globalization it facilitates, with the world more interconnected than ever before. With over 8000 data centers distributed globally, tasks like sending, storing, and retrieving data have become remarkably streamlined [1]. However, despite these advancements, communication channels and data storage systems are susceptible to errors. These errors can occur during data transmission or retrieval when the timing or alignment of data becomes distorted and are known as synchronization errors. The consequences of synchronization errors in communication and storage systems can be far-reaching, impacting both data integrity and system performance. In communication networks, synchronization errors can cause disruptions in real-time applications such as voice calls, video streaming, and online gaming, leading to degraded user experiences and loss of revenue for service providers. In storage systems, synchronization errors can result in data corruption, loss of critical information, and reduced storage efficiency, posing significant risks to data integrity and security. Hence, addressing synchronization errors is crucial for ensuring the seamless operation of communication and storage systems, as well as safeguarding the integrity of critical data.

To mitigate synchronization errors, error correction codes (ECC) play an effective role in ensuring the precise functioning of communication and data storage systems. ECC for synchronization errors are studied from many aspects, however, within the scope of this thesis we focus on the error correction ability of codes. Based on the design of the communication and storage system, several code sets have been introduced as follows. For communication systems, [2] introduced a comprehensive channel model covering all three types of synchronization errors, leading to the development of various synchronization error correction codes. For instance, [3] presented a code class capable of rectifying deletions and substitutions within a byte range, while [4] introduced a moment balance scheme utilizing linear error correction codes to address deletion or insertion errors. Additionally, [5] addressed the challenge of improving bit error rates through distance-preserving mapping in permutation trellis codes. Furthermore, [6] focused on designing a code set ensuring error detection, complementing other communication techniques to guarantee accurate information reception. Moreover, [7] proposed probabilistic methods for correcting multiple deletion or insertion errors in a sequence, and approaches like

[8] or [9] employed marker codes or guess-and-check methods to effectively mitigate synchronization errors.

For flash memories, the permutation-coded rank modulation scheme aims to prevent inadvertent block deletion during overshoot events [10]. Recent developments, such as those outlined in [11, 12], introduce rank-encoded modulation schemes to combat a variety of errors, while Gabrys *et al.* [13] have devised asymptotically optimal permutation codes capable of one deletion correction. Meanwhile, the growing interest in DNA-based (Deoxyribonucleic acid-based) storage systems underscores their potential, although they are susceptible to synchronization errors during synthesizing and sequencing DNA strands. To ensure accurate information retrieval, error correction codes are indispensable, often leveraging linear block codes like Bose-Chaudhuri-Hocquenghem (BCH) code, Reed-Solomon (RS) code, and Low-density parity-check (LDPC) [14–17]. Notably, a class of synchronization error correction codes based on the number theoretic method, pioneered by Varshamov's Varshamov-Tenengolts (VT) code, emerges as a promising solution for addressing one or more synchronization errors within a sequence.

The main goal of this thesis is to explore good error correction codes to shield systems against synchronization errors by using the number theoretic method. In other words, how can we recover precisely the original sequences when a single or multiple synchronization errors? In this thesis, we delve into the code designs to overcome synchronization errors, which are manifested through deletion, insertion, substitution, and transposition errors. More specifically, how to recover the original sent or stored sequences when one deletion and one insertion errors occur simultaneously; when there is an edit error or an adjacent transposition error. Besides, we also suggest the encoding procedures for the proposed codes to overcome all possible error scenarios with specified errors.

1.2 Organization

The dissertation consists of five chapters structured as follows:

In Chapter 1, we sum up all the motivations relevant to synchronization errors and synchronization error correction codes. Then, we show the outline of the dissertation.

In Chapter 2, we provide mathematical preliminaries including some general notations, definitions in coding theory, and related number theory codes that are used in this thesis.

In Chapter 3, a new binary code design to deal with one deletion and insertion errors is presented. We also provide mathematical analysis to prove the error correction ability of the proposed code. In addition, our results include the decoding algorithms for codes to correct one deletion and one insertion in any position within a codeword.

In Chapter 4, we propose a novel nonbinary code design for correcting one edit error or an adjacent transposition error. The proposed code is constructed based on the weights of all elements, even index elements in the codeword, and the high-order syndromes of the codeword. We also provide proof of error correction ability and decoding algorithms for the proposed code.

Finally, in Chapter 5, we give some conclusions on the dissertation and some discussions on the future research directions.

Chapter 2

Preliminaries

2.1 General notation

The set of integers and real numbers are denoted as $\mathbb{Z}$ and $\mathbb{R}$, respectively. The notations, $\mathbb{Z}_+$ and $\mathbb{R}_+$, present the set of positive integers and real numbers, respectively. For a real number $m \in \mathbb{R}$, $\lfloor m \rfloor$ defines the largest integer that is not greater than m. Similarly, $\lceil m \rceil$ denotes the smallest integer which is not smaller than m.

For $m_1, m_2 \in \mathbb{Z}_+$ and $m_1 < m_2$, let $[m_1, m_2]$ be the consecutive indexes from m_1 to m_2, as $m_1, m_1 + 1, \cdots, m_2 - 1, m_2$.

Let $\mathbb{F}$ be an alphabet. For a positive integer n, we define $\mathbb{F}_q^n$ is the set comprising all q-ary sequences ($q \geq 2$) of length n, with $|\mathbb{F}_q^n|$ representing its size. A sequence $\boldsymbol{x} \in \mathbb{F}_q^n$ is given as $\boldsymbol{x} = (x_1, x_2, \cdots, x_n)$. Particularly, if $q = 2$, $\boldsymbol{x}$ is called a binary sequence and $x_k \in \{0, 1\}$ for $1 \leq k \leq n$, and if $q = 4$, $\boldsymbol{x}$ can be known as a quaternary sequence with $x_k \in \{0, 1, 2, 3\}$ for $1 \leq k \leq n$.

2.2 Coding theory

The different types of synchronization errors are delineated as follows.

- Deletion error(s) refer to one or multiple symbols being removed from the original sequence. For example, the original sequence is 011001, and a bit (the bold bit) is removed from the original sequence as

$$011\mathbf{0}01 \xrightarrow{deletion} 01101$$

- Insertion error(s) involve the addition of one or more symbols to the original sequence. For example, a scenario with an insertion error (shown in bold) in the original sequence can be illustrated as

$$011001 \xrightarrow{insertion} 01\mathbf{0}1001$$

- Substitution error(s) occur when one or multiple symbols are replaced by one or multiple symbols with values different from the original ones. Continuing the previous example, a substitution error (depicted in bold) in the original sequence is demonstrated as follows:

$$011001 \xrightarrow{substitution} \mathbf{1}11001$$

- Transposition error(s) are known as swapping the positions of two or more symbols with distinct values. Continuing the previous example, if there is a transposition error (highlighted by the bold bits) in the original sequence, it can be depicted as

$$011001 \xrightarrow{transposition} \mathbf{10}1001$$

The deletion or insertion error can be called the ‘indel’ error, and the ‘edit’ error stands for the deletion, insertion, or substitution error for convenience.

Throughout the thesis, a weight $wt(.)$ of a sequence is defined as a sum of all nonzero elements in the sequence. In other words, for any sequence $\boldsymbol{x} \in \mathbb{F}_q^n$, the weight of $\boldsymbol{x}$ can be expressed as

$$wt(\boldsymbol{x}) = \sum_{k=1}^{n} x_k, \tag{2.1}$$

where the addition is over an integer. For example, a sequence $\boldsymbol{x} \in \mathbb{F}_2^{10}$ as $\boldsymbol{x} = (0, 1, 0, 1, 1, 0, 0, 1, 1, 1)$, the weight of $\boldsymbol{x}$ can be determined as

$$wt(\boldsymbol{x}) = \sum_{k=1}^{10} x_k = 0 + 1 + 0 + 1 + 1 + 0 + 0 + 1 + 1 + 1 = 6.$$

A run of length h in a sequence $\boldsymbol{x} \in \mathbb{F}_q^n$ is defined as a maximal substring of h identical symbols in $\boldsymbol{x}$ for $1 \leq h \leq n$. The run vector $\boldsymbol{r_x}$ of $\boldsymbol{x}$ is defined as $\boldsymbol{r_x} = (r_1, r_2, \cdots, r_n)$, where the elements r_k for $1 \leq k \leq n$ denotes the run to which the k-th bit belongs. For example, a sequence $\boldsymbol{x} \in \mathbb{F}_2^{10}$ as $\boldsymbol{x} = (0, 1, 0, 1, 1, 0, 0, 1, 1, 1)$, the run vector $\boldsymbol{r_x}$ is determined as $\boldsymbol{r_x} = (1, 2, 3, 4, 4, 5, 5, 6, 6, 6)$.

Moreover, we define $\bar{\boldsymbol{r}}_{\boldsymbol{x}} = (\bar{r}_1, \bar{r}_2, \cdots, \bar{r}_n)$ as the run-length vector of $\boldsymbol{x}$, where the elements $\bar{r}_k$ stands for the length of the run to which the k-th bit belongs. For the sequence $\boldsymbol{x} = (0, 1, 0, 1, 1, 0, 0, 1, 1, 1) \in \mathbb{F}_2^{10}$, the run-length vector $\bar{\boldsymbol{r}}_{\boldsymbol{x}}$ is determined as $\bar{\boldsymbol{r}}_{\boldsymbol{x}} = (1, 1, 1, 2, 2, 2, 2, 3, 3, 3)$.

2.3 Number theoretic codes

By using the number theoretic approach, deletion/insertion error correction code designs are considered a timely topic. These codes can correct a limited number of deletion/ insertion errors in a sequence. In this subsection, we summarize the previous studies about deletion/insertion error correction code designs as a literature overview.

Varshamov. *et.al.* [18] first introduced a number theoretic code to correct a single deletion/insertion error. This code is known as the Varshamov-Tenengolts (VT) code, which was the first proposed to generate asymmetric error correction codes with syndrome calculation as shown in Definition 2.1.

Definition 2.1 ([18])**.** For $0 \leq m \leq n$ and $n \geq 3$, the Varshamov-Tenengolts code consists of all binary sequences $\boldsymbol{x} = (x_1, x_2, \cdots, x_n)$ of length n satisfying

$$VT_m(n) \triangleq \left\{ \boldsymbol{x} \in \mathbb{F}_2^n \mid \sum_{k=1}^{n} k \cdot x_k \equiv m \mod (n+1) \right\}. \tag{2.2}$$

Levenshtein adapted the binary VT code into an extended VT (e-VT) code [19], whose capable of rectifying a single insertion, deletion, or substitution error by modifying the modulo value as follows

Definition 2.2 ([19])**.** Given an integer $0 \leq m \leq 2n-1$, for $n \geq 3$ and the binary sequence $\boldsymbol{x} = (x_1, x_2, \cdots, x_n)$ of length n, the e-VT codes $eVT_m(n)$ are defined as

$$eVT_m(n) \triangleq \left\{ \boldsymbol{x} \in \mathbb{F}_2^n \mid \sum_{k=1}^{n} k \cdot x_k \equiv m \mod 2n \right\}. \tag{2.3}$$

Furthermore, by employing the binary VT code, Tenengolts [20] presented a nonbinary single deletion or insertion correcting code, known as q-VT code for $q > 2$. The q-VT code was defined as

Definition 2.3 ([20])**.** Let $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \cdots, \alpha_n)$ be the q-ary codeword, where $\alpha_i \in \{0, 1, \cdots, q-1\}$. Let $\boldsymbol{x} = (x_1, x_2, \cdots, x_n)$ denote a binary sequence mapped to $\boldsymbol{\alpha}$, where x_1 can be any binary bit, as follows

$$x_i = \begin{cases} 1 & \text{if } \alpha_i \geq \alpha_{i-1}, \\ 0 & \text{if } \alpha_i < \alpha_{i-1}. \end{cases} \tag{2.4}$$

For some fixed integers m_1 and m_2, the q-VT codes that can correct a single deletion or insertion error are defined as

$$qVT(n) \triangleq \Big\{ \boldsymbol{\alpha} \in \mathbb{F}_q^n, \boldsymbol{x} \in \mathbb{F}_2^n \mid \sum_{k=1}^{n} \alpha_k \equiv m_1 \bmod q, \tag{2.5}$$

$$\sum_{k=1}^{n} (k-1) \cdot x_k \equiv m_2 \bmod n. \Big\}. \tag{2.6}$$

Chapter 3

Binary code design for one deletion and one insertion errors

3.1 Introduction

Introduced in 1965, the VT code paved the way for numerous subsequent studies in code design. By utilizing syndrome calculation, this code could rectify single deletion or insertion errors within the binary regime [18] as well as within the nonbinary regime [20]. Shortly thereafter, based on VT codes, Levenshtein presented a code to correct either a single deletion, insertion, or substitution error in [19] and also constructed a code that can handle at most two adjacent deletion or insertion errors [21]. Moreover, in [19], Levenshtein explained that any code of length n, which can correct m deletion errors (or m insertion errors), can also correct totally m deletion and insertion errors. Nevertheless, no specific code construction was provided to support this statement. Thanks to the efficiency of the VT code construction, many attempts have been inspired to extend the VT codes to correct multiple deletion or insertion errors. For example, array-type codewords were proposed in [22, 23] for correcting two or more deletion/ insertion errors. By other approaches, the code constructions in [24,25] were provided to address two deletion errors, where the codes in [24] employed some specific substrings in a sequence and a series of constraints. Further, based on a number of constraints or list decoding, the redundancies of the codes in [25] were significantly improved than those in [24]. In addition, [26–28], the authors constructed a concatenated code to correct constant m deletion errors by using VT codes. Moreover, the authors of [29–31] proposed a number-theoretic construction to correct multiple deletion or insertion errors by employing a generalization of the Levenshtein codes. All of these studies [21–24, 26–30] focused on binary codes to correct only one-type errors. However, our goal in this work was to construct codes capable of correcting deletion and insertion errors that occur simultaneously in a codeword. Such codes have many applications in the synchronization of information in communication and storage [14, 32, 33].

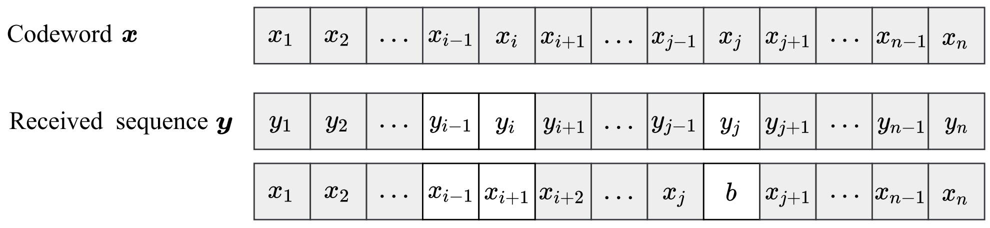
The channel which we consider produces received sequences whose length is the same as the original code length. When the length of the received sequence is not changed, it is regarded that there were no deletion or insertion errors and this could result in missing the errors during communication or the retrieval of storage systems. Therefore, we prioritize this issue in our code construction. Moreover, although proof of correcting multiple deletions using the binary code was previously given [30], reconstructing codewords is a hard problem and an efficient decoding algorithm has not been developed.

Motivated by the above reasons, we focus on solving the cases in which one deletion and one insertion error simultaneously occur in a codeword, thus a novel binary code construction that can correct one deletion and one insertion error at any position in a codeword is proposed. The proposed code design includes three constraints, where two first constraints are employed to isolate the possible error values and positions. The second constraint also provides the distances between the deletion and insertion errors. Then, the third constraint uses syndrome to verify the positions of errors and correct the sequence. Furthermore, we also specifically describe how our proposed code design can address one deletion and one insertion errors at any position. Finally, the thorough decoding procedure of the proposed code is designed for all error scenarios.

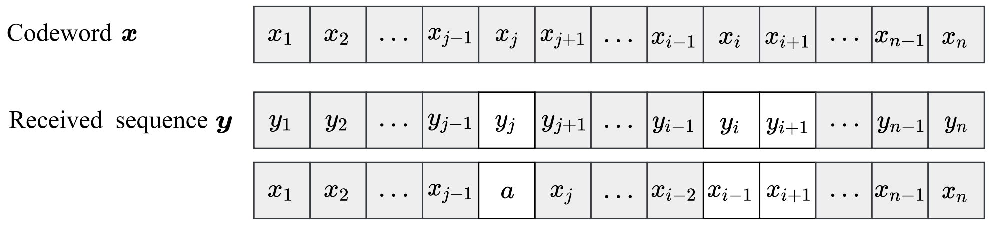
3.2 Channel Model

3.2.1 Channel model

Since the proposed code is designed to correct one deletion and one insertion errors that occur at any position, we first find the distances between the positions of the deletion and insertion errors. It is assumed that for any binary sequence $\boldsymbol{x}$ of length n, the i-th bit in $\boldsymbol{x}$ is deleted; and one bit is inserted at the j-th position in the received sequence $\boldsymbol{y}$. We first note that the error positions, i and j, are mutually irrelevant, which implies that the values of i and j do not affect each other. Thus, there exist two scenarios for the relation of the position indexes i and


Codeword $\boldsymbol{x}$
x_1 x_2 $\dots$ x_{i-1} x_i x_{i+1} $\dots$ x_{j-1} x_j x_{j+1} $\dots$ x_{n-1} x_n
Received sequence $\boldsymbol{y}$
y_1 y_2 $\dots$ y_{i-1} y_i y_{i+1} $\dots$ y_{j-1} y_j y_{j+1} $\dots$ y_{n-1} y_n
x_1 x_2 $\dots$ x_{i-1} x_{i+1} x_{i+2} $\dots$ x_j b x_{j+1} $\dots$ x_{n-1} x_n

(a) *DaIb* scenario.


Codeword $\boldsymbol{x}$
x_1 x_2 $\dots$ x_{j-1} x_j x_{j+1} $\dots$ x_{i-1} x_i x_{i+1} $\dots$ x_{n-1} x_n
Received sequence $\boldsymbol{y}$
y_1 y_2 $\dots$ y_{j-1} y_j y_{j+1} $\dots$ y_{i-1} y_i y_{i+1} $\dots$ y_{n-1} y_n
x_1 x_2 $\dots$ x_{j-1} a x_j $\dots$ x_{i-2} x_{i-1} x_{i+1} $\dots$ x_{n-1} x_n

(b) *IaDb* scenario.

Figure 3.1: Received sequence $\boldsymbol{y}$ for *DaIb* and *IaDb* scenarios.

j as $1 \leq i < j \leq n$ and $1 \leq j < i \leq n$. In scenarios with $1 \leq i < j \leq n$, a deletion error occurs before an insertion error (Deletion-Insertion). Similarly, a deletion error occurs after an insertion error (Insertion-Deletion) for $1 \leq j < i \leq n$. Moreover, when the position of the inserted bit in $\boldsymbol{y}$ is the same as the position of the deleted bit in $\boldsymbol{x}$, it means that a substitution error occurs at the j-th position in $\boldsymbol{y}$. By considering the substitution error as a special error scenario of the Insertion-Deletion case, corresponding to $1 \leq j \leq i \leq n$, the proposed code also corrects this substitution.

In this chapter, the error positions of a codeword $\boldsymbol{x}$ are mapped to the positions of a received sequence. For Deletion-Insertion, when one deletion error occurs at the i-th position of $\boldsymbol{x}$, and one bit is inserted at the j-th position of $\boldsymbol{y}$ for $1 \leq i < j \leq n$, the proposed decoder will try to find the bits which were deleted at between the

$(i-1)$-th and the i-th positions and inserted at the j-th positions in the received sequence. Similarly, for Insertion-Deletion with $1 \le j < i \le n$, the insertion error position is in the j-th index of $\boldsymbol{y}$; and the deletion error occurs between the i-th and the $(i+1)$-th positions in the received sequence $\boldsymbol{y}$.

In this paper, the values of the first error and the second error are denoted as a and b, respectively. Based on the error positions, the error scenarios are divided into two categories as Deletion-Insertion and Insertion-Deletion, where the notations $DaIb$ and $IaDb$ are used for Deletion-Insertion and Insertion-Deletion scenarios, respectively. The received sequences $\boldsymbol{y}$ are shown in Fig. 3.1 for $DaIb$ and $IaDb$ scenarios.

From Fig. 3.1, for $DaIb$ scenarios, the i-th bit x_i of $\boldsymbol{x}$ is a and deleted, and a bit b is inserted in the j-th index in the received sequence $\boldsymbol{y}$ with $i < j$. Thus, the elements y_k of $\boldsymbol{y}$ is expressed via $\boldsymbol{x}$ as

$$y_k = \begin{cases} x_k & \text{for } 1 \le k \le i-1 \text{ and } j+1 \le k \le n, \\ x_{k+1} & \text{for } i \le k \le j-1, \\ b & \text{for } k = j. \end{cases} \tag{3.1}$$

For $IaDb$ scenarios in Fig. 3.1, a bit a is inserted in the j-th index in the received sequence $\boldsymbol{y}$ and the i-th bit x_i of $\boldsymbol{x}$ is b and deleted with $j < i$. The elements y_k of

Table 3.1: The eight possible scenarios when one deletion and one insertion errors occur.

Case	Bit error values		Error positions	
	First error a	**Second error** b	$DaIb$	$IaDb$
1	0	0	$D0I0$	$I0D0$
2	0	1	$D0I1$	$I0D1$
3	1	0	$D1I0$	$I1D0$
4	1	1	$D1I1$	$I1D1$

the received sequence $\boldsymbol{y}$ are given as

$$y_k = \begin{cases} x_k & \text{for } 1 \leq k \leq j-1 \text{ and } i+1 \leq k \leq n, \\ a & \text{for } k = j, \\ x_{k-1} & \text{for } j+1 \leq k \leq i. \end{cases} \tag{3.2}$$

According to the combinations of values and orders of the two errors, eight possible error scenarios are listed in Table 3.1. Firstly, based on the values of the errors, Table 3.1 presents the four combinations of the values of a and b. Secondly, based on the error positions, our method divides the error scenarios into $DaIb$ and $IaDb$.

While designing a binary code, our strategy is first to reduce the possible error scenarios among the eight scenarios listed in Table 3.1. It is a significant step to give information about the error values and the distance between the error positions. Next, the values and positions of errors are determined to correct the sequence based on three constraints. As a result, the proposed code construction can correct one deletion and one insertion error at any position, which will be explained in Section 3.3.1. Detailed proof of each constraint will be given in the next sections with Lemmas 3.1– 3.3. The decoding procedures of our code construction will be provided in Fig. 3.2 in Section 11.

