



저작자표시-비영리-변경금지 2.0 대한민국

이용자는 아래의 조건을 따르는 경우에 한하여 자유롭게

- 이 저작물을 복제, 배포, 전송, 전시, 공연 및 방송할 수 있습니다.

다음과 같은 조건을 따라야 합니다:



저작자표시. 귀하는 원저작자를 표시하여야 합니다.



비영리. 귀하는 이 저작물을 영리 목적으로 이용할 수 없습니다.



변경금지. 귀하는 이 저작물을 개작, 변형 또는 가공할 수 없습니다.

- 귀하는, 이 저작물의 재이용이나 배포의 경우, 이 저작물에 적용된 이용허락조건을 명확하게 나타내어야 합니다.
- 저작권자로부터 별도의 허가를 받으면 이러한 조건들은 적용되지 않습니다.

저작권법에 따른 이용자의 권리는 위의 내용에 의하여 영향을 받지 않습니다.

이것은 [이용허락규약\(Legal Code\)](#)을 이해하기 쉽게 요약한 것입니다.

[Disclaimer](#)

DOCTOR OF PHILOSOPHY

**RESEARCH ON CONSTRAINED AND
ERROR CORRECTION CODES FOR DNA
STORAGE**

The Graduate School
of University of Ulsan

Department of Electrical, Electronic
and Computer Engineering

XIAOZHOU LU

Research on Constrained and Error Correction Codes for DNA Storage

Supervisor: Sunghwan Kim

A Dissertation

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

DOCTOR OF PHILOSOPHY

by

Xiaozhou Lu

**Department of Electrical, Electronic
and Computer Engineering**

University of Ulsan, Korea

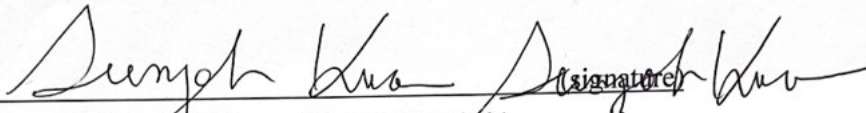
February 2024

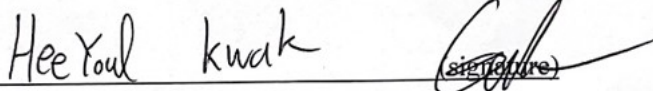
VITA

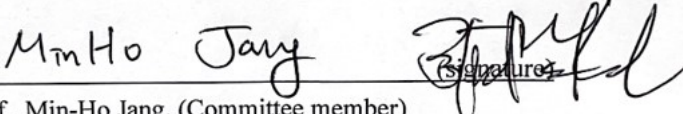
XIAOZHOU LU received the B.S. degree from the University of Ulsan, South Korea, in 2018, where he is currently pursuing the M.S. and Ph.D. combined degree with the Department of Electrical, Electronic, and Computer Engineering, University of Ulsan, South Korea. His research interests include 5G communication, error correction codes, and storage systems.

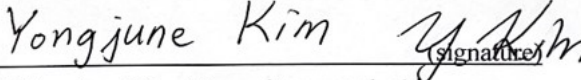
Research on Constrained and Error Correction Codes for DNA Storage

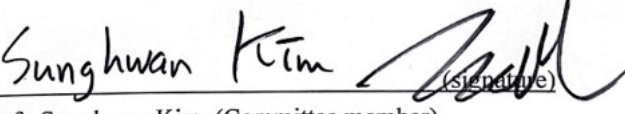
This certifies that the dissertation of Xiaozhou Lu is approved.


Prof . Sungoh Kwon, (Committee Chair)


Prof . Hee-Youl Kwak, (Committee member)


Prof . Min-Ho Jang, (Committee member)


Prof . Yongjune Kim, (Committee member)


Prof . Sunghwan Kim, (Committee member)

Department of Electrical, Electronic
and Computer Engineering

University of Ulsan, Korea

February 2024

ACKNOWLEDGEMENT

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.

Firstly, I would like to express my gratitude to my advisor, Prof. Sunghwan Kim. It is due to his long-term guidance, support, and encouragement that I have made progress in my research topic, ultimately allowing me to refine my work. During my studies, he used his profound theoretical knowledge and extensive experience to continuously help me refine my ideas and provided valuable feedback on my paper writing. I am deeply grateful to him for this.

Secondly, I want to thank my peers. Although we come from different countries, they always share happiness with me and have been a great help in my study.

Additionally, I am profoundly grateful to the University of Ulsan for offering me an exceptional research atmosphere and financial supporting through the BK21+ programs. Moreover, UOU has presented us with the chance to devote into a diverse range of valuable subjects, guided by remarkable instructors. These experiences will furnish me with the expertise required for effective research and professional endeavors throughout my lifetime.

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

ABSTRACT

Research on Constrained and Error Correction Codes for DNA Storage

by

Xiaozhou Lu

Supervisor: Professor Sunghwan Kim

Submitted in Partial Fulfilment of the Requirements for the Degree of
Doctoral Dissertation
January 2024

Due to the increasing demand for data storage, DNA storage systems have begun to attract considerable attention as next-generation storage technologies due to their high densities and longevity. DNA storage technology is a method of storing binary information in the form of DNA strands, which are composed up of DNA sequences and primers. However, common obstacles to DNA storage are caused by insertion, deletion, and substitution errors occurring in DNA synthesis and sequencing. Therefore, reducing the error rates and correcting errors during DNA synthesis and sequencing is inevitable to guarantee reliable data storage in DNA storage. When the DNA strands stored in the DNA pool, efficient random-access desired information from stored DNA strands presents an additional obstacle in DNA storage.

To reduce error rates, a common approach involves imposing constraints on the stored DNA strands, such as ensuring they satisfy GC-balanced and homopolymer run-length constraints, etc. In terms of error correction in DNA storage, error correction codes are employed to enhance the reliability of the DNA synthesis and sequencing processes. Additionally, primers in DNA strands solve the problem of random-access in DNA storage.

This thesis propose a novel code construction method based on the weight distribution of the data and introduce a specific encoding process for both balanced and imbalanced data parts, which enables us to efficiently construct GC-balanced DNA codes. Additionally, to minimize errors in DNA storage processes, we propose a new single insertion/deletion nonbinary systematic error correction code with the maximum run-length constraint and its corresponding encoding algorithm. Finally, to solve the issue that efficient primer design for random-access in synthesized DNA strands, we propose a code design by combining weakly mutually uncorrelated codes with the maximum run length constraint for primer design.

Moreover, we also explore the weakly mutually uncorrelated codes to satisfy combinations of maximum run length constraint with more constraints such as being almost-balanced and having large Hamming distance, which are also efficient constraints for random-access in DNA storage systems.

Contents

List of Figures	ix
List of Tables	x
1 Introduction	1
1.1 Motivation and Objective	1
1.2 DNA storage: system model	3
1.3 The methodology for DNA storage	6
1.3.1 Error-correcting codes for DNA storage	7
1.3.2 Constrained codes for DNA storage	10
1.3.3 Codes for random-access in DNA storage	11
1.4 Contribution and layout	11
2 New construction of balanced codes for DNA storage	13
2.1 Introduction	13
2.2 Preliminaries	14
2.2.1 Related works for binary ϵ -balanced code construction	14
2.3 GC-balanced codes construction	18
2.3.1 Overall encoding processes	18
2.3.2 Flipping functions	19
2.3.3 Encoding and mapping	22
2.3.4 Decoding of the proposed GC-balanced codes	24
2.4 Analysis of the average parity lengths for the proposed codes	27
2.4.1 Average parity length for $f = 0$	27
2.4.2 Average parity length for $f = 1$	29
2.4.3 Average parity length for $f > 1$	30
2.5 Performance evaluation	31
2.5.1 Comparison results of average parity lengths for various ϵ	32
2.5.2 Average parity length comparison results for various f	36
3 Nonbinary single insertion or deletion error correction codes with	

a maximum run-length constraint for DNA storage	40
3.1 Introduction	40
3.2 Preliminaries	42
3.2.1 Proposed code construction and algorithm	43
3.2.2 Construction of sequences with the maximum run-length constraint	44
3.3 Code construction and encoding for a nonbinary SIDEC with a maximum run-length constraint code	46
3.3.1 Code construction	46
3.3.2 Encoding algorithm	50
3.3.3 Special case with $k = 190$ and 191 and the overall of encoding algorithm	55
3.4 Decoding for an insertion or deletion error	59
3.5 Simulation and Comparison results	60
3.5.1 Comparison results of related work [2] and our work	63
4 Weakly mutually uncorrelated codes with maximum run-length constraint for DNA storage	66
4.1 Introduction	66
4.2 Preliminaries	68
4.2.1 MU and WMU code: definitions, constructions, and properties	68
4.3 Constrained WMU code constructions	71
4.3.1 k -WMU code with the maximum run-length l constraint	72
4.3.2 2-WMU code with the maximum run-length l constraint and Hamming distance d	76
4.3.3 Almost-balanced and 3-WMU code with the maximum run-length l constraint	77
4.3.4 Almost-balanced 2-WMU code with the maximum run-length l constraint and Hamming distance d	83
4.4 Applying code construction to primer design in DNA storage	85
4.4.1 2-WMU code construction with maximum run-length l constraint for various lengths	86
4.4.2 The size of the 2-WMU code with maximum run-length l constraint for various lengths	87
4.4.3 Comparison for the maximum run-length constraint	90
5 Conclusions and Future works	93
5.1 Thesis Conclusion	93
5.2 Future research directions	95
Bibliography	97

List of Figures

1.1	The fundamental model of DNA storage systems.	4
1.2	The form of linear block codes apply in DNA storage.	8
1.3	The contribution of the dissertation.	12
2.1	System model of the proposed GC-balanced code construction in DNA storage.	18
2.2	The ratios $\frac{ B_{\epsilon,n}^1 }{2^n}$ for the binary sequences which become $\mathbf{c}^{(1)}$ for $n = 102, 128, 174,$ and 255 and $\epsilon = 0.05$	20
2.3	Encoding functions E_0, E_i for $i = [1, f]$, and E_{final}	24
2.4	Encoding and decoding for $f = 0$	27
2.5	The ratio $\frac{ B_{\epsilon,n}^0 }{2^n}$ for lengths $n = 128, 200,$ and 240	32
2.6	The ratio $\frac{ B_{\epsilon,n}^1 }{2^n}$ for lengths $n = 128, 200,$ and 240	33
2.7	Average parity lengths for $f = 0$ and $n = 128, 200,$ and 240	34
2.8	Average parity lengths for $f = 1$ and $n = 128, 200,$ and 240	35
2.9	Average parity lengths of the proposed codes for various f for lengths $n = 117, 174, 255,$ and $\epsilon = 0.05$	37
2.10	Average parity lengths of the proposed codes for various f for lengths $n = 117, 174, 255,$ and $\epsilon = 0.1$	38
3.1	Overall block diagram of the proposed code.	43
3.2	The flow chart of the proposed encoding algorithm.	57
4.1	Considered DNA storage system and proposed code constructions for primers.	71

List of Tables

3.1	The example of one-to-one mapping between a 48-ary element and q -ary subsequence $\mathbf{z}_{(j-1)r+1}^{jr}$ with r symbols.	45
3.2	Forms of the proposed nonbinary systematic SIDEC code.	46
3.3	The relationship between the message sequence length k , the value of B , the number of parity symbols $m + 1$, and the length of codeword \mathbf{c} , $n + 1$	47
3.4	The relationship between $\alpha_{3(\sigma-1)}^m$ and $\mathbf{c}_{3(\sigma-1)}^m$	53
3.5	The relationship between $\alpha_{3(l-1)}^{3l}$ and $\mathbf{c}_{3(l-1)}^{3l}$ for $l \in [1, \sigma]$ and $m = 3\sigma$	53
3.6	The encoding results for different codeword lengths with $a = 0$ and $b = 1$	62
3.7	The encoding results for different codeword lengths with a is a random number in $[q]$ and b is a random number in $[I_{n+1}]$	62
3.8	The decoding results for different codeword lengths with a single deletion error.	63
3.9	The decoding results for different codeword lengths with a single insertion error.	63
3.10	The comparison between referenced work [2] and our work for CCF and maximum run-length constraint when the codeword length is $n = 150$	64
4.1	The size of the 2-WMU code with maximum run-length $l = 3$ for 18-24 nt.	90
4.2	Different maximum run-lengths results for the codes \mathcal{C}_{R8} of lengths $n_{(2)} = 12$ and $n_{(3)} = 13$, \mathcal{C}_{R8-H} of length $n_{(2)} = 12$, $ \mathcal{C}_{R8-GC} _{(3)}$ of length $n_{(3)} = 13$, and $ \mathcal{C}_{R11-GH} _{(2)}$ of length $n_{(2)} = 12$	91

Notations and Abbreviations

$w(\mathbf{x})$	Weight of the sequence \mathbf{x} : $w(\mathbf{x}) = \sum_{i=1}^n x_i$
ϵ -balanced	Sequence \mathbf{x} satisfy $-\lfloor \epsilon n \rfloor \leq w(\mathbf{x}) - \frac{n}{2} \leq \lfloor \epsilon n \rfloor$
$[i, j]$	Set $\{i, i + 1, \dots, j\}$
$[j]$	Set $[0, j - 1] = \{0, i + 1, \dots, j - 1\}$
$[j] \setminus i$	Set $\{0, 1, \dots, i - 1, i + 1, \dots, j - 1\}$
\mathbf{x}_i^j	Subsequence $(x_i, x_{i+1}, \dots, x_j)$
$\overline{x_i}$	Flipped bit of x_i (i.e. $1 \rightarrow 0$ and $0 \rightarrow 1$)
$\overline{\mathbf{x}_i^j} = (\overline{x_i}, \overline{x_{i+1}}, \dots, \overline{x_j})$	Flipped sequence of \mathbf{x}_i^j
$\lceil \cdot \rceil$	Ceiling function
$\lfloor \cdot \rfloor$	Floor function
$\log_q(\cdot)$	Logarithm of base q
$\log(\cdot)$	Logarithm with base two
mod	Modulo
\cup	Union operation
$\min(\mathbf{x})$	Minimum element in sequence \mathbf{x}
$\max(\mathbf{x})$	Maximum element in sequence \mathbf{x}
$\text{sum}(\mathbf{x})$	Summation all elements in sequence \mathbf{x}
$\mathbb{Z}_q = [q]$	Finite integer ring of size q
\mathbb{Z}_q^n	Set of vectors with n elements over \mathbb{Z}_q
$ \mathbf{x} $	Length of the sequence \mathbf{x}
(\mathbf{x}, \mathbf{y})	Concatenation of sequences \mathbf{x} and \mathbf{y}
A, C, G, and T	DNA nucleotides (correspond to symbols 0, 1, 2, and 3)
Run-length of \mathbf{x}_j^i	All symbols in \mathbf{x}_j^i are same but different with x_{j-1} and x_i
Maximum run-length	Longest run in the sequence \mathbf{x}
Hamming distance	Number of different symbols between any two sequences

DNA	Deoxyribonucleic acid
VT codes	Varshamov-Tenengolts codes
EVT codes	Extended VT codes
nt	Nucleotides
PCR	Polymerase chain reaction
BSC	Binary symmetric channel
BCH codes	Bose–Chaudhuri–Hocquenghem codes
LDPC codes	Low-density parity-check codes
SIDEC codes	Single insertion or deletion error correction codes
MU codes	Mutually uncorrelated codes
WMU codes	Weakly Mutually uncorrelated codes

Chapter 1

Introduction

1.1 Motivation and Objective

With the rapid development of information technology, the total volume of data is also exponentially growing. According to the reports in [1], until 2020, the data volume in the world had reached approximately 44 zettabytes (ZB) (1 ZB = 10^{21} bytes), and it continues to grow at a rate of 2.5 exabytes (Eb) (1 EB = 10^{18} bytes) per day. Although the hardware of data storage systems, such as tapes, hard drives, and flash memory, is also gradually being upgraded to meet the demands of large capacity of data storage, the high-speed growth of data and the high maintenance costs for data centers will soon be challenging to meet people's needs. Therefore, to investigate a new data storage medium is imperative. Deoxyribonucleic acid (DNA) storage technology has opened up a new storage paradigm, and its development plays a crucial role in conserving storage cost and advancing the development of big data storage.

The basic concept of using DNA strands as a storage medium can be traced back to 1953 when it was proposed by Watson and Crick [66], In the following decades, some researchers worked on refining the framework and structure of us-

ing DNA sequences for information storage [47, 48]. In 1996, Davis [14] for the first time encoded a dark and bright pixel image of 35 bits into DNA strands and successfully retrieved the original image from the cellular carrier, marking the initial success of DNA storage technology. In the subsequent years, there was a growing number of successful cases. In 1999, Clelland *et al.* [7] successfully recovered a message hidden in the DNA sequences containing 23 characters. In 2000, Leier *et al.* [37] implemented an encryption and decryption method applicable to DNA barcodes. In 2002, Reif *et al.* [51] first constructed a small DNA database that could be randomly accessed. Uuntil 2017, Organick *et al.* [17] stored 200 megabyte (MB) of data 13 million DNA strands. DNA storage systems have begun to attract a lot of attention again.

With the further research, the advantages of DNA storage become more and more obvious. For instance, it has the advantages of high storage density, longevity, low maintenance cost, and efficient duplication. Although DNA storage offers numerous potential benefits, a series of issues also exist objectively in DNA storage, such as efficiently random-access the stored information among enormous DNA strands and completely retrieve the stored information, given the occurrence of errors during the DNA synthesis and sequencing process [17, 26, 4, 32, 12, 38]. To address these issues, some coding techniques [4, 20] in information theory are considered into DNA storage. Under the goals of errors rate reduction and errors correction, to ensure the high storage density of DNA storage, researchers devote to finding the optimal coding coding techniques.

Overall, the aims of this thesis are to deal with the issues occurred in DNA storage, we propose novel coding coding strategies for DNA sequences and primers, respectively, which is helpful for reducing errors rate, error correction, and efficient random-access in DNA storage. The specific coding strategies are listed as follows.

- To minimize errors in DNA storage, a novel coding scheme is proposed to

construct DNA sequences with a proper balanced GC content. For meet the high information density, the code design with variable parity length is considered based on the data parts, which can efficiently reduce average parity lengths .

- To correct the errors and reduce the errors rate in DNA storage, a new single insertion/deletion nonbinary error correction code with the proper maximum run-length constraint is proposed for the construction of DNA sequences. For the proposed code, we design the fixed maximum run-length in the parity sequence of the proposed code to be three. Additionally, the last parity symbol and the first message symbol are always different. Hence, the overall maximum run-length of the output codeword is guaranteed to be three when the maximum run-length of the message sequence is three.
- To efficiently random-access in synthesized DNA strands, we propose a code design by combining weakly mutually uncorrelated codes with the maximum run-length constraint for the construction of DNA primers. Moreover, we also explore the weakly mutually uncorrelated codes to satisfy combinations of maximum run-length constraint with more constraints such as being almost-balanced and having large Hamming distance, which are also efficient constraints in primer design for random-access in DNA storage systems.

1.2 DNA storage: system model

In nature, the genetic information of organisms is composed of DNA, which consists of four bases: A, T, C, and G. DNA storage employs DNA sequences as a medium, using specific encoding strategies to convert binary files of text, images, sounds, and videos into corresponding DNA sequences of a certain length. In consideration of the cost of DNA synthesis and sequencing, the length of synthesized sequences

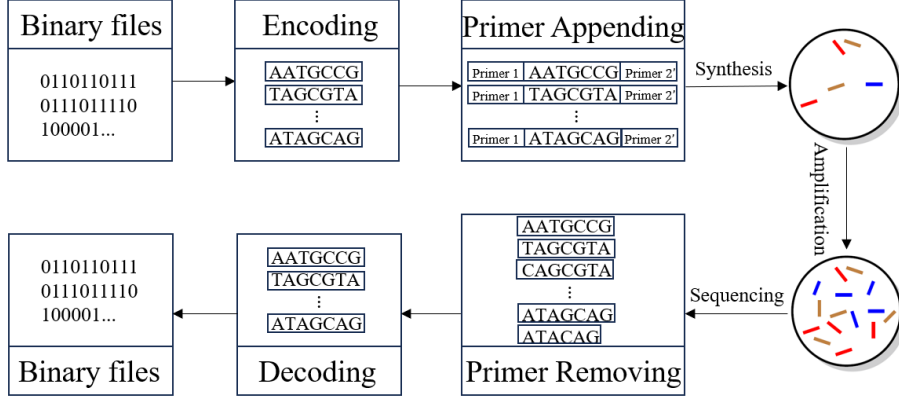


Figure 1.1: The fundamental model of DNA storage systems.

are in range of $10^2 - 10^3$ nucleotide(nt). In Fig. 1.1, we present the fundamental model of DNA storage systems.

During the writing (synthesis) process, the binary files are encoded into DNA sequences. Then, to facilitate random-access to these sequences, pairs of primers are appended to both heads and ends of each DNA sequence, resulting in the entire DNA strands. Finally, DNA strands are synthesized by various DNA synthesis technologies and stored in either *vitro* (DNA pool) or *vivo*. In the reading (sequencing) process, a biological technique called Polymerase Chain Reaction (PCR) is employed to duplicate and amplify synthesized DNA strands, which allows a single stored DNA strand to be duplicated in a short time and increases its quantity exponentially. After obtaining the DNA strands that may contain errors, the primers are attempted to be removed from the DNA strands, and the resulting DNA sequences are decoded back into the original binary files. The details for each step can be checked in [32].

From Fig. 1.1, The process of DNA storage mainly involves synthesis, PCR amplification, and sequencing, etc. This process can be regarded as a channel, which primarily encompasses three types of errors: insertion, for instance, $AACAA \rightarrow AACCATATA$, deletion, for instance, $AACAA \rightarrow AACAA$, and substi-

tution, for instance, $AACAA \rightarrow ATCAT$. Fortunately, the studies [22, 21] had indicated that the probability of errors is in order of magnitude of 10^{-6} , which gives researchers confidence in the future of DNA storage. Since the variety of combinations of errors that can occur during the DNA synthesis and sequencing processes, the exact channel model in DNA storage is determined so far. For the Illumina sequencing technology [17, 22, 21], the studies [22, 21] proposed an asymmetric channel model for DNA storage based on the distribution of the probabilities of the observed insertion, deletion, and substitution errors. In contrast, the study [19] simplified the model by considering a binary symmetric channel (BSC) as a representation of DNA storage. Then, the study [43] considered the deletion (erasure) error in the DNA channel model and proposed binary symmetric erasure channel (BSEC) for DNA storage, where the deletion and substitution errors are obtained from statistics in receiver side. In subsequent years, Nanopore sequencing technology [18, 16, 30, 53] is widely used as the next generation sequencing technology, which reads sequence information by measuring the conductance changes of DNA bases. The study [45] develops a channel model for Nanopore sequencing technology in information theory. This serves as a theoretical framework for studies employing the portable and cost-effective Nanopore sequencing technology in the domain of DNA storage.

Errors in DNA storage are not only caused by external errors in synthesis, amplification, and sequencing technologies, but also from inherent structural properties of DNA. For instance, the imbalance of GC content and the long Homopolymer run in the DNA strands are significant contributors to errors [17, 26, 4, 20, 13, 27]. The GC content is the proportion of G and C in the total of DNA strand. Since the the corresponding bases need to be accurately added in the synthesis process, if the GC content in a DNA strand is too high or too low, it will increase the difficulty of synthesis. During the sequencing process, an imbalanced GC content

is difficult to identifying DNA bases in a strand. This issue is particularly obvious in certain sequencing technologies, resulting in signal saturation or confusion in regions with rich GC content, which can affect the accuracy of sequencing. A long Homopolymer run refers to the same base appearing consecutively in a DNA strand, such as GGG...GGG, etc. In certain sequencing techniques, long Homopolymer run in the DNA strands can lead to an inability to accurately determine the number of received bases, resulting in signal confusion and posing a challenge in accurately identifying and recording the sequence. In general, the GC content needs to meet 45% to 55% and 40% to 60% [26, 4] and the homopolymer run [26, 20, 13] does not exceed 3 or 4 nt, in which cases the error rate will be greatly reduced in DNA storage.

In addition, random-access [17, 41, 61, 68] refers to the capability of retrieving specific data or information stored in DNA sequences without the need to sequentially read the entire DNA sequences in DNA pool. Since the DNA strands are stored unordered in the DNA pool, this ability significantly enhances the speed and efficiency of data retrieval compared to reading the entire DNA strands sequentially in the pool. Therefore, ensuring efficient random-access poses a substantial challenge in DNA storage.

1.3 The methodology for DNA storage

In the previous contents, the challenges faced by DNA storage are presented. In here, The methodologies employed to address these challenges are discussed. For correcting errors in DNA storage, the error-correcting codes are utilized. To ensure GC-balance and Homopolymer run constraints, the constrained codes are employed. Additionally, some codes are designed for achieving random-access in DNA storage.

1.3.1 Error-correcting codes for DNA storage

In DNA storage, error correction is crucial. The studies [26, 13, 5] did not use any error-correcting coding or only simple coding strategies were used, so they ultimately failed to fully recover the original stored information. In traditional digital communication, "noise" (which corresponds to errors in DNA storage) also exists in data both at the transmitter and the receiver. To ensure users receive accurate signals, error-correcting codes such as linear block code are explored. Based on this concept, some researchers have begun investigating the application of error-correcting codes in DNA storage. Although this method is feasible, it's important to note that most error-correcting codes are designed primarily for correcting substitution errors. Therefore, for deletion and insertion errors in DNA storage, researchers have adopted a straightforward approach. They erase the entire sequences which occurs deletion and insertion errors, and replace each of them with a binary DNA sequence consisting of all zero bits. This method effectively treats deletion and insertion errors as substitution errors, ultimately allowing them to be rectified by error-correcting codes. Although this approach proves to be one of the most effective ways to address deletion and insertion errors in DNA storage, in order to deal with deletion and insertion errors more efficiently, researchers are starting to explore algebraic error correction codes.

Linear block codes

Linear block codes are a prevalent form of coding used for error correction, they are composed of q -ary symbols, which can be described as a linear subspace formed by n vectors with dimension k over the finite field F_q . The redundancy of such codes is given by $n - k$. In terms of digital communication, there are a lot of linear block codes has been deeply studied such as Repetition codes, Hamming codes, Bose–Chaudhuri–Hocquenghem (BCH) codes, Reed–Solomon (RS) codes,

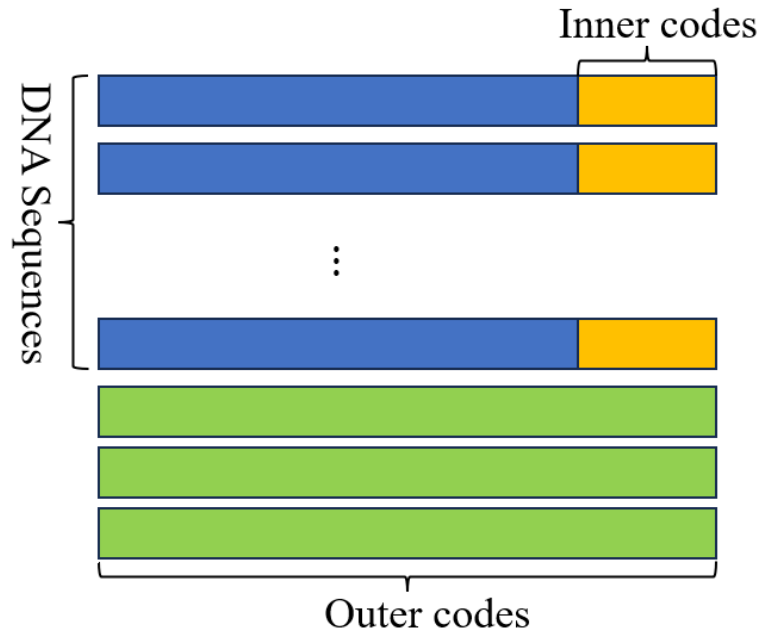


Figure 1.2: The form of linear block codes apply in DNA storage.

and Low-density parity-check (LDPC) codes, etc. Each of these linear block codes exhibits outstanding performance in error correction for digital communication.

In [27], the RS codes were initially applied into DNA storage as inner codes and outer codes, respectively. The authors of [27] not only successfully recovered the original data from the received DNA sequence through RS codes, but also introduced a novel form of linear block codes applied in DNA storage, as outlined in Fig 1.2. This coding scheme continues to find extensive application in subsequent research. To enhance the error-correction capabilities even further, it is logical to employ contemporary coding techniques like LDPC codes [44], which can achieve the channel capacity limit. Since balanced GC-content (45%–55%) and short homopolymer length (less than 3 or 4 nt) are essential for DNA synthesis/sequencing, Deng *et al.* proposed variable-length run-length limited codes combined with LDPC codes [15]. Also, Chandak *et al.* proposed a DNA storage system with LDPC codes [19], where a simple channel model with Poisson dis-

tribution was considered. A machine learning based DNA basecaller [9] was also proposed, which uses convolutional code in DNA storage.

Deletion correction codes

Deletion correction codes are a class of error-correcting codes specifically designed to address deletion errors in data storage systems. These codes utilize algebraic techniques to effectively correct deletion errors, which involve the loss of one or more symbols in a sequence. The study [63] have proven that the deletion correction codes can also correct insertion errors.

The concept of a Varshamov-Tenengolts (VT) code [63], also known as a binary single insertion/deletion error correction (SIDECE) code, was initially introduced in 1965 as shown in Definition 1.3.1.

Definition 1.3.1 ([63]). Given an integer $a \in [0, n]$, for $n \geq 3$ and the binary sequences $\mathbf{c} = (c_1, c_2, \dots, c_n)$ of length n , the VT codes are defined as

$$VT_a(n) = \{\mathbf{c} \in F_2^n : \sum_{i=1}^n i \cdot c_i \equiv a \pmod{(n+1)}\}. \quad (1.1)$$

Levenshtein adapted the binary VT code into an extended VT (EVT) code [40], which is capable of rectifying a single insertion, or deletion, or substitution error. With the in-depth study of deletion correction codes, more and more deletion correction codes [23, 56, 28, 6, 31, 23, 55, 54] have been proposed and the different kinds of the deletion errors can be corrected. For instance, the multiple deletion/insertion (t arbitrary deletion) errors correction codes have been still an open issue, many works [6, 31, 23, 55, 54] have devoted into constructing error correction codes with the optimal redundancy for t arbitrary deletion/insertion errors. The most current and advanced results for t arbitrary deletion/insertion errors correcting code could be found in [54].

1.3.2 Constrained codes for DNA storage

Constrained codes in DNA storage, refer to specialized coding strategies that incorporate specific constraints during the encoding process. These constraints are designed to address particular challenges or characteristics inherent to DNA storage systems. The primary purpose of constrained codes in DNA storage is to guarantee that the encoded DNA sequence adheres to the natural characteristics of DNA, thus ensuring robust and error-resistant data retrieval. This includes considerations such as GC-content balancing and handling long homopolymer runs. An effective constrained codes should proficiently transform binary data into DNA sequences, adhering to biochemical constraints, all while maintaining a high code rate (bits per nucleotide/base).

The methods to meet the constraint in [13] is that a direct mapping of bits 0 and 1 to A/G and C/T, effectively mitigating biased GC content and avoiding long homopolymer runs. However, this method exhibited a low potential of code rate of only 1 bit/nt, falling significantly less than the upper bound of almost 2 bits/nt.

Subsequently, the technique outlined in [26] introduced a ternary-based mapping strategy, thereby increasing the potential of code rate to $\log_2 3$ bits/nt. Building upon this, the authors of in [34] further refined their methodology, developing into the properties and constructions of constrained codes while considering both homopolymer runs and balanced GC content constraints.

More recently, in [57], a novel construction method for devising bio-constrained codes was introduced. This approach achieved a code rate of 1.90 bits/nt, which achieves a higher coding rate. The most current and advanced results for constrained codes could be found in [49] which can achieve a code rate of 1.9333 bits/nt when the length of sequences is 300 nt.

1.3.3 Codes for random-access in DNA storage

Codes for random-access in DNA storage refers to specialized encoding techniques designed to facilitate efficient and precise retrieval of specific data from a pool of stored DNA sequences. The codes are engineered to enable direct access to desired information without the need to sequentially read through the entire DNA sequences.

To achieve random-access to the stored data in the DNA pool, Yazdi *et al* [73]. proposed a new random-access method that appends two address sequences, called ‘primers’, to the head and tail of the synthesized DNA sequences. Yazdi *et al.* [76] first designed mutually uncorrelated (MU) codes as primers in DNA storage systems.

1.4 Contribution and layout

The dissertation consists of five chapters structured as follows, the main contributions of this dissertation are summarized in Fig 1.3.

In Chapter 1, we sum up all the motivation and fundamental knowledge relevant to DNA Storage, and related knowledge such as error correction codes, constrained codes, and the codes for random-access in DNA storage. Then, we show the outline of the dissertation.