3.2.2 Preliminaries

It is assumed that a binary sequence of length n is denoted as $\boldsymbol{g} = (g_1, g_2, \cdots, g_n) \in \mathbb{F}_2^n$. A integer sequence $\boldsymbol{u_g}$ of length n is the accumulation sequence of $\boldsymbol{g}$ as $\boldsymbol{u_g}= (u_1, u_2, \cdots, u_n)$, whose each element u_k for $1 \leq k \leq n$ is determined as

$$u_k = \sum_{l=1}^{k} g_l, \tag{3.3}$$

where the addition is also over the integer. For example, a binary sequence $\boldsymbol{g}$ with length 10 is given as $\boldsymbol{g} = (0,1,0,1,1,0,0,1,1,0) \in \mathbb{F}_2^{10}$. From the definition of $\boldsymbol{u_g}$, $u_1 = g_1 = 0$, and $u_2 = g_1+g_2 = 0+1 = 1$. According to (3.3), the element u_n in $\boldsymbol{u_g}$ is the same as $w(\boldsymbol{g})$. Thus, the sequence $\boldsymbol{u_g}$ is obtained as $\boldsymbol{u_g} = (0,1,1,2,3,3,3,4,5,5)$.

3.3 Code construction

In this section, we present our code construction to correct one deletion and one insertion error and provide the functions of constraints in our code design. Firstly, subsection 3.3.1 shows our code construction with three constraints. Then, we present specific functions of constraints to correct one deletion and one insertion error in the next sections.

3.3.1 Code construction

First, we consider a positive, monotonically increasing integer sequence $\boldsymbol{v}$ of length n, whose element v_k is defined as

$$v_k = \begin{cases} 1 & \text{for } k = 1 \text{ and } k = 2, \\ 2k-3 & \text{for } 3 \le k \le n. \end{cases} \tag{3.4}$$

The sequence $\boldsymbol{v}$ is used for syndrome calculation in the code construction.

Definition 3.1. Let $C(n, c, d, e) \in \mathbb{F}_2^n$ denote the code of length n with parameters $0 \le c \le 2$, $0 \le d \le 2n+1$, and $0 \le e < 4n-6$, which is capable of correcting one deletion and one insertion errors. Let a sequence $\boldsymbol{x} \in C(n, c, d, e)$ be a codeword, and a sequence $\boldsymbol{u_x} = (u_1, u_2, \cdots, u_n)$ is determined by (3.3). Then, three constraints of the proposed code are presented as

$$C(n, c, d, e) \triangleq \Bigg\{ \boldsymbol{x} \in \mathbb{F}_2^n : wt(\boldsymbol{x}) \equiv c \bmod 3, \tag{3.5}$$

$$\sum_{k=1}^{n} u_k \equiv d \bmod (2n+2), \tag{3.6}$$

$$\sum_{k=1}^{n} v_k x_k \equiv e \bmod (4n-5) \Bigg\}. \tag{3.7}$$

For example, for $n = 20$, $c = 0$, $d = 0$, and $e = 0$, $\boldsymbol{x}$ =(0,0,0,0,0,0,1,0,1,0,0,1,0,0,0,0, 1,0,1,1) is a codeword of $C(20, 0, 0, 0)$, since $w(\boldsymbol{x}) \bmod 3 = 0$ and from (3.3), the accumulation sequence $\boldsymbol{u_x}$ of $\boldsymbol{x}$ as $\boldsymbol{u_x}$= (0,0,0,0,0,0,1,1,2,2,2,3,3,3,3,3,4,4,5,6), and $\sum_{k=1}^{20} u_k \bmod 42 = 0$. In addition, with the coefficients v_k determined in (3.4), the syndrome of $\boldsymbol{x}$ as $\sum_{k=1}^{20} v_k x_k \bmod 75 = 0$.

The two constraints in (3.5) and (3.6) are employed to categorize the error scenarios with different positions and values of error pairs as $DaIb$ or $IaDb$, as listed in Table 3.1. Furthermore, from constraint (3.6), the distance between the deletion and insertion error positions can be obtained. Based on this distance and the estimated values of the errors, we can compute feasible combinations of the positions of the errors. The constraint (3.7) is designed to determine the positions of the deleted and inserted bits filtered by constraints (3.5) and (3.6) and to recover codewords. Specific explanations of the three constraints of the proposed code construction are described in the next subsections.

3.3.2 Roles of constraint (3.5)

As mentioned in Section 3.3.1, we first reduce the possible scenarios among the eight scenarios in Table 3.1 according to constraints (3.5) and (3.6). In addition, constraints (3.5) and (3.6) give some potential combinations of the values and positions of the errors. However, if we know that Deletion-Insertion $DaIb$ or Insertion-Deletion $IaDb$ occurs, the constraints (3.5) and (3.6) can distinguish four scenarios of the error values in each case. In this subsection, we provide detailed descriptions of constraint (3.5) in our code construction.

Let a sequence $\boldsymbol{y} \in \mathbb{F}_2^n$ be generated by deleting one bit and inserting one bit from a codeword $\boldsymbol{x} \in C(n, c, d, e)$. Thus, from $\boldsymbol{y}$, a sequence $\boldsymbol{u_y} = (u'_1, u'_2, \cdots, u'_n)$ is obtained by (3.3). The first step to decode the original codeword is to calculate three parameters c', d', and e' as

$$c' \equiv wt(\boldsymbol{y}) \bmod 3, \tag{3.8}$$

$$d' \equiv \sum_{k=1}^{n} u'_k \bmod (2n+2), \tag{3.9}$$

$$e' \equiv \sum_{k=1}^{n} v_k y_k \bmod (4n-5), \tag{3.10}$$

where the coefficients v_k in (3.10) are the same as those in (3.4). Since a decoder knows exact parameters, c, d, and e, the second step is to calculate the difference between parameters as

$$\Delta_c \equiv c - c' \bmod 3, \tag{3.11}$$

$$\Delta_d \equiv d - d' \bmod (2n+2), \tag{3.12}$$

$$\Delta_e \equiv e - e' \bmod (4n-5). \tag{3.13}$$

Regardless of the error positions, deletion and insertion errors can affect the weight of a sequence, as shown in Table 3.2. Since the weight difference belongs to $\{-1, 0, 1\}$ in Table 3.2, the modulo value in the constraint (3.5) was chosen to be 3. The values of Δ_c in (3.11) for four cases are calculated as $\Delta_c \in \{0, 1, 2\}$ in Table 3.2. If Δ_c is calculated as $\Delta_c = 2$, which corresponds to Case 2 then the decoder knows that the deleted and inserted bits are 0 and 1, respectively. Similarly, if $\Delta_c = 1$, the deleted and inserted bits are 1 and 0, respectively. However, if $\Delta_c = 0$, Case 1 $(D0I0, I0D0)$ or Case 4 $(D1I1, I1D1)$ can be an answer, and more constraints are needed to distinguish between Case 1 and Case 4.

Table 3.2: Weight difference Δ_c for error scenarios.

Cases	**Deleted bit**	**Inserted bit**	**Scenario**	$c - c'$	Δ_c
1	0	0	$D0I0, I0D0$	0	0
2	0	1	$D0I1, I1D0$	-1	2
3	1	0	$D1I0, I0D1$	1	1
4	1	1	$D1I1, I1D1$	0	0

3.3.3 Roles of constraint (3.6)

In this subsection, we first explain the necessary parameters of constraint (3.6). Because of similarity, we present only a specific decoding process for $DaIb$; and then shortly explain the other part for $IaDb$.

It is assumed that the error scenarios $DaIb$ occur. According to (3.1) and error positions, $\boldsymbol{u_x}$ can be represented via $\boldsymbol{u_y} = (u'_1, u'_2, \cdots, u'_n)$ for $DaIb$ as

$$u_k = \begin{cases} u'_k & \text{for } 1 \le k \le i-1, \\ u'_{k-1} + a & \text{for } i \le k \le j, \\ u'_k + a - b & \text{for } j+1 \le k \le n. \end{cases} \tag{3.14}$$

For example, when n is 10 and the sequence $\boldsymbol{x} = (1,0,0,0,1,1,0,0,0,0) \in \mathbb{F}_2^{10}$ is transmitted, then the sequence $\boldsymbol{u_x}$ of length 10 is obtained from $\boldsymbol{x}$ as $\boldsymbol{u_x} = (1,1,1,1,2,3,3,3,3,3)$. It is assumed that a received sequence $\boldsymbol{y} = (1,0,\_0,1,\underline{0},1,0,0,0,0) \in \mathbb{F}_2^{10}$, where '_' and '$\underline{0}$' represent the deletion and insertion error, respectively. This means that the sequence $\boldsymbol{y}$ is obtained by deleting bit 0 in the third position and inserting bit 0 in the fifth position in $\boldsymbol{y}$, which corresponds to Deletion-Insertion

DaIb with $(a,b) = (0,0)$. Based on the sequence $\boldsymbol{y} = (1,0,0,1,0,1,0,0,0,0) \in \mathbb{F}_2^{10}$, $\boldsymbol{u_y}$ is determined as $\boldsymbol{u_y} = (1,1,1,2,2,3,3,3,3,3)$. From (3.14), the elements in $\boldsymbol{u_x}$ can be rewritten as

$$u_k = \begin{cases} u'_k & \text{for } 1 \le k \le 2, \\ u'_{k-1} & \text{for } 3 \le k \le 5, \\ u'_k & \text{for } 6 \le k \le 10. \end{cases} \tag{3.15}$$

To present the role of constraint (3.6), Lemmas 3.1 and 3.2 prove that the difference between the accumulation sequences $\boldsymbol{u_x}$ and $\boldsymbol{u_y}$ is always smaller than $2n+2$, then $2n+2$ is used as the modulo value of constraint (3.6). From the definitions of d, d', and Δ_d, Δ_d can be represented as $\Delta_d \equiv \sum_{k=1}^{n} (u_k - u'_k) \mod (2n+2)$.

Lemma 3.1. *Let $\boldsymbol{y}$ be a sequence of length n by deleting the i-th bit x_i of $\boldsymbol{x}$ is a and deleted and a bit b is inserted in the j-th index in $\boldsymbol{y}$ for $1 \le i < j \le n$, which stands for Deletion-Insertion DaIb. Then, the difference between the accumulation sequences $\boldsymbol{u_x}$ and $\boldsymbol{u_y}$ is bounded as*

$$\left| \sum_{k=1}^{n} (u_k - u'_k) \right| < 2n + 2. \tag{3.16}$$

Proof. Based on (3.14) and the sum of element difference between $\boldsymbol{u_x}$ and $\boldsymbol{u_y}$ for

$1 \leq i < j \leq n$, $\sum_{k=1}^{n}(u_k - u'_k)$ in (3.16) can be represented as

$$\begin{aligned}\sum_{k=1}^{n}(u_k - u'_k) &= \sum_{k=1}^{i-1}(u_k - u'_k) + \sum_{k=i}^{j}(u_k - u'_k) + \sum_{k=j+1}^{n}(u_k - u'_k)\\ &= \sum_{k=1}^{i-1}0 + \sum_{k=i}^{j}(u_{k-1} + a - u'_k) + \sum_{k=j+1}^{n}(a - b)\\ &= \sum_{k=i}^{j}(u'_{k-1} - u'_k) + \sum_{k=i}^{j}a + (n-j)(a-b)\\ &= u'_{i-1} - u'_j + (n-i+1)a - (n-j)b\\ &= u'_{i-1} - u'_j + R, \end{aligned} \tag{3.17}$$

where $R = (n-i+1)\,a - (n-j)\,b$. According to binary values, a and b, R can be given as

$$R = \begin{cases} 0 & \text{for } a = 0, b = 0,\\ n-i+1 & \text{for } a = 1, b = 0,\\ -n+j & \text{for } a = 0, b = 1,\\ j-i+1 & \text{for } a = 1, b = 1. \end{cases} \tag{3.18}$$

Since $1 \leq i < j \leq n$, R is bounded as

$$-n + j \leq R \leq n - i + 1. \tag{3.19}$$

Since u'_{i-1} and u'_j are bounded by $0 \leq u'_{i-1} \leq i-1$ and $0 \leq u'_j \leq j$, respectively, the term $u'_{i-1} - u'_j$ in (3.17) is asymptotically expressed as

$$-j \leq u'_{i-1} - u'_j \leq i - 1. \tag{3.20}$$

Based on (3.19) and (3.20), (3.17) is bounded by the following

$$-j - n + j \leq u'_{i-1} - u'_j + R \leq i - 1 + n - i + 1. \tag{3.21}$$

$$-n \leq u'_{i-1} - u'_j + R \leq n. \tag{3.22}$$

Since $-n \leq \sum_{k=1}^{n} (u_k - u'_k) \leq n$ and $|\sum_{k=1}^{n} (u_k - u'_k)| < 2n + 2$, (3.16) is satisfied.

□

Similarly, Lemma 3.2 shows a short explanation of the constraint (3.6) in Insertion-Deletion $IaDb$.

Lemma 3.2. *Let $\boldsymbol{y}$ be a sequence of length n inserting a bit a at the j-th index in $\boldsymbol{y}$ and deleting the i-th bit as b of $\boldsymbol{x}$ for $1 \leq j < i \leq n$, which stands for Insertion-Deletion $IaDb$. Then, the difference between the accumulation sequences $\boldsymbol{u_x}$ and $\boldsymbol{u_y}$ has a bound as below*

$$\left|\sum_{k=1}^{n} (u_k - u'_k)\right| < 2n + 2. \tag{3.23}$$

Proof. For Insertion-Deletion $IaDb$, the elements of $\boldsymbol{u_x}$ can be expressed via $\boldsymbol{u_y}$ as

$$u_k = \begin{cases} u'_k & \text{for } 1 \leq k \leq j-1, \\ u'_{k+1} - a & \text{for } j \leq k \leq i-1, \\ u'_k - a + b & \text{for } i \leq k \leq n. \end{cases} \tag{3.24}$$

From (3.24), $\sum_{k=1}^{n}(u_k - u'_k)$ can be represented by

$$\begin{aligned} \sum_{k=1}^{n}(u_k - u'_k) =& \sum_{k=1}^{j-1}(u_k - u'_k) + \sum_{k=j}^{i-1}(u_k - u'_k) + \sum_{k=i}^{n}(u_k - u'_k) \\ =& u'_i - u'_j - (n-j+1)a + (n-i+1)b. \end{aligned} \tag{3.25}$$

Based on (3.25), the proof steps are similar to Lemma 3.1 with $1 \leq j \leq i \leq n$, and the result is $-n \leq \sum_{k=1}^{n}(u_k - u'_k) \leq n$. Therefore, $|\sum_{k=1}^{n}(u_k - u'_k)| < 2n + 2$ and (3.23) is satisfied. □

3.3.4 Determining error scenarios based on constraints (3.5) and (3.6)

In this subsection, we present how to determine the error scenarios by using constraints (3.5) and (3.6). Cases 1 and 4 in Table 3.2 are distinguished by constraint (3.6), and Cases 2 and 3 are also distinguished by combining constraints (3.5) and (3.6).

To distinguish Cases 1 and 4, we use a threshold value of Δ_d as $n+1 = n+1$. For $DaIb$, Case 1 ($D0I0$) and Case 4 ($D1I1$) can be distinguished by Δ_d and $n+1$ as shown below

1. $D0I0$ with $(a,b) = (0,0)$:

   From (3.17) and (3.18), the term $\sum_{k=1}^{n}(u_k - u'_k)$ in Δ_d for $D0I0$ is expressed as

$$\sum_{k=1}^{n}(u_k - u'_k) = u'_{i-1} - u'_j. \tag{3.26}$$

   From $1 \le i < j \le n$, $u'_{i-1} - u'_j$ in (3.26) is bounded by

$$-j \le u'_{i-1} - u'_j \le 0. \tag{3.27}$$

   However, $u'_{i-1} - u'_j$ can be zero when the bits in the $(i-1)$-th and the j-th positions are zeros and belong to the same run. Then, the received sequence $\boldsymbol{y}$ is the same as $\boldsymbol{x}$. Thus, it can be regarded that no error occurs. Therefore, for $\boldsymbol{y} \ne \boldsymbol{x}$, since u'_j is always larger than u'_{i-1}, the bound (3.27) is modified as

$$-j \le u'_{i-1} - u'_j < 0. \tag{3.28}$$

Table 3.3: Values of Δ_c and Δ_d for eight error scenarios.

Error scenario	Δ_c	Δ_d
$D0I0$	0	$\Delta_d = u'_{i-1} - u'_j + 2n + 2 \geq n + 1$
$D0I1$	2	$\Delta_d = u'_{i-1} - u'_j + n + j + 2 \geq n + 1$
$D1I0$	1	$\Delta_d = u'_{i-1} - u'_j + n - i + 1 < n + 1$
$D1I1$	0	$\Delta_d = u'_{i-1} - u'_j + j - i + 1 < n + 1$
$I0D0$	0	$\Delta_d = u'_i - u'_j < n + 1$
$I0D1$	1	$\Delta_d = u'_i - u'_j + n - i + 1 < n + 1$
$I1D0$	2	$\Delta_d = u'_i - u'_j + j + n + 1 \geq n + 1$
$I1D1$	0	$\Delta_d = u'_i - u'_j + j - i + 2n + 2 \geq n + 1$

From the definition of Δ_d in (3.12) and (3.28), Δ_d of $D0I0$ can be determined as

$$\Delta_d \equiv \left(u'_{i-1} - u'_j\right) \mod (2n+2) = u'_{i-1} - u'_j + 2n + 2. \tag{3.29}$$

Therefore, Δ_d in (3.29) is lower bounded as

$$\Delta_d \geq -j + 2n + 2 \geq -n + 2n + 2 = n + 2 > n + 1. \tag{3.30}$$

2. $D1I1$ with $(a, b) = (1, 1)$:

   From (3.17) and (3.18), the term $\sum_{k=1}^{n}(u_k - u'_k)$ in Δ_d for $D1I1$ is expressed as

$$\sum_{k=1}^{n} (u_k - u'_k) = u'_{i-1} - u'_j + j - i + 1. \tag{3.31}$$

Table 3.4: Error scenarios according to Δ_c and Δ_d.

Δ_c	Δ_d	**Error scenario**
1	$\Delta_d < \lceil {}^{2n+2}/_{2} \rceil$	$D1I0, I0D1$
2	$\Delta_d \geq \lceil {}^{2n+2}/_{2} \rceil$	$D0I1, I1D0$
0	$\Delta_d < \lceil {}^{2n+2}/_{2} \rceil$	$I0D0, D1I1$
	$\Delta_d \geq \lceil {}^{2n+2}/_{2} \rceil$	$I1D1, D0I0$

Based on (3.20), (3.31) can be asymptotically expressed by

$$\begin{aligned} -j + j - i + 1 \leq u'_{i-1} - u'_j + j - i + 1 \leq i - 1 + j - i + 1, \\ 1 - i \leq u'_{i-1} - u'_j + j - i + 1 \leq j. \end{aligned} \tag{3.32}$$

Since $u'_{i-1} - u'_j$ is not larger than $j - i + 1$ and $j \leq n$, the bound (3.32) is expressed as

$$0 \leq u'_{i-1} - u'_j + j - i + 1 \leq n. \tag{3.33}$$

From (3.33), Δ_d of $D1I1$ is $\Delta_d = u'_{i-1} - u'_j + j - i + 1$ and is bounded by

$$0 \leq \Delta_d < n + 1. \tag{3.34}$$

From (3.30) and (3.34), in $DaIb$, if $\Delta_c = 0$ and $\Delta_d \geq n + 1$, the decoder knows that the deleted and inserted bits are zeros, corresponding to Case 1 ($D0I0$). When $\Delta_c = 0$ and $\Delta_d < n+1$, the deleted and inserted bits are ones, corresponding to Case 4 ($D1I1$). Therefore, if we know $DaIb$ occurs, four scenarios, $D0I0, D0I1, D1I0$, and $D1I1$, can be distinguished by the constraints (3.5) and (3.6).

The values of Δ_c and Δ_d for the eight error scenarios are listed in Table 3.3. The values of Δ_d for $DaIb$ and for $IaDb$ are given by (3.17) and (3.25), respectively. The relationship between Δ_d and the threshold value $n+1$ is considered to be similar to the cases of $D0I0$ and $D1I1$.

For decoding, according to Δ_c and Δ_d, the error scenarios are determined and summarized in Table 3.4. From Table 3.4, the decoder knows that $D1I0$ or $I0D1$ errors occur if $\Delta_c = 1$, but it cannot be distinguished by these two constraints of Δ_c and Δ_d. Similarly, the scenario $D0I1$ or $I1D0$ can be also acknowledged if $\Delta_c = 2$ but it cannot distinguish each error scenario exactly. When $\Delta_c = 0$ and $\Delta_d \geq n+1$, two scenarios, $D0I0$ and $I1D1$, are found but cannot be distinguished by these two constraints. Similarly, when $\Delta_c = 0$ and $\Delta_d < n+1$, two scenarios, $D1I1$ and $I0D0$, are also isolated but cannot be determined exactly by using these two constraints. Therefore, we provide constraint (3.7) not only to distinguish $\Delta_c = 0$ cases but also to find the positions of the errors.

3.3.5 Roles of constraint (3.7)

As mentioned in Section 3.3.1, the last constraint (3.7) in the proposed code construction, whose coefficients v_k given by (3.4), is employed to validate the exact values and positions of the errors. From the definition (3.4) of $\boldsymbol{v}$, the coefficients v_k are non-negative and increasing for all $k \geq 1$. Due to these characteristics of $\boldsymbol{v}$ and distances for $DaIb$ and $IaDb$, the possible indexes for the error values and positions result in different syndromes e are determined by the constraint (3.7). Therefore, the constraint (3.7) can confirm only one recovered sequence. The detailed proof will be explained in Section 11. The following lemma gives a necessary parameter for the constraint (3.7).