In Chapter 2, we propose a novel general encoding algorithm to construct GC-balanced DNA codes by using various parity formats and flipping patterns according to the weight distribution of binary data. To validate the effectiveness of our proposed encoding algorithms, we compare the average parity lengths of GC-balanced DNA codes constructed by our proposed encoding algorithm with those in previous works. The comparison results show that the average parity lengths of our proposed encoding algorithm are significantly smaller than parity lengths

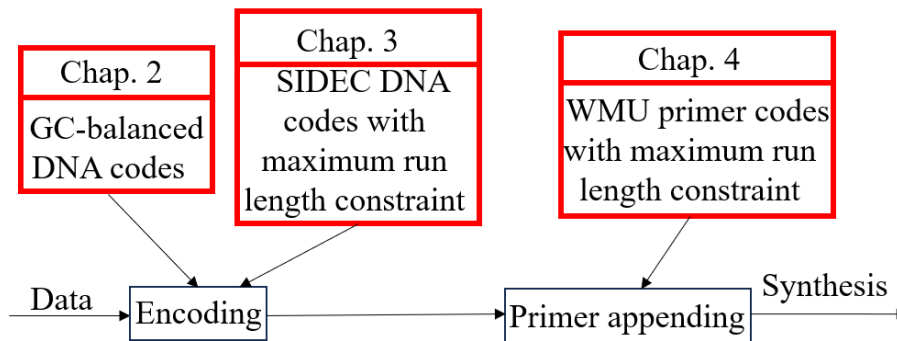


Figure 1.3: The contribution of the dissertation.

of previous works for DNA sequences with GC contents ranging from 45–55% or 40–60%.

In chapter 3, We propose a new construction of a q -ary systematic SIDEC code whose codewords meet the maximum run-length constraint to satisfy the error correction and bio-constrained for DNA storage simultaneously. For the proposed code, we propose systematic encoding algorithm with the parity sequence whose the maximum run-length is always not greater than three.

In chapter 4. we propose a primer codes construction with the crucial maximum run-length constraint. Based on our proposed primer codes with a maximum run-length constraint, we can enhance the codes through simple methods to satisfy the combinations of maximum run-length constraint with other beneficial constraints for primer design, such as GC-balanced and Hamming distance. To apply our proposed codes to primer design in DNA storage, we modify our proposed codes with the maximum run-length constraint to implement various code lengths.

Finally, in chapter 5, we conclude the dissertation and discuss possible future research directions.

Chapter 2

New construction of balanced codes for DNA storage

2.1 Introduction

Most of the advantages of DNA storage have been presented, the DNA sequences are prone to errors during synthesis and sequencing processing [26, 4, 32, 71], which can affect the reliability of the stored data. Some studies [32, 34] have reported that an imbalance in the ratio of GC contents in DNA sequences can lead to errors. However, the studies [20, 32, 57] have suggested that the 45–55% or 40–60% GC contents in DNA sequences can enable a low error rate during synthesis and sequencing processing. Therefore, the efficient way to construct GC-balanced DNA sequences has been a major concern for DNA storage.

Various designs have been studied to generate binary balanced or ϵ -balanced codewords. Look-up tables methods [72, 35] and enumeration techniques [69] were the common ways to construct binary balanced codes. Some studies [57, 64, 42] have also proposed efficient encoding algorithms to construct the GC-balanced codewords for DNA storage by enumeration techniques. Although these methods

can achieve a high code rate, the complexity of codes construction increases with the increase of code length, which limits their practicability. To address this issue, Knuth [36] proposed a simple and linear-time algorithm that converts any binary sequence into a balanced codeword by using a parity tag. Then, Nguyen *et al.* [49] proposed a method to construct binary ϵ -balanced codes by modifying the Knuth methods and applied the binary ϵ -balanced codes to design of GC-balanced DNA codes.

To maintain the high information density of DNA storage, it is crucial to develop constructions of GC-balanced DNA codes with lower parity length. We find a space that can effectively reduce the parity length of GC-balanced codes from weights distribution of binary data and propose novel general encoding algorithms to construct GC-balanced DNA codes. Since the binary ϵ -balanced encoding functions in [36] and [49] are applied into our proposed encoding algorithms, we compare the average parity lengths of the proposed GC-balanced DNA codes with those of the referenced codes [36],[49]. We provide some results to indicate that the average parity lengths of our proposed encoding algorithm decrease dramatically as the value of ϵ increases.

2.2 Preliminaries

2.2.1 Related works for binary ϵ -balanced code construction

We briefly present two referenced methods to construct binary ϵ -balanced codes in [36] and [49], where the parity lengths of [36] and [49] depend on the length n and the value of ϵ , respectively. Then, we also present the relations between binary ϵ -balanced codes and the GC-balanced DNA codes.

In [36], Knuth proposed a linear-time algorithm that converts any binary sequence \mathbf{x} of length n to an exact balanced codeword whose weight is $\frac{n}{2}$ for an even value of n . Although Knuth algorithm was originally designed to construct binary balanced codes, it is still applicable to the general ϵ -balanced codes construction for $0 \leq \epsilon \leq 0.5$. We summarize the Knuth algorithm for constructing general ϵ -balanced codes in Lemma 2.1.1.

Lemma 2.2.1. Let a set S_n be $S_n = \{1, 2, 3, \dots, n\}$. For any binary sequence \mathbf{x} of length n and $0 \leq \epsilon \leq 0.5$, there always exists a smallest element $t_{(1)} \in S_n$ to convert the sequence \mathbf{x} to be a binary ϵ -balanced codeword \mathbf{c} by flipping the first $t_{(1)}$ bits of \mathbf{x} . The binary ϵ -balanced encoding function in [36] is denoted as $\mathcal{E}_1(\cdot)$. Then, the function can convert a binary sequence \mathbf{x} to be a binary ϵ -balanced codeword \mathbf{c} as

$$\mathbf{c} = \mathcal{E}_1(\mathbf{x}) = (\overline{\mathbf{x}_1^{t_{(1)}}}, \mathbf{x}_{t_{(1)}+1}^n) \quad \text{for } t_{(1)} \in S_n. \quad (2.1)$$

Let $\mathcal{E}_1^{-1}(\cdot)$ be a corresponding binary decoding function that recovers the original sequence \mathbf{x} from the ϵ -balanced codeword \mathbf{c} . A decoder can recover the original sequence \mathbf{x} from \mathbf{c} if the index $t_{(1)}$ is transferred to the decoder. Then, a short parity sequence $\mathbf{r}_{(1)}$ of length $R_{(1)}$ is required to represent the value of $t_{(1)}$. Then, the parity length $R_{(1)}$ to represent the value $t_{(1)}$ is given as

$$R_{(1)} = \lceil \log |S_n| \rceil = \lceil \log n \rceil. \quad (2.2)$$

Note that if the original sequences \mathbf{x} are already ϵ -balanced, then the flipped codewords $\mathbf{c} = \mathcal{E}_1(\mathbf{x})$ in (2.1) are also ϵ -balanced. To understand Lemma 2.2.1, we present a simple example of the binary ϵ -balanced code construction.

Example 2.2.2. A sequence \mathbf{x} of length $n = 10$ is given by

$$\mathbf{x} = (0, 1, 0, 1, 1, 1, 1, 1, 1, 0).$$

We consider that ϵ is 0.1. Then, the sequence \mathbf{x} is not 0.1-balanced. For $t_{(1)}$ which is from one to four, the flipped codeword $\mathbf{c} = \mathcal{E}_1(\mathbf{x})$ is still not 0.1-balanced. When $t_{(1)} = 5$, the flipped codeword \mathbf{c} is obtained as

$$\mathbf{c} = \mathcal{E}_1(\mathbf{x}) = (\overline{\mathbf{x}_1^5}, \mathbf{x}_6^{10}) = (1, 0, 1, 0, 0, 1, 1, 1, 1, 0).$$

Since $|w(\mathbf{c}) - 0.5n|$ is $\lfloor \epsilon n \rfloor = \lfloor 0.1 \times 10 \rfloor = 1$, the codeword \mathbf{c} is 0.1-balanced.

The authors of [49] proposed a more efficient way to construct ϵ -balanced codes by modifying the Knuth algorithm for $0 < \epsilon \leq 0.5$. In Lemma 2, we summarize the construction of binary ϵ -balanced codes in [49].

Lemma 2.2.3. Let a set $S_{\epsilon,n}$ be $S_{\epsilon,n} = \{0, 2\lfloor \epsilon n \rfloor, 4\lfloor \epsilon n \rfloor, \dots, n\}$. For any binary sequence \mathbf{x} of length n and $0 < \epsilon \leq 0.5$, an element $t_{(2)} \in S_{\epsilon,n}$ always exists such that the binary codeword \mathbf{c} becomes ϵ -balanced by flipping the first $t_{(2)}$ bits of \mathbf{x} . Then, we denote the binary ϵ -balanced encoding function as $\mathcal{E}_2(\cdot)$ in [49], which can convert \mathbf{x} to be the binary ϵ -balanced codeword \mathbf{c} as

$$\mathbf{c} = \mathcal{E}_2(\mathbf{x}) = (\overline{\mathbf{x}_1^{t_{(2)}}}, \mathbf{x}_{t_{(2)}+1}^n) \quad \text{for } t_{(2)} \in S_{\epsilon,n}. \quad (2.3)$$

The $\mathcal{E}_2^{-1}(\cdot)$ corresponds to a binary decoding function. Additionally, a short parity sequence $\mathbf{r}_{(2)}$ of length $R_{(2)}$ is used to represent the element $t_{(2)}$ in the set $S_{\epsilon,n}$, which can recover the original sequences of \mathbf{x} from \mathbf{c} . Then, the parity length $R_{(2)}$ is calculated as

$$R_{(2)} = \lceil \log |S_{\epsilon,n}| \rceil = \lceil \log(\lfloor \frac{1}{2\epsilon} \rfloor + 1) \rceil \quad \text{for } 0 < \epsilon \leq 0.5. \quad (2.4)$$

Note that if the original sequences \mathbf{x} are ϵ -balanced, the flipped codewords $\mathbf{c} = \mathcal{E}_2(\mathbf{x})$ in (2.3) are also ϵ -balanced for $t_{(2)} = 0$. The relation between binary ϵ -balanced codes and the GC-balanced DNA codes were also presented in [49], which

is summarized in Lemma 2.2.4.

Lemma 2.2.4. Suppose that there are two binary sequences $\mathbf{z}^o = (z_1^o, z_2^o, \dots, z_n^o)$ and $\mathbf{z}^e = (z_1^e, z_2^e, \dots, z_n^e)$ of length n . For $i \in [1, n]$, the elements z_i of a DNA codeword \mathbf{z} are mapped from the elements of two binary sequences as

$$z_i = \phi(z_i^o, z_i^e) = \begin{cases} \text{A} & \text{if } (z_i^o, z_i^e) = (0, 0), \\ \text{T} & \text{if } (z_i^o, z_i^e) = (0, 1), \\ \text{G} & \text{if } (z_i^o, z_i^e) = (1, 0), \\ \text{C} & \text{if } (z_i^o, z_i^e) = (1, 1). \end{cases} \quad (2.5)$$

The inverse mapping of a DNA symbol to two binary bits is denoted as $\phi^{-1}(\cdot)$. Then, if the binary sequence \mathbf{z}^o is ϵ -balanced, the DNA codeword \mathbf{z} is also GC-balanced.

According to Lemma 2.2.4, we present the GC-balanced codes constructed by the binary ϵ -balanced codes in [36] and [49]. It is assumed two binary sequences $\mathbf{a}_{(1)}$ of length n and $\mathbf{b}_{(1)}$ of length $n - R_{(1)}$ are extracted from the binary data in [36]. Similarly, two binary sequences $\mathbf{a}_{(2)}$ of length n and $\mathbf{b}_{(2)}$ of length $n - R_{(2)}$ are given from the binary data in [49]. Then, the sequences $\mathbf{a}_{(1)}$ and $\mathbf{a}_{(2)}$ are inputs of the binary ϵ -balanced encoding function $\mathcal{E}_1(\mathbf{a}_{(1)})$ in (2.1) and $\mathcal{E}_2(\mathbf{a}_{(2)})$ in (2.3), respectively. Then, the binary ϵ -balanced codewords $\mathbf{c}_{(1)}$ and $\mathbf{c}_{(2)}$, and parity sequences $\mathbf{r}_{(1)}$ and $\mathbf{r}_{(2)}$ can be obtained for [36] and [49], respectively. Finally, if we set $\mathbf{z}^o = \mathbf{c}_{(1)}$ and $\mathbf{z}^e = (\mathbf{b}_{(1)}, \mathbf{r}_{(1)})$ or $\mathbf{z}^o = \mathbf{c}_{(2)}$ and $\mathbf{z}^e = (\mathbf{b}_{(2)}, \mathbf{r}_{(2)})$, respectively, the GC-balanced DNA codewords \mathbf{z} can be obtained by the mapping rule $\phi(\cdot)$ in (2.5). Therefore, the parity lengths for GC-balanced DNA codes constructed by [36] and [49] are also $R_{(1)}$ in (2.2) and $R_{(2)}$ in (2.4), respectively.

2.3 GC-balanced codes construction

We first explain a proposed DNA storage system in Fig. 2.1. Then, we present the encoding and decoding algorithms for GC-balanced DNA codes construction in detail, respectively, which are essential parts of the proposed DNA storage systems.

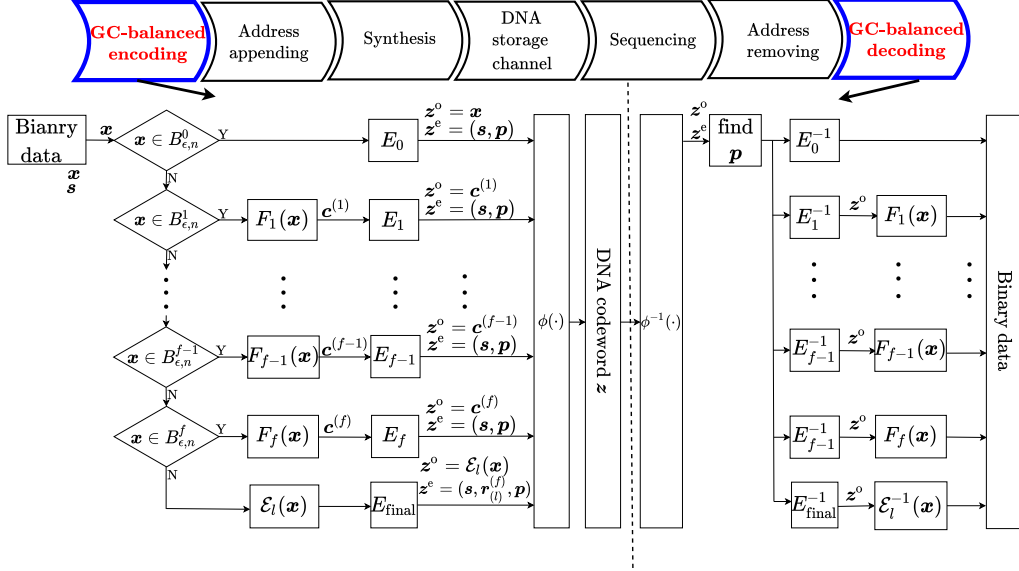


Figure 2.1: System model of the proposed GC-balanced code construction in DNA storage.

2.3.1 Overall encoding processes

The overall system model of the proposed GC-balanced code construction for DNA storage is shown in Fig. 2.1. Let f be the maximum number of flipping levels. Two binary sequences \mathbf{x} of length n and \mathbf{s} of variable length in binary data are used for generating one GC-balanced codeword. The length of \mathbf{s} is determined by flipping levels and parity lengths. $B_{\epsilon, n}^0$ at level zero in Fig. 2.1 is a set that contains the all binary ϵ -balanced sequences of length n among all 2^n sequences. If the sequence \mathbf{x} is in $B_{\epsilon, n}^0$, two binary sequences \mathbf{x} and \mathbf{s} of length $n - 1$ are moved to E_0 . Then, E_0 produces a parity sequence \mathbf{p} of length one and outputs two binary codewords

$\mathbf{z}^o = \mathbf{x}$ and $\mathbf{z}^e = (\mathbf{s}, \mathbf{p})$. If not, the sequence \mathbf{x} is checked to determine whether it belongs to the set $B_{\epsilon,n}^1$ in the first level.

Let $B_{\epsilon,n}^1$ be a set that does not contain any sequence in $B_{\epsilon,n}^0$ and consists of the binary sequences which become ϵ -balanced after flipping $F_1(\mathbf{x})$. If the sequence \mathbf{x} is in $B_{\epsilon,n}^1$, the flipped ϵ -balanced codeword $\mathbf{c}^{(1)}$ and sequence \mathbf{s} of length $n - 2$ are the inputs of E_1 . Then, E_1 produces $\mathbf{z}^o = \mathbf{c}^{(1)}$ and $\mathbf{z}^e = (\mathbf{s}, \mathbf{p})$, where \mathbf{p} is a parity sequence of length two. If not, the sequence \mathbf{x} is determined whether in the set $B_{\epsilon,n}^2$.

For $i \in [2, f]$, let $B_{\epsilon,n}^i$ be a set that does not contain any sequence in $\cup_{j=0}^{i-1} B_{\epsilon,n}^j$ and consists of the binary sequences which become ϵ -balanced after flipping $F_i(\mathbf{x})$. The encoding steps for $i \in [2, f]$ are similar to ones at the first level and outputs of E_i are $\mathbf{c}^{(i)}$, \mathbf{s} of length $n - i - 1$, and \mathbf{p} of length $i + 1$.

Finally, if \mathbf{x} is not included in $\cup_{j=0}^f B_{\epsilon,n}^j$, the binary ϵ -balanced encoding function $\mathcal{E}_l(\cdot)$ in (2.1) or (2.3) for $l = 1$ or $l = 2$ is used. Then, the ϵ -balanced codeword $\mathcal{E}_l(\mathbf{x})$ and parity sequence $\mathbf{r}_{(l)}^{(f)}$ of length $R_{(l)}^{(f)}$ are obtained. E_{final} outputs the sequences $\mathbf{z}^o = \mathcal{E}_l(\mathbf{x})$ and $\mathbf{z}^e = (\mathbf{s}, \mathbf{r}_{(l)}^{(f)}, \mathbf{p})$, where the lengths of binary sequences \mathbf{s} and \mathbf{p} are $n - R_{(l)}^{(f)} - f - 1$ and $f + 1$, respectively.

2.3.2 Flipping functions

We firstly consider a $F_1(\mathbf{x})$ flipping function and subsequently, we discuss the $F_i(\mathbf{x})$ flipping functions for $i \in [2, f]$ for length $n = 2^m$, where a positive integer m is defined as $f \leq m$. Finally, we present the flipping functions for various values of n .

If the sequence \mathbf{x} is not in $B_{\epsilon,n}^0$, the flipping function $F_1(\mathbf{x})$ outputs $\mathbf{c}^{(1)}$ from \mathbf{x} . To reduce the average length of parity, $F_1(\mathbf{x})$ should be designed to maximize the size of $B_{\epsilon,n}^1$. For random $\mathbf{x} \notin B_{\epsilon,n}^0$, we find that any flipping pattern $F_1(\mathbf{x})$ with the same number of flipped bits yields the same size of $B_{\epsilon,n}^1$. Therefore, we

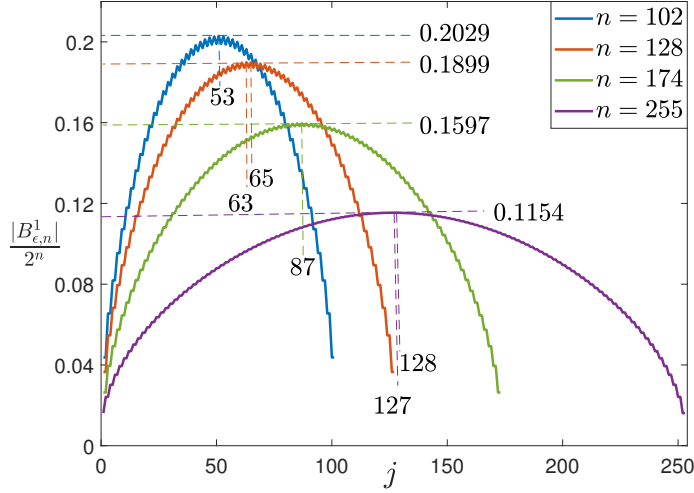


Figure 2.2: The ratios $\frac{|B_{\epsilon,n}^1|}{2^n}$ for the binary sequences which become $\mathbf{c}^{(1)}$ for $n = 102$, 128, 174, and 255 and $\epsilon = 0.05$.

only consider that $F_1(\mathbf{x})$ is flipping the first j bits of \mathbf{x} . If the sequences \mathbf{x} are not ϵ -balanced, the sequences after flipping all bits of \mathbf{x} be also not ϵ -balanced. Then, the values of j should be considered as $j \in [1, n - 1]$. In Fig. 2.2, the ratios $\frac{|B_{\epsilon,n}^1|}{2^n}$ for the binary sequences which become the binary ϵ -balanced codewords $\mathbf{c}^{(1)}$ after flipping $F_1(\mathbf{x})$ are shown when $n = 102$, 128, 174, and 255 and $\epsilon = 0.05$.

In Fig. 2.2, the maximum values of $\frac{|B_{\epsilon,n}^1|}{2^n}$ and the corresponding values j are illustrated. In the case of $n = 102$ and 174, the maximum values of $\frac{|B_{\epsilon,n}^1|}{2^n}$ are 0.2029 and 0.1597 at $j = 53$ and 87, respectively, which correspond to $\frac{n}{2}$ bits. Similarly, for $n = 255$, the maximum values of $\frac{|B_{\epsilon,n}^1|}{2^n}$ is 0.1154 at $j = 127$ or 128, corresponding to $\lfloor \frac{n}{2} \rfloor$ or $\lceil \frac{n}{2} \rceil$ bits, respectively. However, for $n = 128$, at $j = 63$ and 65, which is not corresponding to $\frac{n}{2}$ bits, and the maximum $\frac{|B_{\epsilon,n}^1|}{2^n}$ is 0.1899. At $j = 64$ for $n = 128$, the value of $\frac{|B_{\epsilon,n}^1|}{2^n}$ is 0.1872. The gap between 0.1872 and the maximum $\frac{|B_{\epsilon,n}^1|}{2^n}$ is 0.0027, indicating a relatively small difference. Based on the results in Fig. 2.2, the $\frac{|B_{\epsilon,n}^1|}{2^n}$ can be a maximum value or almost near a maximum value by flipping the first $\lceil \frac{n}{2} \rceil$ bits for various lengths n . Therefore, we finally

determine the $F_1(\mathbf{x})$ for various lengths n as flipping the first $\lceil \frac{n}{2} \rceil$ bits, and the ϵ -balanced codewords $\mathbf{c}^{(1)}$ obtained from $\mathbf{x} \in B_{\epsilon,n}^1$ by $F_1(\mathbf{x})$ are

$$\mathbf{c}^{(1)} = (\overline{\mathbf{x}}_1^{\lceil \frac{n}{2} \rceil}, \mathbf{x}_{\lceil \frac{n}{2} \rceil+1}^n).$$

If the sequence \mathbf{x} is not in $\cup_{j=0}^1 B_{\epsilon,n}^j$, $F_2(\mathbf{x})$ are used to produce binary ϵ -balanced codeword $\mathbf{c}^{(2)}$. To find the optimal flipping pattern, we need to search all possible patterns for $F_2(\mathbf{x})$ to maximize the size of $B_{\epsilon,n}^2$. However, it is not feasible to determine the $F_2(\mathbf{x})$ for various lengths n since there are too many candidates. Therefore, we attempt to develop a systematic approach for designing the flipping methods. We first consider n as $n = 2^m$ and later discuss flipping pattern for general n .

For $n = 2^m$, from the $(m+1) \times n$ generator matrix G of the first-order Reed-Muller codes [50], which is given as

$$G = \begin{bmatrix} \mathbf{g}^{(0)} \\ \mathbf{g}^{(1)} \\ \mathbf{g}^{(2)} \\ \vdots \\ \mathbf{g}^{(m-1)} \\ \mathbf{g}^{(m)} \end{bmatrix} = \begin{bmatrix} \mathbf{1}_1^n \\ (\mathbf{1}_1^{\frac{n}{2}}, \mathbf{0}_1^{\frac{n}{2}}) \\ (\mathbf{1}_1^{\frac{n}{4}}, \mathbf{0}_1^{\frac{n}{4}}, \mathbf{1}_1^{\frac{n}{4}}, \mathbf{0}_1^{\frac{n}{4}}) \\ \vdots \\ (\mathbf{1}_1^2, \mathbf{0}_1^2, \dots, \mathbf{1}_1^2, \mathbf{0}_1^2) \\ (1, 0, 1, 0, \dots, 1, 0, 1, 0) \end{bmatrix}, \quad (2.6)$$

where the vectors $\mathbf{1}_i^j$ and $\mathbf{0}_i^j$ in (2.6) for $1 \leq i \leq j \leq n$ are all ones and zeros vectors of length $j - i + 1$. Since the generator vector $\mathbf{g}^{(0)}$ in (2.6) stands for flipping all bits in the proposed encoding, $\mathbf{g}^{(0)}$ is not used. The generator vector $\mathbf{g}^{(1)}$ corresponds to the flipping $F_1(\mathbf{x})$ and the flipped sequence $\mathbf{c}^{(1)}$ is expressed as

$$\mathbf{c}^{(1)} = \mathbf{x} \oplus \mathbf{g}^{(1)} = (\overline{\mathbf{x}}_1^{\frac{n}{2}}, \mathbf{x}_{\frac{n}{2}+1}^n).$$

where \oplus is a bitwise exclusive-or operation. Similar to $\mathbf{c}^{(1)}$, for the sequences \mathbf{x} of length $n = 2^m$, the flipped sequences $\mathbf{c}^{(i)}$ are obtained as

$$\mathbf{c}^{(i)} = \mathbf{x} \oplus \mathbf{g}^{(i)} \quad \text{for } i \in [2, f].$$

Therefore, the codewords $\mathbf{c}^{(i)}$ obtained from the sequences \mathbf{x} by $F_i(\mathbf{x})$ are

$$\mathbf{c}^{(i)} = (\overline{\mathbf{x}}_1^{\frac{n}{2^i}}, \mathbf{x}_{\frac{n}{2^i}+1}^{\frac{n}{2^{i-1}}}, \dots, \overline{\mathbf{x}}_{n-\frac{n}{2^{i-1}}+1}^{\frac{n-n}{2^i}}, \mathbf{x}_{n-\frac{n}{2^i}+1}^n).$$

Since it is not feasible to exhaustively search the flipping functions $F_i(\mathbf{x})$ for $i \in [2, f]$, it is clear that our determined $F_i(\mathbf{x})$ flipping functions are not optimal, and to find optimal flipping functions $F_i(\mathbf{x})$ will be an interesting future work.

For $n \neq 2^m$, similar to $F_1(\mathbf{x})$ to support various lengths, the flipping $F_i(\mathbf{x})$ for $i \in [2, f]$ are designed to be as similar as possible to generator functions $\mathbf{g}^{(i)}$ in (2.6) by using ceiling and floor functions. Finally, $\mathbf{c}^{(i)}$ for the various lengths are determined as

$$\mathbf{c}^{(i)} = (\overline{\mathbf{x}}_1^{\lceil \frac{n}{2^i} \rceil}, \mathbf{x}_{\lceil \frac{n}{2^i} \rceil + 1}^{\lceil \frac{n}{2^{i-1}} \rceil}, \dots, \overline{\mathbf{x}}_{\lceil \frac{n}{2} \rceil - \lceil \frac{n}{2^i} \rceil + 1}^{\lceil \frac{n}{2} \rceil - \lceil \frac{n}{2^{i-1}} \rceil + 1}, \mathbf{x}_{\lceil \frac{n}{2} \rceil - \lceil \frac{n}{2^i} \rceil + 1}^{\lceil \frac{n}{2} \rceil}, \\ \overline{\mathbf{x}}_{\lceil \frac{n}{2} \rceil + 1}^{\lceil \frac{n}{2} \rceil + \lfloor \frac{n}{2^i} \rfloor}, \mathbf{x}_{\lceil \frac{n}{2} \rceil + \lfloor \frac{n}{2^i} \rfloor + 1}^{\lceil \frac{n}{2} \rceil + \lfloor \frac{n}{2^{i-1}} \rfloor}, \dots, \overline{\mathbf{x}}_{\lceil n \rceil - \lfloor \frac{n}{2^i} \rfloor + 1}^{\lceil n \rceil - \lfloor \frac{n}{2^{i-1}} \rfloor + 1}, \mathbf{x}_{\lceil n \rceil - \lfloor \frac{n}{2^i} \rfloor + 1}^n).$$

2.3.3 Encoding and mapping

If the binary sequences \mathbf{x} are already ϵ -balanced, which means $\mathbf{x} \in B_{\epsilon, n}^0$, the encoding E_0 produces two binary codewords \mathbf{z}^o and \mathbf{z}^e . Then, the encoding E_0 is determined as

$$\begin{aligned} \mathbf{z}^o &= \mathbf{x} \\ \mathbf{z}^e &= (\mathbf{s}, 0), \end{aligned} \tag{2.7}$$

where \mathbf{s} denotes the data with length $n - 1$ and 0 is a bit zero. The parity length of the encoding E_0 is one and parity is $\mathbf{p} = 0$.

For the binary ϵ -balanced codewords $\mathbf{c}^{(1)}$, the encoding E_1 is defined as

$$\begin{aligned}\mathbf{z}^o &= \mathbf{c}^{(1)} \\ \mathbf{z}^e &= (\mathbf{s}, 0, 1),\end{aligned}\tag{2.8}$$

where \mathbf{s} is the data with length $n - 2$ and $\mathbf{p} = (0, 1)$ is parity. The last bit one in parity \mathbf{p} denotes that the original binary sequence which becomes \mathbf{z}^o is not in $B_{\epsilon,n}^0$, and bit zero in parity stands for the original binary sequence which becomes \mathbf{z}^o is in set $B_{\epsilon,n}^1$.

Similar to E_1 , E_i for $i \in [2, f]$ is determined as

$$\begin{aligned}\mathbf{z}^o &= \mathbf{c}^{(i)} \\ \mathbf{z}^e &= (\mathbf{s}, \mathbf{p}),\end{aligned}\tag{2.9}$$

where \mathbf{s} is the data with length $n - i - 1$ and the parity sequence $\mathbf{p} = (0, \mathbf{1}_1^i)$ of length $i + 1$.

If \mathbf{x} is still not in $\cup_{j=0}^f B_{\epsilon,n}^j$, different encoding scheme needs to produce an ϵ -balanced codeword. Then, the encoding E_{final} is defined as

$$\begin{aligned}\mathbf{z}^o &= \mathcal{E}_l(\mathbf{x}) \\ \mathbf{z}^e &= (\mathbf{s}, \mathbf{r}_{(l)}^{(f)}, \mathbf{1}_1^{f+1}),\end{aligned}\tag{2.10}$$

where $\mathcal{E}_l(\cdot)$ in (2.10) for $l = 1$ or $l = 2$ is the binary ϵ -balanced encoding function denoted in (2.1) or (2.3), the sequence $\mathbf{r}_{(l)}^{(f)}$ is the corresponding parity produced by $\mathcal{E}_l(\cdot)$, and the parity sequence \mathbf{p} is $\mathbf{p} = (\mathbf{1}_1^{f+1})$. We summarize the encoding functions E_0 , E_i for $i = [1, f]$, and E_{final} in Fig. 2.3.

For a given f , the average parity lengths of our proposed GC-balanced DNA

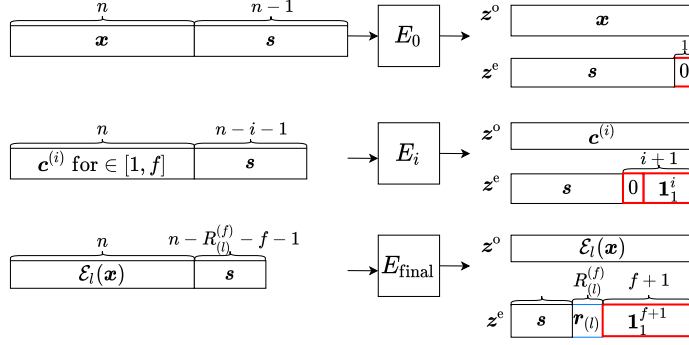


Figure 2.3: Encoding functions E_0 , E_i for $i = [1, f]$, and E_{final} .

code construction, $AR_{(l)}^{(f)}$, can be calculated as

$$AR_{(l)}^{(f)} = \sum_{i=0}^f \left(\frac{|B_{\epsilon, n}^i|}{2^n} (i+1) \right) + \left(1 - \sum_{i=0}^f \frac{|B_{\epsilon, n}^i|}{2^n} \right) (R_{(l)}^{(f)} + f + 1), \quad (2.11)$$

where l is $l = 1$ or $l = 2$ and 2^n represents the total number of the binary sequences of length n .

After obtaining the binary codewords \mathbf{z}^o and \mathbf{z}^e , the GC-balanced DNA codewords \mathbf{z} can be obtained by the mapping rule $\phi(\mathbf{z}^o, \mathbf{z}^e)$ in (2.5). Therefore, we summarize the overall encoding algorithm for GC-balanced DNA codes in Algorithm 1.

2.3.4 Decoding of the proposed GC-balanced codes

We present the overall decoding processes of the proposed GC-balanced codes for DNA storage, which are shown in Fig. 2.1. From the received GC-balanced DNA codeword \mathbf{z} , two binary sequences \mathbf{z}^o and \mathbf{z}^e are obtained by the inverse function $\phi^{-1}(\mathbf{z})$ of the ϕ function in (2.5). Then, from last element z_n^e to z_{n-f}^e in the sequence \mathbf{z}^e , we try to find the index where the first zero occurs. If the index is denoted as σ , the parity \mathbf{p} can be determined as $\mathbf{z}_{n-\sigma}^e$. According to the parity

Algorithm 1: Encoding GC-balanced DNA codes

Input: A fixed value f , and binary data \mathbf{x} and \mathbf{s} .

Output: GC-balanced DNA codewords \mathbf{z} .

```
1 if  $\mathbf{x} \in B_{\epsilon,n}^0$  then
2    $E_0$  produces  $\mathbf{z}^o = \mathbf{x}$  and  $\mathbf{z}^e = (\mathbf{s}, 0)$  in (2.7).
3 else
4   set  $i = 1$  and  $flag = 1$ ,
5   while  $flag$  do
6     Flip  $\mathbf{x}$  according to  $F_i(\mathbf{x})$  and produce  $\mathbf{c}^{(i)}$ .
7     if  $\mathbf{c}^{(i)} \in B_{\epsilon,n}^i$  then
8        $E_i$  produces  $\mathbf{z}^o = \mathbf{c}^{(i)}$  and  $\mathbf{z}^e = (\mathbf{s}, 0, \mathbf{1}_1^i)$  in (2.8) and (2.9),
9        $flag = 0$ .
10    else
11       $i = i + 1$ ,
12      if  $i = f + 1$  then
13         $E_{\text{final}}$  produces  $\mathbf{z}^o = \mathcal{E}_i(\mathbf{x})$  and  $\mathbf{z}^e = (\mathbf{s}, \mathbf{r}_{(l)}^{(f)}, \mathbf{1}_1^{f+1})$  in (2.10),
14         $flag = 0$ .
15 Encode a codeword  $\mathbf{z}$  as  $\mathbf{z} = \phi(\mathbf{z}^o, \mathbf{z}^e)$ .
```

\mathbf{p} , the encoding for sequence \mathbf{z}^o is determined and finally the original binary data can be recovered.

If the parity \mathbf{p} is $\mathbf{p} = \mathbf{z}_n^e = 0$, the decoding E_0^{-1} produces a concatenated sequence by removing the parity \mathbf{p} . Then, the decoding E_0^{-1} is determined as

$$(\mathbf{z}^o, \mathbf{z}_1^{e_{n-1}}) = E_0^{-1}(\mathbf{z}^o, \mathbf{z}^e), \quad (2.12)$$

and according to the parity \mathbf{p} , the sequence \mathbf{z}^o in (2.12) does not undergo any flipping. Therefore, the original binary data is recovered as

$$(\mathbf{x}, \mathbf{s}) = (\mathbf{z}^o, \mathbf{z}_1^{e_{n-1}}). \quad (2.13)$$

When the parity \mathbf{p} is $\mathbf{p} = \mathbf{z}_{n-i}^e = (0, \mathbf{1}_1^i)$ for $i \in [1, f]$, the decoding E_i^{-1} is

Algorithm 2: Decoding GC-balanced DNA codes

Input: The fixed value f and DNA codeword \mathbf{z} .

Output: the original binary data.

- 1 Inverse mapping $\phi^{-1}(\mathbf{z}) \rightarrow$ binary sequences \mathbf{z}^o and \mathbf{z}^e .
 - 2 Determine the parity \mathbf{p} in the subsequence \mathbf{z}_{n-f}^{en} ,
 - 3 **if** $\mathbf{p} = \mathbf{z}_n^e = 0$ **then**
 - 4 └ the original binary data is $(\mathbf{x}, \mathbf{s}) = (\mathbf{z}^o, \mathbf{z}_1^{en-1})$ in (2.13).
 - 5 **else if** $\mathbf{p} = \mathbf{z}_{n-i}^{en} = (0, \mathbf{1}_1^i)$ for $i \in [1, f]$ **then**
 - 6 └ the original binary data is $(\mathbf{x}, \mathbf{s}) = (F_i(\mathbf{z}^o), \mathbf{z}_1^{en-i-1})$ in (2.15).
 - 7 **else**
 - 8 └ the original binary data is $(\mathbf{x}, \mathbf{s}) = (\mathcal{E}_l^{-1}(\mathbf{z}^o), \mathbf{z}_1^{en-R_{(l)}^{(f)}-f-1})$ in (2.17).
-

defined as

$$(\mathbf{z}^o, \mathbf{z}_1^{en-i-1}) = E_i^{-1}(\mathbf{z}^o, \mathbf{z}^e), \quad (2.14)$$

and from the parity \mathbf{p} , it is known that the sequence \mathbf{z}^o in (2.14) is obtained from $F_i(\mathbf{x})$. Therefore, the original binary data is given as

$$(\mathbf{x}, \mathbf{s}) = (F_i(\mathbf{z}^o), \mathbf{z}_1^{en-i-1}). \quad (2.15)$$

Otherwise, if the parity \mathbf{p} is $\mathbf{p} = \mathbf{z}_{n-f}^{en} = \mathbf{1}_1^{f+1}$, the decoding E_{final}^{-1} is determined as

$$(\mathbf{z}^o, \mathbf{z}_1^{en-f-1}) = E_{\text{final}}^{-1}(\mathbf{z}^o, \mathbf{z}^e), \quad (2.16)$$

where the sequence \mathbf{z}^o in (2.16) is obtained from $\mathcal{E}_l(\mathbf{x})$. Therefore, using the parity sequence $\mathbf{z}_{n-R_{(l)}^{(f)}-f}^{en-f-1}$, the original binary data can be obtained by applying the binary ϵ -balanced decoding function $\mathcal{E}_l^{-1}(\cdot)$ as

$$(\mathbf{x}, \mathbf{s}) = (\mathcal{E}_l^{-1}(\mathbf{z}^o), \mathbf{z}_1^{en-R_{(l)}^{(f)}-f-1}). \quad (2.17)$$

We summarize the proposed decoding algorithm for GC-balanced DNA codes in

Algorithm 2.

2.4 Analysis of the average parity lengths for the proposed codes

We present the exact calculation of average parity lengths of our proposed codes construction based on the weights of sequences for $f = 0$ and $f = 1$, respectively. We also derive the average parity lengths by using randomly generated sequences, which are especially convenient for the calculation of the average parity lengths for $f > 1$.

2.4.1 Average parity length for $f = 0$

We first present a simple case of the proposed codes at $f = 0$. For easy understanding, Fig. 2.4. illustrates the encoding and decoding for $f = 0$.

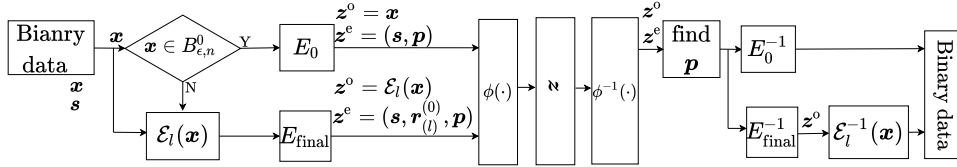


Figure 2.4: Encoding and decoding for $f = 0$.

The average parity length for our proposed GC-balanced DNA codes $AR_{(l)}^{(f)}$ in (2.11) can be rewritten as

$$AR_{(l)}^{(0)} = \frac{|B_{\epsilon,n}^0|}{2^n} \cdot 1 + \left(1 - \frac{|B_{\epsilon,n}^0|}{2^n}\right)(R_{(l)}^{(0)} + 1). \quad (2.18)$$

Since the set $B_{\epsilon,n}^0$ is composed of ϵ -balanced binary sequences among all 2^n

sequences, the size of $B_{\epsilon,n}^0$ for even n can be calculated as

$$|B_{\epsilon,n}^0| = \binom{n}{\frac{n}{2}} + 2 \sum_{i=1}^{\lfloor \epsilon n \rfloor} \binom{n}{\frac{n}{2} + i}. \quad (2.19)$$

For odd n , the size of $B_{\epsilon,n}^0$ is determined as

$$|B_{\epsilon,n}^0| = 2 \sum_{i=0}^{\lfloor \epsilon n \rfloor} \binom{n}{\lceil \frac{n}{2} \rceil + i}. \quad (2.20)$$

As shown in Lemma 2.2.1, the first flipping $t_{(1)}$ in (2.1) can be n in S_n if a sequence \mathbf{x} is ϵ -balanced and $\mathcal{E}_1(\mathbf{x})$ is not ϵ -balanced when $t_{(1)}$ takes the value from 1 to $n - 1$. Therefore, the parity length of [36] should be $R_{(1)} = \lceil \log n \rceil$ in (2.2). However, in our the proposed codes, $t_{(1)}$ does not need to be n in S_n since inputs of $\mathcal{E}_1(\cdot)$ encoding are always not ϵ -balanced. Therefore, the parity length of $R_{(1)}^{(0)}$ in (2.18) produced by $\mathcal{E}_1(\mathbf{x})$ is modified as

$$R_{(1)}^{(0)} = \lceil \log(n - 1) \rceil. \quad (2.21)$$

Based on the values of $|B_{\epsilon,n}^0|$ in (2.19) or (2.20), $R_{(1)}^{(0)}$ in (2.21), and $AR_{(l)}^{(0)}$ in (2.18), the average parity length $AR_{(1)}^{(0)}$ for our proposed codes is derived as

$$AR_{(1)}^{(0)} = \frac{|B_{\epsilon,n}^0|}{2^n} \cdot 1 + \left(1 - \frac{|B_{\epsilon,n}^0|}{2^n}\right) (\lceil \log(n - 1) \rceil + 1). \quad (2.22)$$

Similarly, since $t_{(2)}$ cannot be zero and n , the length of parity $\mathbf{r}_{(l)}^{(0)}$ for $\mathcal{E}_2(\cdot)$ encoding, $R_{(2)}^{(0)}$, is also modified as

$$R_{(2)}^{(0)} = \lceil \log(\lfloor \frac{1}{2\epsilon} \rfloor - 1) \rceil \quad \text{for } 0 < \epsilon \leq 0.5. \quad (2.23)$$

The average parity length $AR_{(2)}^{(0)}$ for our proposed codes is calculated as

$$AR_{(2)}^{(0)} = \frac{|B_{\epsilon,n}^0|}{2^n} \cdot 1 + \left(1 - \frac{|B_{\epsilon,n}^0|}{2^n}\right) (\lceil \log(\lfloor \frac{1}{2\epsilon} \rfloor - 1) \rceil + 1). \quad (2.24)$$

2.4.2 Average parity length for $f = 1$

For the value of $f = 1$, the average parity length $AR_{(l)}^{(1)}$ is given as

$$AR_{(l)}^{(1)} = \frac{|B_{\epsilon,n}^0|}{2^n} \cdot 1 + \frac{|B_{\epsilon,n}^1|}{2^n} \cdot 2 + \left(1 - \frac{|B_{\epsilon,n}^0| + |B_{\epsilon,n}^1|}{2^n}\right) (R_{(l)}^{(1)} + 2). \quad (2.25)$$

The set $B_{\epsilon,n}^1$ contains binary sequences \mathbf{x} that become ϵ -balanced after flipping $F_1(\mathbf{x})$, and do not contain any sequence in $B_{\epsilon,n}^0$. Therefore, the weight of these sequences $w(\mathbf{x})$ falls in the range of $[\lceil \epsilon n \rceil + 1, 0.5n]$. Then, the size of $B_{\epsilon,n}^1$ for even n can be calculated as

$$|B_{\epsilon,n}^1| = 2 \sum_{j=\lceil \epsilon n \rceil + 1}^{0.5n} \sum_{i=t}^{j-t} \binom{\frac{n}{2}}{\lceil \frac{n}{4} \rceil + i} \binom{\frac{n}{2}}{\lceil \frac{n}{4} \rceil + j - i}, \quad (2.26)$$

where t in (2.26) is $t = \lceil \frac{j - \lceil \epsilon n \rceil}{2} \rceil$.

Similarly, for odd n , the size $|B_{\epsilon,n}^1|$ is determined as

$$|B_{\epsilon,n}^1| = 2 \sum_{j=\lceil \epsilon n \rceil + 1}^{0.5n} \sum_{i=t}^{j-t} \binom{\lceil \frac{n}{2} \rceil}{\lceil \frac{n}{4} \rceil + i} \binom{n - \lceil \frac{n}{2} \rceil}{\lceil \frac{n}{4} \rceil + j - i}, \quad (2.27)$$

where t in (2.27) is also $t = \lceil \frac{j - \lceil \epsilon n \rceil}{2} \rceil$.

Since the original input sequences of the output ϵ -balanced codewords $\mathcal{E}_1(\mathbf{x})$ are not in the sets $B_{\epsilon,n}^0$ and $B_{\epsilon,n}^1$, $t_{(1)}$ does not need to be n and $\frac{n}{2}$ in S_n . Then, the length of parity $\mathbf{r}_{(1)}^{(1)}$ produced by $\mathcal{E}_1(\mathbf{x})$ can be modified as

$$R_{(1)}^{(1)} = \lceil \log(n - 2) \rceil. \quad (2.28)$$

Based on the values of $|B_{\epsilon,n}^0|$, $|B_{\epsilon,n}^1|$, $R_{(1)}^{(1)}$ in (2.28), and $AR_{(l)}^{(1)}$ in (2.25), the average parity length $AR_{(1)}^{(1)}$ for the proposed codes can be calculated as

$$AR_{(1)}^{(1)} = \frac{|B_{\epsilon,n}^0|}{2^n} \cdot 1 + \frac{|B_{\epsilon,n}^1|}{2^n} \cdot 2 + \left(1 - \frac{|B_{\epsilon,n}^0| + |B_{\epsilon,n}^1|}{2^n}\right) (\lceil \log(n-2) \rceil + 2). \quad (2.29)$$

Similarly, $t_{(2)}$ does not need to be zero, n , and $\frac{n}{2}$ if $(\frac{n}{2} \bmod \lfloor 2\epsilon n \rfloor) = 0$. Therefore, the length of parity $\mathbf{r}_{(2)}^{(1)}$ produced by $\mathcal{E}_2(\mathbf{x})$ is modified as

$$R_{(2)}^{(1)} = \begin{cases} \lceil \log(\lfloor \frac{1}{2\epsilon} \rfloor - 1) \rceil & \text{if } (\frac{n}{2} \bmod \lfloor 2\epsilon n \rfloor) \neq 0, \\ \lceil \log(\lfloor \frac{1}{2\epsilon} \rfloor - 2) \rceil & \text{if } (\frac{n}{2} \bmod \lfloor 2\epsilon n \rfloor) = 0. \end{cases} \quad (2.30)$$

Then, the average parity length $AR_{(2)}^{(1)}$ for our proposed codes is given as

$$AR_{(2)}^{(1)} = \begin{cases} \frac{|B_{\epsilon,n}^0|}{2^n} \cdot 1 + \frac{|B_{\epsilon,n}^1|}{2^n} \cdot 2 + \left(1 - \frac{|B_{\epsilon,n}^0| + |B_{\epsilon,n}^1|}{2^n}\right) \\ \times (\lceil \log(\lfloor \frac{1}{2\epsilon} \rfloor - 1) \rceil + 2) & \text{if } (\frac{n}{2} \bmod \lfloor 2\epsilon n \rfloor) \neq 0, \\ \frac{|B_{\epsilon,n}^0|}{2^n} \cdot 1 + \frac{|B_{\epsilon,n}^1|}{2^n} \cdot 2 + \left(1 - \frac{|B_{\epsilon,n}^0| + |B_{\epsilon,n}^1|}{2^n}\right) \\ \times (\lceil \log(\lfloor \frac{1}{2\epsilon} \rfloor - 2) \rceil + 2) & \text{if } (\frac{n}{2} \bmod \lfloor 2\epsilon n \rfloor) = 0. \end{cases} \quad (2.31)$$

2.4.3 Average parity length for $f > 1$

To derive the average parity length $AR_{(l)}^{(f)}$ for $f > 1$, the size of $B_{\epsilon,n}^i$ for $i \in [2, f]$ is first calculated. However, for general n and f , it is not possible to calculate the exact size of $B_{\epsilon,n}^i$ for $i \in [2, f]$. To address the average parity length $AR_{(l)}^{(f)}$ for $f > 1$, we derive the calculation of the average parity length by using randomly generated sequences.

Let define \mathcal{B} as the set of randomly generated sequences \mathbf{x} of length n , and $\mathcal{B}_{\epsilon,n}^0$ as the set of all ϵ -balanced sequences \mathbf{x} in set \mathcal{B} . Then, $\mathcal{B}_{\epsilon,n}^i$ for $i \in [1, f]$ denote the set that does not contain any sequence in $\cup_{j=0}^{i-1} \mathcal{B}_{\epsilon,n}^j$ and consists of the binary sequences which become ϵ -balanced after flipping $F_i(\mathbf{x})$. Therefore, by

substituting sequences set sizes 2^n to $|\mathcal{B}|$ and $|B_{\epsilon,n}^i|$ to $|\mathcal{B}_{\epsilon,n}^i|$ for $i = [0, f]$, the average parity length $AR_{(l)}^{(f)}$ in (2.11) is modified as

$$\text{M-}AR_{(l)}^{(f)} = \sum_{i=0}^f \left(\frac{|\mathcal{B}_{\epsilon,n}^i|}{|\mathcal{B}|} (i+1) \right) + \left(1 - \sum_{i=0}^f \frac{|\mathcal{B}_{\epsilon,n}^i|}{|\mathcal{B}|} \right) (R_{(l)}^{(f)} + f + 1). \quad (2.32)$$

Similar to the calculation of $AR_{(l)}^{(f)}$, since the inputs of $\mathcal{E}_l(\cdot)$ encoding for $l = 1$ and $l = 2$ are always not ϵ -balanced, $t_{(1)}$ and $t_{(2)}$ do not need to be some elements of in S_n and $S_{\epsilon,n}$, respectively. For $f = 0$, the values of $R_{(l)}^{(f)}$ in (2.32) are $\lceil \log(n-1) \rceil$ for $l = 1$ and $\lceil \log(\lfloor \frac{1}{2\epsilon} \rfloor - 1) \rceil$ for $l = 2$. Similarly, for $f = 1$, the values of $R_{(l)}^{(f)}$ in (2.32) for $l = 1$ and $l = 2$ are $\lceil \log(n-2) \rceil$ and $\lceil \log(\lfloor \frac{1}{2\epsilon} \rfloor - 2) \rceil$ if $(\frac{n}{2} \bmod \lfloor 2\epsilon n \rfloor) = 0$, respectively. However, for $f > 1$, since the bits in the sequences flipped by $F_i(\mathbf{x})$ for $i \in [2, f]$ are not the first consecutive $t_{(1)}$ or $t_{(2)}$ bits of the sequences, it is not possible remove more elements from S_n and $S_{\epsilon,n}$ for $\mathcal{E}_l(\cdot)$ encoding. Then, the values of $R_{(l)}^{(f)}$ in (2.32) for $f > 1$ should be same with $R_{(l)}^{(f)}$ for $f = 1$.

2.5 Performance evaluation

To evaluate the performance of our proposed codes, we present the comparison results of average parity lengths for our proposed codes and referenced ones for various ϵ . Then, we also present the comparison results of the average parity lengths for various f . Since the studies [20, 71, 57] indicated that the proper ratio of GC contents in a DNA codeword is around 45 – 55% or 40 – 60%, we especially focus on the results for $\epsilon = 0.05$ and 0.1.

2.5.1 Comparison results of average parity lengths for various ϵ

We present the comparison results of average parity lengths of our proposed codes and referenced ones for various ϵ . Since the length of synthesized DNA sequences has been usually determined as 100-300 nt [4, 20, 36], the values of n are considered as $n = 128, 200,$ and 240 .

For the convenience of estimating the average parity lengths $AR_{(1)}^{(0)}$ in (2.22), $AR_{(2)}^{(0)}$ in (2.24), $AR_{(1)}^{(1)}$ in (2.29), and $AR_{(2)}^{(1)}$ in (2.31), we present the values of $\frac{|B_{\epsilon,n}^0|}{2^n}$ and $\frac{|B_{\epsilon,n}^1|}{2^n}$ for various ϵ since they are closely related to the average parity lengths. The results of the ratios $\frac{|B_{\epsilon,n}^0|}{2^n}$ and $\frac{|B_{\epsilon,n}^1|}{2^n}$ for lengths $n = 128, 200,$ and 240 are listed in Figs. 2.5 and 2.6.

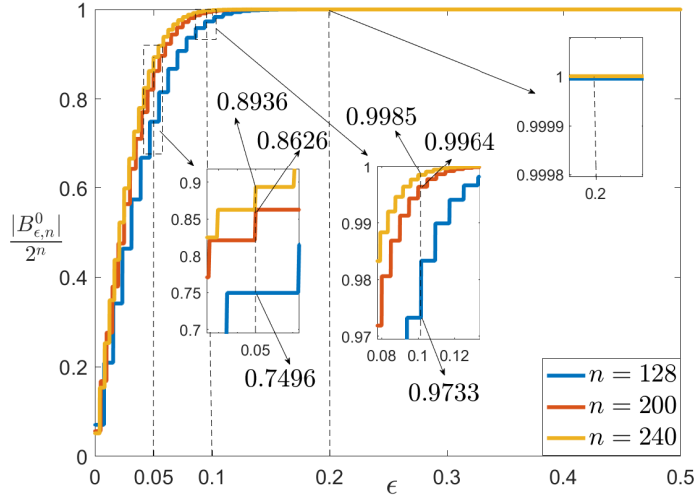


Figure 2.5: The ratio $\frac{|B_{\epsilon,n}^0|}{2^n}$ for lengths $n = 128, 200,$ and 240 .

From Fig. 2.5, as ϵ increases, the values of $\frac{|B_{\epsilon,n}^0|}{2^n}$ monotonically increase and are saturated as 0.9999 if ϵ is over 0.2. For $\epsilon = 0.05$, the values of $\frac{|B_{\epsilon,n}^0|}{2^n}$ are 0.7496, 0.8626, and 0.8936 for $n = 128, 200,$ and 240 , respectively. When ϵ is doubled as 0.1, $\frac{|B_{\epsilon,n}^0|}{2^n}$ are 0.9733, 0.9964, and 0.9985 for $n = 128, 200,$ and 240 , respectively.

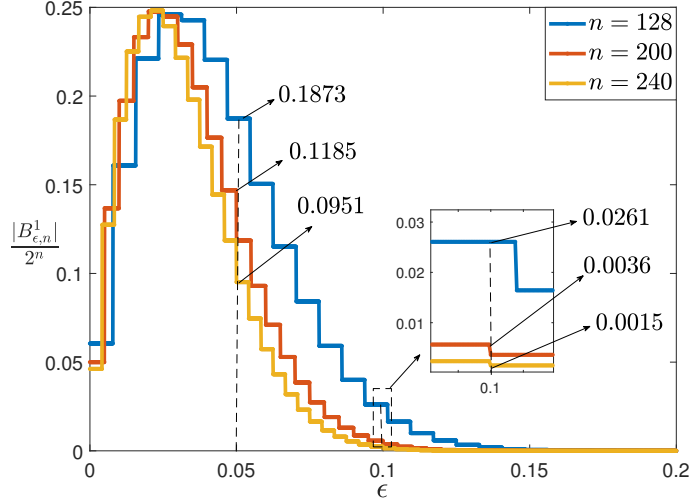


Figure 2.6: The ratio $\frac{|B_{\epsilon,n}^1|}{2^n}$ for lengths $n = 128, 200,$ and 240 .

Based on Fig. 2.5 and the values of $AR_{(1)}^{(0)}$ in (2.22) and $AR_{(2)}^{(0)}$ in (2.24), we can infer that the values of $AR_{(1)}^{(0)}$ and $AR_{(2)}^{(0)}$ will be close to one if the ratio $\frac{|B_{\epsilon,n}^0|}{2^n}$ is near one. Since ϵ larger than 0.2 causes $\frac{|B_{\epsilon,n}^0|}{2^n} \approx 1$ in Fig. 2.5, which will lead $\frac{|B_{\epsilon,n}^i|}{2^n}$ for $i \in [1, f]$ to be saturated as 0, we only consider the range of ϵ as $0 \leq \epsilon \leq 0.2$.

In Fig. 2.6, the ratios $\frac{|B_{\epsilon,n}^1|}{2^n}$ for $n = 128, 200,$ and 240 are 0.1873, 0.1185, and 0.0951, respectively, when $\epsilon = 0.05$. When $\epsilon = 0.1$, $\frac{|B_{\epsilon,n}^1|}{2^n}$ are 0.0261, 0.0036, and 0.0015 for $n = 128, 200,$ and 240 , respectively. From Figs. 2.5 and 2.6, we can obtain that the ratios $(1 - \frac{|B_{\epsilon,n}^0|}{2^n} + \frac{|B_{\epsilon,n}^1|}{2^n})$ for $n = 128, 200,$ and 240 are 0.0631, 0.0189, and 0.0113 when $\epsilon = 0.05$ and 0.0006, 0, and 0 when $\epsilon = 0.1$, which are also important results to calculate the average parity lengths.

The performance of GC-balanced DNA codes constructed by our proposed encoding algorithm is compared with those constructed by the referenced ones [36] and [49] for the same lengths n in order to provide a fair comparison. The comparison results of the average parity lengths $R_{(1)}$ of [36], $R_{(2)}$ of [49], $AR_{(1)}^{(f)}$ in (2.22) or (2.29), and $AR_{(2)}^{(f)}$ in (2.24) or (2.31) for $n = 128, 200,$ and 240 are presented in Figs. 2.7 and 2.8, when $f = 0$ and $f = 1$, respectively.

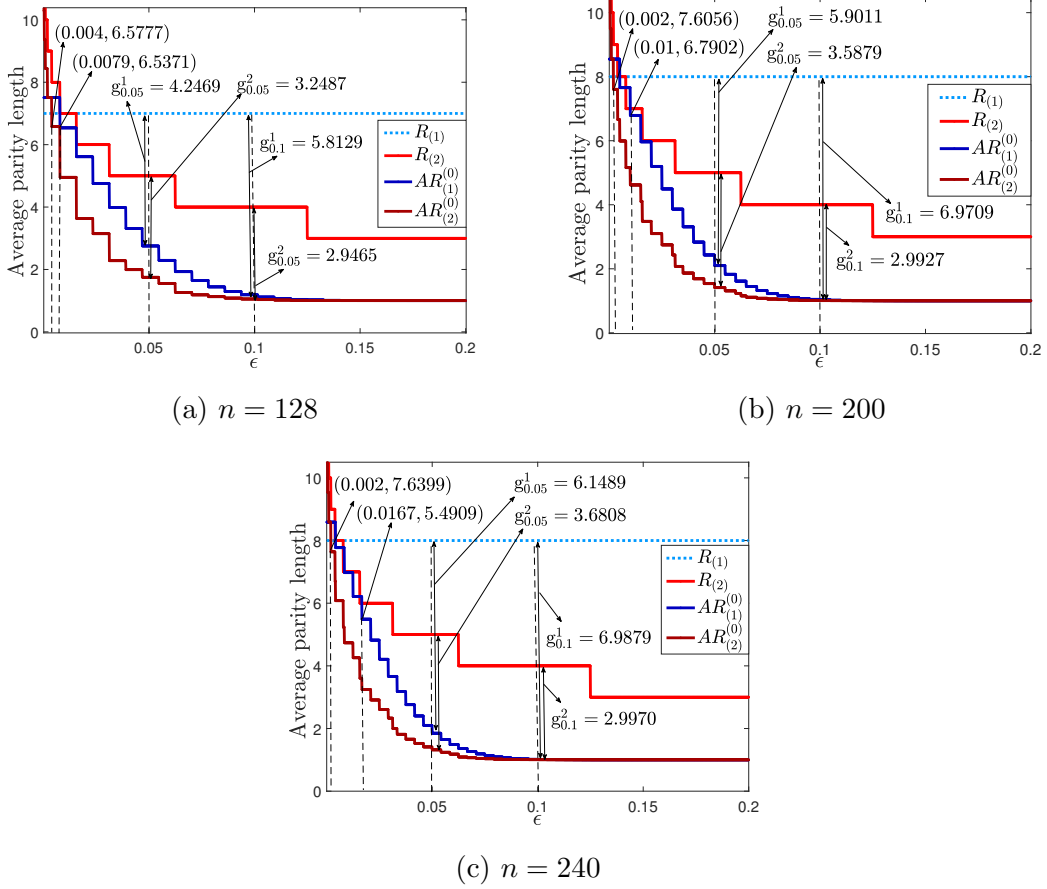


Figure 2.7: Average parity lengths for $f = 0$ and $n = 128, 200,$ and 240 .

In Fig. 2.7, we observe that $R_{(1)}$ of [36] remains fixed for each length n , whereas the average parity lengths $R_{(2)}$ of [49], $AR_{(1)}^{(0)}$ in (2.22), and $AR_{(2)}^{(0)}$ in (2.24) decrease as the ϵ values increase. Two points $(0.004, 6.5777)$ and $(0.0079, 6.5371)$ in Fig. 2.7(a) stands for minimum values of ϵ to ensure that the average parity length $AR_{(1)}^{(0)}$ or $AR_{(2)}^{(0)}$ of our proposed codes is always less than $R_{(1)}$ of [36] and $R_{(2)}$ of [49], respectively, for $n = 128$. Similarly, in Fig. 2.7(b), the points $(0.002, 7.6056)$ and $(0.01, 6.7902)$, and in Fig. 2.7(c), the points $(0.002, 7.6399)$ and $(0.0167, 5.4909)$ show the minimum values of ϵ required to ensure that the average parity lengths of the proposed codes are less than those of [16] and [17]. According to the obtained

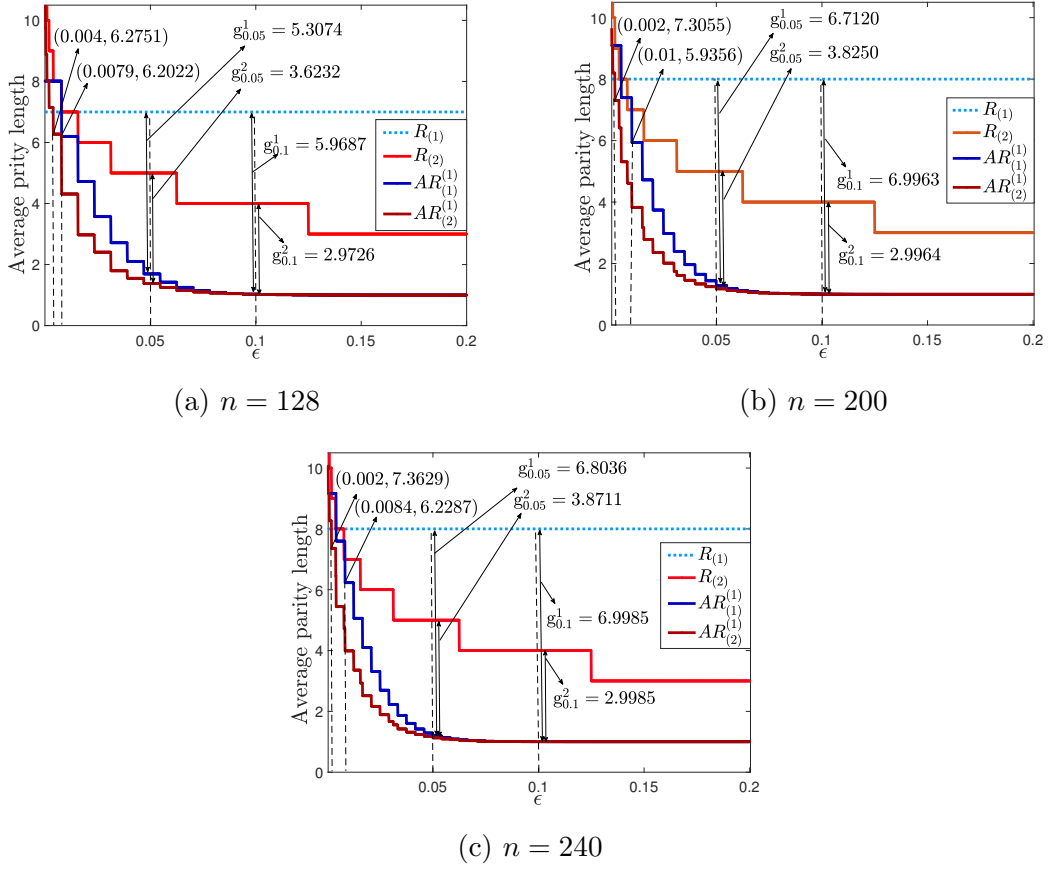


Figure 2.8: Average parity lengths for $f = 1$ and $n = 128, 200,$ and 240 .

results, it can be concluded that when ϵ is slightly larger than zero, specifically at values of 0.002 and 0.0167, the average parity lengths of the proposed codes yield smaller values. This implies that proposed codes are more efficient than two referenced codes at the most values of ϵ .