Lemma 3.3. *Let $\boldsymbol{x}$ and $\boldsymbol{y}$ be a transmitted codeword in $C(n,c,d,e)$ and a received sequence of length n by deleting a bit and inserting a bit in $\boldsymbol{x}$, respectively. Then, according to vector $\boldsymbol{v}$ in* (3.4)*, the difference of syndromes in* (3.7) *of $\boldsymbol{x}$ and $\boldsymbol{y}$ can be bounded as*

$$\left|\sum_{k=1}^{n} v_k(x_k - y_k)\right| < 4n - 5. \tag{3.35}$$

Proof. For *DaIb* case, from (1), the syndrome difference of $\boldsymbol{x}$ and $\boldsymbol{y}$ is presented as

$$\begin{aligned}\sum_{k=1}^{n} v_k(x_k - y_k) =& \sum_{k=1}^{i-1} v_k(x_k - y_k) + \sum_{k=i}^{j-1} v_k(x_k - y_k) + \sum_{k=j}^{n} v_k(x_k - y_k)\\ =&0 + \sum_{k=i}^{j-1} v_k(x_k - x_{k+1}) + v_j(x_j - b) + 0\\ =&v_i x_i - v_j b + \sum_{k=i+1}^{j} x_k(v_k - v_{k-1}).\end{aligned} \tag{3.36}$$

The term $v_i x_i - v_j b$ in (3.36) has bounds as

$$-v_j \le v_i x_i - v_j b \le v_i. \tag{3.37}$$

From (3.4), the term $\sum_{k=i+1}^{j} x_k(v_k - v_{k-1})$ in (3.36) has bounds as

$$0 \le \sum_{k=i+1}^{j} x_k(v_k - v_{k-1}) \le \sum_{k=i+1}^{j} (v_k - v_{k-1}). \tag{3.38}$$

Since $\sum_{k=i+1}^{j}(v_k - v_{k-1})$ in (3.38) is $v_j - v_i$, from (3.37) and $1 \le i < j \le n$, the bounds of (3.36) is derived as

$$-v_n \le \sum_{k=1}^{n} v_k(x_k - y_k) \le v_n.$$

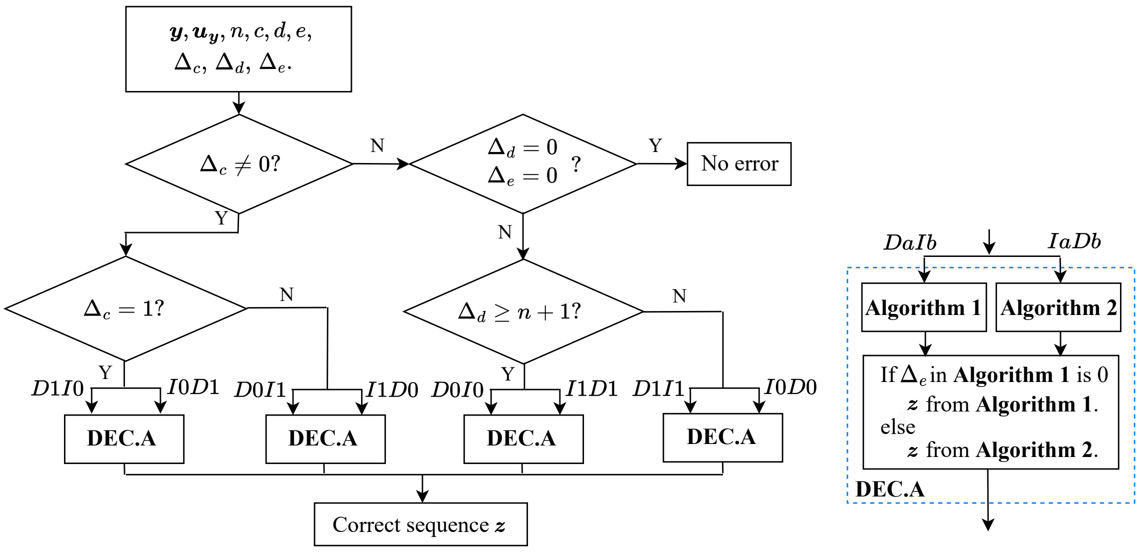

$\boldsymbol{y}, \boldsymbol{u_y}, n, c, d, e,$
Δ_c, Δ_d, Δ_e.
$\Delta_c \neq 0$?
N
$\Delta_d = 0$
$\Delta_e = 0$?
Y
No error
Y
N
$\Delta_c = 1$?
N
$\Delta_d \geq n+1$?
N
Y
Y
$D1I0$
$I0D1$
$D0I1$
$I1D0$
$D0I0$
$I1D1$
$D1I1$
$I0D0$
DEC.A
DEC.A
DEC.A
DEC.A
Correct sequence $\boldsymbol{z}$
$DaIb$
$IaDb$
Algorithm 1
Algorithm 2
If Δ_e in Algorithm 1 is 0
$\boldsymbol{z}$ from Algorithm 1.
else
$\boldsymbol{z}$ from Algorithm 2.
DEC.A

Figure 3.2: The overall decoding procedure of the proposed code.

Thus, $\left|\sum_{k=1}^{n} v_k(x_k - y_k)\right| < 2v_n + 1$.

For $IaDb$ case, based on (3.2), the syndrome difference of $\boldsymbol{x}$ and $\boldsymbol{y}$ is presented as

$$\begin{aligned}\sum_{k=1}^{n} v_k(x_k - y_k) &= \sum_{k=1}^{j-1} v_k(x_k - y_k) + \sum_{k=j}^{i} v_k(x_k - y_k) + \sum_{k=i+1}^{n} v_k(x_k - y_k) \\ &= v_i x_i - v_j a - \sum_{k=j}^{i-1} x_k(v_{k+1} - v_k).\end{aligned}$$

By using a similar method as (3.37) and (3.38), for $IaDb$ case $\left|\sum_{k=1}^{n} v_k\left(x_k - y_k\right)\right| < 2v_n+1$. Moreover, from the definition (3.4) of $\boldsymbol{v}$, when $k = n$, the value of $v_n = 2n-3$, leading to $2v_n + 1 = 4n - 5$. Therefore, (3.35) is always satisfied. □

It is presented to validate the exact values and positions of deletion and insertion errors using the constraint (3.7) in Section 11.

3.4 Overall decoding procedure

In this subsection, we present the decoding procedure of the proposed code based on our strategy described in Section 3.3.1. As mentioned in Section 3, the first step in our strategy is to use Δ_c and Δ_d values to determine the error scenarios based on Table 3.4. Then, based on the error values and Δ_d in the third column of Table 3.3, the possible pairs of the error positions can be obtained. Finally, the constraint (3.7) verifies the error positions to correct the sequence.

The overall decoding procedure of the proposed code is shown in Fig. 3.2. From the received sequence $\boldsymbol{y}$ and the code parameters, the decoder first determines $\boldsymbol{u}_{\boldsymbol{y}}$, Δ_c, Δ_d, and Δ_e in (3.11)–(3.13). Then, if the Δ_c, Δ_d, and Δ_e values are zero, $\boldsymbol{y}$ is regarded as $\boldsymbol{x}$. If not, the decoder traces the error values a and b based on Δ_c and Δ_d, then finds the error positions, and the corrected sequence $\boldsymbol{z}$ is produced. Based on Table 3.4, if $\Delta_c = 1$ the decoder knows that $D1I0$ or $I0D1$ scenarios occur. Similarly, if $\Delta_c = 2$, we can infer that $D0I1$ or $I1D0$ scenarios occur. Thus, when $\Delta_c = 1$ or $\Delta_c = 2$, the decoder uses the Δ_c value to isolate two error scenarios among four error scenarios, $I1D0, D1I0, I0D1$, and $D0I1$. The next step **DEC.A** is described in Fig. 3.2, where two decoding algorithms, **Algorithm 1** and **Algorithm 2**, are used simultaneously for $DaIb$ and $IaDb$, respectively. Moreover, when $\Delta_c = 0$, the Δ_d value is considered to isolate four scenarios, $D0I0, D1I1, I0D0$, and $I1D1$. If $\Delta_c = 0$ and $\Delta_d \geq n+1$, we know that two scenarios, $D0I0$ and $I1D1$, can occur, but cannot determine one of them exactly. Similarly, if $\Delta_c = 0$ and $\Delta_d < n+1$, two scenarios, $D1I1$ and $I0D0$ can occur, but also cannot be distinguished yet. Then, in **DEC.A**, **Algorithm 1** for $D0I0$ and **Algorithm 2** for $I1D1$ or **Algorithm 1** for $D1I1$ and **Algorithm 2** for $I0D0$ are employed.

3.4.1 Reducing candidates of error positions in Algorithms 1 and 2

After determining the error scenarios, the constraint (3.6) provides the distances between the error positions for decoding. According to the accumulated characteristic of $\boldsymbol{u}$ in (3.3), the error positions can be inferred via u_k. Thus, $u'_{i-1} - u'_j$ and $u'_i - u'_j$ present the distances between the deletion and insertion errors in $\boldsymbol{u_y}$ and are expressed via n, Δ_d, and the error positions i and j in the third column of Table 3.3. For decoding, the value u'_j is first considered to find candidates of the insertion error position in the received sequence $\boldsymbol{y}$. Then, from the distances in Table 3.3, the u'_{i-1} or u'_i values are obtained to estimate the deletion error position. Thus, based on these distances and error values, we can isolate possible candidates for the error positions in the sequence.

When errors occur in runs of length one, the candidates for the error positions should be always considered. However, if errors occur in runs with lengths larger than one, the candidates of the error positions can be reduced to design effective decoding, and the received sequence is still corrected successfully. Before explaining how to get the candidates for the error positions, we provide necessary lemmas to reduce the candidates for the error positions.

Lemma 3.4. *For DaIb, it is supposed that the distance between the previous bit of* *the deleted bit a as the $(l-1)$-th bit and the inserted bit b at the m-th position in a received sequence $\boldsymbol{y}$ satisfies the condition for Δ_d in* (3.12)*. It is also assumed that the run length of the m-th bit in $\boldsymbol{y}$ is larger than one. When m_k stands for indexes belonging to the run of the m-th bit in $\boldsymbol{y}$, the candidates at the run of the m-th bit in $\boldsymbol{y}$ can be reduced as one candidate.*

Proof. It is assumed that the run-length of the m-th bit in $\boldsymbol{y}$ is r_m and the indexes belonging to the run of the m-th bit are expressed as $\boldsymbol{m} = (m_1, m_1+1, \cdots, m_1+r_m-1)$. This means that the index m_1 is the first index of the run of the m-th bit. Let m_k be any element in $\boldsymbol{m}$.

For $D0I0$, from Table 3.3, the distances $u'_{l-1} - u'_{m_k}$ for $m_1 \leq m_k \leq m_1 + r_m - 1$ are given as

$$u'_{l-1} - u'_{m_k} = \Delta_d - 2n - 2. \tag{3.39}$$

Since $b = 0$, the run $\boldsymbol{m}$ in $\boldsymbol{y}$ is all zeros, and from (3.3), u'_{m_k} for all m_k in m are the same. Thus, the values $u'_{l-1} - u'_{m_k}$ in (3.39) are same for all m_k in $\boldsymbol{m}$. This means that the multiple candidates in $\boldsymbol{m}$, which are the same distance, can be reduced to one candidate.

For $D0I1$, from Table 3.3, the distances $u'_{l-1} - u'_{m_k}$ are expressed as

$$u'_{l-1} - u'_{m_k} = \Delta_d - n - m_k - 2. \tag{3.40}$$

Since $b = 1$, the run at $\boldsymbol{m}$ in $\boldsymbol{y}$ consists of all ones, and from (3.3), the values u'_{m_k} increase by one for all m_k in $\boldsymbol{m}$. Then, the values of $u'_{l-1} - u'_{m_k}$ in (3.40) decrease by one for all m_k in $\boldsymbol{m}$. However, the u'_{l-1} values to estimate the deletion error position

l are the same for all m_k in $\boldsymbol{m}$, and the insertion bit belongs to the run at $\boldsymbol{m}$. Then, the insertion error can be removed by deleting a bit in any position in the run. This means that the multiple candidates in $\boldsymbol{m}$ can be reduced to one candidate.

For $D1I0$, from Table 3.3, $u'_{l-1} - u'_{m_k}$ for $m_1 \leq m_k \leq m_1 + r_m - 1$ are determined as

$$u'_{l-1} - u'_{m_k} = \Delta_d - n + l - 1. \tag{3.41}$$

Similar to $D0I0$ scenario, the u'_{m_k} values for all m_k in $\boldsymbol{m}$ are same. Then, the values of $u'_{l-1} - u'_{m_k}$ in (3.41) are the same for all m_k in $\boldsymbol{m}$. Thus, the multiple candidates in $\boldsymbol{m}$, which have the same distance, can be reduced to one candidate.

For $D1I1$, from Table 3.3, the distances $u'_{l-1} - u'_{m_k}$ are formulated as

$$u'_{l-1} - u'_{m_k} = \Delta_d + l - m_k - 1. \tag{3.42}$$

Similar to scenario $D0I1$, the values u'_{m_k} increase by one for all m_k in $\boldsymbol{m}$. Then, the terms $u'_{l-1} - u'_{m_k}$ in (3.42) decrease by one for all m_k in $\boldsymbol{m}$. Therefore, the multiple candidates in $\boldsymbol{m}$ can be reduced to one candidate. $\square$

From Lemma 3.4, for the $DaIb$ scenarios, the multiple candidates in the run of the m-th bit can be reduced to one candidate. Thus, among the multiple candidates, the first element at the run of the m-th bit is only considered. In this case, the first element sometimes cannot reveal the actual insertion error position when the insertion error occurs at the second or later bits in the run. Even though the insertion error occurs at the second or later bits, the insertion error can be recovered by deleting the first bit of the run. Therefore, considering the first bit of the run of the m-th bit is enough to correct the insertion bit.

Lemma 3.5. *For DaIb, assuming that the distance between the previous bit of the deleted bit a as the $(l-1)$-th bit and the inserted bit b at the m-th position in a received sequence $\boldsymbol{y}$ satisfies the condition on Δ_d in (3.12). It is assumed that the $(l-1)$-th bit or the l-th bit in $\boldsymbol{y}$ is a. Then, when l_k stands for indexes belonging to the run of the $(l-1)$-th bit of value a or the l-th bit of value a in $\boldsymbol{y}$, the candidates at the run of the $(l-1)$-th bit or the l-th bit can be reduced as one candidate.*

Proof. See Appendix A.1. □

From Lemma 3.5, for the *DaIb* scenarios, the multiple candidates in the run of the $(l-1)$-th bit a or the l-th bit a can be reduced as one candidate. Thus, among the multiple candidates, the first element at the run of the $(l-1)$-th bit a or the l-th bit a is only considered. Similar to the insertion error case in Lemma 3.4, though the deletion error occurs at the second or later bits, the deletion error can be recovered by adding the first bit of the run. Therefore, considering the first bit of the run of the $(l-1)$-th bit a or the l-th bit a is enough to correct the deletion bit.

Lemma 3.6. *For DaIb, let a sequence $\boldsymbol{y}$ of length n be a received sequence by deleting the l-th bit a in a codeword $\boldsymbol{x}$ and inserting the m-th bit b in $\boldsymbol{y}$. It is assumed that the $(l-1)$-th bit or the l-th bit in $\boldsymbol{y}$ is a and the run length of the m-th bit in $\boldsymbol{y}$ is larger than one. Then, when the l_k and m_h stand for indexes belonging to runs of the $(l-1)$-th bit a or the l-th bit a and runs of the m-th bit in $\boldsymbol{y}$, respectively, the candidates at the run of the m-th bit b and at the run of the $(l-1)$-th bit a or the l-th bit a in $\boldsymbol{y}$ can be reduced as one candidate.*

Proof. From Lemmas 3.4 and 3.5, it is clear that when l_k and m_h represent the indexes belonging to runs of the $(l-1)$-th bit a or the l-th bit a and runs of the m-th bit in $\boldsymbol{y}$, respectively, the candidates of error positions can be reduced as one candidate. $\square$

Therefore, when decoding scenarios $DaIb$, the candidates of error positions can be reduced by choosing the first indexes of the runs whose run-length is more than one.

Lemma 3.7. *For $IaDb$, let a sequence $\boldsymbol{y}$ of length n be a received sequence by inserting the m-th bit a and deleting the l-th bit b in a codeword $\boldsymbol{x}$. It is assumed that the run length of the m-th bit in $\boldsymbol{y}$ is larger than one and the l-th bit or the $(l+1)$-th bit in $\boldsymbol{y}$ is b. Then, when m_h and l_k stand for indexes belonging to the runs of the m-th bit and the runs of the l-th bit of value b or the $(l+1)$-th bit of value b in $\boldsymbol{y}$, respectively, the candidates at the run of the m-th bit a and at the run of the l-th bit b or the $(l+1)$-th bit b in $\boldsymbol{y}$ can be reduced as one candidate.*

Proof. Due to the similarity to Lemmas 3.4–3.6, the specific steps for this proof are omitted. Thus, when m_h and l_k stand for indexes belonging to the runs of the m-th bit and the runs of the l-th bit of value b or the $(l+1)$-th bit of value b in $\boldsymbol{y}$, respectively, the candidates of the error positions can be reduced as one candidate. $\square$

Therefore, when decoding $IaDb$ scenarios, the candidates of the error positions can be reduced by choosing the first indexes of the runs whose run-length is larger than one.

3.4.2 Finding candidates of error positions and correcting the sequence in Algorithms 1 and 2

In this subsection, the steps to find the candidates for the error positions are provided as follows

For $DaIb$, from Lemmas 3.4–3.6, the candidates for the insertion position j are determined as

$$S_b = \{m \mid y_m = b, y_{m-1} \neq b, \text{ for } 2 \leq m \leq n\}. \tag{3.43}$$

For the elements m in S_b, the u'_m values are elements in $\boldsymbol{u_y}$ corresponding to candidates m of the insertion position j. Then, the decoder finds a set T_m, whose elements l are the candidates for i. The set T_m is determined by the values of u'_{i-1} and u'_j in Table 3.3 and Lemmas 3.5 and 3.6, where u'_m and u'_{l-1} in Lemmas 3.5 and 3.6 stand for u'_j and u'_{i-1} in Table 3.3, respectively. The set T_m is shown to be

$$\begin{aligned} T_m = \{l \mid u'_{l-1} = \Delta_d + (n+l+1)a + (n-m)b - 2n - 2 + u'_m, \\ y_{l-1} \neq a, \text{ and } l < m\}. \end{aligned} \tag{3.44}$$

Then, a whole candidate set S is defined as

$$S = \{(l, m) \mid m \in S_b, l \in T_m\}. \tag{3.45}$$

Among all of the elements in S, **Algorithm 1** can find the solutions to satisfy the constraint (3.7).

Similarly, from Lemma 3.7 for $IaDb$, an index set S_a in $\boldsymbol{y}$, which are candidates for the insertion position j is expressed as

$$S_a = \{m \mid y_m = a, y_{m-1} \neq a, \text{ for } 1 \leq m \leq n-1\}. \tag{3.46}$$

For all elements m in S_a, the decoder finds a set T_m, whose elements are candidates for the deletion position i and these candidates are determined via u'_l. The set T_m including candidates l of the deletion position is given based on Lemma 3.7 as

$$T_m = \{l \mid u'_l = \Delta_d - (n+m+1)a - (n-l+1)b + u'_m, y_l \neq b, \text{ and } l \geq m\}. \tag{3.47}$$

Then, a whole candidate set S can be defined by

$$S = \{(l,m) \mid m \in S_a, l \in T_m\}. \tag{3.48}$$

Among all of the elements in S, **Algorithm 2** can find the solutions to satisfy the constraint (3.7).

After finding the candidates of error positions, **Algorithm 1** and **Algorithm 2** for *DaIb* and for *IaDb* are provided, respectively. After determining candidates of the error positions, two decoding algorithms correct a sequence by using constraint (3.7). If a sequence $\boldsymbol{z}$ which is recovered from a candidate pair of error positions has syndrome e'' equals to e, ($\Delta_e = e'' - e \bmod (4n-5) = 0$), the decoding is successful and the reconstructed sequence is $\boldsymbol{z}$.

When $\boldsymbol{z}$ is a generated sequence from the exact error positions (i,j) in the received sequence $\boldsymbol{y}$, the sequence $\boldsymbol{z}$ should be the same with the codeword $\boldsymbol{x}$, and satisfy $\Delta_e = 0$. However, the candidate sets S in (3.45) and (3.48) are chosen as the first indexes of the runs according to Lemmas 3.4–3.7. Therefore, we have to check if the codeword $\boldsymbol{z}$ generated from the first bits of the runs of the i-th bit and the j-th bit still satisfies $\Delta_e = 0$ in the following lemmas.

Lemma 3.8. *For DaIb, when the i-th bit is deleted and a bit b is inserted at the j-th bit in the received sequence $\boldsymbol{y}$, then the sequence $\boldsymbol{z}$ that is reconstructed from (l, m), where l and m are the first indexes at the runs of the i-th bit and the j-th bit in* (3.43) *and* (3.44)*, respectively, satisfies $\Delta_e = 0$.*

Proof. From Lemmas 3.4–3.6 and the definition of the candidate set S in (3.45), since the first indexes at the run of the inserted bit b are included in S, the first index m at the run of the exact insertion position j is included in S. And since the first indexes at the run of the deleted bit a are elements in S, the first index l at the run of the exact deletion position i is included in S.