Moreover, we provide performance comparison when values of ϵ are 0.05 and 0.1. In Fig. 2.7(a), $g_{0.05}^1$ and $g_{0.05}^2$ denote the gaps of average parity lengths of the proposed codes in comparison with the codes in [36] and [49], respectively, when ϵ is 0.05. Since the values of $R_{(1)}$ of [36] and $AR_{(1)}^{(0)}$ are 7 and 2.7531, respectively, $g_{0.05}^1$ is 4.2469. This means that the proposed codes can reduce around 61% ($\frac{4.2469}{7} = 0.6067$) of parity lengths for $n = 128$. Similarly, since the values of

$R_{(2)}$ of [49] and $AR_{(2)}^{(0)}$ are 5 and 1.3768, respectively, $g_{0.05}^2$ is 3.2487, which denotes around 65% reduction in parity lengths. For $\epsilon = 0.1$ in Fig. 2.7(a), $g_{0.1}^1$ is 5.8129, which is larger than $g_{0.05}^1$, but $g_{0.1}^2$ is 2.9465, which is smaller than $g_{0.05}^2$. When ϵ is 0.2, the average parity lengths of proposed codes are almost one, while those of [36] and [49] are 7 and 3. In Figs. 2.7(b) and 2.7(c), the gaps, $g_{0.05}^1$, $g_{0.05}^2$, $g_{0.1}^1$, and $g_{0.1}^2$, are also denoted for $n = 200$ and 240, and show similar trends with ones in Fig. 2.7(a). Moreover, as n increases, $g_{0.05}^1$, $g_{0.05}^2$, $g_{0.1}^1$, and $g_{0.1}^2$ also increase.

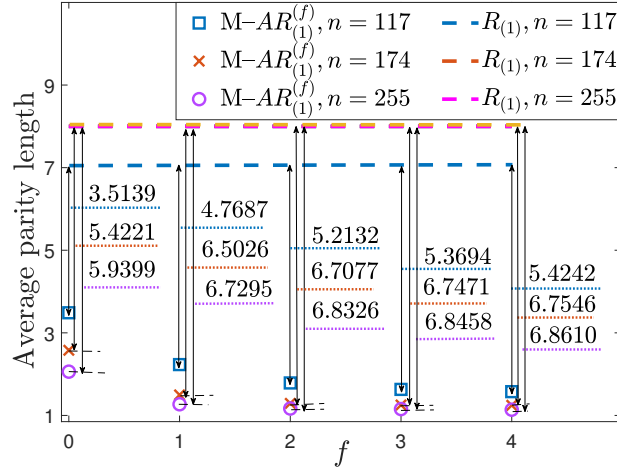
The minimum values of ϵ and gaps of average parity lengths are also denoted in Fig. 2.8 when the value of f is $f = 1$. First, compared with Fig. 2.7, it can be confirmed that the average parity lengths of the proposed codes decrease overall as f increases by one. Therefore, the minimum values of ϵ to ensure that the average parity lengths of the proposed codes are less than those of [36] and [49] in Fig. 2.8 are smaller or equal to ones in Fig. 2.7. Moreover, the gaps of parity length between the proposed codes and referenced ones become larger.

2.5.2 Average parity length comparison results for various f

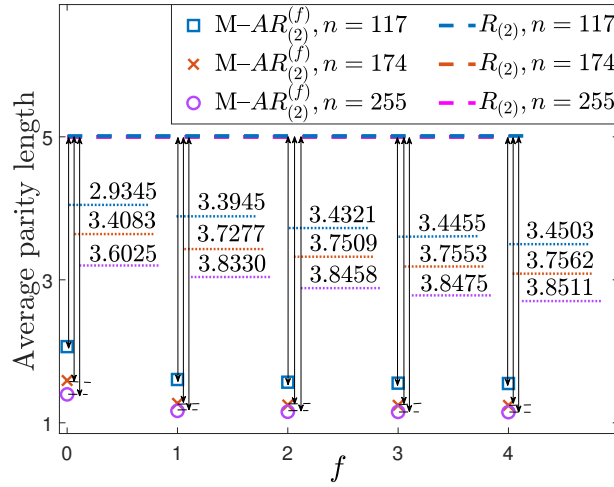
We present the comparison results of average parity lengths of our proposed codes with those of [36] and [49] for various f at $\epsilon = 0.05$ and 0.1, where the maximum value of f is considered up to four. To evaluate the effectiveness of our proposed codes, we determine n as $n = 117, 174,$ and 255 since the same lengths of the DNA codes were considered in [32, 71].

After obtaining the sizes of $\mathcal{B}_{\epsilon,n}^i$ in randomly generated sequences set \mathcal{B} , the corresponding average parity lengths $M-AR_{(l)}^{(f)}$ in (2.32) for $l = 1$ or $l = 2$ can be calculated. The results comparing the average parity lengths $R_{(l)}$ of [36] and [49] with $M-AR_{(l)}^{(f)}$ in (2.32) for various f at $\epsilon = 0.05$ and $\epsilon = 0.1$ are presented in Figs. 2.9 and 2.10, respectively. In Figs. 2.9 and 2.10, the dash lines and markers

are represented for the average parity lengths of $R_{(l)}$ of [36] and [49] and $M-AR_{(l)}^{(f)}$ in (2.32), respectively. The average parity lengths of the proposed codes $M-AR_{(l)}^{(f)}$ of lengths $n = 117, 174,$ and 255 decrease as the f increases and gradually saturate at certain values for $\epsilon = 0.05$ and 0.1 .



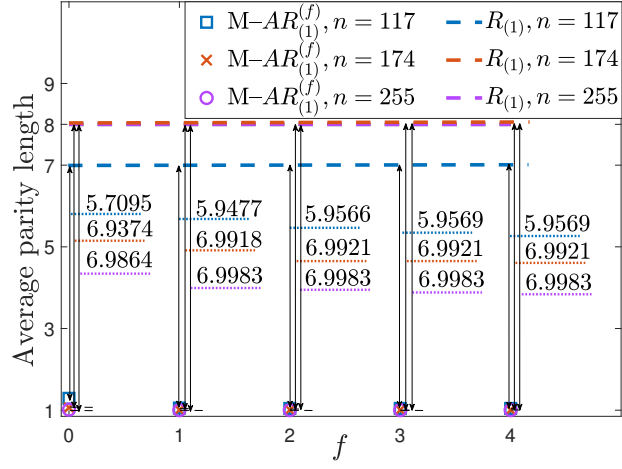
(a) $l = 1$



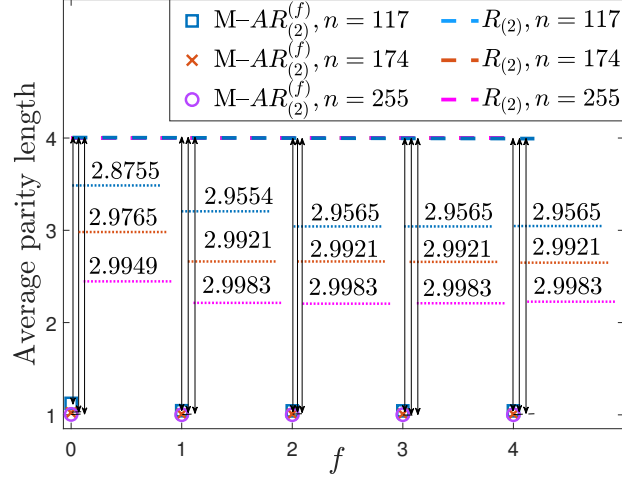
(b) $l = 2$

Figure 2.9: Average parity lengths of the proposed codes for various f for lengths $n = 117, 174, 255,$ and $\epsilon = 0.05$.

Moreover, we present the performance improvement of our proposed codes for



(a) $l = 1$



(b) $l = 2$

Figure 2.10: Average parity lengths of the proposed codes for various f for lengths $n = 117, 174, 255$, and $\epsilon = 0.1$.

various values of f . In Fig. 2.9(a), the values of $R_{(1)}$ of [18] remain fixed at 7 for $n = 117$, while the values of $M-AR_{(1)}^{(f)}$ are 3.4861, 2.2313, 1.7868, 1.6306, and 1.5758 for $f = 0, 1, 2, 3$, and 4, respectively. The gaps between $R_{(1)}$ of [18] and $M-AR_{(1)}^{(f)}$ are also shown above the blue dotted lines, which are 3.5139, 4.7687, 5.2132, 5.3694, and 5.4242, indicating around 50% ($3.5139/7=0.5019$), 68%, 74%,

77%, and 77% reductions in parity lengths, respectively. Additionally, for $n = 174$ and 255, the parity lengths $R_{(1)}$ of [18] are fixed at 8, but the average parity lengths of $M-AR_{(1)}^{(f)}$ decrease as f increases, and the gaps between $R_{(1)}$ of [18] and $M-AR_{(1)}^{(f)}$ are also illustrated above the red and purple dotted lines, respectively. In Fig. 2.9(b), for $n = 117, 174,$ and 255, the parity lengths $R_{(2)}$ of [45] are fixed at 5, while the average parity lengths $M-AR_{(2)}^{(f)}$ also decrease as f increases. For instance, for $n = 117$, the gaps between $R_{(2)}$ of [45] and $M-AR_{(2)}^{(f)}$ are 2.9345, 3.3945, 3.4321, 3.4455, and 3.4503 for $f = 0, 1, 2, 3,$ and 4, respectively, which correspond to 59%, 68%, 69%, 69%, and 69% reductions. From Fig. 2.9, it can be concluded that for $\epsilon = 0.05$, which ensures 45-55% GC-balanced DNA codes, the proposed codes require over 50% reduction in parity lengths compared to two referenced codes when considering the simplest type of the proposed codes, which corresponds to $f = 0$.

For $\epsilon = 0.1$, the average parity lengths of our proposed codes and referenced codes, as well as their corresponding gaps are also presented in Fig. 2.10. Since the value of $|B_{\epsilon,n}^0|$ becomes larger as ϵ is doubled, the average parity lengths for $f = 0$ in Fig. 2.10 become shorter than those in Fig. 2.9, and the corresponding gaps become larger. For example, the average parity length and gap for $n = 117$ in Fig. 2.10(a) are 1.2905 and 5.7095 while those in Fig. 2.9(a) are 3.4861 and 3.5139. Moreover, when f is larger than one in Fig. 2.10(a) and zero in Fig. 2.10(b), the average parity lengths and their gaps become saturated. One interesting point is that the average parity lengths of the proposed codes cannot be less than one, and in Fig. 2.10, the average parity lengths of the proposed codes are slightly larger than one. From Fig. 2.10, it can be concluded that even for $\epsilon = 0.1$, the proposed codes exhibit better performance than two referenced codes, and the simplest type of the proposed codes shows excellent results.

Chapter 3

Nonbinary single insertion or deletion error correction codes with a maximum run-length constraint for DNA storage

3.1 Introduction

Although DNA storage has many potential advantages, recent studies [17, 26, 4, 32] have indicated that insertion, deletion, and substitution errors occur in DNA synthesis and sequencing processes. To improve the robustness of the DNA synthesis and sequencing processes, error correction codes such as fountain codes [20], Reed-Solomon codes [32], Bose-Chaudhuri-Hocquenghem codes [19], and low-density parity-check codes [19, 15, 24] have been applied to DNA storage systems. Since a DNA strand is subject to two constraints, i.e., balanced GC-content (45%–55% or 40%–60%) and short homopolymer length (≤ 3 or 4), which enable low error rates during DNA synthesis and sequencing processes [20], the constrained codes

[4, 33] without the capability of error correction have been considered to reduce the occurrence of insertion, deletion, and substitution errors in DNA storage. Furthermore, the constrained code combined with the error correction code has been considered for DNA storage [70].

A binary single insertion/deletion error correction (SIDECE) code, which is called a Varshamov-Tenengolts (VT) code [63], was first proposed in 1965. Levenshtein modified the binary VT code as an extended VT (EVT) code [40] to correct a single insertion/deletion/substitution error. The nonbinary VT codes [60], which also can correct a single insertion/deletion error, were proposed almost 20 years later. The authors of [2] proposed an efficient systematic encoding algorithm for nonbinary SIDECE codes whose codewords consist of parity symbols and message symbols. A binary EVT code combined with a GC-balanced constrained code [70] and a binary VT code combined with a GC-balanced and r run-length limited constrained code [49] were applied for DNA storage systems. Additionally, a new binary SIDECE code combined with the maximum run-length r constrained code and efficient systematic encoding algorithm was proposed in [59].

All of these studies [19, 15, 49, 33, 70] focused on binary coding schemes for DNA storage systems. So far, there have been few studies focusing on q -ary coding schemes [27] for DNA storage systems. Moreover, insertion and deletion errors are inevitably bound to occur in the process of DNA synthesis and sequencing. For these purposes, we propose a new nonbinary SIDECE code with the maximum run-length r constraint and systematic encoding algorithm. We can apply the proposed code with $q = 4$ and the maximum run-length $r = 3$ which are suitable for DNA storage.

The design of the codes is especially inspired by related works [2] and [59]. First, the nonbinary code design in [2] did not consider the maximum run-length constraint and some of the codewords from the design in [2] cannot be successfully

encoded. However, these issues are addressed in our code construction. Second, although the binary SIDEC code in [59] involved the maximum run-length e constraint, the length of output codewords was limited by the maximum run-length e . However, our proposed code has no limitations on codewords length.

We first present a method to convert the binary data into q -ary symbol sequences with the maximum run-length r constraint. These sequences can be used as inputs for the proposed nonbinary SIDEC code with the maximum run-length r constraint. The construction of the proposed q -ary SIDEC code and its corresponding systematic encoding algorithm are also presented. The output codeword is a sequence of the q -ary SIDEC code with the maximum run-length r constraint. Simulation results show that the encoding algorithm is feasible and the q -ary SIDEC code with the maximum run-length r constraint can correct a single deletion or insertion error.

3.2 Preliminaries

Some special notations are used in this chapter are introduced. For any positive integer a and b , given a an integer c , $\langle a, c, b \rangle$ denotes the vector $(a, a + c, a + 2c, \dots, b)$. The value q is considered as $q = 2^\omega$ for an integer $\omega > 1$. The $\text{dec}_a(\cdot)$ denote decimal representation of base a . Let G_a^b be the set of sequences $\mathbf{v} \in \mathbb{Z}_q^b$ of length b with the maximum run-length a constraint. The Hamming weight of a sequence \mathbf{v} is the number of symbols that are different from the zero symbol. A positive integer $w \in [1, 2^a - 1]$ is represented by $\sum_{i=0}^{a-1} g_i \cdot 2^i$, where $g_i \in \mathbb{Z}_2$ and a is any positive integer. We define a mapping as $\text{Le}_a(w) = (g_0, g_1, \dots, g_{a-1})$ and $\text{Le}_a^{-1}((g_0, g_1, \dots, g_{a-1})) = w$. For example, $\text{Le}_3(1) = (1, 0, 0)$ and $\text{Le}_3^{-1}((1, 0, 0)) = 1$.

3.2.1 Proposed code construction and algorithm

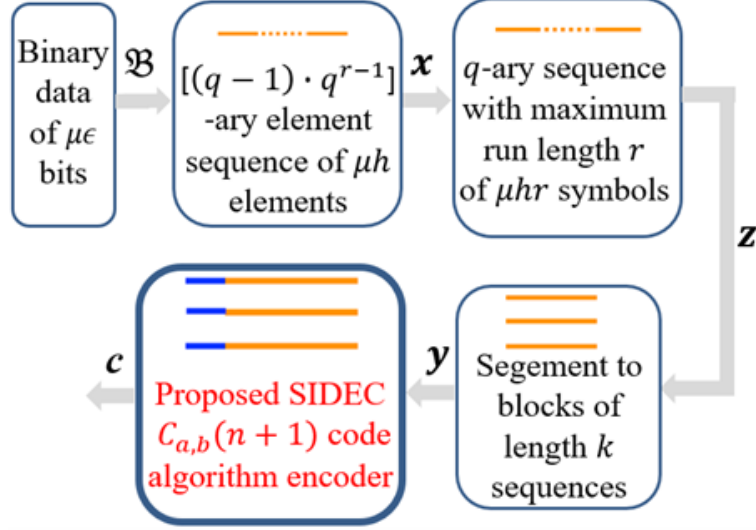


Figure 3.1: Overall block diagram of the proposed code.

The overall block diagram of the proposed code is shown in Fig 3.1. Since the construction method that converts the binary data \mathfrak{B} into a q -ary sequence \mathfrak{z} with the maximum run-length r constraint is similar in [27], we briefly explain the method in Section 3.2.2. We also present an application of a codeword where the maximum run-length r is three for DNA storage. We segment the q -ary sequence with the maximum run-length r constraint into multiple sequences with the specified length; this is the message sequence of the proposed SIDEC code $C_{a,b}(n+1)$ in Section 3.3.1. The message sequences used as the inputs pass through the algorithm of the systematic proposed SIDEC code in Section 3.3.2, and the output codewords have properties of single insertion/deletion error correction and the maximum run-length r constraint. The maximum run-length of the output codewords is $r = 3$, since we design the fixed maximum run-length r to be three in message the parity sequences and the last symbol of the parity sequence is always different with the first symbol of the message sequence. The codewords with maximum run-length $r = 3$ can be used for DNA synthesis due to the constraints for

DNA storage systems [20].

3.2.2 Construction of sequences with the maximum run-length constraint

Our method for constructing the maximum run-length r sequences is similar to the method used in [27]. We extend the method in [27] to the general case to obtain the q -ary maximum run-length r constrained sequence. First, we assume that the binary data \mathfrak{B} can be divided into μ blocks of length ϵ bits, which are represented as $\mathfrak{b}_{(i-1)\epsilon+1}^{i\epsilon} = (\mathfrak{b}_{(i-1)\epsilon+1}, \mathfrak{b}_{(i-1)\epsilon+2}, \dots, \mathfrak{b}_{i\epsilon}) \in \mathbb{Z}_2^\epsilon$ for $i \in [1, \mu]$. Then, we build one-to-one mapping to convert $\mathfrak{b}_{(i-1)\epsilon+1}^{i\epsilon}$ into $((q-1)q^{r-1})$ -ary h elements, where h is given by $h = \lceil \epsilon / \log((q-1)q^{r-1}) \rceil$. The binary data \mathfrak{B} with length $\mu\epsilon$ can be converted into a sequence $\mathbf{x} = (x_1, x_2, \dots, x_{\mu h}) = (\mathbf{x}_1^h, \mathbf{x}_{h+1}^{2h}, \dots, \mathbf{x}_{(i-1)h+1}^{ih}, \dots, \mathbf{x}_{(\mu-1)h+1}^{\mu h})$, where x_j is a $((q-1)q^{r-1})$ -ary element for $j \in [1, \mu h]$. Then, the mapping between the binary subsequence $\mathfrak{b}_{(i-1)\epsilon+1}^{i\epsilon}$ with ϵ bits in \mathfrak{B} and the $((q-1)q^{r-1})$ -ary subsequence $\mathbf{x}_{(i-1)h+1}^{ih}$ with h elements in \mathbf{x} is $\text{dec}_2(\mathfrak{b}_{(i-1)\epsilon+1}^{i\epsilon}) = \text{dec}_{(q-1)q^{r-1}}(\mathbf{x}_{(i-1)h+1}^{ih})$, for $i \in [1, \mu]$. Next, we can build a one-to-one mapping to convert a $((q-1)q^{r-1})$ -ary element x_j in \mathbf{x} into a q -ary subsequence $\mathbf{z}_{(j-1)r+1}^{jr}$ with r symbols, since the subsequence $\mathbf{z}_{(j-1)r+1}^{jr}$ in \mathbf{z} has the property that $z_{j_{r-1}} \neq z_{j_r}$ for $j \in [1, \mu h]$. It can represent $((q-1)q^{r-1})$ different values. Then, the sequence \mathbf{x} can be converted into a q -ary sequence $\mathbf{z} = (z_1, z_2, \dots, z_{\mu hr}) = (\mathbf{z}_1^r, \mathbf{z}_{r+1}^{2r}, \dots, \mathbf{z}_{(j-1)r+1}^{jr}, \dots, \mathbf{z}_{(\mu h-1)r+1}^{\mu hr})$ of length μhr . Since $\mathbf{z}_{(j-1)r+1}^{jr}$ has $z_{j_{r-1}} \neq z_{j_r}$ for $j \in [1, \mu h]$, for any two subsequences $\mathbf{z}_{(j_1-1)r+1}^{j_1 r}$, $\mathbf{z}_{(j_2-1)r+1}^{j_2 r}$ with r symbols in \mathbf{z} for $j_1 \neq j_2$ and $j_1, j_2 \in [1, \mu h]$, the maximum run-length of their concatenation $(\mathbf{z}_{(j_1-1)r+1}^{j_1 r}, \mathbf{z}_{(j_2-1)r+1}^{j_2 r})$ is $r-1$ when the last symbol $z_{j_1 r}$ in $\mathbf{z}_{(j_1-1)r+1}^{j_1 r}$ and the first symbol $z_{(j_2-1)r+1}$ in $\mathbf{z}_{(j_2-1)r+1}^{j_2 r}$ have $z_{j_1 r} \neq z_{(j_2-1)r+1}$, and the maximum run-length is r when $z_{j_1 r} = z_{(j_2-1)r+1}$. Therefore, the q -ary sequence \mathbf{z} is the maximum run-length r sequence if the maximum run-length of any two adjacent subsequences $\mathbf{z}_{(j-1)r+1}^{jr}$ and $\mathbf{z}_{j_{r+1}}^{(j+1)r}$ with concate-

Table 3.1: The example of one-to-one mapping between a 48-ary element and q -ary subsequence $\mathbf{z}_{(j-1)r+1}^{jr}$ with r symbols.

48-ary element	Subsequence $\mathbf{z}_{(j-1)r+1}^{jr}$	48-ary element	Subsequence $\mathbf{z}_{(j-1)r+1}^{jr}$	48-ary element	Subsequence $\mathbf{z}_{(j-1)r+1}^{jr}$
0	AAT	16	TTG	32	GGC
1	AAG	17	TTC	33	GCA
2	AAC	18	TGA	34	GCT
3	ATA	19	TGT	35	GCG
4	ATG	20	TGC	36	CAT
5	ATC	21	TCA	37	CAG
6	AGA	22	TCT	38	CAC
7	AGT	23	TCG	39	CTA
8	AGC	24	GAT	40	CTG
9	ACA	25	GAG	41	CTC
10	ACT	26	GAC	42	CGA
11	ACG	27	GTA	43	CGT
12	TAT	28	GTG	44	CGC
13	TAG	29	GTC	45	CCA
14	TAC	30	GGA	46	CCT
15	TTA	31	GGT	47	CCG

nation $(\mathbf{z}_{(j-1)r+1}^{jr}, \mathbf{z}_{jr+1}^{(j+1)r})$ in the sequence \mathbf{z} is r . Finally, the sequence \mathbf{z} of length μhr is segmented into $\lfloor \mu hr/k \rfloor$ blocks of sequences $\mathbf{y} = (y_1, y_2, \dots, y_k) \in \mathbb{Z}_q^k$ with specified length k , which can be used as the message sequences of the proposed code. Since the sequences \mathbf{y} are a subset of sequence \mathbf{z} and the maximum run-length of sequence \mathbf{z} is r (as long as the maximum run-length of the concatenation of any two adjacent subsequences with r symbols is r), the run-length of sequences \mathbf{y} is less than or equal to r .

Since the alphabet and maximal run-length are popularly used as $q = 4$ and $r = 3$ in DNA storage, respectively, we can consider converting a block of $\epsilon = 16$ bits to 48-ary $h = 3$ elements. If A = 0, T = 1, G = 2, and C = 3, an example of one-to-one mapping between a 48-ary element and q -ary subsequence $\mathbf{z}_{(j-1)r+1}^{jr}$ with r symbols for $j \in [1, \mu h]$ is given in Table 3.1.

3.3 Code construction and encoding for a non-binary SIDEC with a maximum run-length constraint code

We propose a method to construct a nonbinary SIDEC code and then present a systematic encoding algorithm of the proposed nonbinary SIDEC code. We design parity symbols that satisfy the following constraints: the maximum run-length r is three and the symbols can be obtained by mapping through tables. The outputs of the systematic encoding algorithm are the maximum run-length $r = 3$ constraint codewords when the maximum run-length r of the message sequence is not larger than three. Hence, the output codewords have two properties: single insertion/deletion error correction and the maximum run-length r constraint.

3.3.1 Code construction

Table 3.2: Forms of the proposed nonbinary systematic SIDEC code.

\mathbf{I}		I_1	I_2	I_3 (d_3)	\cdots	$I_{3(\sigma-1)}$ ($d_{3(\sigma-1)}$)	\cdots	I_m	I_{m+1}	\cdots	I_n
$\boldsymbol{\alpha}$		α_1	α_2	α_3	\cdots	$\alpha_{3(\sigma-1)}$	\cdots	α_m	α_{m+1}	\cdots	α_n
\mathbf{c}	c_0 (p_0)	c_1 (p_1)	c_2 (p_2)	c_3 (p_3)	\cdots	$c_{3(\sigma-1)}$ ($p_{3(\sigma-1)}$)	\cdots	c_m (p_m)	c_{m+1} (y_1)	\cdots	c_n (y_k)

The form of the proposed nonbinary systematic code $C_{a,b}(n+1)$ is shown in Table 3.2. The proposed code is constructed by the sequence $\mathbf{I} = (I_1, I_2, \cdots, I_{n+1})$ of length $n+1$, the auxiliary sequence $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \cdots, \alpha_n)$ of length n , and the codeword $\mathbf{c} = (c_0, c_1, c_2, \cdots, c_n)$ of length $n+1$. The codeword \mathbf{c} is $(\mathbf{p}_0^m, \mathbf{y}_1^k)$, which is the concatenation of the parity sequence $\mathbf{p}_0^m = (p_0, p_1, \cdots, p_m)$ of length $m+1$ and the message sequence $\mathbf{y}_1^k = (y_1, y_2, \cdots, y_k)$ of length k . To determine codeword \mathbf{c} , monotonically increasing integer sequence \mathbf{I} is explained in Definition 3.3.1. In

Table 3.2, the subsequence \mathbf{I}_1^n of \mathbf{I} is used, and I_{n+1} will be used for syndrome calculation later. When the length of message sequence k is given, I_{n+1} should be first determined for the sequence \mathbf{I} and the number of the parity symbols can be determined by I_{n+1} . However, since I_{n+1} is also recursively calculated after determining \mathbf{I}_1^n in Definition 1, it is not possible to determine I_{n+1} and $B = \lceil \log(I_{n+1}) \rceil$ directly. To solve this problem, we first provide the relationship between the message sequence length k and the value of B in Table 3.3, which can be understood after the sequence generation has ended. We also present the number of parity symbols by (3.5) and the length of codeword \mathbf{c} by (3.6), as shown in Table 3.3.

Table 3.3: The relationship between the message sequence length k , the value of B , the number of parity symbols $m + 1$, and the length of codeword \mathbf{c} , $n + 1$.

Message sequence length k	$B = \lceil \log(I_{n+1}) \rceil$	$\sigma = \lceil \frac{B}{2} \rceil$	Number of parity symbols $m + 1$	Length of codeword \mathbf{c}
$[1, 2] = [2^{B-2} - 1, 2^{B-1} - 2]$	3	2	6	$[7, 8], k + 3\sigma$
$[3, 6] = [2^{B-2} - 1, 2^{B-1} - 2]$	4	2	7	$[10, 13], k + 3\sigma + 1$
$[7, 14] = [2^{B-2} - 1, 2^{B-1} - 2]$	5	3	9	$[16, 23], k + 3\sigma$
$[15, 30] = [2^{B-2} - 1, 2^{B-1} - 2]$	6	3	10	$[25, 40], k + 3\sigma + 1$
$[31, 62] = [2^{B-2} - 1, 2^{B-1} - 2]$	7	4	12	$[43, 74], k + 3\sigma$
$[63, 126] = [2^{B-2} - 1, 2^{B-1} - 2]$	8	4	13	$[76, 139], k + 3\sigma + 1$
$[127, 254] = [2^{B-2} - 1, 2^{B-1} - 2]$	9	5	15	$[142, 269], k + 3\sigma$
$[255, 510] = [2^{B-2} - 1, 2^{B-1} - 2]$	10	5	16	$[271, 526], k + 3\sigma + 1$

Moreover, since the number of parity symbols depends on whether B is even or odd, and around half of B can make a special pattern in the sequence, we need

to define σ . From B , the value of σ is defined as

$$\sigma = \lceil \frac{B}{2} \rceil.$$

Conversely, from σ , the value of B can be defined as

$$B = \begin{cases} 2\sigma - 1 & \text{if } B \text{ is an odd number;} \\ 2\sigma & \text{if } B \text{ is an even number.} \end{cases} \quad (3.1)$$

From the relationship between the message sequence length k and the value of B in Table 3.3, we conclude that

$$k \in [2^{B-2} - 1, 2^{B-1} - 2]. \quad (3.2)$$

Definition 3.3.1. We define a positive, monotonically increasing integer sequence $\mathbf{I} = (I_1, I_2, \dots, I_{n+1})$ of length $n + 1$. When the length of message sequence k is predefined, the value of B can be determined according to (3.2). The sequence \mathbf{I} is defined as

$$\mathbf{I} = (\mathbf{T}_1, \mathbf{T}_2, \dots, \mathbf{T}_{\sigma-1}, \mathbf{L}, \mathbf{I}_m^{n+1}), \quad (3.3)$$

where \mathbf{T}_i for $i \in [1, \sigma - 1]$, \mathbf{L} , m , $n + 1$, and \mathbf{I}_m^{n+1} are expressed as

$$\mathbf{T}_i = (2^{2i-2}, 2^{2i-1}, d_{3i}) \quad \text{for } i \in [1, \sigma - 1], \quad (3.4)$$

$$\mathbf{L} = \begin{cases} (2^{2\sigma-2}) & \text{for } B = 2\sigma - 1; \\ (2^{2\sigma-2}, 2^{2\sigma-1}) & \text{for } B = 2\sigma, \end{cases}$$

$$m = \begin{cases} 3(\sigma - 1) + 2 & \text{for } B = 2\sigma - 1; \\ 3\sigma & \text{for } B = 2\sigma, \end{cases} \quad (3.5)$$

$$n + 1 = m + k + 1, \quad (3.6)$$

$$\mathbf{I}_m^{n+1} = \begin{cases} I_m = \begin{cases} 2^{2\sigma-2} + 1 & \text{for } B = 2\sigma - 1; \\ 2^{2\sigma-1} + 1 & \text{for } B = 2\sigma, \end{cases} \\ I_j = I_{j-1} + 1 & \text{for } j \in [m + 1, n + 1], \end{cases} \quad (3.7)$$

where $d_{3i} \in [2^{2i-1} + 1, 2^{2(i+1)-1} - 1]$ in (3.4) for $i \in [1, \sigma - 1]$.

We present some examples of the construction of the positive, monotonically increasing integer sequence \mathbf{I} . In Example 3.3.2, we can intuitively see that the integer sequence \mathbf{I} is a monotonically increasing sequence. The reason why some indices in \mathbf{T}_i for $i \in [1, \sigma - 1]$ are a multiple of three in (3.4) is that the fixed maximum run-length r is designed to be three in \mathbf{p}_0^m of the proposed code.

Example 3.3.2. It is assumed that the lengths of message sequences are $k = 126$ and 145. Then, according to (3.2), it can be determined that the values of B are 8 and 9, respectively. The values of $m = 12$ and 14, $d_3 \in [3, 3]$, $d_6 \in [9, 15]$, $d_9 \in [33, 61]$, and $d_{12} \in [129, 255]$. Then, the integer sequences \mathbf{I} in Definition 3.3.1 are given as

$$\mathbf{I} = (1, 2, d_3, 4, 8, d_6, 16, 32, d_9, 64, 128, I_m = 129, \\ 130, \dots, 255, 256) \quad \text{for } k = 126,$$

$$\mathbf{I} = (1, 2, d_3, 4, 8, d_6, 16, 32, d_9, 64, 128, d_{12}, 256, \\ I_m = 257, 258, \dots, 402, 403) \quad \text{for } k = 145.$$

Definition 3.3.3. Consider a sequence $\mathbf{c} = (c_0, c_1, c_2, \dots, c_n) \in \mathbb{Z}_q^{n+1}$ and its corresponding auxiliary sequence $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \dots, \alpha_n) \in \mathbb{Z}_2^n$, whose element α_i for $i \in [1, n]$ is given by

$$\alpha_i = \begin{cases} 0 & \text{if } c_i < c_{i-1}; \\ 1 & \text{if } c_i \geq c_{i-1}. \end{cases} \quad (3.8)$$

A syndrome $Syn(\boldsymbol{\alpha})$ of the auxiliary sequence $\boldsymbol{\alpha}$ and the check summation $\text{sum}(\mathbf{c})$ of the codeword \mathbf{c} are defined as

$$Syn(\boldsymbol{\alpha}) = \sum_{i=1}^n I_i \alpha_i \pmod{I_{n+1}}, \quad (3.9)$$

$$\text{sum}(\mathbf{c}) = \sum_{i=0}^n c_i \pmod{q}. \quad (3.10)$$

For $a \in \mathbb{Z}_q$ and $b \in \mathbb{Z}_{I_{n+1}}$, the q -ary SIDEC code with parameters $n + 1$, a , and b is defined as

$$C_{a,b}(n + 1) = \{\mathbf{c} \in \mathbb{Z}_q^{n+1} : Syn(\boldsymbol{\alpha}) = b \pmod{I_{n+1}}, \\ \text{sum}(\mathbf{c}) = a \pmod{q}\}. \quad (3.11)$$

Theorem 3.3.4. *For any codeword $\mathbf{c} \in C_{a,b}(n + 1)$, $a \in \mathbb{Z}_q$ and $b \in \mathbb{Z}_{I_{n+1}}$, the codeword \mathbf{c} has the SIDEC property.*

Proof. The sequence \mathbf{I} is a positive, monotonically increasing sequence. Hence, the sequence \mathbf{I} is a monotonically increasing code [29]. Since monotonically increasing codes are also SIDEC codes [29] and $\text{sum}(\mathbf{c}) = a \pmod{q}$ satisfies the property of nonbinary VT codes [60], $\mathbf{c} \in C_{a,b}(n + 1)$ is a codeword of a SIDEC code.

3.3.2 Encoding algorithm

We present a systematic encoding algorithm for nonbinary SIDEC codes, where the maximum run-length r in the parity sequence is three. To satisfy the run-length constraint between a sequence of parity symbols and a sequence of message symbols, the last parity symbol is designed to be always different from the first message symbol. The output codewords have the properties of nonbinary SIDEC codes and the maximum run-length r which depends on the maximum run-length

of the message sequence.

Overview

This systematic encoding algorithm converts a message sequence \mathbf{y}_1^k with the maximum run-length r constraint into a SIDEC codeword with the maximum run-length r constraint. The output codeword $\mathbf{c} \in C_{a,b}(n+1) \cap G_r^{n+1}$ is defined by $\mathbf{c} = (\mathbf{p}_0^m, \mathbf{y}_1^k)$. The last symbol p_m of \mathbf{p}_0^m is used for avoiding the run-length between \mathbf{p}_0^m and \mathbf{y}_1^k , and $(\sigma - 1)$ symbols of \mathbf{p}_0^m whose indices are a multiple of three are used for avoiding the run-length in the parity sequence. The other symbols of \mathbf{p}_0^m are represented by the value of $Syn(\boldsymbol{\alpha})$.

We explain the encoding algorithm in six steps according to Table 3.2. The parity sequence \mathbf{p}_0^m and message sequence \mathbf{y}_1^k correspond to the partial codeword $\mathbf{c}_0^m = (c_0, c_1, \dots, c_m)$ and $\mathbf{c}_{m+1}^n = (c_{m+1}, c_{m+2}, \dots, c_n)$, respectively.

Encoding algorithm steps

Step 1. The length of the message sequence \mathbf{y}_1^k is predefined as k . The subsequence \mathbf{I}_1^n of \mathbf{I} and the value of I_{n+1} can be determined by Definition 3.3.1. Then, the length $m + 1$ of the parity sequence \mathbf{p}_0^m can be obtained by (3.5) with \mathbf{I} .

This encoding algorithm requires d_{3i} for $i \in [1, \sigma - 1]$, a , b , and message sequence $\mathbf{y}_1^k \in G_r^k$ as input parameters. The output codeword $\mathbf{c} \in C_{a,b}(n+1) \cap G_r^{n+1}$ is obtained from the proposed encoding algorithm. The symbols p_{3i} are used for avoiding the run-length in the parity sequence \mathbf{p}_0^m for $i \in [1, \sigma - 1]$ and the symbol p_m is used for avoiding the run-length between the parity sequence \mathbf{p}_0^m and message sequence \mathbf{y}_1^k . When the message sequence \mathbf{y}_1^k embeds into the partial codeword \mathbf{c}_{m+1}^n , the corresponding partial auxiliary sequence $\boldsymbol{\alpha}_{m+2}^n = (\alpha_{m+2}, \alpha_{m+3}, \dots, \alpha_n)$ with length $k - 1$ can be obtained according to (3.8).

Step 2. Initialize $\alpha_m = 0$, since the parity symbol $c_m < c_{m-1}$ in (3.8) when

$\alpha_m = 0$. The constraint $\alpha_m = 0$ prevents an increase in the maximum run-length r between the parity sequence \mathbf{p}_0^m and message sequence \mathbf{y}_1^k . After initializing $\alpha_m = 0$, we determine the symbol $c_m = \min([q] \setminus y_1)$ according to the desirable value of c_m , as shown in Tables 3.4 and 3.5. The symbol c_m can be determined as $c_m = \max([q] \setminus y_1)$ according to the desirable value of c_m in Tables 3.4 and 3.5 if α_m is flipped from 0 to 1. Meanwhile, the value of α_{m+1} can be determined after the symbol c_m is determined according to (3.8). Then, we set all the values of α_{3i} to 1 for $i \in [1, \sigma - 1]$. Due to the condition $c_j < c_{j-1}$ in (3.8) when $\alpha_j = 0$, if the partial auxiliary sequence $\boldsymbol{\alpha}_{j-q+1}^j = (\alpha_{j-q+1}, \alpha_{j-q+2}, \dots, \alpha_j)$ is $(0, 0, \dots, 0)$ of length q for $j \in [q, n]$, then the symbol c_{j-q} should be larger than $q - 1$, even though the symbol $c_j = 0$ and the value of symbol c_τ increases by one for $\tau = \langle j - 1, -1, j - q + 2 \rangle$. When $\alpha_j = 1$ and the partial auxiliary sequence $\boldsymbol{\alpha}_{j-(q-1)r-1}^j = (\alpha_{j-(q-1)r-1}, \alpha_{j-(q-1)r-2}, \dots, \alpha_j)$ of length $(q-1)r$ is $(1, 1, \dots, 1)$ for $j \in [(q-1)r, n]$, the symbol $c_{j-(q-1)r}$ should be less than 0, even though all symbols in $\mathbf{c}_{j-r+1}^j = (c_{j-r+1}, c_{j-r+2}, \dots, c_j)$ are the same as $q - 1$. Additionally, all symbols in sequence $\mathbf{c}_{j-(\tau-1)r+1}^{j-\tau r} = (c_{j-(\tau-1)r+1}, c_{j-(\tau-1)r+2}, \dots, c_{j-\tau r})$ are the same and the symbol value of $\mathbf{c}_{j-(\tau-1)r+1}^{j-\tau r}$ decreases by one for $\tau = \langle 1, 1, q - 1 \rangle$, since the codeword \mathbf{c} should satisfy the run-length constraint and the condition $c_j \leq c_{j-1}$ in (3.8). We define these two situations, where a symbol should be larger than $q - 1$ or less than zero in the codeword \mathbf{c} due to several ones or zeros in the auxiliary sequence $\boldsymbol{\alpha}$, as **codeword construct failure** (CCF).

Step 3. The remaining bits $(\alpha_1, \alpha_2, \alpha_4, \alpha_5, \dots, \alpha_{m-1})$ in auxiliary sequence $\boldsymbol{\alpha}$ are used for checking that $Syn(\boldsymbol{\alpha}) = b \pmod{I_{n+1}}$ and its corresponding I_j has a

Table 3.4: The relationship between $\alpha_{3(\sigma-1)}^m$ and $c_{3(\sigma-1)}^m$.

$(\alpha_{3(\sigma-1)}, \alpha_{3(\sigma-1)+1}, \alpha_m)$	$c_{3(\sigma-1)}$	$c_{3(\sigma-1)+1}$	Desirable value of c_m	c_m
(0, 0, 0)	$c_m + 2$	$c_m + 1$	$[0, q - 3]$	$c_m \in \min([q] \setminus y_1)$
(0, 1, 0)	0	$q - 1$	$[0, q - 2]$	
(1, 0, 0)	$q - 1$	$c_m + 1$	$[0, q - 3]$	
(1, 1, 0)	$q - 1$	$q - 1$	$[0, q - 2]$	
(0, 0, 1)	$c_{3(\sigma-1)+1} + 1$	0	$[0, q - 1]$	$c_m \in \max([q] \setminus y_1)$
(0, 1, 1)	0	$c_m - 1$	$[1, q - 1]$	
(1, 0, 1)	$q - 1$	0	$[0, q - 1]$	
(1, 1, 1)	$c_m - 1$	$c_m - 1$	$[1, q - 1]$	

Table 3.5: The relationship between $\alpha_{3(l-1)}^{3l}$ and $c_{3(l-1)}^{3l}$ for $l \in [1, \sigma]$ and $m = 3\sigma$.

$(\alpha_{3(l-1)}, \alpha_{3(l-1)+1}, \alpha_{3(l-1)+2}, \alpha_{3l})$	$c_{3(l-1)}$	$c_{3(l-1)+1}$	$c_{3(l-1)+2}$	Desirable value of c_{3l}	$c_{3\sigma}$
(0, 0, 0, 0)	$c_{3l} + 3$	$c_{3l} + 2$	$c_{3l} + 1$	$[0, q - 5]$	$c_m \in \min([q] \setminus y_1)$
(1, 0, 0, 0)	$q - 1$	$c_{3l} + 2$	$c_{3l} + 1$	$[0, q - 4]$	
(1, 0, 1, 0)	$q - 1$	0	$c_{3l} + 1$	$[0, q - 2]$	
(1, 1, 0, 0)	$q - 1$	$q - 1$	$c_{3l} + 1$	$[0, q - 3]$	
(1, 1, 1, 0)	$q - 1$	$q - 1$	$q - 1$	$[0, q - 2]$	
(0, 0, 1, 0)	$c_{3(l-1)+1} + 1$	0	$q - 1$	$[0, q - 2]$	
(0, 1, 0, 0)	0	$q - 1$	$c_{3l} + 1$	$[0, q - 3]$	$c_m \in \max([q] \setminus y_1)$
(0, 1, 1, 0)	0	$q - 1$	$q - 1$	$[0, q - 2]$	
(1, 0, 0, 1)	$q - 1$	$c_{3(l-1)+2} + 1$	0	$[0, q - 1]$	
(1, 0, 1, 1)	$q - 1$	0	c_{3l}	$[1, q - 1]$	
(1, 1, 0, 1)	$q - 1$	$q - 1$	0	$[0, q - 1]$	
(1, 1, 1, 1)	$c_{3l} - 2$	$c_{3l} - 1$	$c_{3l} - 2$	$[1, q - 1]$	
(0, 0, 0, 1)	$c_{3(l-1)+1} + 1$	$c_{3(l-1)+2} + 1$	0	$[0, q - 1]$	
(0, 0, 1, 1)	$c_{3(l-1)+1} + 1$	0	$c_{3l} - 1$	$[1, q - 1]$	
(0, 1, 0, 1)	0	$q - 1$	0	$[0, q - 1]$	
(0, 1, 1, 1)	0	$c_{3l} - 1$	$c_{3l} - 1$	$[1, q - 1]$	

power of 2. Hence, we define β as

$$\begin{aligned}
\beta &= \sum_{j=1}^2 \alpha_j \cdot 2^{j-1} + \sum_{i=2}^{\sigma-1} \sum_{j=3i-2}^{3i-1} \alpha_j \cdot 2^{j-i} \\
&\quad + \sum_{j=3\sigma-2}^{m-1} \alpha_j \cdot 2^{j-\sigma} \pmod{I_{n+1}} \\
&= b - \sum_{i=1}^{\sigma-1} \sum_{j=3i} \alpha_j d_j - \alpha_m I_m \\
&\quad - \sum_{j=1}^k \alpha_{j+m} I_{j+m} \pmod{I_{n+1}}.
\end{aligned} \tag{3.12}$$

From the calculation of (3.12), we can determine the unique remaining bits $(\alpha_1, \alpha_2, \alpha_4, \alpha_5, \dots, \alpha_{m-1})$ to represent the value of β .

Step 4. After obtaining the unique auxiliary sequence α , we first determine the symbols of $\mathbf{c}_{3(\sigma-1)}^m = (c_{3(\sigma-1)}, c_{3(\sigma-1)+1}, c_m)$ related to $\alpha_{3(\sigma-1)}^m = (\alpha_{3(\sigma-1)}, \alpha_{3(\sigma-1)+1}, \alpha_m)$ according to Table 3.4 if the length of $\alpha_{3(\sigma-1)}^m$ is three. If the length of $\alpha_{3(\sigma-1)}^m = (\alpha_{3(\sigma-1)}, \alpha_{3(\sigma-1)+1}, \alpha_{3(\sigma-1)+2}, \alpha_m)$ is four, $\mathbf{c}_{3(\sigma-1)}^m = (c_{3(\sigma-1)}, c_{3(\sigma-1)+1}, c_{3(\sigma-1)+2}, c_m)$ related to $\alpha_{3(\sigma-1)}^m$ is determined according to Table 3.5. After the symbols $\mathbf{c}_{3(\sigma-1)}^m$ are determined, if CCF occurs, we flip α_m .

Step 5. We determine $\mathbf{c}_{3(t-1)}^{3t} = (c_{3(t-1)}, c_{3(t-1)+1}, c_{3(t-1)+2}, c_{3t})$ related to $\alpha_{3(t-1)}^{3t} = (\alpha_{3(t-1)}, \alpha_{3(t-1)+1}, \alpha_{3(t-1)+2}, \alpha_{3t})$ according to Table 3.5, for $t = \langle \sigma - 1, -1, 2 \rangle$. After the symbols $\mathbf{c}_{3(t-1)}^{3t}$ are determined, if CCF occurs or if the maximum run-length $r > 3$, we flip the current α_{3t} .

Step 6. The rest of the symbols $\mathbf{c}_0^2 = (c_0, c_1, c_2)$ in the output codeword \mathbf{c} is used for computing to satisfy the sum $(\mathbf{c}) = a \pmod{q}$. We define γ as

$$\begin{aligned} \gamma &= c_0 + c_1 + c_2 \pmod{q} \\ &= a - \sum_{i=3}^n c_i \pmod{q}. \end{aligned} \tag{3.13}$$

According to the value of γ and the $\alpha_1^3 = (\alpha_1, \alpha_2, \alpha_3)$, we can determine the symbols \mathbf{c}_0^2 . Since sometimes there is not only one case of \mathbf{c}_0^2 that satisfies the value of γ and the α_1^3 , we should determine the case to avoid the run-length issue as much as possible. Similarly, after the symbols \mathbf{c}_0^2 are determined, if CCF occurs or if the maximum run-length $r > 3$, we flip α_3 .

From (3.8), the auxiliary sequence α has a strong relationship with the codeword \mathbf{c} . Obviously, if there are many consecutive ones in α , the codeword construction can fail when the codeword should satisfy the maximum run-length constraint. Hence, we first consider $\alpha_{3(l-1)}^{3l} = (1, 1, 1, 1)$ in Table 3.5 for $l \in [1, \sigma]$

and $m = 3\sigma$. Then, there are a total of 19 cases of $\mathbf{c}_{3^{(l-1)}}^{3l}$ that correspond to $\alpha_{3^{(l-1)}}^{3l} = (1, 1, 1, 1)$, e.g., $\mathbf{c}_{3^{(l-1)}}^{3l} = (c_{3l} - 1, c_{3l}, c_{3l}, c_{3l})$, $\mathbf{c}_{3^{(l-1)}}^{3l} = (c_{3l} - 2, c_{3l}, c_{3l}, c_{3l})$, etc. Based on the trials, we finally determine $\mathbf{c}_{3^{(l-1)}}^{3l} = (c_{3l} - 2, c_{3l} - 1, c_{3l} - 1, c_{3l})$. Similarly, for the case of $\alpha_{3^{(l-1)}}^m = (1, 1, 1)$ in Table 3.4, we determine its corresponding $\mathbf{c}_{3^{(l-1)}}^m = (c_m - 1, c_m - 1, c_m)$. For the concatenation issue, the first symbol $c_{3^{(l-1)}}$ in $\mathbf{c}_{3^{(l-1)}}^{3l}$ will be the last symbol in the partial sequence $\mathbf{c}_{3^{(l-2)}}^{3^{(l-1)}}$. Refer to the value of c_{3l} in Table 3.5, which is the last symbol in $\mathbf{c}_{3^{(l-1)}}^{3l}$, we can determine the first symbol $c_{3^{(l-1)}}$ in $\mathbf{c}_{3^{(l-1)}}^{3l}$ as maximum available symbol (i.e., $q - 1$) and minimum available symbol (i.e., 0) when the first bit $\alpha_{3^{(l-1)}}$ in $\alpha_{3^{(l-1)}}^{3l}$ is 1 and 0, respectively. According to (3.8), we know that $c_i < c_{i-1}$ if $\alpha_i = 0$. Hence, when consecutive zeros occur in the partial auxiliary sequence $\alpha_{3^{(l-1)}}^{3l}$, we determine the symbol corresponding to the last zero position in $\alpha_{3^{(l-1)}}^{3l}$ as the minimum available symbol (i.e., 0) and increase the value of each symbol to the left of this symbol by one step by step. The symbol values of $c_{3^{(l-1)+1}}$ and $c_{3^{(l-1)+2}}$ are determined by $\alpha_{3^{(l-1)+1}}$ and $\alpha_{3^{(l-1)+2}}$. In most cases, we can determine the $c_{3^{(l-1)+1}} = c_{3^{(l-1)+2}} = q - 1$ always satisfy $\alpha_{3^{(l-1)+1}} = \alpha_{3^{(l-1)+2}} = 1$ according to (3.8) except $\alpha_{3^{(l-1)}}^{3l} = (1, 1, 1, 1)$ and $(0, 1, 1, 1)$. If $\alpha_{3^{(l-1)}}^{3^{(l-1)+2}} = (0, 1, 0)$, we can determine the symbol value $c_{3^{(l-1)+1}}$ as $q - 1$ to satisfy $\alpha_{3^{(l-1)+1}} = 1$ according to (3.8). On the contrary, we can determine the symbol value $c_{3^{(l-1)+1}} = 0$ if $\alpha_{3^{(l-1)}}^{3^{(l-1)+2}} = (1, 0, 1)$. Similarity, the symbol value $c_{3^{(l-1)+2}}$ can be determined as $q - 1$ and 0 if $\alpha_{3^{(l-1)+1}}^{3l} = (0, 1, 0)$ and $(1, 0, 1)$, respectively. The value of $\mathbf{c}_{3^{(\sigma-1)}}^m$ in Table 3.4 is determined similarly to $\mathbf{c}_{3^{(l-1)}}^{3l}$ in Table 3.5.

3.3.3 Special case with $k = 190$ and 191 and the overall of encoding algorithm

While simulating the encoding algorithm in Section III.B, we find that in most of cases, codewords can be constructed without flipping α_{3i} for $i \in [1, \sigma - 1]$ in

Steps 5 and 6. However, flipping α_3 in Step 6 occurs at two cases where the message sequence length $k = 190$ and 191 , corresponding to $I_{n+1} = 433$ and 434 , respectively. In these cases, the codewords are not constructed successfully. When $\beta = 431$ in (3.12) for $I_{n+1} = 433$ and 434 , the corresponding partial auxiliary sequence is $\alpha_1^m = (\alpha_1, \alpha_2, \dots, \alpha_m) = (1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0)$. Then, we can acquire the partial codeword $\mathbf{c}_3^m = (c_3, c_4, \dots, c_m) = (1, 2, 2, 3, 0, 2, 3, 0, 2, 3, 3, c_m)$ for $q = 4$ according to Tables 3.4 and 3.5. We assume that $\gamma = 3$ in (3.13) and the partial codeword $\mathbf{c}_0^2 = (c_0, c_1, c_2) = (1, 1, 1)$ to satisfy $\gamma = 3$ when $\alpha_1^3 = (\alpha_1, \alpha_2, \alpha_3) = (1, 1, 1)$. However, since the run-length of \mathbf{c}_0^3 is four, it violates the maximum run-length $r = 3$ in parity sequence. According to the encoding algorithm, α_3 needs to be flipped from 1 to 0, and the value of β is again calculated as $\beta = 1$ and 0 . In this case, the corresponding partial auxiliary sequence $\alpha_1^m = (1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0)$ and $(0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0)$, respectively. Since there are more than three consecutive zeros in the corresponding partial auxiliary sequence α_1^m and $q = 4$, the codeword cannot be constructed successfully due to CCF occurs in Step 6. We conclude that these cases are special cases that cannot be constructed successfully, even if the corresponding α_{3i} is flipped for $i \in [1, \sigma - 1]$ in Steps 5 and 6.

To address special cases where the codeword cannot be constructed successfully, even if the corresponding α_{3i} is flipped for $i \in [1, \sigma - 1]$ in Steps 5 and 6, we make some modifications in Steps 5 and 6. Let $\delta = (\alpha_3, \alpha_6, \dots, \alpha_{3(\sigma-1)})$ be a sequence that consists of the values α_{3i} in auxiliary sequence α for $i \in [1, \sigma - 1]$ and $\delta_i^j = (\alpha_{3i}, \alpha_{3(i+1)}, \dots, \alpha_{3j})$ for $i, j \in [1, \sigma - 1]$ and $i \leq j$. Notice that the corresponding α_{3i} in auxiliary sequence α should also be changed when the value in sequence δ has been changed. The modification is that updating $temp_delta = \delta_t^{\sigma-1} = (\alpha_{3t}, \alpha_{3(t+1)}, \dots, \alpha_{3(\sigma-1)})$ and $\delta_t^{\sigma-1} = Le_{\sigma-t}(Le_{\sigma-t}^{-1}(temp_delta) - 1)$ until the partial codeword $\mathbf{c}_{3(t-1)}^{3t}$ can be constructed successfully for $t = \langle \sigma, -1, 2 \rangle$. For

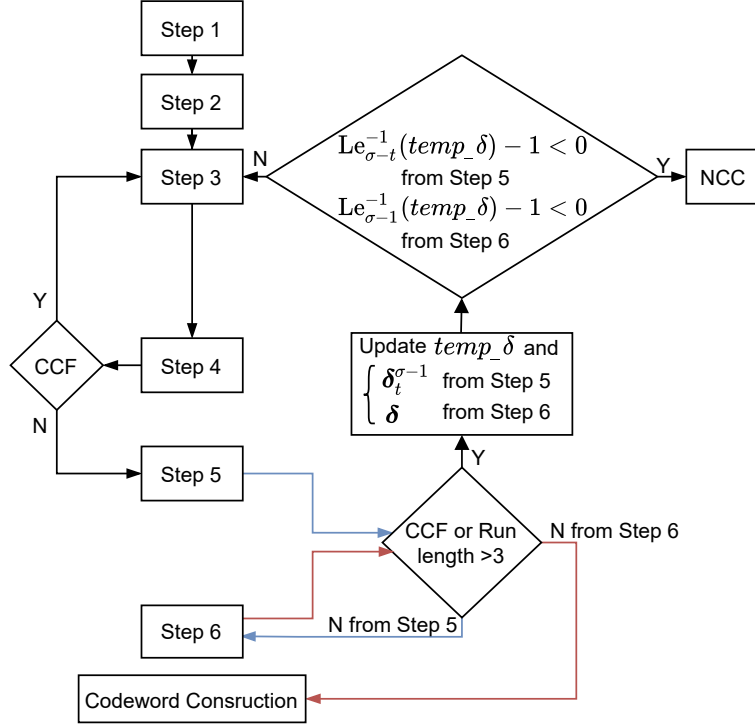


Figure 3.2: The flow chart of the proposed encoding algorithm.

example, from Step 2, the initial sequence δ is set as $\delta = (1, 1, \dots, 1)$. If $\mathbf{c}_{3(t-1)}^{3t}$ mapping from Table 3.5 causes CCF to occur or if run-length $r > 3$, we set $temp_{\delta} = \delta_t^{\sigma-1} = (1, 1, \dots, 1)$ and update $\delta_t^{\sigma-1} = (0, 1, \dots, 1)$, which represent flipping α_{3t} from 1 to 0. We then go back to Step 3. If $\mathbf{c}_{3(t-1)}^{3t}$ causes CCF to occur again or if the run-length still $r > 3$, we update $temp_{\delta} = (0, 1, \dots, 1)$ and $\delta_t^{\sigma-1} = (1, 0, \dots, 1)$, which represents flipping $\alpha_{3(t+1)}$ from 1 to 0, and then initialize α_{3t} (since the codeword cannot be successfully constructed by only flipping α_{3t}). The update will stop until $\mathbf{c}_{3(t-1)}^{3t}$ be constructed successfully. If $\mathbf{c}_{3(t-1)}^{3t}$ always causes CCF to occur or if the run-length $r > 3$ and $Le_{\sigma-t}^{-1}(temp_{\delta}) - 1$ is less than zero, the codeword cannot be constructed and we define this situation as **no code construction** (NCC). The modification in Step 6 is similar to the one in the Step 5. The overall encoding algorithm and the accompanying flow chart are

summarized by Algorithm 3 and Fig. 3.2.

Algorithm 3: Encoding the SIDEC code with the maximum run-length r constraint

Input: d_{3i} for $i \in [1, \sigma - 1]$, a , b , and message sequence $\mathbf{y}_1^k \in G_r^k$.
Output: codeword $\mathbf{c} = (\mathbf{p}_0^m, \mathbf{y}_1^k) \in C_{a,b}(n+1) \cap G_r^k$

- 1 Step 1: Embed message sequence $\mathbf{y}_1^k \rightarrow$ the partial auxiliary sequence α_{m+2}^n .
- 2 Step 2: Set $\alpha_m = 0 \rightarrow c_m = \min([q] \setminus y_1) \rightarrow \alpha_{m+1}$ and all $\alpha_{3i} = 1$ for $i \in [1, \sigma - 1]$.
- 3 Step 3: Calculate β in (3.12) to satisfy $Syn(\alpha) = b \pmod{I_{n+1}}$.
- 4 Step 4: $\alpha_{3(\sigma-1)}^m \rightarrow \mathbf{c}_{3(\sigma-1)}^m$.
- 5 **if** the length of $\alpha_{3(\sigma-1)}^m$ is three **then**
- 6 | determine $\mathbf{c}_{3(\sigma-1)}^m$ according to $\alpha_{3(\sigma-1)}^m$ with Table 3.4.
- 7 **else if** the length of $\alpha_{3(\sigma-1)}^m$ is four **then**
- 8 | determine $\mathbf{c}_{3(\sigma-1)}^m$ according to $\alpha_{3(\sigma-1)}^m$ with Table 3.5.
- 9 | **if** $\mathbf{c}_{3(\sigma-1)}^m$ causes CCF **then**
- 10 | | flip $\alpha_m \rightarrow c_m = \max([q] \setminus y_1) \rightarrow \alpha_{m+1}$.
- 11 | | **goto** Step 3.
- 12 Step 5: $\alpha_{3(t-1)}^{3t} \rightarrow \mathbf{c}_{3(t-1)}^{3t}$
- 13 **for** $t = \langle \sigma - 1, -1, 2 \rangle$ **do**
- 14 | determine $\mathbf{c}_{3(t-1)}^{3t}$ according to $\alpha_{3(t-1)}^{3t}$ with Table 3.5.
- 15 | **if** $\mathbf{c}_{3(t-1)}^{3t}$ causes CCF or run-length $r > 3$ **then**
- 16 | | $temp_delta = \delta_t^{\sigma-1}$,
- 17 | | **if** $Le_{\sigma-t}^{-1}(temp_delta) - 1 < 0$ **then**
- 18 | | | print (NCC), exit.
- 19 | | $\delta_t^{\sigma-1} = Le_{\sigma-t}^{-1}(Le_{\sigma-t}^{-1}(temp_delta) - 1)$.
- 20 | | **goto** Step 3.
- 21 Step 6: Calculate γ in (3.13) to satisfy $sum(\mathbf{c}) = a \pmod{q}$ and $\gamma \rightarrow \mathbf{c}_0^2$ according to α_1^3 .
- 22 **if** \mathbf{c}_0^2 causes CCF or run-length $r > 3$ **then**
- 23 | $temp_delta = \delta$. **if** $Le_{\sigma-1}^{-1}(temp_delta) - 1 < 0$ **then**
- 24 | | print (NCC), exit.
- 25 | $\delta = Le_{\sigma-1}^{-1}(Le_{\sigma-1}^{-1}(temp_delta) - 1)$.
- 26 | **goto** Step 3.

According to the encoding algorithm, the codewords are successfully constructed. The simulation results are shown in Section 3.5. In Fig. 3.2, Y, and N are represented encoding step number, yes, and no, respectively.

3.4 Decoding for an insertion or deletion error

Due to the strong relationship between the auxiliary sequence $\boldsymbol{\alpha}$ and codeword \mathbf{c} of $C_{a,b}(n+1)$ code, when a single insertion/deletion error occurs in the codeword \mathbf{c} of $C_{a,b}(n+1)$ code, a single insertion/deletion error occurs in the corresponding auxiliary sequence $\boldsymbol{\alpha}$ at the same position. Recall that the sequence \mathbf{I} is a monotonically increasing sequence; hence, we can acquire the decoding algorithm by the method used in [58]. Since the $C_{a,b}(n+1)$ code has similar properties as the nonbinary VT codes [60], we can decode for a single insertion/deletion error. Since decoding for an insertion error is similar, we will briefly introduce the decoding method for the deletion error but omit the explanation of decoding for insertion in this paper.

We assume that a deletion error occurs at the position $f \in [1, n]$. From the received codeword $\mathbf{c}' = (c'_0, c'_1, \dots, c'_{f-1}, c'_{f+1}, \dots, c'_{n+1})$ of length n , the auxiliary sequence $\boldsymbol{\alpha}' = (\alpha'_1, \alpha'_2, \dots, \alpha'_{f-1}, \alpha'_{f+1}, \dots, \alpha'_n)$ of length $n-1$ can be recalculated, where $\alpha'_f \in [2]$ and $c'_f \in [q]$. To determine the syndrome, two mappings are defined as

$$L_{\mathbf{I}}^{(0)}(f, \boldsymbol{\alpha}') = \begin{cases} \sum_{j=1}^{f-1} (1 - \alpha'_j) \cdot (I_{j+1} - I_j) & \text{for } i \neq 1; \\ 0 & \text{for } i = 1, \end{cases}$$

$$R_{\mathbf{I}}^{(1)}(f, \boldsymbol{\alpha}') = \begin{cases} \sum_{j=f}^{n-1} \alpha'_j \cdot (I_{j+1} - I_j) & \text{for } i \neq n; \\ 0 & \text{for } i = n. \end{cases}$$

Here, $L_{\mathbf{I}}^{(0)}(f, \boldsymbol{\alpha}')$ and $R_{\mathbf{I}}^{(1)}(f, \boldsymbol{\alpha}')$ represent the operations on all zeros to the

left of the f -th position and on all of the ones to the right of the f -th position in the received auxiliary sequence α' , respectively. We denote $w_{\mathbf{I}}(\alpha') = R_{\mathbf{I}}^{(1)}(1, \alpha')$, which coincides with the Hamming weight of the received auxiliary sequence α' if the integer sequence $\mathbf{I} = (1, 2, \dots, n+1)$. The difference in the syndrome values between α and α' is defined as $\Delta = b - \text{Syn}(\alpha') \pmod{I_{n+1}}$.

The deleted value α'_f is determined by comparing the Δ value and $w_{\mathbf{I}}(\alpha')$ as

$$\alpha'_f = \begin{cases} 0 & \text{for } \Delta \leq w_{\mathbf{I}}(\alpha'); \\ 1 & \text{for } \Delta > w_{\mathbf{I}}(\alpha'). \end{cases}$$

The sets of possible deletion positions f are $\{f \in [1, n], |R_{\mathbf{I}}^{(1)}(f, \alpha') = \Delta\}$ or $\{f \in [1, n], |L_{\mathbf{I}}^{(0)}(f, \alpha') = \Delta - w_{\mathbf{I}}(\alpha') - I_1\}$ if the deletion value $\alpha'_f = 0$ or 1 , respectively.