It is supposed that a reconstructed $\boldsymbol{z}$ is generated from the first indexes (l, m) and the run-length of the i-th bit and the j-th bit in $\boldsymbol{z}$ are r_i and r_j, respectively to find the original codeword $\boldsymbol{x} = (x_1, x_2, \cdots, x_n)$. This means that $l \le i \le l + r_i - 1$ and $m \le j \le m + r_j - 1$. The syndrome $e''_{\boldsymbol{z}}$ of $\boldsymbol{z}$ can be calculated as

$$\begin{aligned} e''_{\boldsymbol{z}} &\equiv \sum_{k=1}^{n} v_k z_k \bmod (4n-5) \\ &= \left(\sum_{k=1}^{l-1} v_k z_k + \sum_{k=l}^{l+r_i-1} v_k z_k + \sum_{k=l+r_i}^{m-1} v_k z_k + \sum_{k=m}^{m+r_j-1} v_k z_k + \sum_{k=m+r_j}^{n} v_k z_k \right) \bmod (4n-5). \end{aligned} \tag{3.49}$$

Since all elements z_k for $1 \le k \le l-1, l + r_i \le k \le m$, and $m + r_i \le k \le n$ are the same as with the codeword bit x_k, the three terms $\sum_{k=1}^{l-1} v_k z_k, \sum_{k=l+r_i}^{m-1} v_k z_k$, and $\sum_{k=m+r_j}^{n} v_k z_k$ in (3.49) are the same as $\sum_{k=1}^{l-1} v_k x_k, \sum_{k=l+r_i}^{m-1} v_k x_k$, and $\sum_{k=m+r_j}^{n} v_k x_k$, respectively. Since the deletion position i is between l and $l + r_i - 1$ and in the same run, the subsequence $(z'_l, z'_{l+1}, \cdots, z'_{l+r_i-1})$ is recovered by inserting a bit a at the index i of the run of length r_i as $(a, a, \cdots, a)$ and this run is the same as $(x_l, x_{l+1}, \cdots, x_{l+r_i-1})$. By inserting a bit a at the index l, we can obtain the subsequence $(z_l, z_{l+1}, \cdots, z_{l+r_i-1})$ as $(a, a, \cdots, a)$ of length r_i. This means that the subsequence $(z_l, z_{l+1}, \cdots, z_{l+r_i-1})$ is the same with $(x_l, x_{l+1}, \cdots, x_{l+r_i-1})$ and $\sum_{k=l}^{l+r_i-1} v_k z_k = \sum_{k=l}^{l+r_i-1} v_k x_k$. Similarly, for the insertion part, $\sum_{k=m}^{m+r_j-1} v_k z_k = \sum_{k=m}^{m+r_j-1} v_k x_k$. Therefore, since $\sum_{k=1}^{n} v_k z_k = \sum_{k=1}^{n} v_k x_k$, the syndrome $e''_{\boldsymbol{z}}$ in (3.49) is the same as e of $\boldsymbol{x}$ and $\Delta_e = 0$. □

Lemma 3.9. *For IaDb, when a bit a is inserted in the j-th index of $\boldsymbol{y}$, and deletion error occurs between the i-th and the $(i+1)$-th positions in the received sequence $\boldsymbol{y}$, a sequence $\boldsymbol{z}$ that is reconstructed from (l, m). The elements l and m are the first indexes at the runs of the deleted bit and the inserted bit in* (3.46) *and* (3.47)*, respectively, satisfying $\Delta_e = 0$.*

Proof. See Appendix A.2 □

The inputs of **Algorithm 1** are the error values a and b, parameters $\Delta_c, \Delta_d, \Delta_e$, $\boldsymbol{y} \in \mathbb{F}_2^n$, and $\boldsymbol{u_y}$. In **Algorithm 1**, the decoder finds an index set S_b according to (3.43), where each element of S_b is a candidate to estimate the insertion position j. For each element m in S_b, depending on the deletion value a, the decoder finds a set T_m by (3.44), whose elements l are candidates for i. Then, a set S comprises pairs of candidates m for j and candidates l for i. For each pair (p, q) of S, a sequence of length n as $Temp\_y$ is obtained by removing the q-th bit and adding a bit with

Algorithm 1: Correcting sequence for *DaIb*

```
Input: $\boldsymbol{y}, \boldsymbol{u_y}, n, a, b, d, c, e, \Delta_c, \Delta_d, \Delta_e, i < j$.
Output: Correct sequence $\boldsymbol{z}, \Delta_e$.
1  Find a set $S_b$ according to (3.43).
2  for all $m \in S_b$ do
3      Find a set $T_m \neq \emptyset$ according to (3.44).
4      Create a set $S$ according to (3.45).
5  for all $(p, q) \in S$ do
6      $Temp\_y = (y_1, \cdots, y_{p-1}, a, y_p, \cdots, y_{q-1}, y_{q+1}, \cdots, y_n)$.
7      $e'' = \sum_{k=1}^{n} v_k y'_k \bmod (4n-5)$ with $y'_k \in Temp\_y$.
8      if $e'' \neq e$ then
9          goto line 5.
10     else
11         $\boldsymbol{z} = Temp\_y, \Delta_e = 0$; stop.
```

the a value in the p-th position in $\boldsymbol{y}$. To correct the sequence with error positions (p, q), the syndrome e'' of $Temp\_y$ is calculated as in line 7 in **Algorithm 1**, and compared to e in (3.7). If e'' and e are not equal, the next pair of (p, q) in S needs to be checked. If e'' equals e, a sequence $\boldsymbol{z}$ is recovered from $\boldsymbol{y}$ as $\boldsymbol{z} = Temp\_y$, Δ_e is updated as zero, and the algorithm is finished. Therefore, the final output of **Algorithm 1** is the reconstructed sequence $\boldsymbol{z}$.

Due to the similarity of the approaches to **Algorithm 1**, the descriptions for the procedure of **Algorithm 2** are omitted.

We note that **Algorithms 1** and **2** are designed to produce a unique sequence $\boldsymbol{z}$ from the received sequence $\boldsymbol{y}$, which corresponds to the codeword $\boldsymbol{x}$. Since from Lemmas 3.8 and 3.9, there is always an element in S to satisfy $\Delta_e = 0$, we need to check whether two algorithms produce only one solution or not. However, the '**stop**' at line 11 in the two algorithms seems to stop the algorithms and to find the first solution even though there are multiple solutions to satisfy $\Delta_e = 0$. In this case, the two algorithms cannot recover the codeword $\boldsymbol{x}$. Therefore, we need to investigate that two algorithms without '**stop**' produce a unique solution and the proof of unique decoding will be explained in the next subsection. The '**stop**' in the two algorithms infers that the first solution is always the same as $\boldsymbol{x}$ and it is not necessary to search the other candidates in S.

3.4.3 Unique correct sequence of Algorithms 1 and 2

In this subsection, we discuss that the overall decoding is uniquely decodable, which means that the proposed decoder produces only one correct codeword from the received sequence. It also means that other error positions that are not in the runs and generate a sequence satisfying $\Delta_e = 0$ from $\boldsymbol{y}$ do not exist. Therefore, the term '**stop**' in the two algorithms is temporally removed and the whole solutions for the error positions are searched to check the number of correct sequences.

The following theorems show that **Algorithms 1** and **2** without '**stop**' provide a unique correct sequence. To explain Theorems 3.1 and 3.2, we first present the following two lemmas.

Algorithm 2: Correcting sequence for $IaDb$

```
Input: $\boldsymbol{y}, \boldsymbol{u_y}, n, a, b, d, c, e, \Delta_c, \Delta_d, \Delta_e, j \leq i$.
Output: Correct sequence $\boldsymbol{z}, \Delta_e$.
1  Find a set $S_a$ according to (3.46).
2  for all $m \in S_a$ do
3      Find a set $T_m \neq \emptyset$ according to (3.47).
4      Create a set $S$ according to (3.48).
5  for all $(p, q) \in S$ do
6      $Temp\_y = (y_1, \cdots, y_{q-1}, y_{q+1}, \cdots, y_p, b, y_{p+1}, \cdots, y_n)$.
7      $e'' = \sum_{k=1}^{n} v_k y'_k \bmod (4n-5)$ with $y'_k \in Temp\_y$.
8      if $e'' \neq e$ then
9          goto line 5.
10     else
11         $\boldsymbol{z} = Temp\_y, \Delta_e = 0;$ stop.
```

Lemma 3.10. *For a sequence* $\boldsymbol{y}$ *of length* n*, a partial sum* $\sum_{k=p+1}^{q} v_k(y_k - y_{k-1})$ *with* $1 < p < q \leq n$ *is conditioned as*

$$\sum_{k=p+1}^{q} v_k(y_k - y_{k-1}) \bmod (4n-5) \neq 0. \tag{3.50}$$

Proof. See Appendix A.3. □

Lemma 3.11. *For a sequence* $\boldsymbol{y}$ *of length* n*, a partial sum* $\sum_{k=p+1}^{q} v_k(y_{k-1} - y_{k+1})$

with $1 < p < q \leq n$ *satisfies*

$$\sum_{k=p+1}^{q} v_k(y_{k-1} - y_{k+1}) \bmod (4n-5) \neq 0. \tag{3.51}$$

Proof. Using a similar approach to Lemma 3.10, the specific steps to prove (3.51) are omitted. Then, (3.51) is always satisfied. □

Theorem 3.1. *For the decoding of DaIb, with a received sequence* $\boldsymbol{y}$*,* $\boldsymbol{u_y}$*, parameters* n, c, d, e *and* $(4n-5)$*,* ***Algorithm 1*** *without '****stop****' gives a unique correct sequence by using the constraint* (3.7)*.*

Proof. From Lemma 3.8, let (i_1, j_1) be one correct solution of **Algorithm 1** without '**stop**'. It is assumed that $\boldsymbol{z}$ is a codeword that is recovered by removing the j_1-th bit b and inserting the i_1-th bit a in $\boldsymbol{y}$ when $i_1 < j_1$. This means that according to **Algorithm 1**, syndrome e'' of $\boldsymbol{z}$ as $e''_{\boldsymbol{z}}$ equals e.

It is assumed that a pair (i_2, j_2) is also another solution of **Algorithm 1** without '**stop**' and two solutions satisfy $u'_{i_1-1} \neq u'_{i_2-1}$ or $u'_{j_1} \neq u'_{j_2}$. Then, $\boldsymbol{w}$ is recovered from the error positions (i_2, j_2) with $i_2 < j_2$, and $\boldsymbol{w} \neq \boldsymbol{z}$. This means that syndrome e'' of $\boldsymbol{w}$ as $e''_{\boldsymbol{w}}$ equals to e. Since two values $e''_{\boldsymbol{z}}$ and $e''_{\boldsymbol{w}}$ should be the same and equal to e, Δ_e is expressed as

$$\Delta_e = e''_{\boldsymbol{z}} - e''_{\boldsymbol{w}} \bmod (4n-5) = 0. \tag{3.52}$$

From (3.10), Δ_e in (3.52) can be rewritten as

$$\Delta_e = \sum_{k=1}^{n} v_k(z_k - w_k) \bmod (4n-5) = 0. \tag{3.53}$$

Since (i_1, j_1) is a correct solution of error positions, to prove that **Algorithm 1** without '**stop**' produces a unique correct sequence, (i_2, j_2) cannot be a solution for **Algorithm 1** without '**stop**'. This means that (3.53) should be not satisfied.

First, the condition $u'_{i_1-1} \neq u'_{i_2-1}$ or $u'_{j_1} \neq u'_{j_2}$ is classified into nine cases as $i_1 = i_2 < j_1 < j_2, i_1 = i_2 < j_2 < j_1, i_1 < i_2 < j_1 = j_2, i_2 < i_1 < j_1 = j_2, i_1 < i_2 < j_1 < j_2, i_1 < j_1 < i_2 < j_2, i_2 < i_1 < j_1 < j_2, i_1 < i_2 < j_2 < j_1$, and $i_2 < i_1 < j_2 < j_1$. Then, for the different locations of i_1, j_1, i_2, and j_2, (3.53) can be rewritten as below

1. For $i_1 = i_2 < j_1 < j_2$, the term $\sum_{k=1}^{n} v_k(z_k - w_k)$ in (3.53) is presented as

$$\begin{aligned}\sum_{k=1}^{n} v_k(z_k - w_k) =& \sum_{k=1}^{i_1-1} v_k(z_k - w_k) + \sum_{k=i_1}^{j_1} v_k(z_k - w_k) + \sum_{k=j_1+1}^{j_2} v_k(z_k - w_k) \\ &+ \sum_{k=j_2+1}^{n} v_k(z_k - w_k) \\ =& 0 + 0 + \sum_{k=j_1+1}^{j_2} v_k(z_k - w_k) + 0 \\ =& \sum_{k=j_1+1}^{j_2} v_k(y_k - y_{k-1}). \end{aligned} \tag{3.54}$$

   Since $\boldsymbol{w} \neq \boldsymbol{z}$, there is always at least one position k for $j_1 + 1 \leq k \leq j_2$ to $z_k \neq w_k$. From Lemma 3.10, the term $\sum_{j_1+1}^{j_2} v_k(y_k - y_{k-1}) \bmod (4n - 5)$ in (3.54) is always not zero. Thus, (3.53) is not satisfied.

2. Since the proof steps for $i_1 = i_2 < j_2 < j_1, i_1 < i_2 < j_1 = j_2$, and $i_2 < i_1 < j_1 = j_2$ are similar to those for $i_1 = i_2 < j_1 < j_2$ and are omitted. For these cases, (3.54) is also not satisfied.

3. For $i_1 < i_2 < j_1 < j_2$, the term $\sum_{k=1}^{n} v_k(z_k - w_k)$ in (3.53) is expressed as

$$\begin{aligned}\sum_{k=1}^{n} v_k(z_k - w_k) =& \sum_{k=1}^{i_1-1} v_k(z_k - w_k) + \sum_{k=i_1}^{i_2} v_k(z_k - w_k) + \sum_{k=i_2+1}^{j_2} v_k(z_k - w_k) \\ &+ \sum_{k=j_2+1}^{n} v_k(z_k - w_k) \\ =& 0 + \sum_{k=i_1}^{i_2} v_k(z_k - w_k) + 0 + \sum_{k=j_1+1}^{j_2} v_k(z_k - w_k) + 0 \\ =& v_{i_1}(a - y_{i_1}) + v_{i_2}(y_{i_2-1} - a) + \sum_{k=j_1+1}^{i_2-1} v_k(y_{k-1} - y_k) \\ &+ \sum_{k=j_1+1}^{j_2} v_k(y_k - y_{k-1}). \end{aligned} \quad (3.55)$$

Due to $\boldsymbol{w} \neq \boldsymbol{z}$, there is always at least one position k for $i_1 \leq k \leq i_2$ or $j_1 + 1 \leq k \leq j_2$ to $z_k \neq w_k$. Thus, (3.55) needs to be not equal to zero.

Since i_2 is the first index of the run of deleted bit in $\boldsymbol{w}$ and $y_{i_2-1} \neq a$, $v_{i_2}(y_{i_2-1} - a) \neq 0$. Moreover, since $v_{i_1}(a - y_{i_1})$ can have values $\{-v_{i_1}, 0, v_{i_1}\}$ and $v_{i_1} < v_{i_2}$, $v_{i_1}(a - y_{i_1}) + v_{i_2}(y_{i_2-1} - a) \neq 0$. Since the values v_k are nonnegative and increasing for $i_1 \leq k \leq i_2$, $v_{i_1}(a - y_{i_1}) + v_{i_2}(y_{i_2-1} - a) + \sum_{k=i_1+1}^{i_2-1} v_k(y_{k-1} - y_k)$ in (3.55) is not always zeros. From Lemma 3.10 and owing to the fact that v_k for $j_1 + 1 \leq k \leq j_2$ is always larger than v_k for $i_1 \leq k \leq i_2$, the right side of (3.55) cannot be zero. Thus, (3.55) is always not equal to zero, and (3.53) is not satisfied.

4. Similarly, for the four cases $i_1 < j_1 < i_2 < j_2, i_2 < i_1 < j_1 < j_2, i_1 < i_2 < j_2 < j_1$, and $i_2 < i_1 < j_2 < j_1$, the steps in the proof are almost the same to those for $i_1 < i_2 < j_1 < j_2$. Therefore, the specific proof steps are omitted.

For these cases, (3.53) is also not satisfied.

From steps 1) to 4), any pair (i_2, j_2) with $u'_{i_1-1} \neq u'_{i_2-1}$ or $u'_{j_1} \neq u'_{j_2}$ cannot be a solution of error positions to satisfy $\Delta_e = 0$ for **Algorithm 1** without '**stop**'. Therefore, **Algorithm 1** without '**stop**' produces a unique correct sequence as $\boldsymbol{z}$ and $\boldsymbol{z}$ is reconstructed by removing the j_1-th bit b and inserting the i_1-th bit a and in $\boldsymbol{y}$ with $i_1 < j_1$. □

Theorem 3.2. *For the decoding of IaDb, from a received sequence $\boldsymbol{y}$, $\boldsymbol{u_y}$, parameters n, c, d, e and $(4n-5)$,* ***Algorithm 2*** *without '**stop**' gives a unique correct sequence by using the constraint* (3.7)*.*

Proof. From Lemma 3.9, the exact position (i_1, j_1) is one solution of **Algorithm 2** without '**stop**'. It is assumed that $\boldsymbol{z}$ is a sequence that is recovered by removing the j_1-th bit a and inserting a bit b between the i_1-th and the (i_1+1)-th bits in $\boldsymbol{y}$ when $j_1 \leq i_1$. This means that from **Algorithm 2**, syndrome e'' of $\boldsymbol{z}$ as $e''_{\boldsymbol{z}}$ equals e.

It is assumed that a pair (i_2, j_2) is also another solution of **Algorithm 2** without '**stop**', and two solutions satisfy $u'_{i_1+1} \neq u'_{i_2+1}$ or $u'_{j_1} \neq u'_{j_2}$. Then, $\boldsymbol{w}$ is recovered with error positions (i_2, j_2) with $j_2 \leq i_2$, and $\boldsymbol{w} \neq \boldsymbol{z}$. This means that syndrome e'' of $\boldsymbol{w}$ as $e''_{\boldsymbol{w}}$ equals e.

Since the proof approach is similar to Theorem 3.1, the specific proof steps are omitted. Therefore, **Algorithm 2** without '**stop**' produces a unique correct sequence $\boldsymbol{z}$, and $\boldsymbol{z}$ is retrieved by removing the j_1-th bit a and inserting a bit b between the i_1-th and the (i_1+1)-th bits in $\boldsymbol{y}$ with $j_1 \leq i_1$. □

3.4.4 The decoding steps in DEC.A

As mentioned in Section 11, when $\Delta_c = 1$, based on Table 3.4, the two scenarios, $D1I0$ and $I0D1$, need to be investigated. Similarly, when $\Delta_c = 2$, the two scenarios, $D0I1$ and $I1D0$, are determined. For $\Delta_c = 0$ and $\Delta_d < n+1$, based on Table 3.4, we need to isolate two cases as Deletion-Insertion $DaIb$ with the error values $(a, b) = (1, 1)$ and Insertion-Deletion $IaDb$ with $(a, b) = (0, 0)$. Similarly, when $\Delta_c = 0$ and $\Delta_d \geq n + 1$, there exists two possible cases as Deletion-Insertion $DaIb$ with $(a, b) = (0, 0)$ and Insertion-deletion $IaDb$ with $(a, b) = (1, 1)$. When decoding these cases, we suggest decoding steps of **DEC.A** with two decoding algorithms, **Algorithm 1** for $DaIb$ and **Algorithm 2** for $IaDb$. For example, when $\Delta_c = 0$ and $\Delta_d < n + 1$ in Fig. 3.2, there are two possible error scenarios, $D1I1$ and $I0D0$. Then, **Algorithm 1** is used for $D1I1$ and **Algorithm 2** is for $I0D0$ simultaneously.

The following theorem shows that when the value Δ_c produces one unique correct sequence.

Theorem 3.3. *Let $\boldsymbol{y}$ be a received sequence of length n with parameters n, c, d, e and $(4n - 5)$. It is assumed that Δ_c and Δ_d are shown in Fig. 3.2. By using the constraint (3.7), the decoding of **DEC.A** gives a unique correct sequence.*

Proof. In Fig. 3.2, when Δ_c and Δ_d are given, the decoding steps of **DEC.A** with two decoding algorithms are used simultaneously as **Algorithm 1** for $DaIb$ and **Algorithm 2** for $IaDb$. From Lemma 3.8, the (i_1, j_1) is one solution of **Algorithm 1**. It is assumed that $\boldsymbol{z}$ is a recovered sequence by removing the j_1-th bit b and inserting bit a between the $(i_1 - 1)$-th and i_1-th bits in $\boldsymbol{y}$ when $i_1 < j_1$. This means that according to **Algorithm 1**, syndrome e'' of $\boldsymbol{z}$ denoted by $e''_{\boldsymbol{z}}$ equals e.

It is assumed that a pair (i_2, j_2) is a solution of **Algorithm 2** and two solutions satisfy $i_1 \neq i_2$ or $j_1 \neq j_2$. Then, $\boldsymbol{w}$ is recovered with error positions (i_2, j_2) with $j_2 \leq i_2$, and $\boldsymbol{w} \neq \boldsymbol{z}$. This means that syndrome e'' of $\boldsymbol{w}$ denoted by $e''_{\boldsymbol{w}}$ equals e.

Similar to Theorem 3.1, since two values $e''_{\boldsymbol{z}}$ and $e''_{\boldsymbol{w}}$ should be the same and equal to e, Δ_e is expressed as in (3.53). Since (i_1, j_1) is a solution, to prove that the two parallel decoding algorithms in each algorithm produce a unique solution, (i_2, j_2) cannot be a solution for the two parallel decoding algorithms. This means that (3.53) should not be satisfied.

First, the condition $i_1 \neq i_2$ or $j_1 \neq j_2$ is classified as $i_1 = j_2 < j_1 = i_2, i_1 = j_2 < j_1 < i_2, i_1 = j_2 < i_2 < j_1, i_1 < j_2 < j_1 = i_2, j_2 < i_1 < i_2 = j_1, i_1 < j_2 < j_1 < i_2, i_1 < j_2 < i_2 < j_1, j_2 < i_1 < j_1 < i_2$, and $j_2 < i_1 < i_2 < j_1$. Then, (3.53) can be expressed for the different locations of i_1, j_1, i_2, and j_2 as following

1. For $i_1 = j_2 < j_1 = i_2$, the term $\sum_{k=1}^{n} v_k(z_k - w_k)$ in (3.53) is written as

$$\begin{aligned}\sum_{k=1}^{n} v_k(z_k - w_k) =& \sum_{k=1}^{i_1-1} v_k(z_k - w_k) + \sum_{k=i_1}^{j_1} v_k(z_k - w_k) + \sum_{k=j_1+1}^{n} v_k(z_k - w_k) \\ =& 0 + \sum_{k=i_1}^{j_1} v_k(z_k - w_k) + 0 \\ =& v_{i_1}(a - y_{i_1+1}) + v_{j_1}(y_{j_1-1} - b) + \sum_{k=i_1+1}^{j_1-1} v_k(y_{k-1} - y_{k+1}).\end{aligned} \tag{3.56}$$

Since $\boldsymbol{w} \neq \boldsymbol{z}$, there is always at least one position k for $i_1 \leq k \leq j_1$ to be $z_k \neq w_k$. Due to $-v_{i_1} \leq v_{i_1}(a - y_{i_1+1}) \leq v_{i_1}$ and $v_{j_1}(y_{j_1-1} - b) \neq 0$, $-v_{i_1} - v_{j_1} \leq v_{i_1}(a - y_{i_1+1}) + v_{j_1}(y_{j_1-1} - b) \leq v_{i_1} + v_{j_1}$. Since the value v_k

is nonnegative and increasing for $i_1 \leq k \leq j_1$, the right side of (3.56) is not always zero. Thus, (3.56) is not always zero. Therefore, (3.53) is not satisfied.