After we find the deleted value and possible position of the recalculated auxiliary sequence α' , we can infer the deleted value and the possible position in the received codeword \mathbf{c}' from the relationship between the deleted symbol c'_f and its left symbol c'_{f-1} according to (3.8). Similar to the decoding of nonbinary VT codes, we define the deleted symbol c'_f as

$$c'_f = a - \text{sum}(\mathbf{c}') \pmod{q}.$$

With the above method, we can successfully decode a deletion or insertion error, and these results are shown in Section 3.5.

3.5 Simulation and Comparison results

We first present the simulation results for our proposed code construction, determine the feasibility of the proposed encoding algorithm, and verify successful

decoding when a single insertion/deletion error occurs in the codeword. Then, we present the comparison results between relevant work [2] and our work to present the improvement of our code design.

In this simulation, we determine the values of $q = 4$ and $r = 3$. We segment message sequences \mathbf{y}_1^k into different lengths $k = 60, 120, 127, 135, 145, 155, 165, 175, 185$, and 400 . The reason why the length k is chosen as 127 and 135 is that, even though the two adapters of 24 nt [20] are appended to the head and tail of encoded codeword, respectively, the total length still meets the length of strands in DNA storage, which ranges from 160 - 200 nt [4, 20, 18]. Additionally, the length k is chosen from 145 to 185 because the adapters are not considered, and the length of encoded codeword is satisfied the length of strands in DNA storage. Moreover, the reason why we choose k as $60, 120$, and 400 is that different values B (i.e., $7, 8$, and 10 , respectively) need to be considered for general cases. Since the total number of message sequences is very large, we randomly choose 1.5×10^8 message sequences from the entire message sequences set in order to verify the feasibility of our proposed encoding algorithm. Then, the lengths of the encoded output codewords are $72, 133, 138, 150, 160, 170, 180, 190, 200$, and 415 , respectively. Since the predefined parameters $a \in [q]$ and $b \in [I_{n+1}]$ have many combinations, we consider two representative scenarios: one where $a = 0$ and $b = 1$ and another where a is a random number in $[q]$ and b is a random number in $[I_{n+1}]$. The results of the successful encoding are listed in Tables 3.6 and 3.7.

✓ and ✗ in Tables 3.6 and 3.7 denote successful encoding result and encoding failure result, respectively. From these results, the length of 190 codewords corresponds to $I_{n+1} = 433$, and the proposed encoding algorithm can generate the codewords construct successfully.

We assume that a single deletion or insertion error occurs in the received codewords. The position of a single deletion or insertion error is randomly determined

Table 3.6: The encoding results for different codeword lengths with $a = 0$ and $b = 1$.

Number of sequences	Message length	Codeword length	Successful encoding
1.5×10^8	60	72	✓
1.5×10^8	120	133	✓
1.5×10^8	127	138	✓
1.5×10^8	135	150	✓
1.5×10^8	145	160	✓
1.5×10^8	155	170	✓
1.5×10^8	165	180	✓
1.5×10^8	175	190	✓
1.5×10^8	185	200	✓
1.5×10^8	400	415	✓

Table 3.7: The encoding results for different codeword lengths with a is a random number in $[q]$ and b is a random number in $[I_{n+1}]$.

Number of sequences	Message length	Codeword length	Successful encoding
1.5×10^8	60	72	✓
1.5×10^8	120	133	✓
1.5×10^8	127	138	✓
1.5×10^8	135	150	✓
1.5×10^8	145	160	✓
1.5×10^8	155	170	✓
1.5×10^8	165	180	✓
1.5×10^8	175	190	✓
1.5×10^8	185	200	✓
1.5×10^8	400	415	✓

and the inserted symbol is randomly chosen from $[q]$ in the received codewords. Then, the decoding algorithm in Section IV is used to correct a single deletion or insertion. The simulation decoding results for single deletion and single insertions are listed in Tables 3.8 and 3.9, respectively. ✓ and X in Tables 3.8 and 3.9 also denote successful decoding result and decoding failure result, respectively. From Tables 3.8 and 3.9, all codewords with a single deletion/insertion are successfully

decoded.

Table 3.8: The decoding results for different codeword lengths with a single deletion error.

Number of received sequences	Received codeword length	Successful decoding
1.5×10^8	71	✓
1.5×10^8	132	✓
1.5×10^8	137	✓
1.5×10^8	149	✓
1.5×10^8	159	✓
1.5×10^8	169	✓
1.5×10^8	179	✓
1.5×10^8	189	✓
1.5×10^8	199	✓
1.5×10^8	414	✓

Table 3.9: The decoding results for different codeword lengths with a single insertion error.

Number of received sequences	Received codeword length	Successful decoding
1.5×10^8	73	✓
1.5×10^8	134	✓
1.5×10^8	139	✓
1.5×10^8	151	✓
1.5×10^8	161	✓
1.5×10^8	171	✓
1.5×10^8	181	✓
1.5×10^8	191	✓
1.5×10^8	201	✓
1.5×10^8	416	✓

3.5.1 Comparison results of related work [2] and our work

Among the related works, the systematic encoding method for nonbinary SIDEC code [2] is most relevant and we provide comparison results with our work. For fair

comparison with the related work [2], we still consider the 1.5×10^8 message sequences with maximum run-length $r = 3$ as inputs, and then make codewords with length $n = 150$ by the systematic encoding method in [2]. The code construction in [2] did not consider the maximum run-length constraint and CCF issues which are caused by some parity symbols c_{2^j} located between two message symbols for $j \in [2, \lceil \log n + 1 \rceil]$. Then, the parity symbols c_{2^j} should be less than zero according to (3.8) if the message symbol $c_{2^{j-1}} = 0$ and auxiliary bit $\alpha_{2^j} = 0$. The ratios of the CCF occurrence and maximum run-length violation in [2] are compared with ones in our work.

Among the 1.5×10^8 outputs codewords from [2], the numbers of codewords which only occur CCF and only violate the maximum run-length constraint are 5.779×10^7 and 3.158×10^7 , respectively. The number of codewords which occur CCF and violate the run-length constraint simultaneously is 2.599×10^7 , and the number of codewords do not occur CCF and satisfy the maximum run-length constraint is 3.442×10^7 . However, the whole codewords of our proposed SIDEC code do not occur CCF and satisfy the maximum run-length constraint. We present the ratios of the CCF occurrence, violation of run-length constraint, and no CCF occurrence and no maximum run-length constraint violation of output codewords with $n = 150$ in Table 3.10.

Table 3.10: The comparison between referenced work [2] and our work for CCF and maximum run-length constraint when the codeword length is $n = 150$.

	Ref.[2]	Our work
CCF only	38.66%	0%
CCF and maximum run-length constraint violation	17.33%	0%
Maximum run-length constraint violation	21.05%	0%
No CCF and no maximum run-length constraint violation	22.96%	100%

From Table 3.10, about 22.96% codewords of referenced work [2] (Ref.[2]) can be used as DNA strands since the CCF do not occur and the maximum run-length

constraint is satisfied. Therefore, the outputs codewords in Ref.[2] are not suitable for DNA storages. However, the whole codewords from our systematic encoding method for our proposed nonbinary SIDEC code do not occur CCF and satisfy the maximum run-length constraint, which has the potential advantages for DNA storages.

Chapter 4

Weakly mutually uncorrelated codes with maximum run-length constraint for DNA storage

4.1 Introduction

While the advantages of DNA storage are appealing, such as the ability to store $10^{15} - 10^{20}$ bytes of information per gram [73], it is difficult to read specific segments of data in the massive stored DNA sequences since the DNA sequences which contain the data are unordered in the DNA pool. To achieve random-access to the stored data in the DNA pool, Yazdi *et al.* [73] proposed a new random-access method that appends two address sequences, called ‘primers’, to the head and tail of the synthesized DNA sequences. Yazdi *et al.* [76] first designed mutually uncorrelated (MU) codes as primers in DNA storage systems.

The MU code could be traced back to 1964 when it was first proposed by Levenshtein [39]. It is a code in which the prefix of one codeword is different from the suffix of itself or another codeword. In subsequent studies, Massey [46] and Wijn-

gaarden *et al.* [67] later rediscovered the code and applied it to frame-synchronous communication. Blackburn [3] later proved an optimal explicit construction for the code. Although MU code has been used as primers for DNA storage, its supply is gradually becoming insufficient to meet the growing demand for assembling synthesized DNA sequences. Therefore, Yazdi *et al.* [74] proposed a variant of MU code, called the k -weakly mutually uncorrelated (WMU) code, for some positive integers $k \geq 2$. This k -WMU code relaxes the prefix-suffix constraints of the MU code to improve the size of the code, which is beneficial for the increasing demand of DNA sequences assembly.

In certain DNA storage systems, some studies [62, 75] indicated that almost-balanced GC-contents (with a ratio of around 45-55% or 40-60%) and large Hamming distance can also be helpful for random-access in DNA storage. Although codes for GC-balanced and large Hamming distance have been studied in great depth, there has been no work on combining MU or WMU codes with GC-balanced constraint and large Hamming distance for primer design. Therefore, Yazdi *et al.* [74] proposed WMU codes with Hamming distance, GC-balanced and WMU codes, and GC-balanced and WMU codes with Hamming distance. The authors of [11] argued that the codes in [74] do not present the explicit code constructions, and it is unclear whether efficient encoding is possible. Hence, they proposed a new code construction for the almost-balanced code with Hamming distance.

Some studies [18, 65, 68, 8] in DNA storage systems indicated that the maximum run-length constraint has also been an important characteristic for primer design since long run-length in DNA storage systems is easy to lead to errors in primers synthesis and sequencing. Unfortunately, previous studies [74, 11] did not consider the maximum run-length constraint for k -WMU code, which can potentially affect the accuracy and reliability of primers in DNA storage systems. To address this issue. our work aims to design a code to combine the k -WMU code

with maximum run-length constraint. Additionally, the studies [74, 8] indicated that the occurrences of errors can be reduced to a greater extent during primers synthesis and sequencing when multiple beneficial constraints are satisfied simultaneously. Thus, based on the k -WMU code with maximum run-length constraint, we present simple ways to design a class of WMU codes to satisfy the maximum run-length constraint and the proposed constraints in [74] and [11] simultaneously. We also provide explicit constructions for the proposed codes, considering that certain explicit code constructions in [74] were not taken into account.

4.2 Preliminaries

Some special notations are used in this chapter are introduced. Let $\mathcal{L}_{\llbracket n \rrbracket}$ be an $\llbracket n, a, d \rrbracket$ -linear code of length n for any positive integers $a < n$ and $d < n$, which has an a -dimensional vector subspace over \mathbb{Z}_q^n , any two codewords in $\mathcal{L}_{\llbracket n \rrbracket}$ have Hamming distance at least d , and the redundancy of $\mathcal{L}_{\llbracket n \rrbracket}$ is $n - a$.

4.2.1 MU and WMU code: definitions, constructions, and properties

We present the definitions, constructions, and some properties of the MU and WMU codes from [74]. The codes are introduced for the primer design in DNA data storage systems with certain criteria to achieve more efficient random-access. The definitions of MU and WMU codes are provided in Definition 4.2.1.

Definition 4.2.1 ([74]). A code $\mathcal{C} \subseteq \mathbb{Z}_q^n$ is a k -WMU code such that for any two codewords $\mathbf{a}, \mathbf{b} \in \mathcal{C}$, not necessarily distinct, no proper prefix of length m of \mathbf{a} appears as a suffix of length m of \mathbf{b} for all $k \leq m \leq n$, which is expressed as

$$\mathbf{a}_1^m \neq \mathbf{b}_{n-m+1}^n.$$

If $k = 1$, the code \mathcal{C} is an MU code.

Let the lengths of the MU code and k -WMU code be $n_{(1)}$ and $n_{(k)}$, respectively. The construction of q -ary MU codes of length $n_{(1)}$ in Construction 1 (C. 1) was originally proposed by Levenshtein [39]. Later, Gilbert [25], Blackburn [62] and Chee *et al.* [10] proved that the code in C. 1 can achieve the maximum size of the code for the specific m and $n_{(1)}$, where m is given by $k \leq m \leq n_{(1)}$.

Construction 1 (*Construction 1* in [39]). For two integers $n_{(1)}$ and f such that $n_{(1)} \geq 2$ and $f \in [2, n_{(1)} - 1]$, all codewords $\mathbf{a} = (a_1, a_2, \dots, a_{n_{(1)}})$ of a MU code $\mathcal{C}_1 \subseteq \mathbb{Z}_q^{n_{(1)}}$ satisfy the following three conditions:

- The prefix of \mathbf{a} with length f holds that $\mathbf{a}_1^f = \mathbf{0}^f$, where $\mathbf{0}^f$ is an all-zero sequence of length f .
- The symbols a_{f+1} and $a_{n_{(1)}}$ hold that $a_f \in [q] \setminus 0$ and $a_{n_{(1)}} \in [q] \setminus 0$.
- Any subsequence $\mathbf{a}_{f+2}^{n_{(1)}-1}$ does not contain $\mathbf{0}^f$.

In [74], Yazdi *et al.* proposed a class of k -WMU codes for $k \geq 2$ by modifying the MU codes and presented the code construction of k -WMU codes based on the MU codes, which is summarized in Construction 2 (C. 2).

Construction 2 (*Construction 4* in [4]). It is assumed that $\mathcal{C}_1 \subseteq \mathbb{Z}_q^{n_{(1)}}$ is an MU code of length $n_{(1)}$ in **C. 1**. Let $n_{(k)}$ be $n_{(k)} = n_{(1)} + (k - 1)$. Then, a code $\mathcal{C}_2 \subseteq \mathbb{Z}_q^{n_{(k)}}$ is a k -WMU code if each codeword in \mathcal{C}_2 is constructed by a simple concatenation of a codeword $\mathbf{a} \in \mathcal{C}_1$ of length $n_{(1)}$ and a sequence $\mathbf{b} \in \mathbb{Z}_q^{k-1}$ of length $k - 1$. The form of the k -WMU code \mathcal{C}_2 is given as

$$\mathcal{C}_2 = \{(\mathbf{a}, \mathbf{b}) | \mathbf{a} \in \mathcal{C}_1, \mathbf{b} \in \mathbb{Z}_q^{k-1}\}. \quad (4.1)$$

For $k \geq 2$, the codeword (\mathbf{a}, \mathbf{b}) in (4.1) can also be expressed as

$$(\mathbf{a}, \mathbf{b}) = (\mathbf{a}, \mathbf{b}_1^{j-1}, \mathbf{b}_j^{k-1}). \quad (4.2)$$

According to **C. 2**, the subsequence $(\mathbf{a}, \mathbf{b}_1^{j-1})$ in (4.2) is a j -WMU codeword of length $n_{(j)} = n_{(1)} + j - 1$ for $j \leq k$. Therefore, the k -WMU codes are also constructed by a concatenation of the j -WMU codes of length $n_{(j)}$ and the sequences of length $k - j$. \square

Moreover, Yazdi *et al.* also presented some properties for the coupling construction of the k -WMU code in [74]. We summarize them in Lemma 4.2.2.

Lemma 4.2.2 ([74]). Suppose that $\mathbf{c}^o = (c_1^o, c_2^o, \dots, c_{n_{(k)}}^o) \in \mathcal{C}^o \subseteq \mathbb{Z}_2^{n_{(k)}}$ and $\mathbf{c}^e = (c_1^e, c_2^e, \dots, c_{n_{(k)-1}}^e) \in \mathcal{C}^e \subseteq \mathbb{Z}_2^{n_{(k)}}$ are two binary codes of length $n_{(k)}$. For $i \in [1, n_{(k)}]$, an element c_i of a quaternary codeword $\mathbf{c}^4 \in \mathcal{C}^4 \subseteq \mathbb{Z}_4^{n_{(k)}}$ is mapped from the elements of two binary sequences as

$$c_i^4 = \phi(c_i^o, c_i^e) = \begin{cases} 0 & \text{if } (c_i^o, c_i^e) = (0, 0), \\ 1 & \text{if } (c_i^o, c_i^e) = (0, 1), \\ 2 & \text{if } (c_i^o, c_i^e) = (1, 0), \\ 3 & \text{if } (c_i^o, c_i^e) = (1, 1). \end{cases} \quad (4.3)$$

Let \mathcal{C}^D be a DNA code, where an element c_i^D of a codeword $\mathbf{c}^D \in \mathcal{C}^D$ is defined as

$$c_i^D = \begin{cases} A & \text{if } c_i^4 = 0, \\ T & \text{if } c_i^4 = 1, \\ G & \text{if } c_i^4 = 2, \\ C & \text{if } c_i^4 = 3. \end{cases}$$

Then, the codes \mathcal{C}^4 and \mathcal{C}^D satisfy the following three properties:

(P.1) If \mathcal{C}^o is balanced, \mathcal{C}^D is GC-balanced.

(P.2) If either \mathcal{C}^o or \mathcal{C}^e is a k -WMU code, \mathcal{C}^4 is also a k -WMU code.

(P.3) If \mathcal{C}^o and \mathcal{C}^e have minimum Hamming distances d_o and d_e , respectively, the minimum Hamming distance of \mathcal{C}^4 is $d_4 = \max(d_o, d_e)$. ■

4.3 Constrained WMU code constructions

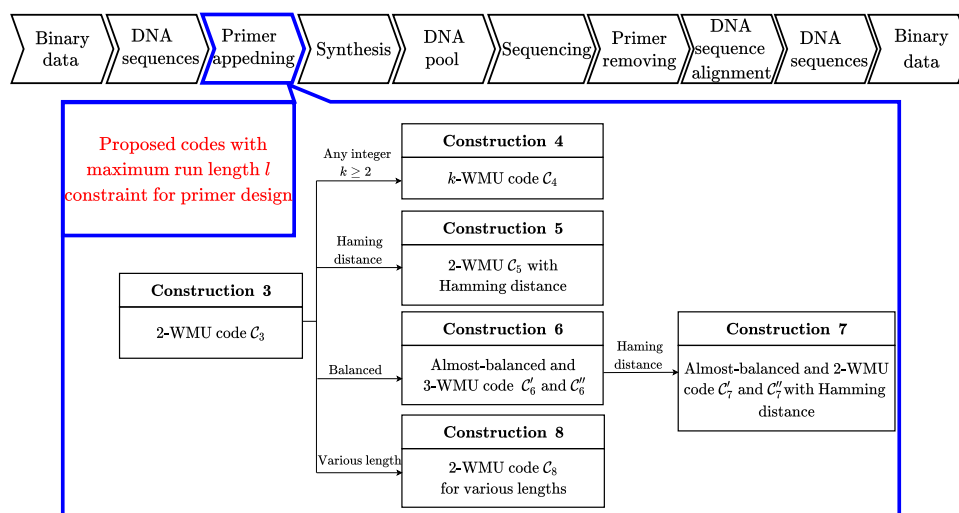


Figure 4.1: Considered DNA storage system and proposed code constructions for primers.

In Fig 4.1, we present proposed the class of WMU code constructions for primer designs of DNA storage. First, we present the construction of 2-WMU codes with the maximum run-length l constraint and explain a method to construct k -WMU codes with the maximum run-length l constraint for any positive integer $k \geq 2$ in subsection 4.3.1. Additionally, we also extend the proposed 2-WMU codes to satisfy combinations of the maximum run-length constraint with the Hamming distance constraint, the GC-balanced constraint, and both the GC-balanced and Hamming distance constraints in subsections 4.3.2, 4.3.3, and 4.3.4, respectively.

4.3.1 k -WMU code with the maximum run-length l constraint

We present our proposed code of length $n_{(2)}$ in Construction 3 (C. 3), which combines the q -ary 2-WMU codes with the maximum run-length l constraint. Then, we also briefly introduce the method to generate the k -WMU codes with the maximum run-length l constraint for any positive integer $k \geq 2$.

Construction 3. Let l , r , and $n_{(2)}$ be positive integers such that $l \geq 3$ and $n_{(2)} = (r + 1)l + 1$. A set \mathcal{S} with sequences $\mathbf{s} = (s_1, s_2, \dots, s_{l-2}, s_{l-1}, s_l)$ of length l is defined as

$$\mathcal{S} = \{\mathbf{s} | s_{l-2} + s_{l-1} \neq 0 \pmod{q} \text{ and } s_{l-2} \neq s_l\}. \quad (4.4)$$

Then, the concatenated sequence $\mathbf{x} = (x_1, x_2, \dots, x_{n_{(2)}})$ of length $n_{(2)}$ in \mathcal{X} is defined as

$$\begin{aligned} \mathbf{x} &= (\mathbf{0}^l, [q] \setminus 0, \mathbf{s}_1, \mathbf{s}_2, \dots, \mathbf{s}_r) \\ &= (\mathbf{x}_1^{l+1}, \mathbf{x}_{l+2}^{(1+1)l+1}, \dots, \mathbf{x}_{rl+2}^{(r+1)l+1}), \end{aligned} \quad (4.5)$$

where the subsequences \mathbf{s}_i for $i = 1, 2, \dots, r$ are elements of the set \mathcal{S} in (4.4) and may not necessarily be distinct. According to (4.4) and (4.5), the run-length of the sequence \mathbf{x} is guaranteed to be at most l . Then, the q -ary codeword $\mathbf{y} = (y_1, y_2, \dots, y_{n_{(2)}})$ in the 2-WMU code with maximum run-length l constraint, \mathcal{C}_3 , is obtained as

$$y_i = \begin{cases} y_1 = x_1 & \text{for } i = 1, \\ y_i = x_{i-1} + x_i \pmod{q} & \text{for } 2 \leq i \leq n_{(2)}. \end{cases} \quad (4.6)$$

Since $\mathbf{y} = (\mathbf{y}_1^{l+1}, \mathbf{y}_{l+2}^{n_{(2)}}) = (\mathbf{y}_1^{l+1}, \mathbf{y}_{l+2}^{(1+1)l+1}, \dots, \mathbf{y}_{rl+2}^{(r+1)l+1})$ is obtained from \mathbf{x} and $\mathbf{x}_1^{l+1} = (\mathbf{0}^l, [q] \setminus 0)$, the first $l + 1$ symbols of \mathbf{y} , \mathbf{y}_1^{l+1} , also start with $(\mathbf{0}^l, [q] \setminus 0)$.

For an easy understanding of C. 3, we present Example 4.3.1.

Example 4.3.1. It assumed that $q = 4$, $l = 3$, and $r = 5$. Then, the length of

the sequence \mathbf{x} in (4.5) is calculated by $n_{(2)} = (r + 1)l + 1 = 19$. From the set \mathcal{S} in (4.4), the sequences \mathbf{s}_i for $i = 1, 2, 3, 4$, and 5 are chosen as $\mathbf{s}_1 = (0, 0, 1)$, $\mathbf{s}_2 = (0, 0, 2)$, $\mathbf{s}_3 = (1, 1, 2)$, $\mathbf{s}_4 = (1, 2, 2)$, and $\mathbf{s}_5 = (3, 1, 1)$ respectively. Then, the concatenated sequence \mathbf{x} in the set of \mathcal{X} from (4.5) is given by

$$\mathbf{x} = (0, 0, 0, 1, 0, 0, 1, 0, 0, 2, 1, 1, 2, 1, 2, 2, 3, 1, 1).$$

The codeword \mathbf{y} from (4.6) can be obtained as

$$\mathbf{y} = (0, 0, 0, 1, 1, 0, 1, 1, 0, 2, 3, 2, 3, 3, 3, 0, 1, 0, 2).$$

The maximum run-length of \mathbf{y} is three, which is same with $l = 3$.

From Example 4.3.1, we can intuitively observe that the maximum run-length of symbols $[q] \setminus 0$ in the q -ary codeword \mathbf{y} obtained by (4.6) is l . Additionally, we can observe that the maximum run-length of the symbol zero in subsequences \mathbf{y}_1^{l+1} and $\mathbf{y}_{l+2}^{n_{(2)}}$ are l and $l - 1$, respectively. By referring to the definition of k -WMU code in Definition 4.2.1 and the code constructions of the k -WMU codes in Constructions 1 and 2, we can easily infer that the codeword \mathbf{y} in \mathcal{C}_3 has the maximum run-length l and becomes a 2-WMU codeword since y_1 can be same with $y_{n_{(2)}}$.

Theorem 4.3.2. *Let $\mathbf{y} \in \mathcal{C}_3 \subseteq \mathbb{Z}_q^{n_{(2)}}$ from (4.6) be the codewords generated from sequences \mathbf{x} in C. 3. Then, the codewords \mathbf{y} in \mathcal{C}_3 are 2-WMU codewords with the maximum run-length l constraint.*

Proof. From (4.4) and (4.5), we have $x_{(i+1)l-1} + x_{(i+1)l} \neq 0 \pmod{q}$ and $x_{(i+1)l-1} \neq x_{(i+1)l+1}$ for $i = 1, 2, \dots, r$. Then, from (4.6), we can derive that $y_{(i+1)l} \neq 0$ and $y_{(i+1)l} \neq y_{(i+1)l+1}$ for $i = 1, 2, \dots, r$. Therefore, the maximum run-lengths of symbols $[q] \setminus 0$ are $l - 1$ in the subsequence $\mathbf{y}_{il+2}^{(i+1)l+1}$ due to $y_{(i+1)l} \neq y_{(i+1)l+1}$ for $i = 1, 2, \dots, r$,

and the maximum run-length of symbol zero is $l - 2$ in the subsequence $\mathbf{y}_{il+2}^{(i+1)l+1}$ because of $y_{(i+1)l} \neq 0$ for $i = 1, 2, \dots, r$. Since the maximum run-lengths of the symbols $[q] \setminus 0$ and zero in the subsequences $\mathbf{y}_{il+2}^{(i+1)l+1}$ for $i = 1, 2, \dots, r$ are $l - 1$ and $l - 2$, the maximum run-lengths of the symbols $[q] \setminus 0$ and zero in the concatenated subsequences $(y_{il+1}, \mathbf{y}_{il+2}^{(i+1)l+1})$ for $i = 1, 2, \dots, r$ correspond to l and $l - 1$, respectively. Since $y_l = 0$, $y_{l+1} = [q] \setminus 0$, and $y_{(i+1)l} \neq y_{(i+1)l+1}$ for $i = 1, 2, \dots, r$, we always have $y_{il} \neq y_{il+1}$ for $i = 1, 2, \dots, r + 1$. According to the conditions that $y_{il} \neq y_{il+1}$ for $i = 1, 2, \dots, r + 1$ and the run-lengths of the subsequences $(y_{il+1}, \mathbf{y}_{il+2}^{(i+1)l+1})$ for $i = 1, 2, \dots, r$ are at most l , it can be concluded that the codeword \mathbf{y} satisfies the maximum run-length l constraint.

Since the codeword \mathbf{y} also starts with $(\mathbf{0}^l, [q] \setminus 0)$ and the maximum run-length of symbol zero in the subsequence $\mathbf{y}_{l+2}^{n(2)}$ is $l - 1$, $\mathbf{0}^l$ does not exist in the subsequence $\mathbf{y}_{l+2}^{n(2)}$. Since y_1 can be the same as $y_{n(2)}$, it can be concluded that the codewords \mathbf{y} are 2-WMU according to Definition 4.2.1.

Finally, since the codewords \mathbf{y} satisfy both the maximum run-length l constraint and the 2-WMU constraint, the codewords $\mathbf{y} \in \mathcal{C}_3 \subseteq \mathbb{Z}_q^{n(2)}$ in C. 3 are 2-WMU codewords with maximum run-length l constraints.

From C. 2, it is known that the k -WMU codes can be constructed by a concatenation of the j -WMU codes of length $n_{(j)}$ and the sequences of length $k - j - 1$ for $j \leq k - 1$. Therefore, from the design of 2-WMU codes with maximum run-length l in C. 3, we briefly introduce the extension to k -WMU codes with the maximum run-length l constraint in Construction 4 (C. 4) for any positive values of $k \geq 2$.

Construction 4. It is assumed that a sequence of length $k - 2$, $\mathbf{e} \in \mathbb{Z}_q^{k-2}$, satisfies the maximum l run-length constraint. Let \mathbf{y} be a codeword in \mathcal{C}_3 as $\mathbf{y} = (y_1, y_2, \dots, y_{n(2)})$. Moreover, a concatenated sequence $(y_{n(2)}, \mathbf{e})$ is also assumed to satisfy the maximum l run-length constraint. Then, the general form of k -WMU codes \mathcal{C}_4 of length $n_{(k)} = n_{(2)} + k - 2$ with the maximum run-length l constraint

is obtained by concatenating the sequences \mathbf{y} and \mathbf{e} , which is given as

$$\mathcal{C}_4 = \{\mathbf{c} = (\mathbf{y}, \mathbf{e}) \mid \mathbf{y} \in \mathcal{C}_3 \subseteq \mathbb{Z}_q^{n_{(2)}}, \mathbf{e} \in \mathbb{Z}_q^{k-2}\}. \quad (4.7)$$

Theorem 4.3.3. *Let $\mathbf{c} \in \mathcal{C}_4 \subseteq \mathbb{Z}_q^{n_{(k)}}$ in (4.7) be the codeword generated by C. 4. Then, the codewords \mathbf{c} in \mathcal{C}_4 are k -WMU codewords with the maximum run-length l constraint.*

Proof. Since \mathbf{y} in (4.6) is a sequence with the maximum run-length l constraint, the sequences \mathbf{e} and $(y_{n_{(2)}}, \mathbf{e})$ are sequences with the maximum run-length l constraint, and $y_{n_{(2)}-1} \neq y_{n_{(2)}}$ from Theorem 4.3.2, the codewords $\mathbf{c} = (\mathbf{y}, \mathbf{e}) \in \mathcal{C}_4 \subseteq \mathbb{Z}_q^{n_{(k)}}$ satisfy the maximum run-length l constraint.

According to C. 2, the k -WMU codes can be constructed by a concatenation of the 2-WMU codes of length $n_{(2)}$ and the sequences in \mathbb{Z}_q^{k-2} . Since \mathbf{y} in (4.6) is 2-WMU sequence and the sequence \mathbf{e} of length $k - 2$ is one of the sequences in \mathbb{Z}_q^{k-2} , the concatenated codewords $\mathbf{c} = (\mathbf{y}, \mathbf{e}) \in \mathcal{C}_4 \subseteq \mathbb{Z}_q^{n_{(k)}}$ are k -WMU codewords.

Finally, it can be concluded that the codewords $\mathbf{c} \in \mathcal{C}_4$ are k -WMU codewords with the maximum run-length l constraint.

In C. 3 and C. 4, we present code constructions methods for 2-WMU code \mathcal{C}_3 and k -WMU code \mathcal{C}_4 with the maximum run-length l constraint, respectively. From \mathcal{C}_3 and \mathcal{C}_4 , the construction for general k -WMU codes with the maximum run-length l constraint is similar to the one for the 2-WMU code. Moreover, the explicit code design for 2-WMU codes is easier than the one for general k -WMU codes. Therefore, from the following subsection, we still focus on the design of 2-WMU codes with the maximum run-length l constraint and we will explain other cases of k -WMU codes if necessary.

4.3.2 2-WMU code with the maximum run-length l constraint and Hamming distance d

Due to the error-prone nature of DNA synthesis and sequencing processing [73, 75], it is beneficial to ensure the Hamming distance between primers. From this aspect, we provide a method to extend our proposed binary 2-WMU codes with the maximum run-length l constraint in C. 3 to quaternary 2-WMU with the maximum run-length l constraint and Hamming distance d in Construction 5 (C. 5), which was inspired by Construction B in [11].

Construction 5. Let p , d , and $n_{(2)}$ be positive integers such that $p < n_{(2)}$ and $d < n_{(2)}$. It is assumed that the binary sequence $\mathbf{y} \in \mathcal{C}_3 \subseteq \mathbb{Z}_2^{n_{(2)}}$ from (4.6) in C. 3 as the input sequence. Let $\mathcal{L}_{\llbracket n_{(2)}+p \rrbracket}$ be a binary $\llbracket n_{(2)}+p, n_{(2)}, d \rrbracket$ -linear code, which is an $n_{(2)}$ -dimensional vector subspace of $\mathbb{Z}_2^{n_{(2)}+p}$, equipped with a systematic encoder $\mathcal{E}_{\llbracket p \rrbracket}: \mathbb{Z}_2^{n_{(2)}} \rightarrow \mathbb{Z}_2^p$. Then, a concatenated sequence (\mathbf{y}, \mathbf{p}) of length $n_{(2)} + p$ can be obtained as

$$(\mathbf{y}, \mathbf{p}) = (\mathbf{y}, \mathcal{E}_{\llbracket p \rrbracket}(\mathbf{y})), \quad (4.8)$$

where the subsequence $\mathbf{p} = \mathcal{E}_{\llbracket p \rrbracket}(\mathbf{y})$ of length p is the parity sequence of \mathbf{y} and (\mathbf{y}, \mathbf{p}) is the corresponding codeword in binary $\llbracket n_{(2)}+p, n_{(2)}, d \rrbracket$ -linear code $\mathcal{L}_{\llbracket n_{(2)}+p \rrbracket}$.

Similarly, a binary $\llbracket n_{(2)}, p, d \rrbracket$ -linear code $\mathcal{L}_{\llbracket n_{(2)} \rrbracket}$ is assumed to be equipped with a systematic encoder $\mathcal{E}_{\llbracket n_{(2)}-p \rrbracket}: \mathbb{Z}_2^p \rightarrow \mathbb{Z}_2^{n_{(2)}-p}$. After $\mathbf{p} \in \mathbb{Z}_2^p$ is obtained from (4.8), the sequence \mathbf{p} is used as the input of the encoder $\mathcal{E}_{\llbracket n_{(2)}-p \rrbracket}$, and a binary sequence $\mathbf{u} \in \mathcal{L}_{\llbracket n_{(2)} \rrbracket} \subseteq \mathbb{Z}_2^{n_{(2)}}$ with Hamming distance d can be obtained as

$$\mathbf{u} = (\mathbf{p}, \mathcal{E}_{\llbracket n_{(2)}-p \rrbracket}(\mathbf{p})). \quad (4.9)$$

Finally, the 2-WMU quaternary codeword $\mathbf{c} = (c_0, c_1, \dots, c_{n_{(2)}-1}) \in \mathcal{C}_5$ of length $n_{(2)}$ with Hamming distance d and maximum run-length l can be obtained by

applying the mapping function $\phi(\cdot)$ in (4.3) to the binary sequences $\mathbf{y} \in \mathcal{C}_3$ in (4.8) and $\mathbf{u} \in \mathcal{L}_{\llbracket n(2) \rrbracket}$ in (4.9), which is given as

$$\mathcal{C}_5 = \{\mathbf{c} | \mathbf{c} = \phi(\mathbf{y}, \mathbf{u}) \in \mathbb{Z}_4^{n(2)}\}. \quad (4.10)$$

Theorem 4.3.4. *Let $\mathbf{c} \in \mathcal{C}_5 \subseteq \mathbb{Z}_4^{n(2)}$ in (4.10) be the codeword generated by C. 5. Then, the quaternary codewords \mathbf{c} in \mathcal{C}_5 are 2-WMU codewords with the maximum run-length l constraint and Hamming distance d .*

Proof. According to the obtained codewords $\mathbf{c} = \phi(\mathbf{y}, \mathbf{u})$ in (4.10), the maximum run-lengths of the symbols two and three in the codewords \mathbf{c} are smaller than or equal to the maximum run-lengths of bit one in \mathbf{y} , while the maximum run-lengths of the symbols zero and one in the codewords \mathbf{c} are also smaller than or equal to the maximum run-lengths of bit zero in \mathbf{y} . Therefore, if the sequence \mathbf{y} satisfies the maximum run-length l constraint, the codeword \mathbf{c} must satisfy the maximum run-length l constraint. According to (P.2) in Lemma 2.2.1, the codeword \mathbf{c} is also in the 2-WMU code since \mathbf{y} is a binary 2-WMU code. Thus, the codewords \mathbf{c} in the set \mathcal{C}_5 are 2-WMU codewords with the maximum run-length l constraint.

According to (P.3) in Lemma 2.2.1, since the codewords $\mathbf{u} \in \mathcal{L}_{\llbracket n(2) \rrbracket}$ have Hamming distance at least d , any two codewords $\mathbf{c} = \phi(\mathbf{y}, \mathbf{u})$ in \mathcal{C}_5 also have Hamming distance at least d .

4.3.3 Almost-balanced and 3-WMU code with the maximum run-length l constraint

For the DNA storage systems, the sequences that satisfy the GC-balanced and the maximum run-length constraints simultaneously are more suitable during the synthesis and sequencing processing. The Knuth's balancing technique [36] was known as an excellent algorithm to construct a binary balanced sequence of length n with

a small redundancy of $\lceil \log_2 n \rceil$ bits. Then, the authors of [33] extended the Knuth's algorithm to construct the almost GC-balanced sequences for DNA storage. Based on these balanced methods in [36, 33], we propose an almost GC-balanced and 3-WMU code with the maximum run-length l constraint in Construction 6 (C. 6).

To design the code in C. 6, we start with the sequences $\mathbf{y} \in \mathcal{C}_3 \subseteq \mathbb{Z}_q^{n_{(2)}}$ from (4.6) in C. 3 as the inputs for $l \geq 3$. To ensure that the codewords satisfy the WMU constraint, we exclude some input sequences \mathbf{y} in \mathcal{C}_3 , for which the subsequence $\mathbf{y}_{l+2}^{n_{(2)}}$ contains all-one string $\mathbf{1}^l$ if $q = 2$, and all-two string $\mathbf{2}^l$ if $q = 4$. Then, We first keep the first $l + 1$ symbols \mathbf{y}_1^{l+1} of the input \mathbf{y} , and then design the subsequence $\mathbf{y}_{l+2}^{n_{(2)}}$ of \mathbf{y} of length $n_{(2)} - l - 1$ so that they are exactly 50%-balanced if $n_{(2)} - l - 1$ is an even value. The codeword in C. 6 will be almost-balanced since the ratio of one is $(0.5 - \frac{l-1}{n_{(2)}+2\lceil \log_q(rl) \rceil})$ or $(0.5 - \frac{l-1}{n_{(2)}+2\lceil \log_q(rl) \rceil+2})$ for $q = 2$ and the ratio of GC contents is $(0.5 - \frac{l\pm 1}{n_{(2)}+2\lceil \log_q(rl) \rceil})$ or $(0.5 - \frac{l\pm 1}{n_{(2)}+2\lceil \log_q(rl) \rceil+2})$ for $q = 4$, respectively. If $n_{(2)}$ is much larger than l , we can easily obtain GC-balanced codes.

For convenience in explaining C. 6, we first introduce Definition 4.3.5 as follows.

Definition 4.3.5 ([33]). Let the flipping rule in Knuth's balancing technique as $f(0) = 1$ and $f(1) = 0$ for $q = 2$, and $f(0) = 2$, $f(1) = 3$, $f(2) = 0$, and $f(3) = 1$ for $q = 4$, respectively. Then, for a sequence $\mathbf{x} = (x_1, x_2, \dots, x_{|\mathbf{x}|})$ of length $|\mathbf{x}|$, $f_t(\mathbf{x})$ denotes the sequence $(f(\mathbf{x}_1^t), \mathbf{x}_{t+1}^{|\mathbf{x}|}) = (f(x_1), \dots, f(x_t), x_{t+1}, \dots, x_{|\mathbf{x}|})$, which is obtained by flipping the first t symbols of the sequence \mathbf{x} for the value of $t \in [1, |\mathbf{x}|]$.

Construction 6. Let $q = \{2, 4\}$, l , r , and $n_{(2)}$ be positive integers such that $l \geq 3$ and $n_{(2)} = (r + 1)l + 1$. Given a q -ary sequence $\mathbf{y} \in \mathcal{C}_3 \subseteq \mathbb{Z}_q^{n_{(2)}}$ from (4.6) in C. 3 as the input, the subsequence $\mathbf{y}_{l+2}^{n_{(2)}}$ does not contain $\mathbf{1}^l$ if $q = 2$ and $\mathbf{2}^l$ if $q = 4$. First, we keep the first $l + 1$ symbols \mathbf{y}_1^{l+1} of the sequence \mathbf{y} to ensure that the generated codeword maintains the WMU constraint. Next, for $t \in [1, n_{(2)} - l - 1]$, by increasing t one by one, the smallest t can be determined so that $f_t(\mathbf{y}_{l+2}^{n_{(2)}})$ is almost 50%-balanced.

If the t -th symbol $f(y_{l+t+1})$ of $f_t(\mathbf{y}_{l+2}^{n_{(2)}})$ is not equal to the $(t+1)$ -th symbol y_{l+t+2} , the almost-balanced sequence with maximum run-length l constraint of length $n_{(2)}$ is given as

$$\mathbf{y}' = (\mathbf{y}_1^{l+1}, f_t(\mathbf{y}_{l+2}^{n_{(2)}})) = (\mathbf{y}_1^{l+1}, f(\mathbf{y}_{l+2}^{l+t+1}), \mathbf{y}_{l+t+2}^{n_{(2)}}). \quad (4.11)$$

Otherwise, the almost-balanced sequence with maximum run-length l constraint of length $n_{(2)} + 1$ is given as

$$\mathbf{y}'' = (\mathbf{y}_1^{l+1}, f(\mathbf{y}_{l+2}^{l+t+1}), \alpha, \mathbf{y}_{l+t+2}^{n_{(2)}}), \quad (4.12)$$

where α is a symbol which is different with both of the last symbol of $f(\mathbf{y}_{l+2}^{l+t+1})$ and the first symbol of $\mathbf{y}_{l+t+2}^{n_{(2)}}$.

In order to find the number of symbols flipped in receiver side, the value t should be transferred to recover the input sequence. Since the maximum value of t can be $n_{(2)} - l - 1 = rl$, a q -ary vector $\boldsymbol{\tau}$ of length $\lceil \log_q(rl) \rceil$ to represent the value of t is given as

$$\boldsymbol{\tau} = (\tau_1, \tau_2, \dots, \tau_{\lceil \log_q(rl) \rceil}). \quad (4.13)$$

To ensure that the symbol α in (4.12) and the sequence $\boldsymbol{\tau}$ in (4.13) are also balanced, the sequences $\boldsymbol{\delta}'$ of length $2\lceil \log_q(rl) \rceil$ and $\boldsymbol{\delta}''$ of length $2\lceil \log_q(rl) \rceil + 1$ are respectively determined as

$$\boldsymbol{\delta}' = (\tau_1, f(\tau_1), \tau_2, f(\tau_2), \dots, \tau_{\lceil \log_q(rl) \rceil}, f(\tau_{\lceil \log_q(rl) \rceil})), \quad (4.14)$$

$$\boldsymbol{\delta}'' = (\tau_1, f(\tau_1), \tau_2, f(\tau_2), \dots, \tau_{\lceil \log_q(rl) \rceil}, f(\tau_{\lceil \log_q(rl) \rceil}), f(\alpha)). \quad (4.15)$$

Finally, for the almost-balanced and WMU codewords with maximum run-length l constraint, \mathbf{c}' in set $\mathcal{C}'_6 \subseteq \mathbb{Z}_q^{n_{(2)}+2\lceil \log_q(rl) \rceil}$ or \mathbf{c}'' in set $\mathcal{C}''_6 \subseteq \mathbb{Z}_q^{n_{(2)}+2\lceil \log_q(rl) \rceil+2}$ can be generated by concatenating the \mathbf{y}' in (4.11) with the sequence $\boldsymbol{\delta}'$ in (4.14)

or the sequence \mathbf{y}'' in (4.12) with the sequence $\boldsymbol{\delta}''$ in (4.15), respectively, which are given as

$$\mathbf{c}' = (\mathbf{y}', \boldsymbol{\delta}'), \quad (4.16)$$

$$\mathbf{c}'' = (\mathbf{y}'', \boldsymbol{\delta}''). \quad (4.17)$$

The last symbol of $\boldsymbol{\delta}'$ in (4.14) and the last two symbols of $\boldsymbol{\delta}''$ in (4.15) can be zero, which implies the prefix of generated codewords can be same as the suffix of the codewords with a length of at most two. Therefore, we can conclude that the codes \mathcal{C}'_6 and \mathcal{C}''_6 are the almost-balanced 3-WMU codes with maximum run-length l constraint.

For convenience in understanding C. 6, we present Example 4.3.6.

Example 4.3.6. It is assumed that $q = 4$, $l = 3$, and $r = 10$. The sequence $\mathbf{y} \in \mathcal{C}_3$ of length $n_{(2)} = 34$ from (4.6), which does not contain $\mathbf{2}^l$ in subsequence $\mathbf{y}_{l+1}^{n_{(2)}}$, is assumed as

$$\mathbf{y} = (0, 0, 0, 2, 1, 1, 0, 0, 2, 1, 0, 1, 1, 1, 3, 0, 1, 2, 2, 1, 1, 0, 1, 2, 1, 3, 2, 3, 3, 1, 0, 1, 1, 0).$$

We keep the first $l + 1$ symbols $\mathbf{y}_1^{l+1} = (0, 0, 0, 2)$. Then, for $t \in [1, 7]$, the flipped sequence $f_t(\mathbf{y}_{l+2}^{n_{(2)}})$ is still not 50%-balanced. When $t = 8$, the flipped sequence $f_8(\mathbf{y}_{l+2}^{n_{(2)}})$ is obtained as

$$f_8(\mathbf{y}_{l+2}^{n_{(2)}}) = (3, 3, 2, 2, 0, 3, 2, 3, 1, 1, 3, 0, 1, 2, 2, 1, 1, 0, 1, 2, 1, 3, 2, 3, 3, 1, 0, 1, 1, 0),$$

which is exact 50%-balanced. Since the symbol $f(y_{l+t+1}) = 3$ is not same with $y_{l+t+2} = 1$, the almost-balanced sequence \mathbf{y}' in (4.11) with the maximum run-

length l constraint is obtained as

$$\mathbf{y}' = (0, 0, 0, 2, 3, 3, 2, 2, 0, 3, 2, 3, 1, 1, 3, 0, 1, 2, 2, 1, 1, 0, 1, 2, 1, 3, 2, 3, 3, 1, 0, 1, 1, 0).$$

The number of $t = 8$ can be represented as $\boldsymbol{\tau} = (0, 2, 0)$ of length $\lceil \log_q(rl) \rceil = \lceil \log_4(10 \times 3) \rceil = 3$ from (4.13), and then the balanced sequence $\boldsymbol{\delta}' = (0, 2, 2, 0, 0, 2)$ in (4.14) can be obtained. The almost-balanced and 3-WMU codewords $\mathbf{c}' = (\mathbf{y}', \boldsymbol{\delta}')$ with maximum run-length l constraint in (4.16) can be obtained as

$$\begin{aligned} \mathbf{c}' = & (0, 0, 0, 2, 3, 3, 2, 2, 0, 3, 2, 3, 1, 1, 3, 0, 1, 2, 2, 1, \\ & 1, 0, 1, 2, 1, 3, 2, 3, 3, 1, 0, 1, 1, 0, 0, 2, 2, 0, 0, 2). \end{aligned}$$

From Example 4.3.6, we can intuitively observe that though the ratio of GC contents in the original sequence \mathbf{y} is 29.41%. Since $\mathbf{c}' = (\mathbf{y}', \boldsymbol{\delta}') = (\mathbf{y}_1^{l+1}, f_8(\mathbf{y}_{l+2}^{n_{(2)}}), \boldsymbol{\delta}')$ and both subsequences $f_8(\mathbf{y}_{l+2}^{n_{(2)}})$ and $\boldsymbol{\delta}'$ are 50%-balanced, the output codeword \mathbf{c}' is a $(50 - \frac{l-1}{n_{(2)}+2\lceil \log_q(rl) \rceil})\%$ = $(50 - \frac{3-1}{34+6})\%$ = 45% GC-balanced WMU codeword with the maximum run-length $l = 3$ constraint, where $l - 1$ denotes the number of difference symbols between symbols zero and one and symbols two and three in \mathbf{y}_1^l and $n_{(2)} + 2\lceil \log_q(rl) \rceil$ is the length of the codeword \mathbf{c}' .

Theorem 4.3.7. *Let $\mathbf{c}' \in \mathcal{C}'_6$ in (4.16) and $\mathbf{c}'' \in \mathcal{C}''_6$ in (4.17) be the codewords generated by C. 6. Then, the codewords in \mathcal{C}'_6 and \mathcal{C}''_6 are almost GC-balanced 3-WMU codewords with the maximum run-length l constraint.*

Proof. We first consider the subsequence of $(\mathbf{y}_1^{l+1}, f(\mathbf{y}_{l+2}^{l+t+1}))$ of \mathbf{y}' in (4.11) and \mathbf{y}'' in (4.12). If $|f(\mathbf{y}_{l+2}^{l+t+1})| = t \leq l - 1$, the maximum run-length of $f(\mathbf{y}_{l+2}^{l+t+1})$ is also less than or equal to $l - 1$. Since the last symbol y_{l+1} in \mathbf{y}_1^{l+1} is $[q] \setminus 0$ and $|f(\mathbf{y}_{l+2}^{l+t+1})| \leq l - 1$, the maximum run-length of $(y_{l+1}, f(\mathbf{y}_{l+2}^{l+t+1}))$ can be l and $l - 1$ when all the symbols of sequence $f(\mathbf{y}_{l+2}^{l+t+1})$ are $[q] \setminus 0$ and zero, respectively.

If $|f(\mathbf{y}_{l+2}^{l+t+1})| = t \geq l$, the maximum run-lengths of flipped symbols in $f(\mathbf{y}_{l+2}^{l+t+1})$ should be same as the maximum run-lengths of unflipped symbols in subsequence \mathbf{y}_{l+2}^{l+t+1} . Since there are no strings $\mathbf{1}^l$ and $\mathbf{2}^l$ in subsequence \mathbf{y}_{l+2}^{l+t+1} for $q = 2$ and $q = 4$, respectively, the string $\mathbf{0}^l$ does not appear in the sequence $f(\mathbf{y}_{l+2}^{l+t+1})$ after flipping, and the maximum run-length of symbol zero in $f(\mathbf{y}_{l+2}^{l+t+1})$ is $l - 1$. The flipped symbols $f(y_{2l})$ and $f(y_{2l+1})$ always satisfy $f(y_{2l}) \neq f(y_{2l+1})$ since the corresponding unflipped symbols y_{2l} and y_{2l+1} have a relation as $y_{2l} \neq y_{2l+1}$. Because of three conditions that the last symbol y_{l+1} of \mathbf{y}_1^{l+1} is $[q] \setminus 0$, the symbols $f(y_{2l})$ and $f(y_{2l+1})$ satisfy $f(y_{2l}) \neq f(y_{2l+1})$, and the maximum run-length of symbol zero in $f(\mathbf{y}_{l+2}^{l+t+1})$ is $l - 1$, the maximum run-lengths of the symbols $[q] \setminus 0$ and zero in $(y_{l+1}, f(\mathbf{y}_{l+2}^{l+t+1}))$ are also l and $l - 1$, respectively.

According to Lemma 5 in [33] and Corollary 2 in [33], the maximum run-lengths of the symbols $[q] \setminus 0$ and zero in the subsequences $(f(\mathbf{y}_{l+2}^{l+t+1}), \mathbf{y}_{l+t+2}^{n(2)})$ of length $n(2) - l - 1$ in (4.11) and $(f(\mathbf{y}_{l+2}^{l+t+1}), \alpha, \mathbf{y}_{l+t+2}^{n(2)})$ of length $n(2) - l$ in (4.12) are also l and $l - 1$, respectively.

Next, we consider the subsequences $(\mathbf{y}_{l+t+2}^{n(2)}, \delta')$ in (4.16) and $(\mathbf{y}_{l+t+2}^{n(2)}, \delta'')$ in (4.17). If $|\mathbf{y}_{l+t+2}^{n(2)}| \geq 2$, we always have $y_{n(2)-1} \neq y_{n(2)}$. Since $y_{n(2)-1} \neq y_{n(2)}$ and $(\tau_i, f(\tau_i))$ in δ' and δ'' satisfies $\tau_i \neq f(\tau_i)$ for $i \in [1, \lceil \log_q r l \rceil]$, the maximum run-lengths of the subsequences $(\mathbf{y}_{n(2)-1}^{n(2)}, \delta')$ and $(\mathbf{y}_{n(2)-1}^{n(2)}, \delta'')$ are two if $y_{n(2)} = \tau_0$. Since we consider the maximum run-length l as $l \geq 3$ and the maximum run-lengths of the symbols $[q] \setminus 0$ and zero in $\mathbf{y}_{l+t+2}^{n(2)}$ are l and $l - 1$, the maximum run-lengths of the symbols $[q] \setminus 0$ and zero in the subsequences $(\mathbf{y}_{l+t+2}^{n(2)}, \delta')$ and $(\mathbf{y}_{l+t+2}^{n(2)}, \delta'')$ are also l and $l - 1$, respectively. Similarly, If $|\mathbf{y}_{l+t+2}^{n(2)}| = 1$, the maximum run-lengths of the symbols $[q] \setminus 0$ and zero in the subsequences of $(\mathbf{y}_{l+t+2}^{n(2)}, \delta')$ and $(\mathbf{y}_{l+t+2}^{n(2)}, \delta'')$ are also l and $l - 1$, respectively.

Based on the above analysis, the codewords \mathbf{c}' and \mathbf{c}'' satisfy the maximum run-length l constraint since all of the subsequences $(\mathbf{y}_1^{l+1}, f(\mathbf{y}_{l+2}^{l+t+1}))$, $(f(\mathbf{y}_{l+2}^{l+t+1}), \mathbf{y}_{l+t+2}^{n(2)})$,

$(\mathbf{y}_{l+t+2}^{n(2)}, \boldsymbol{\delta}')$, $(f(\mathbf{y}_{l+2}^{l+t+1}), \alpha, \mathbf{y}_{l+t+2}^{n(2)})$, and $(\mathbf{y}_{l+t+2}^{n(2)}, \boldsymbol{\delta}'')$ satisfy the maximum run-length l constraint. Additionally, since the codewords \mathbf{c}' and \mathbf{c}'' satisfy three conditions: they start with $(\mathbf{0}^l, [q] \setminus 0)$, the maximum run-length of symbol zero is $l - 1$ in subsequences $\mathbf{c}'_{l+2}{}^{n(2)+2\lceil\log_q(rt)\rceil-1}$ and $\mathbf{c}''_{l+2}{}^{n(2)+2\lceil\log_q(rt)\rceil}$, and the last symbol of \mathbf{c}' and the last two symbols of \mathbf{c}'' can be zero. These conditions ensure that the codewords \mathbf{c}' and \mathbf{c}'' are 3-WMU codewords with maximum run-length l constraint.

Since the subsequences $(f(\mathbf{y}_{l+2}^{l+t+1}), \mathbf{y}_{l+t+2}^{n(2)})$ in (4.11) and (4.12) are almost 50%-balanced, and $(\alpha, f(\alpha))$ and $\boldsymbol{\tau}$ are exactly 50% balanced, the codewords \mathbf{c}' and \mathbf{c}'' are almost-balanced, where the ratio of one is either $(0.5 - \frac{l-1}{n_{(2)}+2\lceil\log_q(rt)\rceil})$ or $(0.5 - \frac{l-1}{n_{(2)}+2\lceil\log_q(rt)\rceil+2})$ for $q = 2$ and the ratio of GC contents is either $(0.5 - \frac{l\pm 1}{n_{(2)}+2\lceil\log_q(rt)\rceil})$ or $(0.5 - \frac{l\pm 1}{n_{(2)}+2\lceil\log_q(rt)\rceil+2})$ for $q = 4$. Therefore, the codewords in $\mathbf{c}' \in \mathcal{C}'_6$ and $\mathbf{c}'' \in \mathcal{C}''_6$ are almost-balanced and 3-WMU codewords with the maximum run-length l constraint.

4.3.4 Almost-balanced 2-WMU code with the maximum run-length l constraint and Hamming distance d

From Theorem 4.3.7, it is known that both \mathbf{y}' in (4.11) and \mathbf{y}'' in (4.12) start with $(\mathbf{1}^l, [q] \setminus 0)$. Moreover, since except for the first $l + 1$ symbols, the string $\mathbf{0}^l$ is not included in \mathbf{y}' and \mathbf{y}'' and last symbol is zero, it can be concluded that the binary sequences of \mathbf{y}' and \mathbf{y}'' are almost-balanced 2-WMU sequences with the maximum run-length l constraint. In this subsection, we present a code design that incorporates more constraints for DNA storage, which can improve the efficiency of random-access. To this end, we extend binary almost-balanced 2-WMU sequences \mathbf{y}' and \mathbf{y}'' that satisfy the maximum run-length l constraint in C. 6 to quaternary almost-balanced 2-WMU codewords with the maximum run-length l constraint and Hamming distance d in Construction 7 (C. 7), where the code construction is similar to that of Section III.B.

Construction 7. Let p , d , and $n_{(2)}$ be positive integers such that $p < n_{(2)}$ and $d < n_{(2)}$. This code construction has three input components which are binary sequences \mathbf{y}' in (4.11) and \mathbf{y}'' in (4.12) from C. 6, a binary vector $\boldsymbol{\tau}$ of length $\lceil \log_q(rl) \rceil$ in (4.13) which corresponds to the number of flipped symbols in \mathbf{y}' or \mathbf{y}'' , and two binary systematic encoders $\mathcal{E}_{\llbracket p' \rrbracket}$ for $\llbracket n_{(2)} + p', n_{(2)}, d \rrbracket$ -linear code $\mathcal{L}_{\llbracket n_{(2)} + p' \rrbracket}$ and $\mathcal{E}_{\llbracket p'' \rrbracket}$ for $\llbracket n_{(2)} + 1 + p'', n_{(2)} + 1, d \rrbracket$ -linear code $\mathcal{L}_{\llbracket n_{(2)} + 1 + p'' \rrbracket}$. Then, concatenated sequences $(\mathbf{y}', \mathbf{p}')$ of length $n_{(2)} + p'$ and $(\mathbf{y}'', \mathbf{p}'')$ of length $n_{(2)} + 1 + p''$ can be obtained as

$$(\mathbf{y}', \mathbf{p}') = (\mathbf{y}', \mathcal{E}_{\llbracket p' \rrbracket}(\mathbf{y}')), \quad (4.18)$$

$$(\mathbf{y}'', \mathbf{p}'') = (\mathbf{y}'', \mathcal{E}_{\llbracket p'' \rrbracket}(\mathbf{y}')), \quad (4.19)$$

where the subsequences \mathbf{p}' of length p' and \mathbf{p}'' of length p'' are the parity parts of \mathbf{y}' and \mathbf{y}'' , respectively.

Similarly, a binary $\llbracket n_{(2)}, \lceil \log_q(rl) \rceil + p', d \rrbracket$ -linear code $\mathcal{L}_{\llbracket n_{(2)} \rrbracket}$ with a systematic encoder $\mathcal{E}_{\llbracket n_{(2)} - \lceil \log_q(rl) \rceil - p' \rrbracket}$ and a binary $\llbracket n_{(2)} + 1, \lceil \log_q(rl) \rceil + p'', d \rrbracket$ -linear code $\mathcal{L}_{\llbracket n_{(2)} + 1 \rrbracket}$ with a systematic encoder $\mathcal{E}_{\llbracket n_{(2)} + 1 - \lceil \log_q(rl) \rceil - p'' \rrbracket}$ are considered to ensure a minimum Hamming distance. Then, the concatenated sequences $(\boldsymbol{\tau}, \mathbf{p}')$ and $(\boldsymbol{\tau}, \mathbf{p}'')$ are as the inputs of the encoders $\mathcal{E}_{\llbracket n_{(2)} - \lceil \log_q(rl) \rceil - p' \rrbracket}$ and $\mathcal{E}_{\llbracket n_{(2)} + 1 - \lceil \log_q(rl) \rceil - p'' \rrbracket}$, respectively, and the sequences $\mathbf{v}' \in \mathcal{L}_{\llbracket n_{(2)} \rrbracket}$ and $\mathbf{v}'' \in \mathcal{L}_{\llbracket n_{(2)} + 1 \rrbracket}$ can be obtained as

$$\mathbf{v}' = (\boldsymbol{\tau}, \mathbf{p}', \mathcal{E}_{\llbracket n_{(2)} - \lceil \log_q(rl) \rceil - p' \rrbracket}(\boldsymbol{\tau}, \mathbf{p}')), \quad (4.20)$$

$$\mathbf{v}'' = (\boldsymbol{\tau}, \mathbf{p}'', \mathcal{E}_{\llbracket n_{(2)} + 1 - \lceil \log_q(rl) \rceil - p'' \rrbracket}(\boldsymbol{\tau}, \mathbf{p}'')). \quad (4.21)$$

Finally, by applying the mapping function $\phi(\cdot)$ in (4.3) for the binary sequences \mathbf{y}' and \mathbf{v}' in (4.20) or \mathbf{y}'' and \mathbf{v}'' in (4.21), the quaternary almost-balanced 2-WMU codes $\mathbf{c}' \in \mathcal{C}'_7$ of length $n_{(2)}$ and $\mathbf{c}'' \in \mathcal{C}''_7$ of length $n_{(2)} + 1$ with maximum run-length

l constraint and Hamming distance d , which are given as

$$\mathbf{c}' = (\mathbf{y}', \mathbf{v}'), \quad (4.22)$$

$$\mathbf{c}'' = (\mathbf{y}'', \mathbf{v}''). \quad (4.23)$$

Theorem 4.3.8. *The codes \mathcal{C}'_7 and \mathcal{C}''_7 in C. 7 are quaternary almost-balanced 2-WMU codes with the maximum run-length l constraint and Hamming distance d .*

Proof. According to (P.1) and (P.2) in Lemma 2.2.1 and Theorem 4.3.4, it can be easily concluded that the codes $\mathcal{C}'_7 \subseteq \mathbb{Z}_4^{n_{(2)}}$ and $\mathcal{C}''_7 \subseteq \mathbb{Z}_4^{n_{(2)}+1}$ in C. 7 are quaternary almost-balanced 2-WMU codes with the maximum run-length l constraint and Hamming distance d .

4.4 Applying code construction to primer design in DNA storage

According to some research works [32, 57, 33, 52], the available lengths of primers for DNA storage are 18-24 nt and the proper maximum run-length of l should be set as $l = 3$. However, the length $n_{(2)} = (r + 1)l + 1$ of the proposed code \mathcal{C}_3 in C. 3 cannot support various lengths of primers, 18-24 nt, for some positive integers r and l . In this section, we first modify the code \mathcal{C}_3 in C. 3 to \mathcal{C}_8 in Construction 8 (C. 8), which can achieve 2-WMU constraint, maximum run-length l constraint, and various code lengths. We also provide the size for the code \mathcal{C}_8 , which is essential for deriving the sizes of our proposed codes. Then, we present the sizes for the 2-WMU code \mathcal{C}_8 with maximum run-length $l = 3$ constraint for lengths 18-24 nt, which can support primers for DNA storage.

4.4.1 2-WMU code construction with maximum run-length l constraint for various lengths

To support various lengths of the proposed codes, we modify the code design in C. 3 to C. 8, which can satisfy the 2-WMU constraint, maximum run-length l constraint, and various code lengths.

Construction 8. For positive integers l and r , $l \geq 3$, we consider primer lengths ranging in 18-24 nt with the maximum run-length l in DNA storage. We impose the condition $r \geq l$ since the value of $(r+1)l+1$ should be bounded as $(r+1)l+1 \geq 18$. Let μ be the target length of primer. Then, for a nonnegative integer value of $\lambda \in [l-1]$, the target length μ can always be expressed as

$$\mu = (r+1)l + 1 - \lambda. \quad (4.24)$$

If $\lambda = 0$, we can directly apply the code design of **C. 3** for the primer since μ in (4.24) is equal to the code length $n_{(2)}$. If $\lambda \in [1, l-1]$, we define a sequence $\mathbf{g} = (g_1, \dots, g_{l-3}, g_{l-2}, g_{l-1})$ of length $l-1$ in a set \mathcal{G} , which has the similar constraint with the subsequences of $\mathbf{s} \in \mathcal{S}$ in (4.4) and is defined as

$$\mathcal{G} = \{\mathbf{g} | g_{l-3} \neq g_{l-1}\}. \quad (4.25)$$

When $l = 3$, the set \mathcal{G} in (4.25) becomes $\mathcal{G} = \{\mathbf{g} | g_1 \neq g_2\}$.

Then, a sequence \mathbf{h} in a set \mathcal{H} can be defined as

$$\begin{aligned} \mathcal{H} &= \{\mathbf{h} | (h_1, h_2, \dots, h_\mu) \\ &= (\mathbf{0}^l, [q] \setminus \{0, \mathbf{g}_1, \mathbf{g}_2, \dots, \mathbf{g}_\lambda, \mathbf{s}_{\lambda+1}, \dots, \mathbf{s}_r\})\}, \end{aligned} \quad (4.26)$$

where \mathbf{g}_i for $i = 1, 1, \dots, \lambda$ are sequences of the set \mathcal{G} in (4.25), \mathbf{s}_i for $i = \lambda+1, \lambda+$

$2, \dots, r$ are sequences of the set \mathcal{S} in (4.4), and both of the sequences, \mathbf{g}_i and \mathbf{s}_i , are not necessarily distinct.