2. For $i_1 = j_2 < j_1 < i_2$, the term $\sum_{k=1}^{n} v_k(z_k - w_k)$ is written as

$$\begin{aligned}\sum_{k=1}^{n} v_k(z_k - w_k) =& \sum_{k=1}^{i_1-1} v_k(z_k - w_k) + \sum_{k=i_1}^{j_1} v_k(z_k - w_k) + \sum_{k=j_1+1}^{i_2} v_k(z_k - w_k) \\ &+ \sum_{k=i_2+1}^{n} v_k(z_k - w_k) \\ =&0 + \sum_{k=i_1}^{j_1} v_k(z_k - w_k) + \sum_{k=j_1+1}^{i_2} v_k(z_k - w_k) + 0 \\ =&v_{i_1}(a - y_{i_1+1}) + v_{i_2}(y_{i_2} - b) + \sum_{k=i_1+1}^{j_1} v_k(y_{k-1} - y_{k+1}) \\ &+ \sum_{k=j_1+1}^{i_2-1} v_k(y_k - y_{k+1}). \end{aligned} \tag{3.57}$$

Since $\boldsymbol{w} \neq \boldsymbol{z}$, there is always at least one position k for $i_1 \leq k \leq i_2$ to be $z_k \neq w_k$. Since $y_{i_2} \neq b$, $v_{i_2}(y_{i_2} - b) \neq 0$. Based on Lemma 3.11, the right side of (3.57) is not always zero since v_k for $j_1 + 1 \leq k \leq i_2$ is always larger than v_k for $i_1 \leq k \leq j_1$. Thus, (3.53) is not satisfied.

3. Similarly, for the remaining cases as $i_1 = j_2 < i_2 < j_1, i_1 < j_2 < j_1 = i_2, j_2 < i_1 < i_2 = j_1, i_1 < j_2 < j_1 < i_2, i_1 < j_2 < i_2 < j_1, j_2 < i_1 < j_1 < i_2$, and $j_2 < i_1 < i_2 < j_1$, the proof steps are similar to the ones for $i_1 = j_2 < j_1 < i_2$. Therefore, specific proof steps are omitted. For these cases, (3.53) is not also satisfied.

From 1) to 3), any pair (i_2, j_2) with $i_2 \neq i_1$ or $j_2 \neq j_1$ cannot give a solution of the two parallel decoding algorithms in each algorithm. Therefore, two parallel decoding

Table 3.5: Lower bound of cardinality, the redundancy, and the code rate of the proposed code for some given values n.

Length n	**Lower bound of cardinality**	**Redundancy** (bits)	**Code rate**
100	5.296e+24	19	0.810
150	2.648e+39	20	0.867
200	1.676e+54	21	0.895
250	1.207e+69	21	0.916
300	9.439e+83	22	0.927
1000	4.466e+293	25	0.975

algorithms produce a unique correct sequence as $\boldsymbol{z}$ from the received sequence $\boldsymbol{y}$. □

3.4.5 Cardinality analysis, redundancy, and results comparisons of the proposed code

Let $M(n,2)$ be the cardinality of the proposed code of length n, with a maximum possible number of codewords, which can correct one deletion and one insertion error. The lower bound of the cardinality is determined by the potential values of the weight, accumulation value, and syndrome in the code construction. Hence, by applying $0 \leq c \leq 2$, $0 \leq d \leq 2n+1$, and $0 \leq e \leq 4n-6$, we can obtain the lower bound for the cardinality $M(n,2)$ of the proposed code as

$$M(n,2) \geq \frac{2^n}{3(2n+2)(4n-5)}.$$

Table 3.6: Comparisons of the referenced codes and our proposed code.

	Number m of errors	**Length of the received sequence**	**Redundancy**
Levenshtein [19]	1 indel	$n-1$ or $n+1$	$\log_2(n+1)$
	1 edit	$n+1$, $n+1$, or n	$\log_2 n+1$
Gabrys *et al.*[24]	2 deletions	$n-2$	$8\log_2 n+O(\log_2\log_2 n)$
Guruswami *et al.* [25]	2 deletions	$n-2$	$4\log_2 n+O(\log_2\log_2 n)$
			$3\log_2 n+O(\log_2\log_2 n)$
Brakensiek *et al.* [28]	$m\geq 2$ deletions	$n-m$ or $n+m$	$O(m^2\log_2 m)\log_2 n$
Helberg *et al.* [29]	$m\geq 2$ indel	$n-m$ or $n+m$	–
A.-Ghaffar *et al.* [30]	$m\geq 2$ indel	$n-m$ or $n+m$	–
Proposed code	1 deletion + 1 insertion	n	$2\log_2 n+5$

The redundancy of the proposed code can be estimated as

$$n-\log_2|C(n,c,d,e)|\leq n-\log_2\frac{2^n}{3(2n+2)(4n-5)}\approx 2\log_2 n+5.$$

Accordingly, Table 3.5 presents the lower bound of cardinality, the redundancy, and the code rate of the proposed code $C(n,c,d,e)$ for some finite values of length n.

Table 3.6 compares the referenced codes and proposed code. The referenced codes [19, 24, 25, 28–30] focused solely on specific error types such as indel or edit errors, possibly overlooking certain error combinations. For instance, if the length of the received sequence matches that of the codeword, errors might go undetected as the decoder assumes no error. Moreover, in codes using the generated VT code, such as those mentioned in [29, 30], determining redundancies becomes challenging due to exponential increases in coefficient sequences. In contrast, the proposed code can directly ascertain redundancy through three constraints, offering enhanced redundancy compared to the referenced codes.

Chapter 4

Nonbinary code design for one edit or one adjacent transposition error

4.1 Introduction

In spite of many challenges of involved correction code designs [34, 35], a lot of efforts have focused on designing efficient and robust approaches to cope with a single deletion, or insertion error (known as an indel error), a single deletion, insertion, or substitution error (called an edit error), or adjacent transposition error. For instance, in 1965, a binary VT code [18] was first presented to correct a single indel error by using syndrome calculation. Shortly, based on the VT code, Levenshtein proposed a binary code (known e-VT code) that can correct a single edit error in [19]. Later, a q-ary code ($q > 2$) that can handle one indel error was presented in [20] by employing the VT code and a mapping function from q-ary sequence to binary sequence. From the inspiration of the Levenshtein code, the authors of [36, 37] designed a quaternary code to cope with a single edit error by using array-type codewords with several Levenshetein codes [19]. Nonetheless, these codes were limited to addressing a single indel or edit error. Moreover, the codes had an inherent issue of being unable to correct errors that maintain the received sequence length such as substitution or transposition errors.

A transposition error is defined as two or more symbols being swapped leading to a significant effect on communication and storage systems. Especially, a single adjacent transposition error can be seen as a distinct scenario where two adjacent substitution errors occur, and these adjacent values must be different. For instance, the adjacent transposition errors can cause distortion, noise, or interference in the received signals, and reduce the quality of service and performance of the wireless, optical, or molecular systems [38,39]. In storage systems, the adjacent transposition errors can depress the integrity and security of data stored in flash memories [40], or cause strand breakage, data corruption, and loss, leading to reduce the storage density and efficiency of DNA-based data storage [41, 42]. To cope with the adjacent transposition errors, the binary code constructions in [41, 42] were studied to deal with a single deletion or adjacent transposition error by various approaches, whereas the codes in [41] used a generalization of the Levenshtein codes. Besides that, by employing the hash functions, VT sketches, and markers, the redundancies of the codes in [42] were improved than those in [41]. Nonetheless, the codes presented in [41, 42] could not rectify substitution errors in cases where the length of the received sequence is unchanged. This limitation arises due to several following reasons. Scenarios involving substitution errors and transposition errors in [41] could not be reliably differentiated based on the syndromes. In [42], substitution errors might alter the markers, consequently causing an inaccurate determination of error positions.

In this chapter, our purpose is to design codes capable of correcting a single deletion, insertion, substitution, or adjacent transposition error that might arise within a codeword. In other words, we propose a novel code design to correct a single edit or adjacent transposition error. When a deletion or insertion error occurs over the channel, the decoder finds whether a deletion or insertion error occurs based on the length of the received sequence, then the decoder can correct the error. In scenarios where the received sequences have the same length as the original codewords, the proposed code can detect either a substitution or adjacent transposition error. This has the potential to significantly enhance the efficiency of information synchronization in communication systems or the retrieval process in storage systems. Such codes can give rise to numerous applications in synchronizing information across communication and storage contexts [14, 33, 35].

Motivated by the above reasons, we investigate the q-ary ($q \geq 2$) scheme and correcting one edit or adjacent transposition error in a codeword. The proposed q-ary code design is constructed by three constraints. The proposed code can address the inability of codes to correct an edit error in [18, 20, 41, 42]; an adjacent transposition error in [18–20, 36, 37]. To the authors' knowledge, our work is the first try for q-ary code design to tackle one edit or adjacent transposition error. According to the code design, the strategy to deal with errors includes two main steps: i) to isolate the error scenarios based on the length of the received sequence and the first constraint, and ii) to find the error values or the error value difference based on the first two constraints and error positions by the last constraint. Furthermore, to verify the error-correcting capacity of the proposed code, a comprehensive mathematical methodology is specifically provided. Furthermore, a specific decoding procedure for the proposed code that can deal with all error scenarios is also suggested.

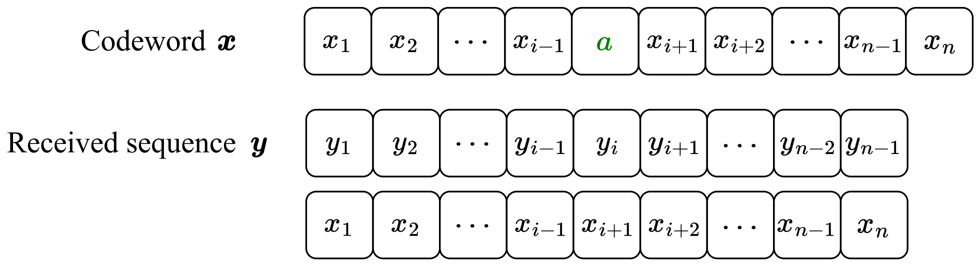
4.2 Preliminaries

A codeword $\boldsymbol{x} \in \mathbb{F}_q^n$ is given as $\boldsymbol{x} = (x_1, x_2, \cdots, x_n)$ and $p = q - 1$. A q-ary sequence $\boldsymbol{y} = (y_1, y_2, \cdots, y_{n'}) \in \mathbb{F}_q^{n'}$ is denoted as the received sequence of $\boldsymbol{x}$ after a single deletion, insertion, substitution, or adjacent transposition error occurring in $\boldsymbol{x}$. To outline the channel model, we provide the error scenarios via Fig. 4.1 and mathematical presentations as follows.

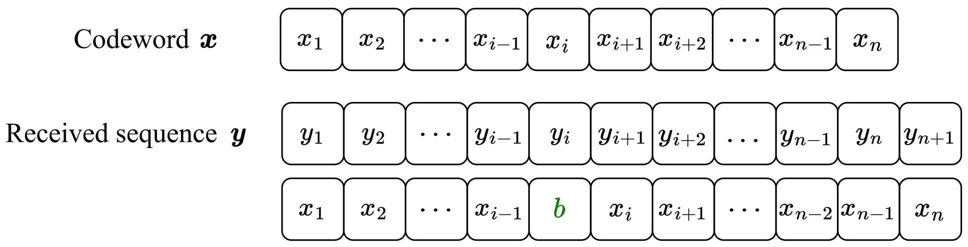
When a deletion error of value $0 \le a \le p$ occurs in the i-th position of $\boldsymbol{x}$ for $1 \le i \le n$, this error scenario is denoted as Da. Thus, the relation between $\boldsymbol{y} \in \mathbb{F}_q^{n-1}$ and $\boldsymbol{x}$ for Da can be expressed by mathematical presentation as

$$y_k = \begin{cases} y_k & \text{for } 1 \le k \le i-1, \\ y_{k+1} & \text{for } i \le k \le n-1. \end{cases} \tag{4.1}$$

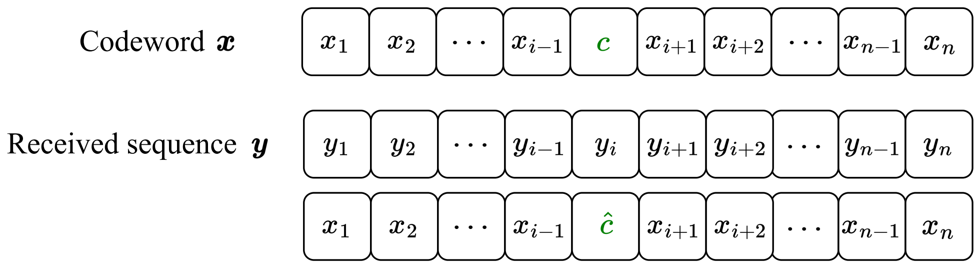
Conversely, Ib denotes the insertion error scenario, where a value $0 \le b \le p$ is inserted in the i-th position of $\boldsymbol{x}$ for $1 \le i \le n+1$. Then, the element x'_k of $\boldsymbol{y} \in \mathbb{F}_q^{n+1}$


Codeword $\boldsymbol{x}$
x_1 x_2 $\cdots$ x_{i-1} a x_{i+1} x_{i+2} $\cdots$ x_{n-1} x_n
Received sequence $\boldsymbol{y}$
y_1 y_2 $\cdots$ y_{i-1} y_i y_{i+1} $\cdots$ y_{n-2} y_{n-1}
x_1 x_2 $\cdots$ x_{i-1} x_{i+1} x_{i+2} $\cdots$ x_{n-1} x_n

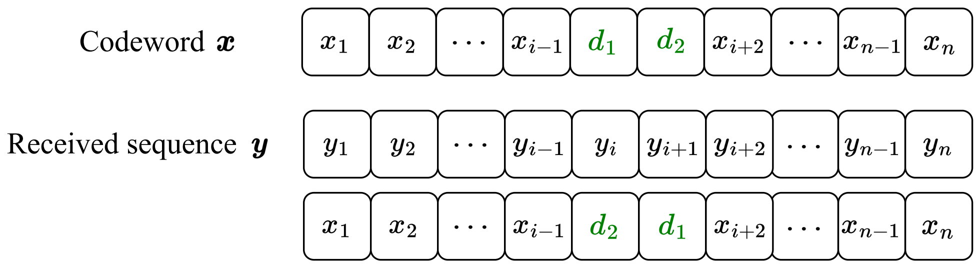
(a) Da scenario.


Codeword $\boldsymbol{x}$
x_1 x_2 $\cdots$ x_{i-1} x_i x_{i+1} x_{i+2} $\cdots$ x_{n-1} x_n
Received sequence $\boldsymbol{y}$
y_1 y_2 $\cdots$ y_{i-1} y_i y_{i+1} y_{i+2} $\cdots$ y_{n-1} y_n y_{n+1}
x_1 x_2 $\cdots$ x_{i-1} b x_i x_{i+1} $\cdots$ x_{n-2} x_{n-1} x_n

(b) Ib scenario.


Codeword $\boldsymbol{x}$
x_1 x_2 $\cdots$ x_{i-1} c x_{i+1} x_{i+2} $\cdots$ x_{n-1} x_n
Received sequence $\boldsymbol{y}$
y_1 y_2 $\cdots$ y_{i-1} y_i y_{i+1} y_{i+2} $\cdots$ y_{n-1} y_n
x_1 x_2 $\cdots$ x_{i-1} $\hat{c}$ x_{i+1} x_{i+2} $\cdots$ x_{n-1} x_n

(c) S$\hat{c}$ scenario.


Codeword $\boldsymbol{x}$
x_1 x_2 $\cdots$ x_{i-1} d_1 d_2 x_{i+2} $\cdots$ x_{n-1} x_n
Received sequence $\boldsymbol{y}$
y_1 y_2 $\cdots$ y_{i-1} y_i y_{i+1} y_{i+2} $\cdots$ y_{n-1} y_n
x_1 x_2 $\cdots$ x_{i-1} d_2 d_1 x_{i+2} $\cdots$ x_{n-1} x_n

(d) Td_2d_1 scenario.

Figure 4.1: Received sequence $\boldsymbol{y}$ for Da, Ib, S$\hat{c}$, and Td_2d_1 scenarios.

is presented via $\boldsymbol{x}$ as

$$x'_k = \begin{cases} x_k & \text{for } 1 \leq k \leq i-1, \\ b & \text{for } k = i, \\ x_{k-1} & \text{for } i+1 \leq k \leq n+1. \end{cases} \tag{4.2}$$

When one symbol of value $0 \leq c \leq p$ in the i-th position in $\boldsymbol{x}$ is replaced by a symbol of value $0 \leq \hat{c} \leq p$ and $c \neq \hat{c}$, and is defined as S$\hat{c}$. Similarly, the codeword $\boldsymbol{x}$ and received sequence $\boldsymbol{y} \in \mathbb{F}_q^n$ for S$\hat{c}$ are presented as

$$x'_k = \begin{cases} x_k & \text{for } 1 \leq k \leq i-1 \text{ and } i+1 \leq k \leq n, \\ \hat{c} & \text{for } k = i. \end{cases} \tag{4.3}$$

When two adjacent symbols of the respective values d_1 and d_2 at the i-th and $(i+1)$-th positions in $\boldsymbol{x}$ are transposed, where $0 \leq d_1, d_2 \leq p$ and $d_1 \neq d_2$ for $1 \leq i \leq n-1$, a adjacent transposition error occurs and is denoted as Td_2d_1. For convenience, the transposition error stands for the adjacent transposition error during the remaining parts of this work. The relation between the received sequence $\boldsymbol{y} \in \mathbb{F}_q^n$ and $\boldsymbol{x}$ for Td_2d_1 can be expressed as

$$x'_k = \begin{cases} x_k & \text{for } 1 \leq k \leq i-1 \text{ and } i+2 \leq k \leq n, \\ d_2 & \text{for } k = i, \\ d_1 & \text{for } k = i+1. \end{cases} \tag{4.4}$$

4.3 Proposed nonbinary code design

4.3.1 Code design

Definition 4.1. Let $C(q,n)$ be the proposed q-ary code of length n, with parameters e, f, and s, where $0 \leq e, f \leq p$, $0 \leq s \leq 2pn^2$, and $p = q-1$. The proposed code $C(q,n)$ is designed to address one deletion, insertion, substitution, or transposition error as

$$C(q,n) \triangleq \Bigg\{ \boldsymbol{x} \in \mathbb{F}_q^n : \sum_{k=1}^{n} x_k \equiv e \bmod q, \tag{4.5}$$

$$\sum_{k=1}^{\lfloor n/2 \rfloor} x_{2k} \equiv f \bmod q, \tag{4.6}$$

$$\sum_{k=1}^{n} k^2 x_k \equiv s \bmod (2pn^2+1) \Bigg\}. \tag{4.7}$$

For example, a sequence $\boldsymbol{x}$ =(2,2,1,2,0,1,2,0,3,3) can be one of the codewords in a quaternary code $C(4,10)$ with $q=4, n=10, e=0, f=0$, and $s=127$, since $\boldsymbol{x}$ satisfies three constraints (4.5)–(4.7). Particularly, $\sum_{k=1}^{10} x_k \bmod 3 = 16 \bmod 4 = 0$ and $\sum_{k=1}^{\lfloor 10/2 \rfloor} x_{2k} \bmod 4 = 8 \bmod 4 = 0$ satisfy (4.5) and (4.6), respectively. Moreover, $\boldsymbol{x}$ has the syndrome $\sum_{k=1}^{n} k^2 x_k \bmod 601 = 127$.

The values e and f in (4.5) and (4.6) present the weight of $\boldsymbol{x}$ and the weight of the even indexes in $\boldsymbol{x}$, respectively. Since the proposed code $C(q,n)$ focuses on the q-ary code, the modulo values in (4.5) and (4.6) are set as q. The constraint (4.7) presents the syndrome of $\boldsymbol{x}$ with high-order coefficients. Thus, our strategy is first to identify the error scenarios according to the length of the received sequence and (4.5). Then, the error values for Da and Ib scenarios are identified by (4.5). The difference between c and $\hat{c}$ in S$\hat{c}$ is given by (4.5), and the difference between d_2 and d_1 in Td_2d_1 is determined by (4.6). Finally, based on the information about the error values, the constraint (4.7) produces the error positions and corrects the codewords. Detailed descriptions of the three constraints in the proposed code $C(q, n)$ and our decoding method will be provided in the next subsections.

4.3.2 Rules of constraints

As mentioned in subsection 4.3.1, we first reduce the error scenarios among the four scenarios according to the length of the received sequence and the constraint (4.5). In addition, the constraints (4.5) and (4.6) give information about the error value or the difference of error values. Based on this information, the constraint (4.7) identifies the error positions and corrects the errors. In this subsection, we also thoroughly explain how the proposed code can distinguish and correct a substitution error and a transposition error when the length of the received sequence is the same as the codeword length.

Let $\boldsymbol{y} \in \mathbb{F}_q^{n'}$ be the received sequence after one edit or transposition error occurs in a codeword $\boldsymbol{x} \in C(q, n)$. Three parameters e', f', and s' for $\boldsymbol{y}$ are calculated as

$$\begin{aligned} e' &\equiv \sum_{k=1}^{n} y_k \bmod q, \\ f' &\equiv \sum_{k=1}^{\lfloor n/2 \rfloor} y_{2k} \bmod q, \\ s' &\equiv \sum_{k=1}^{n} k^2 y_k \bmod 2pn^2 + 1. \end{aligned}$$

Since the values e, f, and s are known in the decoder, the differences between the

Table 4.1: Determining error scenarios by n' and Δ_e.

Length n'	Δ_e	**Error scenario**
$n+1$	–	Da
$n-1$	–	Ib
n	$\neq 0$	S$\hat{c}$
	0	Td_2d_1

parameters of $\boldsymbol{x}$ and $\boldsymbol{y}$ are given as

$$\Delta_e \equiv e - e' \bmod q, \tag{4.8}$$

$$\Delta_f \equiv f - f' \bmod q, \tag{4.9}$$

$$\Delta_s \equiv s - s' \bmod 2pn^2 + 1. \tag{4.10}$$

When decoding, if $n' = n-1$ or $n' = n+1$, inferring to Da or Ib scenario, respectively, the error values are determined by (4.8) as $a = \Delta_e$ or $b = -\Delta_e \bmod q$, respectively. Then, from Δ_s in (4.10), the error position can be determined, and the received sequence will be corrected.

If $n' = n$, S$\hat{c}$ or Td_2d_1 scenario can be a solution. Since the weight of the sequence is not affected by the transposition error, Δ_e for Td_2d_1 scenario is given as $\Delta_e = 0$. Since $c \neq \hat{c}$, Δ_e for S$\hat{c}$ cannot be zero as $\Delta_e \neq 0$. Therefore, if $n' = n$ and $\Delta_e \neq 0$, the decoder knows that S$\hat{c}$ scenario occurs. If $n' = n$ and $\Delta_e = 0$, Td_2d_1 scenario is determined. Table 4.1 summarizes that four error scenarios, Da, Ib, S$\hat{c}$, and Td_2d_1, can be distinguished by the length n' of the received sequence and Δ_e.

As mentioned in Section 4.3.1, the constraint (4.7) is designed to produce the error position and correct the sequence. We provide Lemma 4.1, which calculates the differences in syndromes between the codeword and the received sequence for each scenario. From these differences, the reason for choosing $2pn^2+1$ as the modulo value in (4.7) can be clarified. Furthermore, Lemma 4.1 also offers valuable insights for identifying error positions for each error scenario.

Lemma 4.1. *Let $\boldsymbol{x}$ and $\boldsymbol{y}$ be a codeword in $C(q,n)$ and a received sequence of length n' by deleting a symbol, inserting a symbol, replacing a symbol, or transposing two adjacent symbols in $\boldsymbol{x}$, respectively. Then, the syndrome difference in* (4.7) *between $\boldsymbol{x}$ and $\boldsymbol{y}$ is bounded as*

$$\left|\sum_{k=1}^{n} k^2 x_k - \sum_{k=1}^{n'} k^2 y_k\right| \le 2pn^2 + 1. \tag{4.11}$$

Proof. Let R be $R = \sum_{k=1}^{n} k^2 x_k - \sum_{k=1}^{n'} k^2 y_k$ in (4.11). To prove (4.11), we need to investigate the syndrome difference for Da, Ib, S$\hat{c}$, and Td_2d_1 scenarios as follows.

For Da scenario, since $n' = n-1$, $R = \sum_{k=1}^{n} k^2 x_k - \sum_{k=1}^{n-1} k^2 y_k$. From (4.1), R can be expressed as

$$\begin{aligned} R &= \sum_{k=1}^{i-1} k^2 y_k + i^2 a + \sum_{k=i+1}^{n} k^2 y_{k-1} - \sum_{k=1}^{i-1} k^2 y_k - \sum_{k=i}^{n-1} k^2 y_k \\ &= i^2 a + \sum_{k=i+1}^{n} k^2 y_{k-1} - \sum_{k=i}^{n-1} k^2 y_k \\ &= i^2 a + \sum_{k=i}^{n-1} y_k((k+1)^2 - k^2) \\ &= i^2 a + \sum_{k=i}^{n-1} y_k(2k+1). \end{aligned} \tag{4.12}$$

Since $1 \le a \le p$ and $1 \le y_k \le p$ are q-ary symbols for $i \le k \le n-1$, R in (4.12) is bounded as

$$
\begin{aligned}
&0 \leq R \leq pi^2 + p\sum_{k=i}^{n-1}(2k+1),\\
&0 \leq R \leq pi^2 + p(n-i) + p(n^2 - n - i^2 + i),\\
&0 \leq R \leq pn^2. \qquad (4.13)
\end{aligned}
$$

From (4.13), $\left|\sum_{k=1}^{n} k^2 x_k - \sum_{k=1}^{n-1} k^2 y_k\right| < 2pn^2 + 1$ and (4.11) is satisfied.

For Ib scenario with $n' = n+1$, $R = \sum_{k=1}^{n} k^2 x_k - \sum_{k=1}^{n+1} k^2 y_k$. From (4.2), R can be written as

$$
\begin{aligned}
R &= \sum_{k=1}^{i-1} k^2 y_k + \sum_{k=i}^{n} k^2 y_{k+1} - \sum_{k=1}^{i-1} k^2 y_k - i^2 b - \sum_{k=i+1}^{n+1} k^2 y_k\\
&= \sum_{k=i}^{n} k^2 y_{k+1} - i^2 b - \sum_{k=i+1}^{n+1} k^2 y_k\\
&= -i^2 b + \sum_{k=i}^{n} y_{k+1}(k^2 - (k+1)^2)\\
&= -i^2 b - \sum_{k=i}^{n} y_{k+1}(2k+1). \qquad (4.14)
\end{aligned}
$$

Since $1 \leq b \leq p$ and $1 \leq y_{k+1} \leq p$ for $i \leq k \leq n$, the boundaries of (4.14) are given by

$$
\begin{aligned}
-pi^2 - p\sum_{k=i}^{n}(2k+1) \leq R \leq 0,&\\
-p(n+1)^2 \leq R \leq 0.& \qquad (4.15)
\end{aligned}
$$

From (4.15), $\left|\sum_{k=1}^{n} k^2 y_k - \sum_{k=1}^{n+1} k^2 y_k\right| < 2pn^2 + 1$ and (4.11) is satisfied.

For S$\hat{c}$ scenario with $n' = n$, R is expressed as $R = \sum_{k=1}^{n} k^2 x_k - \sum_{k=1}^{n} k^2 y_k$.

Based on (4.3), R is represented as

$$\begin{aligned} R &= \sum_{k=1}^{i-1} k^2 y_k + i^2 c + \sum_{k=i}^{n} k^2 y_k - \sum_{k=1}^{i-1} k^2 y_k - i^2 \hat{c} - \sum_{k=i+1}^{n} k^2 y_k \\ &= i^2 c - i^2 \hat{c} \\ &= i^2 (c - \hat{c}). \end{aligned} \tag{4.16}$$

Since $c \neq \hat{c}$ and $1 \leq i \leq n$, (4.16) has bounds as

$$\begin{aligned} -pi^2 \leq R \leq pi^2, \\ -pn^2 \leq R \leq pn^2. \end{aligned} \tag{4.17}$$

Based on (4.17), $\left|\sum_{k=1}^{n} k^2 y_k - \sum_{k=1}^{n} k^2 y'_k\right| \leq 2pn^2 + 1$ and (4.11) is satisfied.

For T$d_2 d_1$ scenario with $n' = n$, R is rewritten as $R = \sum_{k=1}^{n} k^2 x_k - \sum_{k=1}^{n} k^2 y_k$. Based on (4.4), R is represented as

$$\begin{aligned} R &= \sum_{k=1}^{i-1} k^2 y_k + i^2 x_i + (i+1)^2 x_{i+1} + \sum_{k=i+2}^{n} k^2 y_k - \sum_{k=1}^{i+1} k^2 y_k - \sum_{k=i+2}^{n} k^2 y_k \\ &= i^2 x_i + (i+1)^2 x_{i+1} - i^2 y_i - (i+1)^2 y_{i+1} \\ &= (2i+1) y_i - (2i+1) y_{i+1} \\ &= (2i+1) d_2 - (2i+1) d_1 \\ &= (2i+1)(d_2 - d_1). \end{aligned} \tag{4.18}$$

Since $d_1 \neq d_2$ and $1 \leq i \leq n-1$, (4.18) has bounds as

$$\begin{aligned} -p(2i+1) \leq R \leq p(2i+1), \\ -p(2n-1) \leq R \leq p(2n-1). \end{aligned} \tag{4.19}$$

According to (4.19), $\left|\sum_{k=1}^{n} k^2 x_k - \sum_{k=1}^{n} k^2 y_k\right| < 2pn^2 + 1$ and (4.11) is satisfied. $\square$

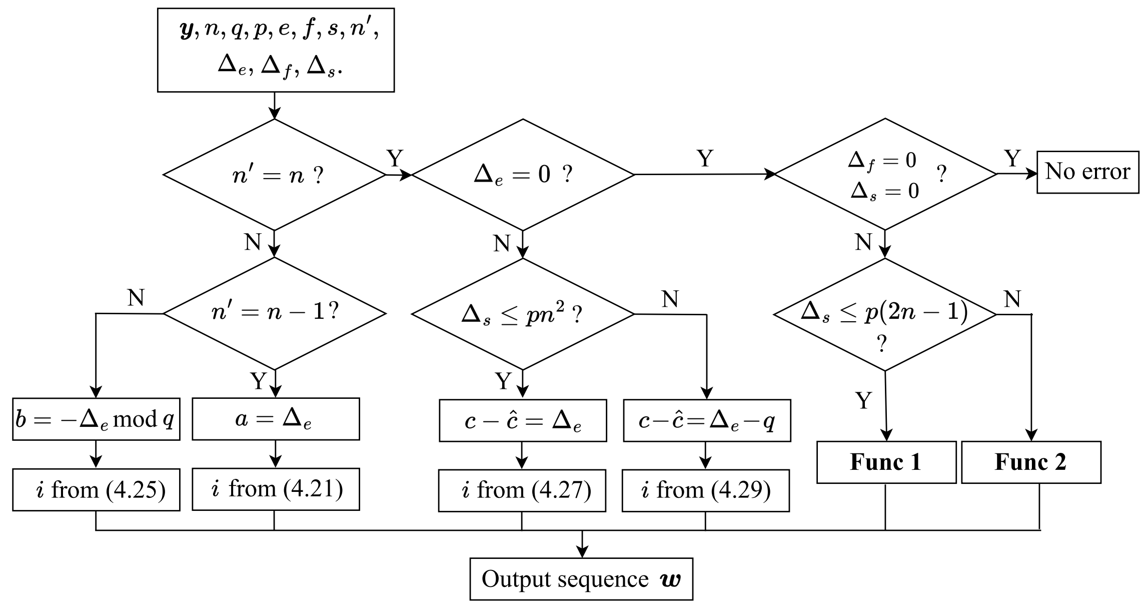


Figure 4.2: Overall decoding procedure of the proposed code $C(q, n)$.

4.4 Decoding of the proposed code design

The overall decoding procedure for the proposed code $C(q, n)$ is provided in Fig. 4.2. The decoding strategy consists of two main steps: 1) classify the error scenarios and 2) determine error value and position then correct the sequence. It is noted that if $n' = n$ and three parameters, Δ_e, Δ_f, and Δ_s, are all zeros, $\boldsymbol{y}$ is regarded to be same as the codeword $\boldsymbol{x}$.

4.4.1 Decoding for a deletion error

If $n' = n - 1$, inferring to Da scenario occurs, the deleted value a is determined as $a = \Delta_e$. As mentioned in [43], decoding for deletion-correcting codes can not be always successful in determining the exact location of the deleted symbol. If a codeword with the large runs is sent and one deletion occurs in a large run, multiple candidates of the deleted position can be detected. However, the candidates of the error position can be reduced for effective decoding and the received sequence is still corrected successfully. In this work, we prioritize choosing the first index at the runs as the candidate for the deleted position. Besides that, based on (4.12) and (4.13), Δ_s in (4.10) for Da can be presented as

$$\Delta_s = i^2 a + \sum_{k=i}^{n-1} y_k(2k+1). \tag{4.20}$$

Let a set D include the values $1 \leq l \leq n$ satisfying Δ_s in (4.20), which are candidates for the deleted position i. Then, the set D is expressed as

$$D = \{l \mid \Delta_s = l^2 a + \sum_{k=l}^{n-1} y_k(2k+1), y_{l-1} \neq a, \text{ for } 1 \leq l \leq n\}. \tag{4.21}$$

Plugging a symbol a into the l-th position in $\boldsymbol{y}$, the output sequence $\boldsymbol{w} = (y_1, y_2, \cdots, y_{l-1}, a, y_l, \cdots, y_{n-1})$ is determined and the decoding is finished.

From (4.22), multiple candidates that belong to the same run are removed and only the first element in each run is checked. However, there may be multiple candidates in different runs, which can be answered for the constraints. This means that unique decoding is not guaranteed. To solve this problem, we provide Lemma 4.2 to demonstrate that the proposed code design can produce the unique decoding for Da scenario.

Lemma 4.2. *For decoding of Da, with a received sequence $\boldsymbol{y}$ and the given parameters n, e, f, and s, the proposed decoding can output a unique correct sequence by* (4.7).

Proof. Let i_1 be one correct solution for error position from (4.24). Inserting a symbol of value a into the i_1-th position in $\boldsymbol{y}$, a sequence $\boldsymbol{w}$ is assumed to be reconstructed. The syndrome of $\boldsymbol{w}$ is $s_{\boldsymbol{w}} = \sum_{k=1}^{n} k^2 w_k \bmod 2pn^2+1$ and $s_{\boldsymbol{w}}$ equals s.

Suppose that i_2 is also another solution for error position from (4.21) and i_2 and i_1 should not be in the same run as $r_{i_1} \neq r_{i_2}$. Then, let $\boldsymbol{z}$ be recovered from i_2 and $\boldsymbol{z} \neq \boldsymbol{w}$. Thus, the syndrome $s_{\boldsymbol{z}}$ of $\boldsymbol{z}$ is $s_{\boldsymbol{z}} = \sum_{k=1}^{n} k^2 z_k \bmod 2pn^2+1 = s$. Since two values $s_{\boldsymbol{w}}$ and $s_{\boldsymbol{z}}$ should be the same and equal to s, Δ_s'' is given as

$$\Delta_s'' \equiv \sum_{k=1}^{n} k^2(w_k - z_k) \bmod 2pn^2+1 = 0. \tag{4.22}$$

To prove that the constraint (4.7) produces a unique output sequence, i_2 cannot be a solution for (4.21). As a result, (4.22) should not be satisfied.

The condition $r_{i_1} \neq r_{i_2}$ is classified into two cases as $i_1 < i_2$ and $i_1 > i_2$. Let Q be $Q = \sum_{k=1}^{n} k^2(w_k - z_k)$ in (4.22). For $i_1 < i_2$, Q is expressed as

$$\begin{aligned} Q &= \sum_{k=1}^{i_1-1} k^2(w_k - z_k) + \sum_{k=i_1}^{i_2} k^2(w_k - z_k) + \sum_{k=i_2+1}^{n} k^2(w_k - z_k) \\ &= a(i_1^2 - i_2^2) + \sum_{k=i_1}^{i_2-1} y_k(2k+1) \\ &= \sum_{k=i_1}^{i_2-1} (2k+1)(y_k - a). \end{aligned} \tag{4.23}$$

To satisfy (4.22), (4.23) should be zero. As the term $(2k+1)$ in (4.23) is positive and increasing for $i_1 \leq k \leq i_2 - 1$, (4.23) becomes zero if $y_k = a$ for $\forall k \in [i_1, i_2 - 1]$. Note that as $y_{i_2-1} \neq a$ due to (4.23), (4.23) cannot be zero and (4.22) is not satisfied.

For $i_1 > i_2$, Q is given as $Q = \sum_{k=i_2}^{i_1-1} (2k+1)(y_k - a)$. The proof steps of

$\sum_{k=i_2}^{i_1-1}(2k+1)(y_k - a) \bmod 2pn^2+1 \neq 0$ are similar to those for case $i_1 < i_2$, and present that (4.22) is also invalid. $\square$

Example 4.1. (*Continue the example in Section 4.3.1*) The given parameters are $n = 10$, $e = 0$, $f = 0$, and $s = 127$. It is assumed that the received sequence $\boldsymbol{y} = (2, 2, 1, \_0, 1, 2, 0, 3, 3) \in \mathbb{F}_4^9$, where notation '_' indicates the deleted symbol. The parameters e', f', and s' of $\boldsymbol{y}$ are $e' = 2, f' = 3$, and $s' = 551$, respectively. Then, $\Delta_e = 2, \Delta_f = 1$, and $\Delta_s = 177$.

Step 1: clarifying the error scenario.

Since $n' = n - 1 = 9$, Da scenario occurs.

Step 2: determining error value and position, then correcting sequence.

- The deleted value is determined as $a = \Delta_e = 2$.
- A set D includes the candidates $1 \leq l \leq n$ for the deleted position and is determined by (4.21) as $D = \{4\}$. Thus, the deleted position is the fourth position in $\boldsymbol{y}$.
- The correct sequence $\boldsymbol{w}$ is given by adding a symbol '2' between the third and the fourth positions in $\boldsymbol{y}$ as $\boldsymbol{w} = (2, 2, 1, 2, 0, 1, 2, 0, 3, 3)$.

4.4.2 Decoding for an insertion error

When $n' = n + 1$, inferring to Ib scenario, the inserted symbol is determined as $b = -\Delta_e \bmod q$. Since the decoding steps for Ib scenario are similar to those for Da scenario, we only provide the step to determine the candidates for the inserted position instead of presenting detailed explanations for each step to correct Ib scenario.

Moreover, from (4.14) and (4.15), Δ_s in (4.10) is expressed as

$$\Delta_s = -i^2 b - \sum_{k=i}^{n} y_{k+1}(2k+1) + 2pn^2 + 1. \tag{4.24}$$

Since the inserted value b is known via Δ_e, the candidates for the inserted position i are defined from Δ_s in (4.24) as

$$I = \{l \mid \Delta_s = -l^2 b - \sum_{k=l}^{n} y_{k+1}(2k+1) + 2pn^2 + 1, y_l = b, y_{l-1} \neq b, \quad \text{for } 1 \leq l \leq n+1\}. \tag{4.25}$$

Similar to Da scenario, a unique candidate l in (4.7) satisfying Δ_s is identified. The output sequence $\boldsymbol{w} = (y_1, y_2, \cdots, y_{l-1}, y_{l+1}, y_{l+2}, \cdots, y_{n+1})$ is determined by deleting the l-th symbol in $\boldsymbol{y}$.

Example 4.2. (*Continue the example in Section 4.3.1*) The given parameters are $n = 10$, $e = 0$, $f = 0$, and $s = 127$. It is assumed that the received sequence $\boldsymbol{y} = (2, \mathbf{3}, 2, 1, 2, 0, 1, 2, 0, 3, 3) \in \mathbb{F}_4^{11}$, where the bold symbol '**3**' indicates the inserted symbol. The parameters e', f', and s' of $\boldsymbol{y}$ are $e' = 3, f' = 1$, and $s' = 337$, respectively. Then, $\Delta_e = 1, \Delta_f = 3$, and $\Delta_s = 391$.

Step 1: clarifying the error scenario.

Since $n' = n + 1 = 11$, Ib scenario occurs.

Step 2: determining error value and position, then correcting sequence.

- The inserted value is determined as $b = -\Delta_e \bmod 4 = 1$.

- A set I consists of the candidates $1 \leq l \leq n+1$ for the inserted position and is verified by (4.25) as $I = \{2\}$. Thus, the inserted position is the second position in $\boldsymbol{y}$.
- The correct sequence $\boldsymbol{w}$ is produced by removing a symbol '**3**' in the second position in $\boldsymbol{y}$ as $\boldsymbol{w} = (2,2,1,2,0,1,2,0,3,3)$.

4.4.3 Decoding for a substitution error

If $n' = n$ and $\Delta_e \neq 0$, the decoder infers that S$\hat{c}$ occurs as shown in Table 4.1. For S$\hat{c}$ scenario, the weight difference between $\boldsymbol{x}$ and $\boldsymbol{y}$ corresponds to the difference between c in $\boldsymbol{x}$ and $\hat{c}$ in $\boldsymbol{y}$, inferring to $(c-\hat{c}) \bmod q = \Delta_e$. Moreover, the condition $c \neq \hat{c}$ can be classified into two cases as $c > \hat{c}$ and $c < \hat{c}$. Thus, based on (4.16) and (4.17), and $(c-\hat{c}) \bmod q = \Delta_e$, Δ_s in (4.10) is represented as below.

- If $c > \hat{c}$, meaning $(c-\hat{c}) > 0$, the term $(c-\hat{c})$ in (4.16) is given as $(c-\hat{c}) = \Delta_e$. Then, Δ_s can be given as

  $$\Delta_s = i^2 \Delta_e. \tag{4.26}$$

  Since $0 < \Delta_e \leq p$ and $1 \leq i \leq n$, the bounds of Δ_s in (4.26) are determined as

  $$0 < \Delta_s \leq pn^2.$$

  Thus, if S$\hat{c}$ is determined and $\Delta_s \leq pn^2$, the error position i is calculated for $1 \leq i \leq n$ by (4.26) as

  $$i = \sqrt{\frac{\Delta_s}{\Delta_e}}. \tag{4.27}$$

From (4.27), the determined value i is the unique value. Moreover, based on (4.27), $c - \hat{c} = \Delta_e$, then the i-th position of value $\hat{c}$ in $\boldsymbol{y}$ is replaced by a symbol of value $(\hat{c} + \Delta_e)$. Then, the output sequence $\boldsymbol{w}$ is given as $\boldsymbol{w} = (y_1, y_2, \cdots, y_{i-1}, (\hat{c} + \Delta_e), y_{i+1}, \cdots, y_n)$, and the decoding is finished.

- If $c < \hat{c}$, corresponding to $c - \hat{c} < 0$, the term $(c - \hat{c})$ in (4.16) is given as $c - \hat{c} = \Delta_e - q$. Then, Δ_s can be given as

$$\Delta_s = i^2(\Delta_e - q) + 2pn^2 + 1. \tag{4.28}$$

Since $0 < \Delta_e \leq p$ and $1 \leq i \leq n$, the value Δ_s in (4.28) is bounded as

$$pn^2 + 1 \leq \Delta_s \leq (2p-1)n^2 + 1.$$

Hence, when S$\hat{c}$ occurs and $\Delta_s > pn^2$, the error position i for $1 \leq i \leq n$ is determined by (4.28) as

$$i = \sqrt{\frac{\Delta_s - 2pn^2 - 1}{\Delta_e - q}}. \tag{4.29}$$

From (4.29), the determined value i is unique. From (4.27), $c - \hat{c} = \Delta_e - q$, then the symbol $\hat{c}$ in the i-th position in $\boldsymbol{y'}$ is replaced by a symbol of value $(\hat{c}+\Delta_e-q)$. Then, the output sequence $\boldsymbol{w}$ is given as $\boldsymbol{w} = (y_1, y_2, \cdots, y_{i-1}, (\hat{c}+\Delta_e - q), y_{i+1}, \cdots, y_n)$, and the decoding is finished.

Therefore, according to the error position i in (4.27) and (4.29), the decoder can produce the corrected sequence.

Example 4.3. (*Continue the example in Section 4.3.1*) The given parameters are $n = 10$, $e = 0$, $f = 0$, and $s = 127$. It is supposed that the received sequence

$\boldsymbol{y} = (2, 2, 1, \mathbf{1}, 0, 1, 2, 0, 3, 3) \in \mathbb{F}_4^{10}$, where the bold symbols indicates the error symbols. The parameters e', f', and s' of $\boldsymbol{y}$ are $e' = 3, f' = 3$, and $s' = 111$, then $\Delta_e = 1, \Delta_f = 1$, and $\Delta_s = 16$.

Step 1: clarifying the error scenario. Since $n' = n = 10, \Delta_e = 1 \neq 0$, S$\hat{c}$ scenario occurs.

Step 2: determining error value and position, then correcting the sequence.