Finally, the q -ary codeword $\mathbf{w} = (w_1, w_2, \dots, w_\mu)$ in the 2-WMU code with maximum run-length l , \mathcal{C}_8 , is defined as

$$w_i = \begin{cases} w_1 = h_1 & \text{for } i = 1, \\ w_i = h_{i-1} + h_i \pmod{q} & \text{for } 2 \leq i \leq \mu. \end{cases} \quad (4.27)$$

By defining $\mathbf{g} \in \mathcal{G}$ in (4.25), we can ensure that the codewords \mathbf{w} can be 2-WMU codewords with maximum run-length l constraint. Similar to the analysis in the proof of Theorem 4.3.2, we can easily prove that the codewords \mathbf{w} in \mathcal{C}_8 are 2-WMU codewords with maximum run-length l constraint, and we omit the detailed proof in here.

4.4.2 The size of the 2-WMU code with maximum run-length l constraint for various lengths

In order to satisfy more constraints, the size of the code should be smaller. Although the proposed codes satisfy the maximum run-length l constraint, they are not suitable for primer design if the size of the code becomes much smaller. In this subsection, we investigate the size of the 2-WMU code \mathcal{C}_8 with maximum run-length l constraint in for various lengths in C. 8. Then, we provide the values of the sizes of the codes \mathcal{C}_8 for $l = 3$ and lengths 18-24 nt, which indicate that our proposed code can provide a sufficiently large number of primers for DNA storage. The sizes of the proposed 2-WMU codes of length $n_{(2)}$ with maximum run-length l constraint in Constructions 3 and 8 are given in Theorem 4.4.1.

Theorem 4.4.1. *Let $|\mathcal{C}_8|_{(2)}$ be the size of the code \mathcal{C}_8 of length $n_{(2)} = \lceil \frac{\mu-1}{l} \rceil l + 1 - \lambda$ from C. 8 for $l \geq 3$, where the value of μ is the target primer length given in (4.24).*

Then, $|\mathcal{C}_8|_{(2)}$ is determined as

$$|\mathcal{C}_8|_{(2)} = (q-1)(q^{l-2}(q-1))^\lambda (q^{l-2}(q-1)^2)^{r-\lambda}. \quad (4.28)$$

Moreover, the size of the code \mathcal{C}_3 , $|\mathcal{C}_3|_{(2)}$, is similarly defined as

$$|\mathcal{C}_3|_{(2)} = (q-1)(q^{l-2}(q-1)^2)^r. \quad (4.29)$$

Proof. Since the codeword $\mathbf{w} \in \mathcal{C}_8$ in (4.27) is obtained by the concatenated sequence \mathbf{h} in (4.26), we can determine the size of the code $|\mathcal{C}_8|_{(2)}$ from the number of the concatenated sequence \mathbf{h} . The number of subsequences $\mathbf{s}_i \in \mathcal{S}$ for $i = \lambda + 1, \lambda + 2, \dots, r$ in (4.26) can be calculated as

$$|\mathcal{S}| = q^{l-3} \cdot q \cdot (q-1) = q^{l-2}(q-1)^2. \quad (4.30)$$

Since the first $l-3$ symbols \mathbf{s}_1^{l-2} of sequences $\mathbf{s} = (s_1, s_2, \dots, s_l)$ in \mathcal{S} have no constraints, the cases of these $l-3$ symbols denote q^{l-3} choices. Since $s_{l-2} + s_{l-1} \neq 0 \pmod{q}$ and the $(l-1)$ -th symbol s_{l-2} of the sequence \mathbf{s} in \mathcal{S} is assumed to have q choices, then the (l) -th symbol s_{l-1} only has $q-1$ choices. Similarly, when the symbol s_{l-2} is determined, the l -th symbol s_l of the sequence \mathbf{s} in \mathcal{S} only has $q-1$ choices due to $s_{l-2} \neq s_l$.

Similar to the analysis for the number of subsequences $\mathbf{s}_i \in \mathcal{S}$ in (4.26), the number of subsequences $\mathbf{g}_i \in \mathcal{G}$ for $i = 2, 2, \dots, \lambda$ of length $l-1$ in (4.26) can be calculated as

$$|\mathcal{G}| = q^{l-2}(q-1). \quad (4.31)$$

From (4.30) and (4.31), the number of the concatenated sequences $\mathbf{h} \in \mathcal{H}$ in (4.26)

can be calculated as

$$|\mathcal{H}| = (q-1)(q^{l-2}(q-1))^\lambda (q^{l-2}(q-1)^2)^{r-\lambda}. \quad (4.32)$$

Finally, from (4.32), the size of the code $|\mathcal{C}_8|_{(2)}$ is given as

$$|\mathcal{C}_8|_{(2)} = (q-1)(q^{l-2}(q-1))^\lambda (q^{l-2}(q-1)^2)^{r-\lambda}.$$

According to (4.4), (4.5), (4.26), and (4.27), we can obtain that the size of the code in C. 3 is the special case of (4.28) with $\lambda = 0$ in (4.24), and the size of the code \mathcal{C}_3 in C. 3 is given as

$$|\mathcal{C}_3|_{(2)} = (q-1)(q^{l-2}(q-1)^2)^r.$$

As other works [32, 57, 33, 52] set a code length of 18-24 nt and a maximum run-length constraint of $l = 3$ in their DNA storage systems, we consider the parameters and list the results of the size of the 2-WMU code with maximum run-length $l = 3$ in Table 4.1. From Table 4.1, the code of lengths $n_{(2)} = 19$ and 22 can be constructed from C. 3 for the maximum run-length $l = 3$ and the codes of the other lengths can be constructed from C. 8. For 20 nt which is popularly used for primers length, the size of the code is 7.2559×10^8 . Since no previous work in WMU code has presented its the size of the code, it is not possible to compare the sizes of the referenced codes with the proposed codes. If the proposed codes satisfy all the constraints which the referenced codes already satisfy and additionally the maximum run-length l constraint, the sizes of the proposed codes will decrease. However, as we can see in detail proof of Theorem 4.4.1, the expected size loss of the proposed codes is relatively small.

Table 4.1: The size of the 2-WMU code with maximum run-length $l = 3$ for 18-24 nt.

$n_{(2)}$ (nt)	$ \mathcal{C}_8 _{(2)}$	$n_{(2)}$ (nt)	$ \mathcal{C}_8 _{(2)}$
18	6.0467×10^7	22	6.5303×10^9
19	1.8140×10^8	23	2.6121×10^{10}
20	7.2559×10^8	24	7.8364×10^{10}
21	2.1768×10^9		

4.4.3 Comparison for the maximum run-length constraint

To examine the characteristics of the codes from the perspective of the maximum run-length constraint for DNA storage, we present the number of codewords with different maximum run-lengths results for the k -WMU code in [74] as well as the almost GC-balanced WMU code in [74], considering $q = 4$. The results reveal that a significant ratio of the codewords in the referenced codes fail to meet the requirements for DNA storage, specifically the maximum run-length $l = 3$.

To easily distinguish different referenced codes, we denote the k -WMU code in [74], almost GC-balanced k -WMU code in [74], k -WMU code with Hamming distance in [74], and almost GC-balanced k -WMU codes with Hamming distance in [74] and in [11] as \mathcal{C}_{R8} , \mathcal{C}_{R8-GC} , \mathcal{C}_{R8-H} , \mathcal{C}_{R8-GH} , and \mathcal{C}_{R11-GH} , respectively. Similarly, the corresponding the sizes of the codes are defined as $|\mathcal{C}_{R8}|_{(k)}$, $|\mathcal{C}_{R8-GC}|_{(k)}$, $|\mathcal{C}_{R8-H}|_{(k)}$, $|\mathcal{C}_{R8-GH}|_{(k)}$, and $|\mathcal{C}_{R11-GH}|_{(k)}$.

Since the explicit construction of the code \mathcal{C}_{R8} was not given, we have devised a method to determine the number of codewords with different maximum run-lengths results for this code. We follow the MU code \mathcal{C}_1 in C. 1 and enumerate all the codewords of length $n_{(k)}$ for \mathcal{C}_{R8} by concatenating the MU code with any sequence of length $k - 1$. This approach allows us to search the number of codewords with different maximum run-lengths for \mathcal{C}_{R8} . To ensure a fair comparison with our proposed codes, we consider two cases: the 2-WMU code of length $n_{(2)}$ and the 3-WMU code of length $n_{(3)} = n_{(2)} + 1$. Since the computational complexity

of enumerating all codewords for \mathcal{C}_{R8} increases as the length $n_{(k)}$ increases, we determine small lengths of $n_{(2)} = 12$ and $n_{(3)} = 13$ for this purpose. Similarly, as an explicit construction for the code $\mathcal{C}_{\text{R8-GC}}$ was not provided, for a fair comparison with our proposed almost GC-balanced and 3-WMU codes \mathcal{C}'_6 and \mathcal{C}''_6 , we enumerate the entire set of codewords for $\mathcal{C}_{\text{R8-GC}}$ among from the codewords for \mathcal{C}_{R8} with a length of $n_{(3)}$, where the ratio of GC contents in codewords is $(0.5 \pm \frac{l \pm 1}{n_{(3)}})$. Then, we present the number of codewords with different maximum run-lengths results for the codes \mathcal{C}_{R8} of lengths $n_{(2)}$ and $n_{(3)}$ as well as the code $\mathcal{C}_{\text{R8-GC}}$ of length $n_{(3)}$ in Table 4.2. The explicit constructions of the codes $\mathcal{C}_{\text{R8-H}}$ and $\mathcal{C}_{\text{R11-GH}}$ allow us to determine the number of codewords with different maximum run-lengths. However, the sizes of the codes $\mathcal{C}_{\text{R8-H}}$ and $\mathcal{C}_{\text{R11-GH}}$ primarily depend on the value of k . To fairly compare with our proposed codes, we would need to set $k = 2$, resulting in a small size of at most q .

Table 4.2: Different maximum run-lengths results for the codes \mathcal{C}_{R8} of lengths $n_{(2)} = 12$ and $n_{(3)} = 13$, $\mathcal{C}_{\text{R8-H}}$ of length $n_{(2)} = 12$, $|\mathcal{C}_{\text{R8-GC}}|_{(3)}$ of length $n_{(3)} = 13$, and $|\mathcal{C}_{\text{R11-GH}}|_{(2)}$ of length $n_{(2)} = 12$

	$ \mathcal{C}_{\text{R8}} _{(3)}$	%	$ \mathcal{C}_{\text{R8}} _{(2)}$	%	$ \mathcal{C}_{\text{R8-H}} _{(2)}$	%	$ \mathcal{C}_{\text{R8-GC}} _{(3)}$	%	$ \mathcal{C}_{\text{R11-GH}} _{(2)}$	%
$l' \leq 3$	2600280	81.86	658476	82.92	–	–	745449	78.80	–	–
$l' = 4$	421413	13.27	99711	12.56	–	–	148272	15.68	1	25
$l' = 5$	115797	3.65	27021	3.40	–	–	41418	4.38	2	50
$l' = 6$	30195	0.95	6963	0.88	–	–	9690	1.02	1	25
$l' = 7$	5562	0.18	1215	0.15	–	–	90	0.01		
$l' = 8$	2619	0.08	603	0.08	3	75	981	0.10		
$l' = 9$	519	0.01	114	0.01	–	–	81	0.01		
$l' = 10$	138	–	30	–	–	–				
$l' = 11$	18	–	3	–	1	25				
$l' = 12$	3	–								

In Tables 4.2, the maximum run-lengths l for the 2-MWU code and 3-WMU code in [4] are $l = n_{(2)} - 1 = 11$ and $l = n_{(3)} - 1 = 12$, respectively, since the codewords in \mathcal{C}_{R8} can consist of the form $(0, ([q] \setminus 0)^{n_{(k)}-1})$. The ratios of the codewords with the run-lengths greater than three in 2-MWU code and 3-WMU code in [4] are 17.08% and 18.14%, respectively. This means that it is difficult to

use the code \mathcal{C}_{R8} for satisfying the maximum run-length constraint $l = 3$ in DNA storage systems. The 3-WMU code can be obtained from the 2-WMU code by concatenating any sequence of length one, which leads to some 2-WMU codewords with run-length $l' = 3$ become 3-WMU codewords with run-length $l' = 4$ after concatenating any sequence of length one. Therefore, the number of codewords in 3-WMU code decreases for the run-lengths $l \leq 3$. However, from the proofs of Theorems 4.3.2 and 4.3.3, it can be observed that the maximum run-lengths of our proposed codes \mathcal{C}_4 and \mathcal{C}_8 are fixed to l . In other words, if $l = 3$, all codewords (100%) in the proposed codes \mathcal{C}_4 and \mathcal{C}_8 satisfy the requirements for DNA storage.

From Table 4.2, when comparing the code \mathcal{C}_{R8} of length $n_{(3)}$ with the code \mathcal{C}_{R8-GC} , it is evident that the size of \mathcal{C}_{R8-GC} significantly decreases due to the fulfillment of the additional GC-balanced constraint. The size loss of the code is about $(1 - \frac{|\mathcal{C}_{R8-GC}|_{(3)}}{|\mathcal{C}_{R8}|_{(3)}}) = 70.22\%$. Similarly, it can be also observed that the maximum run-length l of the referenced code \mathcal{C}_{R8-GC} is nine. Additionally, the ratio of codewords with run-lengths greater than three in \mathcal{C}_{R8-GC} is measured to be 21.20%. These findings indicate that the code \mathcal{C}_{R8-GC} also fails to satisfy the maximum run-length constraint required for DNA storage. On the other hand, as shown in the proof of Theorem 4.3.7, our proposed almost GC-balanced and 3-WMU codes \mathcal{C}'_6 and \mathcal{C}''_6 meet the requirements for DNA storage.

Moreover, all codewords in the codes \mathcal{C}_{R8-H} and \mathcal{C}_{R11-GH} have run-lengths greater than three, which violates the maximum run-length constraint for DNA storage. The maximum run-lengths l of the codes \mathcal{C}_{R8-H} and \mathcal{C}_{R11-GH} are eleven and six, respectively.

Chapter 5

Conclusions and Future works

5.1 Thesis Conclusion

DNA storage, as the next-generation storage method, possesses a lot of advantages. Although errors may occur in the received DNA strands due to external factors such as synthesis, amplification, and sequencing technologies, as well as internal factors such as GC-content imbalance and long homopolymer run, the error rate caused by external factors is gradually decreasing with technological development. Additionally, some coding techniques have successfully generated DNA sequences that meet GC-balanced and short homopolymer run criteria, even at the expense of very small code rates. In this dissertation, we propose several coding methods for DNA storage, which is beneficial for constructing GC-balanced codes, error correction codes with maximum run-length constraint, and the primers codes with with maximum run-length constraint. The conclusions of this dissertation are list as follows.

- First, we propose a novel encoding algorithm to construct GC-balanced DNA codes based on weight distribution of binary data. To verify effectiveness of our proposed encoding algorithm, we compare the average parity lengths

of our proposed codes to those of referenced codes. From the comparison results of the average parity lengths, our proposed codes exhibits a better performance for most values of ϵ , which indicates that our proposed codes can provide an efficient way to construct GC-balanced codes for DNA storage. We also present a simple method to evaluate the average parity lengths of our proposed codes for various numbers of flipping levels, which shows that the average parity lengths decrease as the number of flipping levels increases and gradually tend to saturate. Especially, the proposed codes can achieve excellent parity lengths for small values of f .

- Next, we propose a nonbinary SIDEC code with the maximum run-length r constraint and its systematic encoding algorithm for the proposed code. Moreover, we verify the feasibility of the encoding and decoding processes of the proposed code. The proposed code can be applied to DNA storage systems due to the properties of this code design and the error-prone nature of DNA channels.
- Finally, we propose a class of codes that satisfy the maximum run length l and 2-WMU constraints and we also present the methods to extend 2-WMU codes with maximum run length l constraints to general k -WMU codes with maximum run length l constraints. Moreover, we extend the proposed codes to satisfy almost-balanced, the maximum run length l , k -WMU, and larger Hamming distance d constraints, simultaneously. The proofs corresponding to the proposed code constructions for code are also presented. Since the proposed code is designed to be used for primer in DNA storage, we modify the code construction to achieve the various lengths and list the sizes of our proposed codes. Finally, we compare the referenced codes with our proposed codes to verify that our proposed codes always satisfy the maximum run

length constraint for DNA storage.

5.2 Future research directions

Deletion, insertion, as well as substitution errors occurring in the received DNA strands can pose significant challenges for DNA storage systems. However, a determined channel model for DNA storage has not yet been established. Some researchers assume that the DNA channel to involve specific t and arbitrary t deletion errors for some positive integers t . Some certain algebraic error correction codes are designed to address these errors based on the assumption of the deletion errors channel. It is crucial to note that combinations of deletion and substitution errors can also occur simultaneously in DNA storage systems. While some algebraic error correction codes have been proposed to address combinations of errors, the diverse nature of error combinations remains an open issue in the design of error correction codes for DNA storage systems. The exploration of novel DNA assumptions based on combinations of errors, along with the development of corresponding error correction codes for these assumptions, which can be a potential future research direction in DNA storage. For proposed error correction codes, researchers are actively working on optimizing these proposed codes to achieve minimal redundancy to attain a higher code rate. Therefore, to optimize the redundancy for the proposed codes can also be a future research direction in DNA storage.

List of Publications

SCI(E) Journals

- [R1] X. Lu and S. Kim. “Weakly mutually uncorrelated codes with maximum run length constraint for DNA storage”. *Computers in Biology and Medicine*, 165(1):107439, 2023.
- [R2] X. Lu and S. Kim. “New Construction of Balanced Codes Based on Weights of Data for DNA Storage”. Accepted to *IEEE Transactions on Emerging Topics in Computing*, 2023.
- [R3] X. Lu and S. Kim. “Design of nonbinary error correction codes with a maximum run-length constraint to correct a single insertion or deletion error for DNA storage”. *IEEE Access*, 9(10):135354-135363, 2021.
- [R4] X. Lu, J. Jeong, J. W. Kim, J. S. No, H. Park, A. No, and S. Kim. “Error rate-based log-likelihood ratio processing for low-density parity-check codes in DNA storage”. *IEEE Access*, 8(10):162892-162902, 2020.

International Conferences

- [R5] X. Lu and S. Kim. “Log-likelihood Ratio for low-density parity-check codes under binary symmetric erasure channel in DNA storage,”. In *2022 International Conference on Advanced Technologies for Communications (ATC)*, pages 171-176, 2022.

Bibliography

- [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] M. Abroshan, R. Venkataramanan, and A. G. I. Fabregas. Efficient systematic encoding of non-binary VT codes. In *2018 IEEE International Symposium on Information Theory (ISIT)*, pages 91–95. IEEE, 2018.
- [3] S. R. Blackburn. Non-overlapping codes. *IEEE Transactions on Information Theory*, 61(9):4890–4894, 2015.
- [4] M. Blawat, K. Gaedke, I. Hütter, X.-M. Chen, B. Turczyk, S. Inverso, B. W. Pruitt, and G. M. Church. Forward error correction for DNA data storage. *Procedia Computer Science*, 80(3):1011–1022, 2016.
- [5] J. Bornholt, R. Lopez, D. M. Carmean, G L. Ceze, Seelig, and K. Strauss. A DNA-based archival storage system. *ACM SIGOPS Operating Systems Review*, 50(2):637–649, 2016.
- [6] J. Brakensiek, V. Guruswami, and S. Zbarsky. Efficient low-redundancy codes for correcting multiple deletions. *IEEE Transactions on Information Theory*, 64(5):3403–3410, 2018.

- [7] V. Risca C. T. Clelland and C. Bancroft. Hiding messages in DNA microdots. *Nature*, 399(33):533–534, 1999.
- [8] B. Cao, P. Shi, Y. Zheng, and Q. Zhang. FMG: An observable DNA storage coding method based on frequency matrix game graphs. *Computers in Biology and Medicine*, 151(1):106269, 2022.
- [9] S. Chandak, J. Neu, K. Tatwawadi, J. Mardia, B. Lau, M. Kubit, R. Hulett, P. Griffin, M. Wootters, T. Weissman, and H. Ji. Overcoming high nanopore basecaller error rates for DNA storage via basecallerdecoder integration and convolutional codes. In *2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 8822–8826. IEEE, 2020.
- [10] Y. M. Chee, H. M. Kiah, P. Purkayastha, and C. Wang. Cross-bifix-free codes within a constant factor of optimality. *IEEE Transactions on Information Theory*, 59(7):4668–4674, 2013.
- [11] Y. M. Chee, H. M. Kiah, and H. Wei. Efficient and explicit balanced primer codes. *IEEE Transactions on Information Theory*, 66(9):5344–5357, 2020.
- [12] Y. Choi, T.n Ryu, A. C. Lee, H. Choi, H. Lee, J. Park, S.H. Song, S. Kim, H. Kim, W. Park, and S. Kwon. High information capacity DNA-based data storage with augmented encoding characters using degenerate bases. *Scientific Reports*, 9(1):6582, 2019.
- [13] G. M. Church, Y. Gao, and S. Kosuri. Next-generation digital information-storage in DNA. *Science*, 337(6102):1628, 2018.
- [14] J. Davis. Microvenus. *Art Journal*, 55(1):70–74, 1996.

- [15] 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*, 7(1):84107–84121, 1997.
- [16] A. Doricchi *et al.*. Emerging approaches to DNA data storage: challenges and prospects. *ACS Nano*, 16(1):17552 – 17571, 2022.
- [17] L. Organick *et al.*. Random access in large-scale DNA data storage. *Nature Biotechnology*, 36(3):242–248, 2018.
- [18] R. Lopez *et al.*. DNA assembly for nanopore data storage readout. *Nature Communications*, 10(1):2933–2942, 2019.
- [19] S. Chandak *et al.*. 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)*, pages 147–156. IEEE, 2019.
- [20] Y. Erlich and D. Zielinski. DNA fountain enables a robust and efficient storage architecture. *Science*, 355(6328):950–954, 2017.
- [21] P. Fei and Z. Wang. LDPC codes for portable DNA storage. In *2019 IEEE International Symposium on Information Theory (ISIT)*, pages 76–80. IEEE, 2019.
- [22] R. Gabrys, H. M. Kiah, and O. Milenkovic. Asymmetric lee distance codes for DNA-based storage. In *2015 IEEE International Symposium on Information Theory (ISIT)*, pages 909–913. IEEE, 2015.
- [23] R. Gabrys and F. Sala. Codes correcting two deletions. *IEEE Transactions on Information Theory*, 65(2):965–974, 2019.

- [24] M. Gholami and M. Samadieh. Design of binary and nonbinary codes from lifting of girth-8 cycle codes with minimum lengths. *IEEE Communications Letters*, 17(4):777–780, 2013.
- [25] E. Gilbert. Synchronization of binary messages. *IEEE Transactions on Information Theory*, 6(4):470–477, 1960.
- [26] N. Goldman, P. Bertone, S. Chen, C. Dessimoz, E. M. LeProust, B. Sipos, and E. Birney. Towards practical, high-capacity, low-maintenance information storage in synthesized DNA. *Nature*, 494(7435):77–80, 2013.
- [27] R. N. Grass, R. Heckel, M. Puddu, D. Paunescu, and W. J. Stark. Robust chemical preservation of digital information on DNA in silica with errorcorrecting-codes. *Angewandte Chemie International Edition*, 54(8):2552–2555, 2015.
- [28] V. Guruswami and J. Håstad. Explicit two-deletion codes with redundancy matching the existential bound. *IEEE Transactions on Information Theory*, 67(10):6384–6394, 2021.
- [29] M. Hagiwara. On ordered syndromes for multi insertion/deletion error correcting codes. In *2016 IEEE International Symposium on Information Theory (ISIT)*, pages 625–629. IEEE, 2016.
- [30] B. Hamoum, E. Dupraz, L. C. Canencia, and D. Lavenier. Channel model with memory for DNA data storage with nanopore sequencing. In *2021 11th International Symposium on Topics in Coding (ISTC)*, pages 1–5, 2021.
- [31] S. K. Hanna and S. El Rouayheb. Guess & check codes for deletions, insertions, and synchronization. *IEEE Transactions on Information Theory*, 65(1):3–15, 2019.

- [32] R. Heckel, G. Mikutis, and R. N. Grass. A characterization of the DNA data storage channel. *Scientific Reports*, 9(1):9663, 2019.
- [33] K. A. S. Immink and K. Cai. Design of capacity-approaching constrained codes for DNA-based storage systems. *IEEE Communications Letters*, 22(2):224–227, 2018.
- [34] K. A. S. Immink and K. Cai. Properties and constructions of constrained codes for DNA-based data storage. *IEEE Access*, 8(1):49523–49531, 2020.
- [35] K.A.S. Immink. A practical method for approaching the channel capacity of constrained channels. *IEEE Transactions on Information Theory*, 43(5):1389–1399, 1997.
- [36] D. Knuth. Efficient balanced codes. *IEEE Transactions on Information Theory*, 32(1):51–53, 1986.
- [37] A. Leier, C. Richter, and H. Rauhe W. Banzhaf. Cryptography with DNA binary strands. *Biosystems*, 57(1):13–22, 2000.
- [38] A. Lenz, L. Welter, and S. Puchinger. Achievable rates of concatenated codes in DNA storage under substitution errors. In *2020 IEEE International Symposium on Information Theory and Its Applications (ISITA)*, pages 269–273. IEEE, 2020.
- [39] V. Levenshtein. Decoding automata, invariant with respect to the initial state. *Problemy Kibernet*, 12(1):125–136, 1964.
- [40] V. I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Doklady Akademii Nauk SSSR*, 163(4):845–848, 1965.

- [41] M. Li, J. Wu, J. Dai, Q. Jiang, Q. Qu, X. Huang, and Y. Wang. A self-contained and self-explanatory DNA storage system. *Scientific Reports*, 11(1):18063, 2021.
- [42] Y. J. Liu, x. He, and X. H. Tang. Capacity-achieving constrained codes with gc-content and runlength limits for DNA storage. In *2022 IEEE International Symposium on Information Theory (ISIT)*, pages 198–203. IEEE, 2022.
- [43] X. Lu and S. Kim. Log-likelihood ratio for low-density parity-check codes under binary symmetric erasure channel in DNA storage. In *2022 International Conference on Advanced Technologies for Communications (ATC)*, pages 171–176. IEEE, 2022.
- [44] D. J. C. MacKay and R. M. Neal. Near shannon limit performance of low-density parity-check codes. *Electronics Letters*, 33(6):457–458, 1997.
- [45] W. Mao, S. N. Diggavi, and S. Kannan. Models and information-theoretic bounds for nanopore sequencing. *IEEE Transactions on Information Theory*, 64(4):3216–3236, 2018.
- [46] J. Massey. Optimum frame synchronization. *IEEE Transactions on Communications*, 20(2):115–119, 1972.
- [47] M. S. Neiman. Some fundamental issues of microminiaturization. *Radiotekhnika*, 132(1):3–12, 1964.
- [48] M. S. Neiman. On the molecular memory systems and the directed mutations. *Nature*, 103(6):1–8, 1965.
- [49] T. T. Nguyen, K. Cai, K. A. Schouhamer Immink, and H. M. Kiah. Capacity-approaching constrained codes with error correction for DNA-based data storage. *IEEE Communication Letters*, 67(8):5602–5613, 2021.

- [50] I. Reed. A class of multiple-error-correcting codes and the decoding scheme. *Transactions of the IRE Professional Group on Information Theory*, 4(4):38–49, 1954.
- [51] J. H. Reif, T. H. LaBean, M. Pirrung, V. S. Rana, B. Guo, C. Kingsford, and G. S. Wickham. Experimental construction of very large scale DNA databases with associative search capability. In *The 7th International Workshop on DNA-Based Computers, Tampa, USA*, pages 231–247. SPRINGER LINK, 2002.
- [52] A. E. Shaikh, M. Welzel, D. Heider, and B. Seeger. High-scale random access on DNA storage systems. *NAR Genomics and Bioinformatics*, 4(1):126–136, 2022.
- [53] B. Shen, P. Piskunen, S. Nummelin, Q. Liu, M. A. Kostiainen, and V. Linko. Advanced DNA nanopore technologies. *ACS applied bio materials*, 3(1):5606–5619, 2020.
- [54] J. Sima and J. Bruck. On optimal k -deletion correcting codes. *IEEE Transactions on Information Theory*, 67(6):3360–3375, 2021.
- [55] J. Sima, R. Gabrys, and J. Bruck. Optimal systematic t -deletion correcting codes. In *2020 IEEE International Symposium on Information Theory (ISIT)*, pages 769–774. IEEE, 2020.
- [56] J. Sima, N. Raviv, and J. Bruck. Two deletion correcting codes from indicator vectors. *IEEE Transactions on Information Theory*, 66(4):2375–2391, 2020.
- [57] W. Song, K. Cai, M. Zhang, and C. Yuen. Codes with run-length and gc-content constraints for DNA-based data storage. *IEEE Communication Letters*, 22(10):2004–2007, 2018.

- [58] H. Takahashi and M. Hagiwara. Decoding algorithms of monotone codes and azinv codes and their unified view. In *2020 IEEE International Symposium on Information Theory and Its Applications (ISITA)*, pages 284–288. IEEE, 2020.
- [59] R. Takemoto and T. Nozaki. Encoding algorithm for run-length limited single insertion/deletion correcting code. In *2020 IEEE International Symposium on Information Theory and Its Applications (ISITA)*, pages 294–298. IEEE, 2020.
- [60] G. M. Tenengolts. Nonbinary codes, correcting single deletion or insertion. *IEEE Transactions on Information Theory*, 30(5):766–769, 1984.
- [61] K. J. Tomek, K. Volkel, A. Simpson, A. G. Hass, E. W. Indermaur, and A. J. Keung. Driving the scalability of DNA-based information storage systems. *ACS Synthetic Biology*, 32(1):1241–1248, 2019.
- [62] D. Tulpan, D. H. Smith, and R. Montemanni. Thermodynamic post-processing versus GC-content pre-processing for DNA codes satisfying the Hamming distance and reverse-complement constraints. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 11(2):441–452, 2014.
- [63] R. R. Varshamov and G. M. Tenengolts. Codes correcting single asymmetric errors. *Avtomatika i Telemekhanika*, 26(2):288–292, 1965.
- [64] Y. Wang, M. Noor-A-Rahim, E. Gunawan, Y. L. Guan, and C. L. Poh. Construction of bio-constrained code for DNA data storage. *IEEE Communications Letters*, 23(6):963–966, 2019.
- [65] Y. Wang, M. Noor-A-Rahim, J. Zhang, E. Gunawan, Y. L. Guan, and C. L. Poh. Oligo design with single primer binding site for high-capacity DNA-

- based data storage. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 17(6):2176–2182, 2020.
- [66] J. D. Watson and F. H. Crick. Molecular structure of nucleic acids. *Nature*, 171(5):737–738, 1953.
- [67] A. J. D. L. V. Wijngaarden and T. J. Willink. Frame synchronization using distributed sequences. *IEEE Transactions on Communications*, 48(12):2127–2138, 2000.
- [68] C. Winston, L. Organick, D. Ward, L. Ceze, K. Strauss, and Y.-J. Chen. Combinatorial PCR method for efficient, selective oligo retrieval from complex oligo pools. *ACS Synthetic Biology*, 302(1):10482, 2022.
- [69] Y. Xin and I. J. Fair. Algorithms to enumerate codewords for DC/sup 2/-constrained channels. *IEEE Transactions on Information Theory*, 47(7):3020–3025, 2001.
- [70] T. Xue and F. C. M. Lau. Construction of GC-balanced DNA with deletion/insertion/mutation error correction for DNA storage system. *IEEE Access*, 8(2):140972–140980, 2020.
- [71] Z. Yan, C. Liang, and H. Wu. A segmented-edit error-correcting code with re-synchronization function for DNA-based storage systems. *IEEE Transactions on Emerging Topics in Computing*, 2012. early access.
- [72] C. N. Yang and D. J. Lee. Some new efficient second-order spectral-null codes with small lookup tables. *IEEE Transactions on Computers*, 55(7):924–927, 2006.
- [73] S. M. H. T. Yazdi, R. Gabrys, and O. Milenkovic. Portable and error-free DNA-based data storage. *Scientific Reports*, 7(1):5011, 2017.

- [74] S. M. H. T. Yazdi, H. M. Kiah, R. Gabrys, and O. Milenkovic. Mutually uncorrelated primers for DNA-based data storage. *IEEE Transactions on Information Theory*, 64(9):6283–6296, 2018.
- [75] S. M. H. T. Yazdi, H. M. Kiah, E. Garcia-Ruiz, J. Ma, H. Zhao, and O. Milenkovic. DNA-based storage: Trends and methods. *IEEE Transactions on Molecular, Biological, and Multi-Scale Communications*, 1(3):230–248, 2015.
- [76] S.M.H.T. Yazdi, Y. Yuan, H.Z. J. Ma, and O. Milenkovic. A rewritable, random-access DNA-based storage system. *Scientific Reports*, 5(3):14138, 2015.