- Since $\Delta_s = 16 < pn^2 = 3.100 = 300$, the value difference between c and $\hat{c}$ is determined as $c - \hat{c} = \Delta_e = 1$.
- The substituted position is obtained by (4.27) as $i = \sqrt{\Delta_s/\Delta_e} = \sqrt{16/1} = 4$. Thus, the error position for S$\hat{c}$ is the fourth position in $\boldsymbol{y}$.
- Since $y_4 = \hat{c} = 1$ and $c - \hat{c} = 1$, the original symbol c equals as $c = \hat{c} + 1 = 2$. Thus, the correct sequence $\boldsymbol{w}$ is produced by swapping the fourth and fifth symbols in $\boldsymbol{y}$ as $\boldsymbol{w} = (2, 2, 1, 2, 0, 1, 2, 0, 3, 3)$.

4.4.4 Decoding for a transposition error

When $n' = n$, $\Delta_e = 0$, and $\Delta_f\Delta_s \neq 0$, the decoder infers that Td_2d_1 occurs. For Td_2d_1 scenario, since two adjacent symbols in the i-th and $(i+1)$-th positions are transposed, one of two indexes i and $(i+1)$ is even and the other is odd. Hence, the difference between two transposed symbols can be identified based on the weight difference of the even elements in the sequence. Thus, the difference between two transposed symbols is computed as

$$|d_2 - d_1| \bmod q = \Delta_f, \tag{4.30}$$

where if the first transposed position i is the even index, (4.30) becomes $d_1 - d_2 \bmod q = \Delta_f$. Conversely, if the second transposed position $(i+1)$ is the even index, (4.30) is rewritten as $d_2 - d_1 \bmod q = \Delta_f$. Due to similarity, the detailed descriptions for the case $d_1 - d_2 \bmod q = \Delta_f$ are presented, and the descriptions for $d_2 - d_1 \bmod q = \Delta_f$ are briefly given.

It is assumed that $d_1 - d_2 \bmod q = \Delta_f$, the condition $d_1 \neq d_2$ can be classified into two cases as $d_1 < d_2$ and $d_1 > d_2$. Thus, from (4.18)–(4.19), the value of Δ_s in (4.10) can be expressed as follows.

- If $d_1 < d_2$, corresponding to $(d_1 - d_2) < 0$, the term $(d_2 - d_1)$ in (4.18) is identified as $d_2 - d_1 = q - \Delta_f$, and Δ_s can be presented as

$$\Delta_s = (2i+1)(q - \Delta_f). \tag{4.31}$$

  Since $0 < \Delta_f \leq p$, $p = q - 1$, and $1 \leq i \leq n-1$, the value Δ_s in (4.31) is bounded as

$$0 < \Delta_s \leq p(2n-1).$$

  Since $\Delta_f \neq 0$ and $\Delta_s \leq p(2n-1)$, from (4.31), a unique error position i satisfying as an even index and $1 \leq i \leq n-1$ is determined by

$$i = \frac{1}{2}\left(\frac{\Delta_s}{q - \Delta_f} - 1\right). \tag{4.32}$$

- If $d_1 > d_2$, then $(d_1 - d_2) > 0$, the term $(d_2 - d_1)$ in (4.18) is given as $d_2 - d_1 = -\Delta_f$, and Δ_s in (4.10) is given as

$$\Delta_s = (2i+1)(-\Delta_f) + 2pn^2 + 1. \tag{4.33}$$

From the conditions $0 < \Delta_f \leq p$ and $1 \leq i \leq n-1$, the bounds of Δ_s in (4.33) are determined as

$$2pn^2 + 1 - p(2n-1) \leq \Delta_s \leq 2pn^2 - 2n + 2. \tag{4.34}$$

Since $n^2 > 2n - 1$ and $0 \leq p < q$, $2pn^2 > 2p(2n-1)$ and then the term $2pn^2+1-p(2n-1)$ in (4.34) is always satisfied as $2pn^2+1-p(2n-1) > p(2n-1)$. Thus, (4.34) can be rewritten as

$$p(2n-1) < \Delta_s \leq 2pn^2 - 2n + 2.$$

Furthermore, from (4.34), a unique error position i satisfying as an odd index and $1 \leq i \leq n-1$ is determined as

$$i = \frac{1}{2}\left(\frac{2pn^2 + 1 - \Delta_s}{\Delta_f} - 1\right). \tag{4.35}$$

From the unique error positions i are determined by (4.32) and (4.35), a unique output sequence $\boldsymbol{w}$ is obtained by transposing the i-th and $(i+1)$-th symbols of values d_2 and d_1 in $\boldsymbol{y}$, respectively, as $\boldsymbol{w} = (y_1, y_2, \cdots, y_{i-1}, d_1, d_2, y_{i+2}, \cdots, y_n)$, and the decoding is finished.

By using a similar method, if (4.30) is given as $d_2 - d_1 \bmod q = \Delta_f$, the value of Δ_s in (4.10) can be expressed as follows.

- If $d_2 > d_1$, inferring to $(d_2 - d_1) > 0$, the term $(d_2 - d_1)$ in (4.18) is identified as $d_2 - d_1 = \Delta_f$, and $0 < \Delta_s = (2i+1)\Delta_f \leq p(2n-1)$. Since $\Delta_f \neq 0$ and $\Delta_s \leq p(2n-1)$, a unique error position i satisfying as an even index and $1 \leq i \leq n-1$ is determined by

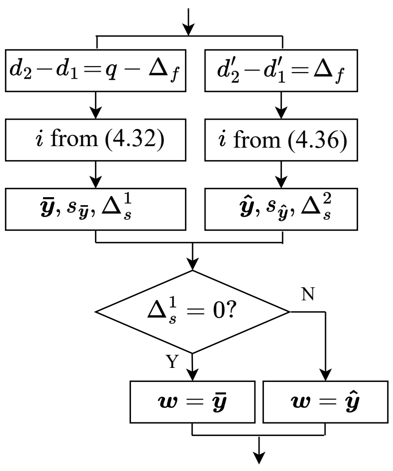
$$i = \frac{1}{2}\left(\frac{\Delta_s}{\Delta_f} - 1\right). \tag{4.36}$$

- If $d_2 < d_1$, then $(d_2 - d_1) < 0$, the term $(d_2 - d_1)$ in (4.18) is given as $d_2 - d_1 = \Delta_f - q$, and $\Delta_s = (2i+1)(\Delta_f - q) + 2pn^2 + 1 > p(2n-1)$. Furthermore, since $-\Delta_f \neq 0$ and $\Delta_s > p(2n-1)$, a unique error position i satisfying as an odd index and $1 \leq i \leq n-1$ is determined as

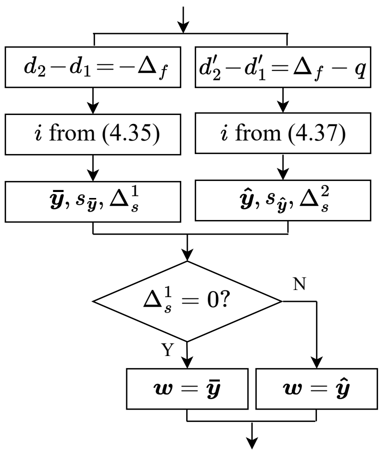
$$i = \frac{1}{2}\left(\frac{\Delta_s - 2pn^2 - 1}{\Delta_f - q} - 1\right). \tag{4.37}$$

From the unique error positions i are determined by (4.36) and (4.37), a unique output sequence $\boldsymbol{w}$ is obtained by transposing the i-th and $(i+1)$-th symbols of values d_2 and d_1 in $\boldsymbol{y}$, respectively, as $\boldsymbol{w} = (y_1, y_2, \cdots, y_{i-1}, d_1, d_2, y_{i+2}, \cdots, y_n)$, and the decoding is finished.

Therefore, if $\Delta_s \leq p(2n-1)$, there are two cases of value $d_2 - d_1$ as $d_2 - d_1 = q - \Delta_f$ (corresponding to the first transposed position i is even index) and $d_2 - d_1 = \Delta_f$ (corresponding to the first transposed position i is odd index). Then, the decoder employs a parallel decoding step, called **Func 1**, to correct the sequence. To avoid confusion, for i as an odd index, we replace $d_2 - d_1 = \Delta_f$ with $d_2' - d_1' = \Delta_f$. Similarly, if $\Delta_s > p(2n-1)$, the decoder uses a parallel decoding step, named **Func 2**, with one branch for $d_2 - d_1 = -\Delta_f$ (corresponding to the first transposed position


$d_2-d_1=q-\Delta_f$
$d'_2-d'_1=\Delta_f$
i from (4.32)
i from (4.36)
$\bar{\boldsymbol{y}}, s_{\bar{y}}, \Delta_s^1$
$\hat{\boldsymbol{y}}, s_{\hat{y}}, \Delta_s^2$
$\Delta_s^1=0?$
N
Y
$\boldsymbol{w}=\bar{\boldsymbol{y}}$
$\boldsymbol{w}=\hat{\boldsymbol{y}}$

(a) **Func 1**


$d_2-d_1=-\Delta_f$
$d'_2-d'_1=\Delta_f-q$
i from (4.35)
i from (4.37)
$\bar{\boldsymbol{y}}, s_{\bar{y}}, \Delta_s^1$
$\hat{\boldsymbol{y}}, s_{\hat{y}}, \Delta_s^2$
$\Delta_s^1=0?$
N
Y
$\boldsymbol{w}=\bar{\boldsymbol{y}}$
$\boldsymbol{w}=\hat{\boldsymbol{y}}$

(b) **Func 2**

Figure 4.3: Decoding steps **Func 1** and **Func 2**.

i is even index) and the other for $d'_2-d'_1=\Delta_f-q$ (meaning that the first transposed position i is odd index). The decoding steps of two parallel decoding steps, **Func 1** and **Func 2**, are shown in Fig. 4.3.

In Fig. 4.3a, **Func 1** includes two parallel decoding branches as one for $d_2-d_1=q-\Delta_f$ and the other for $d'_2-d'_1=\Delta_f$. For $d_2-d_1=q-\Delta_f$, the first transposed position i is determined by (4.32), then the recovered sequence $\bar{\boldsymbol{y}}$ is obtained by transposing the i-th and $(i+1)$-th symbols in $\boldsymbol{y}$. The syndrome $s_{\bar{\boldsymbol{y}}}$ can be calculated as $s_{\bar{\boldsymbol{y}}}=\sum_{k=1}^{n}k^2\bar{y}_k \bmod (2pn^2+1)$, then $\Delta_s^1=s-s_{\bar{\boldsymbol{y}}} \bmod (2pn^2+1)$. Similarly, for $d'_2-d'_1=\Delta_f$, the first transposed position i is determined by $\hat{y}$, the syndrome $s_{\hat{\boldsymbol{y}}}$, and Δ_s^2 are obtained. If $\Delta_s^1=0$, corresponding to $s_{\bar{\boldsymbol{y}}}=s$, the correct sequence is $\boldsymbol{w}=\bar{\boldsymbol{y}}$ and the decoding is finished. If not, meaning that $s_{\hat{\boldsymbol{y}}}=s$, the correct sequence is $\boldsymbol{w}=\hat{\boldsymbol{y}}$ and the decoding is finished.

It is noted that **Func 1** can produce the unique correct sequence and is proved by Lemma 4.3. It means that a unique sequence $\boldsymbol{w}$ has syndrome $s_{\boldsymbol{w}} = \sum_{k=1}^{n} k^2 w_k \bmod (2pn^2+1)$ and $s_{\boldsymbol{w}} = s$.

Lemma 4.3. *When decoding Td_2d_1 scenario for a received sequence $\boldsymbol{y}$, parameters n, e, f and s, **Func 1** gives a unique correct sequence by using constraint* (4.7)*.*

Proof. From Fig. 4.3a, it is supposed that $1 \leq i_1 \leq n-1$ is obtained by (4.32). It is assumed that $\boldsymbol{w}$ is a codeword that is recovered by transposing the i_1-th and (i_1+1) symbols in $\boldsymbol{y}$. Then, the syndrome of $\boldsymbol{w}$ is calculated as $s_{\boldsymbol{w}} = \sum_{k=1}^{n} k^2 w_k \bmod (2pn^2+1)$ and $s_{\boldsymbol{w}} = s$.

It is assumed that $1 \leq i_2 \leq n-1$ and $i_2 \neq i_1$ is obtained by (4.36). Then, $\boldsymbol{z}$ is a codeword that is generated by transposing the i_2-th and (i_2+1) symbols in $\boldsymbol{y}$. The syndrome of $\boldsymbol{z}$ is given as $s_{\boldsymbol{z}} = \sum_{k=1}^{n} k^2 z_k \bmod (2pn^2+1) = s$.

Since two values $s_{\boldsymbol{w}}$ and $s_{\boldsymbol{z}}$ should be the same and equal to s, Δ_s'' is expressed as

$$\Delta_s'' = s_{\boldsymbol{w}} - s_{\boldsymbol{z}} \bmod (2pn^2+1) = 0. \tag{4.38}$$

From (4.7), Δ_s'' in (4.38) can be rewritten as

$$\Delta_s'' = \left(\sum_{k=1}^{n} k^2 w_k - \sum_{k=1}^{n} k^2 z_k \right) \bmod (2pn^2+1) = 0. \tag{4.39}$$

To prove that **Func 1** produces a unique correct sequence, (4.39) should be not satisfied. Additionally, the condition $i_2 \neq i_1$ can be classified into two cases as $i_1 < i_2$ and $i_1 > i_2$. Then, for the different locations of i_1 and i_2, let H be $H = \sum_{k=1}^{n} k^2 w_k - \sum_{k=1}^{n} k^2 z_k$ in (4.39) and H can be rewritten as below

1. For $i_1 < i_2$, H is presented as

$$
\begin{aligned}
H =& \sum\nolimits_{k=1}^{i_1-1} k^2 w_k + i_1^2 d_1 + (i_1+1)^2 d_2 + \sum\nolimits_{k=i_1+2}^{n} k^2 w_k - \sum\nolimits_{k=1}^{i_2-1} k^2 z_k - i_2^2 d_1' \\
&- (i_2+1)^2 d_2' - \sum_{k=i_2+2}^{n} k^2 z_k \\
=& i_1^2 d_1 + (i_1+1)^2 d_2 + i_2^2 y_{i_2} + (i_2+1)^2 y_{i_2+1} - i_1^2 y_{i_1} - (i_1+1)^2 y_{i_1+1} - i_2^2 d_1' \\
&- (i_2+1)^2 d_2' \\
=& i_1^2 d_1 + (i_1+1)^2 d_2 + i_2^2 d_2' + (i_2+1)^2 d_1' - i_1^2 d_2 - (i_1+1)^2 d_1 - i_2^2 d_1' - (i_2+1)^2 d_2' \\
=& i_1^2 (d_1 - d_2) + (i_1+1)^2 (d_2 - d_1) + i_2^2 (d_2' - d_1') + (i_2+1)^2 (d_1' - d_2') \\
=& i_1^2 (\Delta_f - q) + (i_1+1)^2 (q - \Delta_f) + i_2^2 \Delta_f + (i_2+1)^2 (-\Delta_f) \\
=& -\Delta_f (2i_1 + 2i_2 + 2) + q(2i_1 + 1). \qquad (4.40)
\end{aligned}
$$

According to (4.31), i_1 and i_2 satisfy $\Delta_s = (2i_1+1)(d_2-d_1) = (2i_2+1)(d_2'-d_1')$. Thus, (4.40) can be rewritten as

$$
\begin{aligned}
H =& -\Delta_f \left(\frac{\Delta_s}{q-\Delta_f} + \frac{\Delta_s}{\Delta_f} \right) + q \frac{\Delta_s}{q-\Delta_f} \\
=& -\Delta_f - \frac{\Delta_s(\Delta_f - 1)}{q - \Delta_f} \\
=& -\Delta_s \frac{q-1}{q-\Delta_f}. \qquad (4.41)
\end{aligned}
$$

Since $0 < \Delta_s \leq p(2n-1)$, $q-1 > 0$, and $q - \Delta_f > 0$, the term $-\Delta_f \frac{q-1}{q-\Delta_f}$ in (4.41) is always negative, then (4.39) is not satisfied.

2. For $i_1 > i_2$, the proof method is similar to case $i_1 < i_2$, then the specific steps are omitted. The result is that (4.39) is also not satisfied.

Therefore, one of i from (4.32) or i from (4.36) can satisfy constraint (4.7). This means that **Func 1** can produce a unique correct sequence. □

Similarly, the decoding steps in **Func 2** are shown in Fig. 4.3b and the detailed descriptions are omitted for short. Additionally, **Func 2** can also provide a unique sequence and is proved by Lemma 4.4.

Lemma 4.4. *When decoding* Td_2d_1 *scenario for a received sequence* $\boldsymbol{y}$*, parameters* n, e, f *and* s*,* ***Func 2*** *gives a unique correct sequence by using constraint* (4.7)*.*

Proof. From Fig. 4.3b, it is supposed that $1 \leq i_1 \leq n-1$ is obtained by (4.35). It is assumed that $\boldsymbol{w}$ is a codeword that is recovered by transposing the i_1-th and (i_1+1) symbols in $\boldsymbol{y}$. Then, the syndrome of $\boldsymbol{w}$ is calculated as $s_{\boldsymbol{w}} = \sum_{k=1}^{n} k^2 w_k \bmod (2pn^2+1)$ and $s_{\boldsymbol{w}} = s$.

It is assumed that $1 \leq i_2 \leq n-1$ and $i_2 \neq i_1$ is obtained by (4.37). Then, $\boldsymbol{z}$ is a codeword that is generated by transposing the i_2-th and (i_2+1) symbols in $\boldsymbol{y}$. The syndrome of $\boldsymbol{z}$ is given as $s_{\boldsymbol{z}} = \sum_{k=1}^{n} k^2 z_k \bmod (2pn^2+1) = s$.

Since the proof approach is similar to Lemma 4.3, the specific proof steps are omitted. Therefore, one of i from (4.35) or i from (4.37) can satisfy constraint (4.7). This means that **Func 2** can produce a unique correct sequence. □

Example 4.4. (*Continue the example in Section 4.3.1*) The given parameters are $n = 10$, $e = 0$, $f = 0$, and $s = 127$. It is supposed that the received sequence $\boldsymbol{y} = (2,2,1,2,0,1,2,\mathbf{3},\mathbf{0},3) \in \mathbb{F}_4^{10}$, where the bold symbols indicates the error symbols. The parameters e', f', and s' of $\boldsymbol{y}$ are $e' = 0, f' = 3$, and $s' = 76$, then $\Delta_e = 0, \Delta_f = 1$, and $\Delta_s = 51$.

Step 1: clarifying the error scenario.

Since $n' = n = 10, \Delta_e = 0$, and $\Delta_f \Delta_s \neq 0$, Td_2d_1 scenario occurs.

Step 2: determining error value and position, then correcting the sequence. Since $\Delta_s = 51 < p(2n-1) = 3(20-1) = 57$, **Func 1** is used with two parallel decoding branches as

- The first branch is for $d_2 - d_1 = q - \Delta_f = 3$. Then, the first transposed position is verified by (4.32) is $i = \frac{1}{2}(\frac{51}{4-1} - 1) = 8$ and i satisfies as an even index.

  The recovered sequence is $\bar{\boldsymbol{y}} = (2, 2, 1, 2, 0, 1, 2, \mathbf{0}, \mathbf{3}, 3)$ by swapping the eighth and ninth symbols in $\boldsymbol{y}$.

  The syndrome of $\bar{\boldsymbol{y}}$ is calculated as $s_{\bar{\boldsymbol{y}}} = \sum_{k=1}^{10} k^2 \bar{y}_k \bmod 601{=}127(= s)$.

- The second branch is for $d'_2 - d'_1 = \Delta_f = 1$. Then, the first transposed position is verified by (4.36), however, $\frac{1}{2}(\frac{51}{1} - 1) = 25 > 10$ contradicts the condition $1 \le i \le 9$. Thus, there is no value i satisfying Δ_s when $d'_2 - d'_1 = \Delta_f$.

The correct sequence $\boldsymbol{w}$ is outputted as $\boldsymbol{w} = \bar{\boldsymbol{y}}$.

4.5 Redundancy symbols and code rates

In addition, the redundancy of the proposed code $C(q, n)$ can be specified by the potential of three parameters $0 \le e, f \le p$, and $0 \le s \le 2pn^2$ in (4.5)–(4.7) as

$$n - \log_q |C(q, n)| \le n - \log_q \frac{q^n}{q^2(2pn^2 + 1)} \approx 2\log_q n + 4.$$

The proposed code $C(q, n)$ requires at most $2\log_q n + 4$ redundancy symbols to correct one edit or transposition error. For particular, we provide the redundancies and code rates of the proposed code $C(q, n)$ for given values of length n when $q = 2$ and $q = 4$ in Table 4.2.

Table 4.2: The redundancies and code rates of the proposed code $C(q, n)$ for length n when $q = 2, q = 3$ and $q = 4$.

n	$q = 2$		$q = 3$		$q = 4$	
	Redundancy	**Code rate**	**Redundancy**	**Code rate**	**Redundancy**	**Code rate**
100	18	0.820	13	0.870	11	0.890
150	19	0.873	14	0.907	12	0.920
200	20	0.900	14	0.930	12	0.940
250	20	0.920	15	0.940	12	0.952
300	21	0.930	15	0.950	13	0.957
1000	24	0.976	17	0.983	14	0.986

Table 4.3: Comparisons between the referenced works and proposed code.

	Type of code	**Type of error**	**Length of received sequence**	**Redundancy**
Varshamov *et al.* [18]	Binary	1 indel	$n-1$ or $n+1$	$\log_2(n+1)$ bits
Levenshtein [19]	Binary	1 edit	$n-1$ or $n+1$ or n	$\log_2 n + 1$ bits
Tenengolts [20]	q-ary ($q > 2$)	1 indel	$n-1$ or $n+1$	$\log_q n + 1$ symbols
Nguyen *et al.* [37]	Quaternary	1 edit	$n-1$ or $n+1$ or n	$\log_2 n + 1$ symbols
Gabrys *et.al* [42]	Binary	1 deletion or transposition	$n-1$ or n	$\log_2 n + O(\log_2 \log_2 n)$ bits
Proposed code	q-ary ($q \geq 2$)	1 edit or transposition	$n-1$ or $n+1$ or n	$2\log_q n + 4$ symbols

To highlight our contributions to this code design, Table 4.3 provides a comparison between the proposed code and the referenced works, where codes were also constructed by several constraints. As shown in Table 4.3, the proposed code investigates the scenarios involving one edit or transposition error, which were not addressed in the referenced works. Furthermore, we propose the q-ary code scheme with arbitrary $q \geq 2$, expanding beyond the limitations of prior works [18–20, 37] which focused solely on indel (deletion or insertion) or edit (deletion/insertion/substitution) errors and a predetermined scheme as binary, quaternary, or q-ary with $q > 2$. While the codes in [42] extended the scope of the error to include deletion or transposition errors, the scheme remained binary. Additionally, this code cannot handle a substitution error since the substitution error can alter the markers, leading to an inaccurate determination of error scenarios and positions. Despite requiring more redundancy, the proposed code's broader error coverage justifies this necessity. Although the proposed code requires more redundancy, its broader scheme and scope of error correction confirm this requirement.

Chapter 5

Summary of contributions and future works

5.1 Thesis conclusion

The communication and storage systems are prone to synchronization errors, especially deletion, insertion, substitution, and adjacent transposition errors, which are the most popular errors affecting information synchronization in these systems. These errors can cause data errors and loss of synchronization, as well as reduce the efficiency of communication, flash memories, and DNA-based data storage systems. Designing the error correction codes to deal with these errors is an essential challenge to improve the performance of the communication and storage systems.

Throughout this dissertation, two classes of code designs are introduced to mitigate the specific number of errors involving deletion, insertion, substitution, and adjacent transposition errors. Specifically, we first construct a binary code design to tackle one deletion and one insertion errors co-occurring in a codeword. The second class of code design is a nonbinary code that can address an edit or adjacent transposition error. The conclusions of this dissertation are listed as follows.

- We propose a novel binary code design to correct one deletion and one insertion errors in any position in a codeword. In comparison with the referenced code designs, the proposed code can solve the error cases that have been considered by any specific design. Furthermore, the proposed code requires competitive redundancy bits. To highlight the error correction ability of the proposed code, we also present the mathematical proofs for each constraint and the decoding procedure for all error scenarios.

- We study a nonbinary code design to mitigate an edit or adjacent transposition error with three constraints. Moreover, we verify the feasibility of the error correction ability and decoding processes of the proposed code by mathematical proofs. To compare with the referenced code designs, the proposed code can solve the limitations in distinguishing the substitution error and transposition error scenarios. The proposed code with $q = 4$ can be directly applied to DNA storage systems due to the properties of this code design and the error-prone nature of DNA channels.

5.2 Future research directions

As information and storage systems demand higher quality standards, synchronization emerges as a crucial concern, underscoring the significance of rectifying deletion, insertion, substitution, and transposition errors. Building upon proposed error correction codes, several directions on error correction designs open, including:

- While significant efforts have been made to devise error correction codes for distinct error types, the simultaneous occurrence of discrete and burst errors within a codeword has received limited attention. Consequently, the investigation of codes capable of mitigating both discrete and burst errors holds significant importance in communication and storage systems. For example, designing codes can correct $m_1 \geq 1$ deletion and a burst of $m_2 \geq 2$ insertion, substitution, or transposition errors.

- Furthermore, the proposed codes currently address a predetermined number of errors. One future direction is to develop a code capable of rectifying unstable errors, a more challenging study than addressing stable errors, such as designing a code capable of correcting at most one edit error and one transposition error, or at most $m_1 \geq 1$ deletion and at most $m_2 \geq 1$ substitution or transposition errors.

- Another future direction is to apply error correction codes to DNA-based data storage systems. Some researchers consider specific m and arbitrary m deletion errors and certain algebraic error correction codes have been designed under this assumption. However, the co-occurrence of at least two of deletion, insertion, and substitution, transposition errors pose a significant challenge in DNA storage. Future research could explore novel assumptions about DNA errors and corresponding error correction codes to address these scenarios. Besides designing codes to overcome synchronization errors that may occur during synthesis and sequencing in DNA-based data storage, some rules of codewords in DNA are also considered such as homopolymer run, GC-balance content, and secondary structure.

Publications

Journals

[1] T. -H. Khuat and S. Kim, "A quaternary code correcting a burst of at most two deletion or insertion errors in DNA storage," *Entropy*, vol. 23, no. 12, Nov. 2021.

[2] T. -H. Khuat, H. Park and S. Kim, "New binary code design to correct one deletion and one insertion error," *IEEE Transactions on Communication*, vol. 71, no. 7, pp. 3807–3820, Jul. 2023.

[3] T. -H. Khuat and S. Kim, "Novel q-ary code design for one edit or adjacent transposition error," *IEEE Open Journal of the Communications Society*, vol. 4, pp. 2830–2837, Oct. 2023.

International Conferences

[4] T. -H. Vu, T. Q. Duong, T. -H. Khuat, T. H. Nguyen and S. Kim, "Short-Packet Communication for Cooperative UAV-Based MIMO Systems With Rate Splitting Multiple Access," in Proc. *2022 IEEE Ninth International Conference on Communications and Electronics (ICCE)*, Nha Trang, Vietnam, 2022, pp. 113–117.

[5] K. -T. Nguyen, T. -H. Vu, T. -H. Khuat, H. -C. Le and S. Kim, "Performance Analysis of Uplink MISO Systems with RIS Selections and MRC/SC Configurations," in Proc. *2022 IEEE Ninth International Conference on Communications and Electronics (ICCE)*, Nha Trang, Vietnam, 2022, pp. 145–148.

[6] T. -H. Khuat and S. Kim, "One indel or one-deletion and one-insertion error-correcting code," in Proc. *2023 International Conference on Advanced Technologies for Communications (ATC)*, Da Nang, Vietnam, 2023, pp. 36–40.

References

[1] "How much data is produced every day?" 2016, http://www.northeastern.edu/levelblog/2016/05/13/how-much-data-produced-every-day/ [Accessed: (13 May)].

[2] R. G. Gallager, "Sequential decoding for binary channels with noise and synchronization errors," in *Lincoln Lab Group Report*, 1961.

[3] K. Iwamura and H. Imai, "A code to correct synchronization errors," *Electronics and Communications in Japan (Part III: Fundamental Electronic Science)*, vol. 76, no. 6, pp. 60–71, 1993.

[4] H. C. Ferreira, K. A. S. Abdel-Ghaffar, L. Cheng, T. G. Swart, and K. Ouahada, "Moment balancing templates: Constructions to add insertion/deletion correction capability to error correcting or constrained codes," *IEEE Transactions on Information Theory*, vol. 55, no. 8, pp. 3494–3500, 2009.

[5] H. Ferreira, A. Vinck, T. Swart, and I. de Beer, "Permutation trellis codes," *IEEE Transactions on Communications*, vol. 53, no. 11, pp. 1782–1789, 2005.

[6] S. Konstantinidis, S. Perron, and L. 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.

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

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

[9] S. K. Hanna and S. E. Rouayheb, "Guess & check codes for deletions, insertions, and synchronization," *IEEE Transactions on Information Theory*, vol. 65, no. 1, pp. 3–15, 2019.

[10] A. Jiang, R. Mateescu, M. Schwartz, and J. Bruck, "Rank modulation for flash memories," in *2008 IEEE International Symposium on Information Theory*, 2008, pp. 1731–1735.

[11] A. Jiang, M. Schwartz, and J. Bruck, "Correcting charge-constrained errors in the rank-modulation scheme," *IEEE Transactions on Information Theory*, vol. 56, no. 5, pp. 2112–2120, 2010.

[12] H. Zhou, A. Jiang, and J. Bruck, "Systematic error-correcting codes for rank modulation," in *2012 IEEE International Symposium on Information Theory Proceedings*, 2012, pp. 2978–2982.

[13] R. Gabrys, E. Yaakobi, F. Farnoud, F. Sala, J. Bruck, and L. Dolecek, "Codes

correcting erasures and deletions for rank modulation," *IEEE Transactions on Information Theory*, vol. 62, no. 1, pp. 136–150, 2016.

[14] M. Blawat, K. Gaedke, I. Huetter, X.-M. Chen, B. Turczyk, S. Inverso, B. Pruitt, and G. Church, "Forward error correction for DNA data storage," *Procedia Computer Science*, vol. 80, pp. 1011–1022, 12 2016.

[15] 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, 2015.

[16] 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.

[17] S. Chandak, H. Ji, K. Tatwawadi, B. Lau, J. Mardia, M. Kubit, J. Neu, P. Griffin, M. Wootters, and T. Weissman, "Improved read/write cost tradeoff in DNA-based data storage using LDPC codes," in *2019 57th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*. IEEE Press, 2019, p. 147–156.

[18] R. R. Varshamov, "A code for correcting a single asymmetric error," *Avtomatika i Telemekhanika*, 1965.

[19] V. I. Levenshtein *et al.*, "Binary codes capable of correcting deletions, inser-

tions, and reversals," in *Soviet physics doklady*, vol. 10, no. 8. Soviet Union, 1966, pp. 707–710.

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

[21] V. Levenshtein, "Asymptotically optimum binary code with correction for losses of one or two adjacent bits," *Problemy Kibernetiki*, vol. 19, pp. 293–298, 1967.

[22] L. Cheng, T. G. Swart, H. C. Ferreira, and K. A. Abdel-Ghaffar, "Codes for correcting three or more adjacent deletions or insertions," in *2014 IEEE International Symposium on Information Theory.* IEEE, 2014, pp. 1246–1250.

[23] C. Schoeny, A. Wachter-Zeh, R. Gabrys, and E. Yaakobi, "Codes correcting a burst of deletions or insertions," *IEEE Transactions on Information Theory*, vol. 63, no. 4, pp. 1971–1985, 2017.

[24] R. Gabrys and F. Sala, "Codes correcting two deletions," *IEEE Transactions on Information Theory*, vol. 65, no. 2, pp. 965–974, 2019.

[25] V. Guruswami and J. Håstad, "Explicit two-deletion codes with redundancy matching the existential bound," *IEEE Transactions on Information Theory*, vol. 67, no. 10, pp. 6384–6394, 2021.

[26] J. Sima and J. Bruck, "On optimal k-deletion correcting codes," *IEEE Transactions on Information Theory*, vol. 67, no. 6, pp. 3360–3375, 2020.

[27] J. Sima, R. Gabrys, and J. Bruck, "Optimal systematic t-deletion correcting codes," in *2020 IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2020, pp. 769–774.

[28] J. Brakensiek, V. Guruswami, and S. Zbarsky, "Efficient low-redundancy codes for correcting multiple deletions," *IEEE Transactions on Information Theory*, vol. 64, no. 5, pp. 3403–3410, 2018.

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

[30] K. A. S. Abdel-Ghaffar, F. Paluncic, H. C. Ferreira, and W. A. Clarke, "On Helberg's generalization of the levenshtein code for multiple deletion/insertion error correction," *IEEE Transactions on Information Theory*, vol. 58, no. 3, pp. 1804–1808, 2012.

[31] K. Bibak and O. Milenkovic, "Explicit formulas for the weight enumerators of some classes of deletion correcting codes," *IEEE Transactions on Communications*, vol. 67, no. 3, pp. 1809–1816, 2019.

[32] M. Abroshan, R. Venkataramanan, and A. G. i Fàbregas, "Multilayer codes for synchronization from deletions and insertions," *IEEE Transactions on Information Theory*, vol. 67, no. 6, pp. 3342–3359, 2021.

[33] X. Jiao, H. Liu, J. Mu, H. Han, and Y.-C. He, "On prefixed Varshamov-Tenengolts codes for segmented edit channels," *IEEE Transactions on Communications*, vol. 70, no. 3, pp. 1535–1545, 2022.

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

[35] J.-P. Thiers and J. Freudenberger, "Code-based cryptography with generalized concatenated codes for restricted error values," *IEEE Open Journal of the Communications Society*, vol. 3, pp. 1528–1539, 2022.

[36] K. Cai, Y. M. Chee, R. Gabrys, H. M. Kiah, and T. T. Nguyen, "Correcting a single indel/edit for DNA-based data storage: Linear-time encoders and order-optimality," *IEEE Transactions on Information Theory*, vol. 67, no. 6, pp. 3438–3451, 2021.

[37] T. T. Nguyen, K. Cai, K. A. S. Immink, and other, "Capacity-approaching constrained codes with error correction for DNA-based data storage," *IEEE Transactions on Information Theory*, 2021.

[38] W. Haselmayr, N. Varshney, A. T. Asyhari, A. Springer, and W. Guo, "On the impact of transposition errors in diffusion-based channels," *IEEE Transactions on Communications*, vol. 67, no. 1, pp. 364–374, 2019.

[39] L. 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.

[40] F. Farnoud, V. Skachek, and O. Milenkovic, "Error-correction in flash memories

via codes in the Ulam metric," *IEEE Transactions on Information Theory*, vol. 59, no. 5, pp. 3003–3020, 2013.

[41] R. Gabrys, E. Yaakobi, and O. Milenkovic, "Codes in the Damerau distance for deletion and adjacent transposition correction," *IEEE Transactions on Information Theory*, vol. 64, no. 4, pp. 2550–2570, 2018.

[42] R. Gabrys, V. Guruswami, J. Ribeiro, and K. Wu, "Beyond single-deletion correcting codes: Substitutions and transpositions," *IEEE Transactions on Information Theory*, vol. 69, no. 1, pp. 169–186, 2023.

[43] T.-H. Khuat and S. Kim, "A quaternary code correcting a burst of at most two deletion or insertion errors in DNA storage," *Entropy*, vol. 23, no. 12, 2021.

Appendix A

Proof in Chapter 3

A.1 Proof for Lemma 3.5

It is assumed that the indexes belonging to the run of the $(l-1)$-th bit zero or the l-th bit zero in $\boldsymbol{y}$ are expressed as $\boldsymbol{l} = (l_1, l_1+1, \cdots, l_1+r-1)$, where the length of the run is r and l is one element in $\boldsymbol{l}$.

For $D0I0$, from Table 3.3, the distances $u'_{l_k-1} - u'_m$ for $l_1 \leq l_k \leq l_1 + r - 1$ are given as

$$u'_{l_k-1} - u'_m = \Delta_d - 2n - 2. \tag{A.1}$$

For $D0I1$, from Table 3.3, the distances $u'_{l_k-1} - u'_m$ are presented as

$$u'_{l_k-1} - u'_m = \Delta_d - n - m - 2. \tag{A.2}$$

Since the run $\boldsymbol{l}$ is all zeros, from (3.3), u'_{l_k-1} for all l_k in $\boldsymbol{l}$ are the same. Thus, the terms $u'_{l_k-1} - u'_m$ in (A.2) are the same for all l_k in $\boldsymbol{l}$ and similarly, u'_{l_k-1} for all l_k in (A.2) are same for all l_k in $\boldsymbol{l}$. This means that the multiple candidates for the deletion error position, which have the same distance, can be reduced to one candidate.

For $D1I0$, the distances $u'_{l_k-1} - u'_m$ are given as

$$u'_{l_k-1} - u'_m = \Delta_d - n + l_k + 1. \tag{A.3}$$

Since the run $\boldsymbol{l}$ is all ones, from (3.3), u'_{l_k-1} increases by one for all l_k in $\boldsymbol{l}$. Then, the term $u'_{l_k-1} - u'_m$ in (A.3) increases by one for all l_k in $\boldsymbol{l}$. However, l_k is an index belonging to the run $\boldsymbol{l}$ in $\boldsymbol{y}$, and the values u'_m of the insertion error are the same. Thus, the deletion error can be recovered by adding a bit in any position in the run. This means that the multiple candidates in $\boldsymbol{l}$ can be reduced to one candidate.

For $D1I1$, the distances $u'_{l_k-1} - u'_m$ are represented as below

$$u'_{l_k-1} - u'_m = \Delta_d + l_k - m - 1. \tag{A.4}$$

Similar to the scenario $D1I0$, the term u'_{l_k-1} increases by one for all l_k in $\boldsymbol{l}$, then the term $u'_{l_k-1} - u'_m$ in (A.4) increases by one for all l_k in $\boldsymbol{l}$. Though the distances are not the same for all l_k in $\boldsymbol{l}$, the multiple candidates in $\boldsymbol{l}$ can be reduced to one candidate.

A.2 Proof for Lemma 3.9

Similar to Lemma 3.8, according to the definitions of set S_a, T_m, and S in (3.46)–(3.48), since the first indexes at the runs of the inserted bit a are included in S, the first index m at the run of the exact insertion position j is also included in S. Since the first indexes at the runs of the deleted bit b are elements in S, the first index l

at the run of the exact deletion position i is also included in S. It supposed that a codeword $\boldsymbol{z}$ generated from the first indexes (l, m) and the run-length of the i-th bit and j-th bit in $\boldsymbol{y}$ are r_i and r_j, respectively to find the original codeword $\boldsymbol{x} = (x_1, x_2, \cdots, x_n)$. This means that $m \leq j \leq m + r_j - 1$ and $l \leq i \leq l + r_i - 1$. The syndrome $e''_{\boldsymbol{z}}$ of $\boldsymbol{z}$ is calculated as

$$\begin{aligned} e''_{\boldsymbol{z}} &\equiv \sum_{k=1}^{n} v_k z_k \bmod (4n-5) \\ &= \left(\sum_{k=1}^{m-1} v_k z_k + \sum_{k=m}^{m+r_j-1} v_k z_k + \sum_{k=m+r_j}^{l-1} v_k z_k + \sum_{k=l}^{l+r_i-1} v_k z_k + \sum_{k=l+r_i}^{n} v_k z_k \right) \bmod (4n-5). \end{aligned} \tag{A.5}$$

By a similar approach to Lemma 3.8, the three terms $\sum_{k=1}^{m-1} v_k z_k$, $\sum_{k=l}^{l+r_i-1} v_k z_k$, and $\sum_{k=l+r_i}^{n} v_k z_k$ in (A.5) are the same as $\sum_{k=1}^{m-1} v_k x_k$, $\sum_{k=l}^{l+r_i-1} v_k x_k$, and $\sum_{k=l+r_i}^{n} v_k x_k$, respectively. Since the insertion position j is between m and $m + r_j - 1$ and in the same run, $\sum_{k=m}^{m+r_j-1} v_k z_k = \sum_{k=m}^{m+r_j-1} v_k x_k$. Moreover, the deletion position i is between l and $l + r_i - 1$ and in the same run, $\sum_{k=l}^{l+r_i-1} v_k z_k = \sum_{k=l}^{l+r_i-1} v_k x_k$. Therefore, $\sum_{k=1}^{n} v_k z_k = \sum_{k=1}^{n} v_k x_k$, and the syndrome $e''_{\boldsymbol{z}}$ in (A.5) is the same as $e_{\boldsymbol{x}}$ and $\Delta_e = 0$.

A.3 Proof for Lemma 3.10

For $p < k \leq q$, y_k and y_{k-1} are the same or different. If y_k and y_{k-1} are the same, y_k and y_{k-1} belong to the same run, and term $v_k(y_k - y_{k-1})$ is also zero. If y_k and y_{k-1} are different, y_k and y_{k-1} belong to different runs and the term $y_k - y_{k-1}$ in (3.50) is 0-1=-1 or 1-0=1. Therefore, if $\boldsymbol{h} = (h_1, h_2, \cdots)$ is an index vector which composes of increasing elements and denotes whole indexes where y_k and y_{k-1} belong to the different runs for $p < k \le q$, $\sum_{k=p+1}^{q} = v_k(y_k - y_{k-1})$ is expressed as

$$\sum_{k=p+1}^{q} v_k(y_k - y_{k-1}) = \begin{cases} v_{h_1} - v_{h_2} + v_{h_3} - v_{h_4} + \cdots & \text{for } y_p = 0, \\ -v_{h_1} + v_{h_2} - v_{h_3} + v_{h_4} - \cdots & \text{for } y_p = 1. \end{cases} \tag{A.6}$$

Since the number of elements of $\boldsymbol{h}$ can be even or odd, we consider the number of elements of $\boldsymbol{h}$ as $2l$ or $2l+1$. For even cases, (A.6) is rewritten as

$$\sum_{k=p+1}^{q} v_k(y_k - y_{k-1}) = \begin{cases} v_{h_1} - v_{h_2} + \cdots + v_{h_{2l-1}} - v_{h_{2l}} & \text{for } y_p = 0, \\ -v_{h_1} + v_{h_2} - \cdots - v_{h_{2l-1}} + v_{h_{2l}} & \text{for } y_p = 1. \end{cases} \tag{A.7}$$

The terms $(v_{h_1} - v_{h_2} + \cdots + v_{h_{2l-1}} - v_{h_{2l}})$ and $-(v_{h_1} - v_{h_2} + \cdots + v_{h_{2l-1}} - v_{h_{2l}})$ are always negative and positive, respectively, since the sequence $\boldsymbol{v}$ is positive and increasing and the number of signs in the terms is the same.

For odd cases, (A.6) is rewritten as below

$$\sum_{k=p+1}^{q} v_k(y_k - y_{k-1}) = \begin{cases} v_{h_1} - v_{h_2} + \cdots - v_{h_{2l}} + v_{h_{2l+1}} & \text{for } y_p = 0, \\ -v_{h_1} + v_{h_2} - \cdots + v_{h_{2l}} - v_{h_{2l+1}} & \text{for } y_p = 1. \end{cases} \tag{A.8}$$

The term $(v_{h_1} - v_{h_2} + \cdots - v_{h_{2l}} + v_{h_{2l+1}})$ in (A.8) can be regarded as $v_{h_1} + (-v_{h_2} + \cdots - v_{h_{2l}} + v_{h_{2l+1}})$. Since the number of sign change in $(-v_{h_2} + \cdots - v_{h_{2l}} + v_{h_{2l+1}})$ is the same, $(-v_{h_2} + \cdots - v_{h_{2l}} + v_{h_{2l+1}})$ is always positive. Moreover, since v_{h_1} is also positive, the term $(v_{h_1} - v_{h_2} + \cdots - v_{h_{2l}} + v_{h_{2l+1}})$ in (A.8) is always positive. Similarly, the term $(-v_{h_1} + v_{h_2} - \cdots + v_{h_{2l}} - v_{h_{2l+1}})$ in (A.8) can be rewritten as $-v_{h_1} - (-v_{h_2} + \cdots - v_{h_{2l}} + v_{h_{2l+1}})$, and it is always negative. Thus, the term $(-v_{h_1} + v_{h_2} - \cdots + v_{h_{2l}} - v_{h_{2l+1}})$ in (A.8) is always negative.

Therefore, from (A.7), (A.8), and the sequence $\boldsymbol{v}$ is non-negative and increasing, $\sum_{k=p+1}^{q} v_k(y_k - y_{k-1})$ is not equal to zero. Since $\left|\sum_{k=p+1}^{q} v_k(y_k - y_{k-1})\right| < (4n-5)$ according to Lemma 3.3, $\sum_{k=p+1}^{q} v_k(y_k - y_{k-1}) \bmod (4n-5)$ is also not zero.